

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ЕЛЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. И.А. БУНИНА»

**И.И. Васильева, С.Е. Попов**

# **ТЕОРИЯ И ПРАКТИКА ПРОГРАММИРОВАНИЯ**

**Учебно-методическое пособие**

Елец - 2022

УДК  
ББК

*Печатается по решению редакционно-издательского совета  
Елецкого государственного университета им. И.А. Бунина  
от \_\_\_\_\_, протокол № \_\_\_\_\_*

### **Рецензенты:**

**И.И. Васильева, С.Е. Попов**

\_\_\_\_\_ Теория и практика программирования: учебно-методическое пособие. – Елец: Елецкий государственный университет им. И.А. Бунина, 2022. - \_\_\_\_\_ с.

В учебно-методическом пособии приводятся задачи, решаемые с помощью языка программирования JavaScript. В конце каждой главы приводятся контрольные вопросы и задания по теме. Данное учебное пособие предназначено для студентов СПО, обучающихся по специальностям 09.02.03 – «Программирование в компьютерных системах», 09.02.07 – «Информационные системы и программирование». Оно может быть использовано учителями информатики в рамках факультативных занятий в старших классах, а также преподавателями в колледжах и училищах.

УДК  
ББК

© Елецкий государственный университет им. И.А. Бунина, 2022

© И.И. Васильева, С.Е. Попов, 2022

## Оглавление

Введение .....	6
Глава 1. Основы программирования на JavaScript. Линейные программы. ....	9
1.1. Синтаксис и семантика языка .....	9
1.2. Основные операции и операторы .....	11
1.3 Типы данных.....	13
1.4. Класс Math и математические функции.....	15
1.5. Организация ввода-вывода.....	16
1.6. Связь функций ввода-вывода с технологией DOM.....	22
1.7. Ввод-вывод с помощью консольной библиотеки Node.js .....	27
1.8 Линейные скрипты и их внедрение на веб-страницу .....	29
1.9 Контрольные вопросы и тесты .....	36
1.10 Задачи для самостоятельной работы.....	44
Глава 2. Организация ветвлений.....	49
2.1. Логические операторы и операции отношений .....	49
2.2. Оператор ветвления .....	52
2.3. Оператор выбора варианта.....	57
2.4. Скрипты с разветвляющимися алгоритмами .....	61
2.5. Контрольные вопросы и тесты .....	78
2.6. Задачи для самостоятельной работы.....	83
Глава 3. Циклические конструкции.....	90
3.1. Цикл с параметром и его особенности.....	90
3.2. Цикл с предусловием .....	98
3.3. Цикл с постусловием .....	101
3.4. Условия досрочного выхода из цикла .....	104
3.5. Вложенные операторы цикла.....	110
3.6. Скрипты с циклическими конструкциями .....	115
3.7. Контрольные вопросы и тесты .....	125
3.8. Задачи для самостоятельной работы.....	131
Глава 4. Массивы.....	137
4.1. Одномерные массивы .....	137

Определение массива.....	138
4.2 Циклы при работе с массивами .....	146
4.3. Основные функции работы с массивами.....	155
4.4. Двумерные массивы.....	159
4.5. Скрипты, содержащие массивы.....	161
4.6. Контрольные вопросы и тесты .....	170
4.7. Задачи для самостоятельной работы.....	175
Глава 5. Функции пользователя .....	182
5.1. Синтаксис функций пользователя в JavaScript .....	182
5.2. Область видимости объектов и переменных.....	185
5.3. Характеристики функций .....	186
5.4. Упаковка и распаковка аргументов.....	190
5.5. Юнит-тесты.....	196
5.6. Скрипты, содержащие функции пользователя .....	200
5.7. Контрольные вопросы и тесты .....	208
5.8. Задачи для самостоятельной работы.....	214
Глава 6. Работа с символами и строками .....	222
6.1. Организация работы со строковыми объектами в JS .....	222
6.2. Класс String .....	226
6.3. Регулярные выражения.....	232
6.4. Работа с числами и массивами с помощью строк.....	235
6.5. Скрипты, содержащие символы и строки .....	236
6.6. Контрольные вопросы и тесты .....	243
6.7. Задачи для самостоятельной работы.....	248
Глава 7. Файловая система и динамические структуры данных.....	253
7.1. Работа с файлами с помощью языка Node.js.....	253
7.2. Коллекции данных .....	262
7.3. Классы и объекты.....	271
7.4. Итераторы и генераторы последовательностей .....	274
7.5. Динамические структуры .....	277

7.6. Скрипты, содержащие структуры и объекты.....	281
7.7. Контрольные вопросы и тесты .....	294
7.8. Задачи для самостоятельной работы.....	299
Глава 8. Работа с графикой в JS .....	306
8.1. Возможности изображения фигур на холсте .....	306
8.2. Рекурсия и фракталы .....	318
8.3. Работа с модулем «Черепашка».....	321
8.4. Анимация .....	323
8.5. Загрузка графических файлов.....	326
8.6. Скрипты, содержащие графические примитивы .....	331
8.7. Контрольные вопросы и тесты .....	345
8.8. Задачи для самостоятельной работы.....	350
Список используемых источников.....	355

## Введение

Жизнь людей постепенно перемещается в интернет. Статический текст уходит на второй план. Ввести учетные данные социальной сети, купить или продать товар, пообщаться онлайн с живым человеком или даже программой-роботом. За всеми этими действиями стоят специальные скрипты – программы, запускаемые пользователем или сервером, прямо со странички браузера.

Самый популярный язык веб-программирования на сегодняшний день – JavaScript и его модификации. В первую очередь он предназначен для взаимодействия пользователя с всемирной паутиной. Но может ли он стать заменой классическим «настольным» системам?

В настоящем пособии приводятся классические задачи программирования с учетом особенностей frontend-разработки. Но некоторые задачи с помощью этих средств решить нельзя, например, через браузер не поддерживается работа с файлами пользователя. Поэтому в качестве backend-разработки для отдельных задач можно взять язык Node.js.

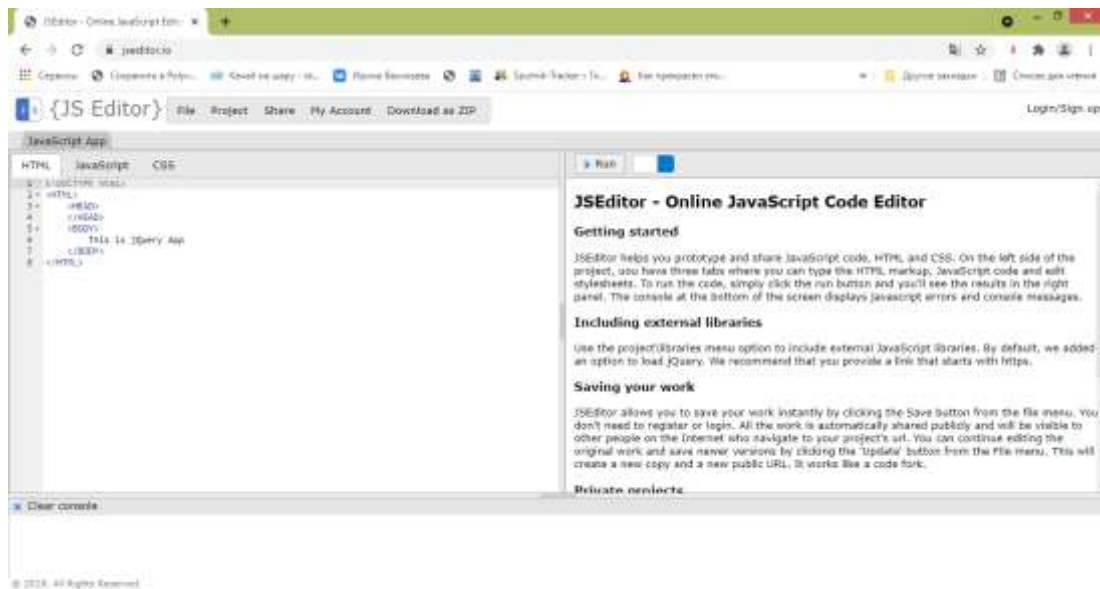
В каноническом варианте разработки скрипт записывается в текстовый файл с расширением .js и подгружается к веб-странице. За отладкой скрипта и проверкой результата приходится следить с помощью инструмента браузера – Консоль разработчика. Но для ускорения работы над кодом удобнее пользоваться онлайн-средами разработки.

Наиболее популярные из них:

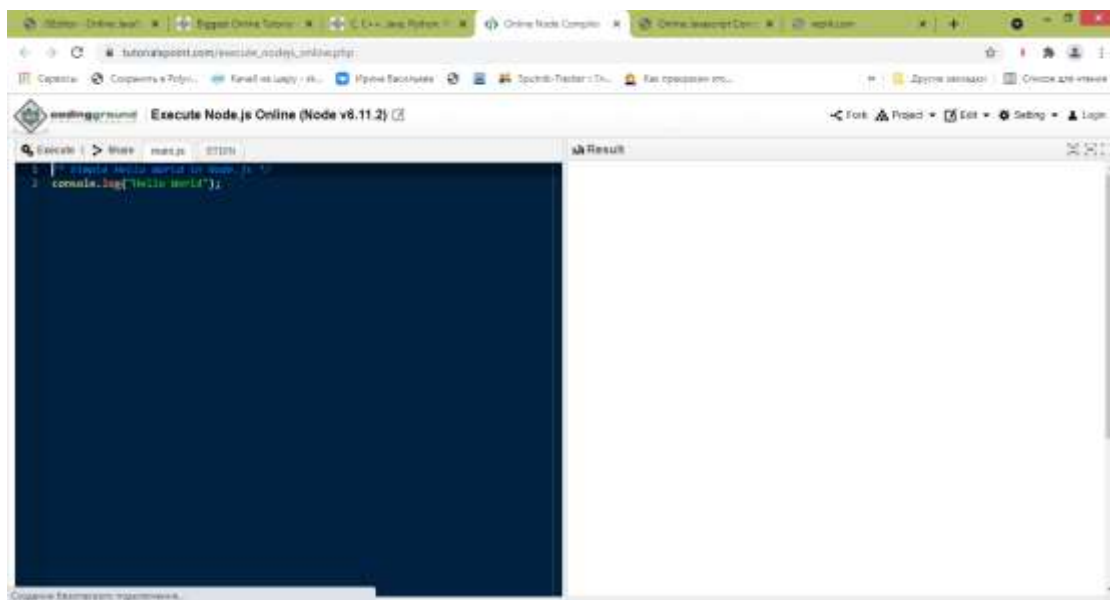
JSEditor ([jseditor.io](https://jseditor.io));

Tutorialspoint ([tutorialspoint.com/codingground.htm](https://www.tutorialspoint.com/codingground.htm));

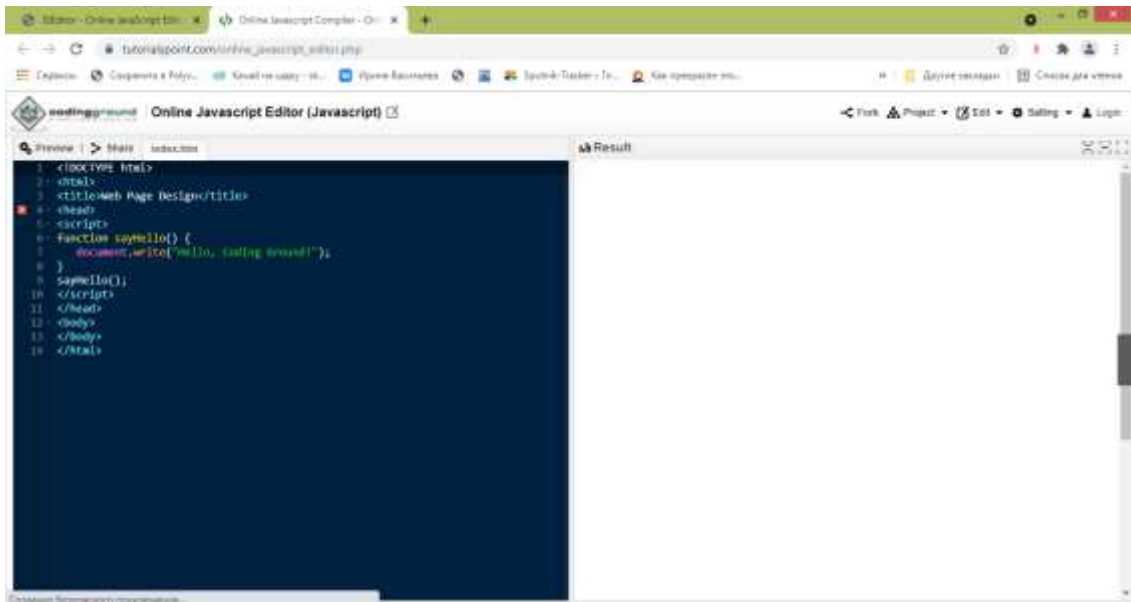
Repl.it! ([repl.it.com](https://repl.it)).



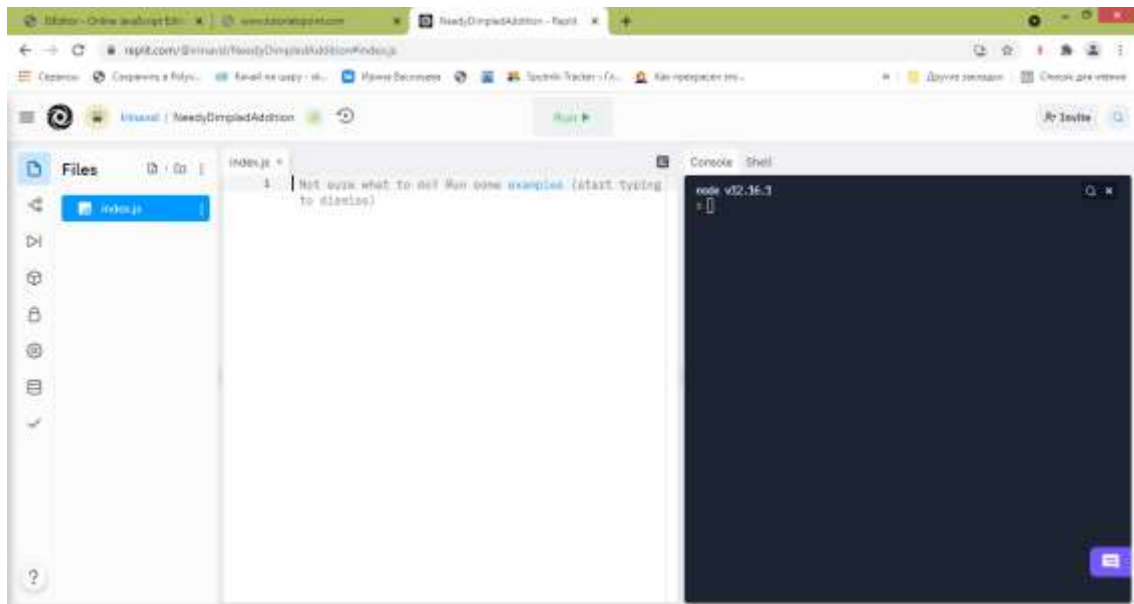
Окно JSEditor



Окно Tutorialspoint (Node.js)



Окно Tutorialspoint (JavaScript)



Окно Repl.it (Node.js)



# Глава 1. Основы программирования на JavaScript.

## Линейные программы.

### 1.1. Синтаксис и семантика языка

Изучения любого языка, не только для программирования, начинается с алфавита. Код на языке JavaScript записывается в виде последовательности символов, к числу которых относятся буквы латинского алфавита, арабские цифры, знаки препинания, знаки математических операций и специальные символы. Для обозначения исходных данных и результатов вычислений употребляются идентификаторы. Существует ряд ключевых слов (например, `if`, `for`, `float`), которые нельзя использовать в качестве идентификаторов...

Стоп. Где-то уже такое было. Да где угодно! В том же C++. Но чем отличается код программы, написанной на C++, от скрипта JavaScript? Особенности использования конструкций и вложенным в них смыслом.

Синтаксис – это правила написания кода, по которым компилятор или интерпретатор сможет перевести программу на машинный язык.

Семантика – это логический смысл программы. Что дает в итоге та или иная программная конструкция?

Например, запись вида

```
c=a+b
```

может трактоваться по-разному в зависимости от языка программирования.

В Паскале знак равенства использовался в качестве логической операции при сравнении двух операндов. В C++ это – операция сложения двух числовых аргументов. А в JavaScript – это соединение любых объектов в зависимости от их начальных значений. Чтобы сложить два числа с помощью операции «+», надо явно указать тип операндов, иначе можно получить значение 35 вместо 8.

Снова вернемся к идентификаторам. Что означают `a`, `b`, `c` в приведенном выше примере? По сути – это переменные, т.е. элементы программы, предназначенные для хранения данных в процессе выполнения кода.

Это определение верно для языков программирования со строгой типизацией. Точнее, для структурных языков.

Записали в том же C++ оператор:

```
int a,b,c;
```

и тем самым определили ячейки памяти для хранения трех целых чисел, значения которых можно менять в процессе написания или исполнения программы.

Какие операции определены над целыми числами? Строго говоря, сложение, вычитание и умножение. И то при условии, что результат тоже попадет в диапазон допустимых значений, не вызывая переполнения памяти.

С делением начинаются проблемы. Как 5 разделить на 2? Невежливо попросить переопределить тип результата на вещественный (Паскаль), отбросить дробную часть, округляя ответ до целого (C++). А можно ли обойтись без этих сложностей?

В JavaScript можно.

```
let a=5
```

```
let b=2
```

```
let c=a/b
```

```
console.log(c)
```

Были целочисленные операнды, стал вещественный результат.

Такое решение стало возможным благодаря тому, что JS не является строго типизированным языком, да и понятия переменной в нём тоже нет. Есть термин объект – некий вид данных, над которым можно выполнять различные операции,

порой приводящие не только к изменению значений, но и к изменению самого объекта.

Новички в программировании привыкают к этому сразу, а приверженцы Паскаля и Си называют объект переменной, но понимая, что эти понятия не всегда синонимы.

## 1.2. Основные операции и операторы

Операция присваивания перекочевала в JS из C++:

Объект = Выражение

Кроме того, доступны и множественные присваивания. Например:

```
let a=b=5
```

```
let c=a/b
```

```
console.log(c)
```

И составные присваивания:

```
let a=5
```

```
let b=2
```

```
a/=b
```

```
console.log(a)
```

Чтобы перебрать все варианты, надо рассмотреть все математические, логические и прочие операции. Они почти стандартны для многих языков программирования. По крайней мере, в JS и C++ всё одинаково.

+ - это и сложение чисел, и соединение строк, конечно, если количество операндов не менее двух. В унарном исполнении «+» - конвертация объекта в число. В двойном написании – инкремент.

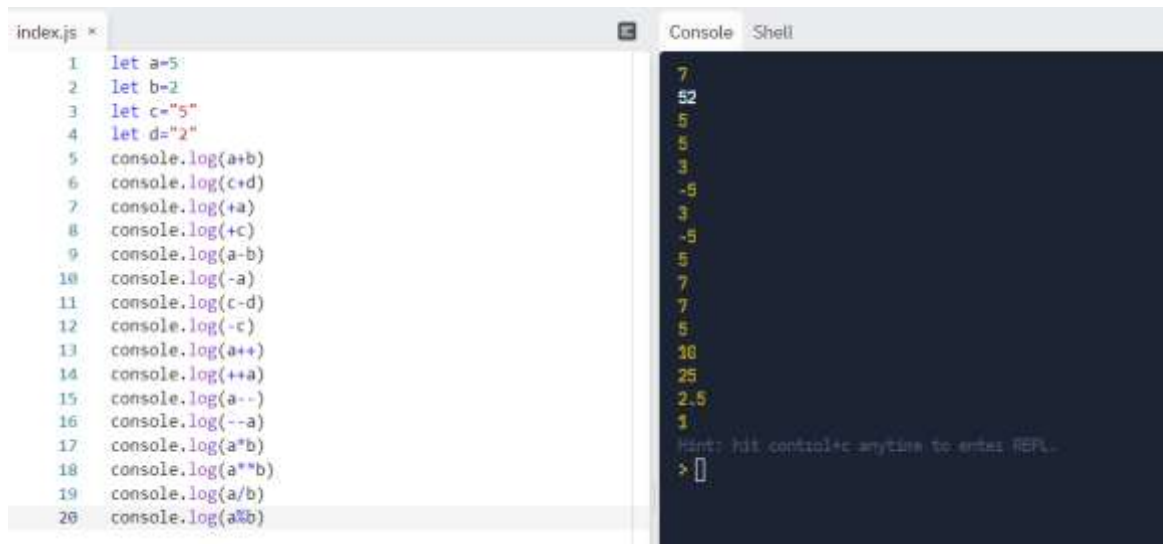
- - не только разность двух чисел, но и смена знака для одного числа. В двойном написании – декремент.

\* - умножение чисел. В двойном написании – возведение числа в степень.

/ - уже знакомое нам деление

% - получение остатка от деления первого числа на второе

Выведем все операции в одном скрипте:



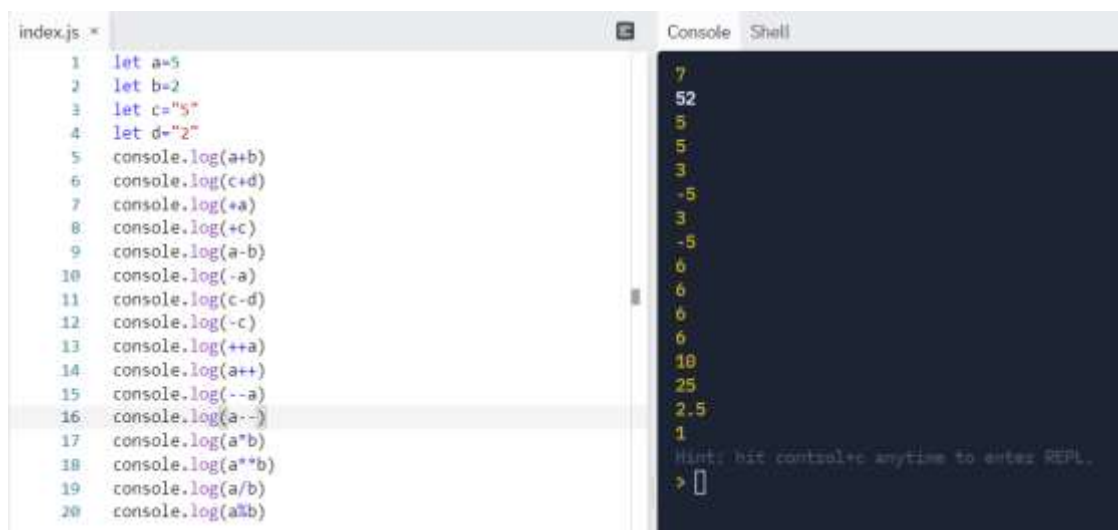
```
index.js *
1 let a=5
2 let b=2
3 let c="5"
4 let d="2"
5 console.log(a+b)
6 console.log(c+d)
7 console.log(+a)
8 console.log(+c)
9 console.log(a-b)
10 console.log(-a)
11 console.log(c-d)
12 console.log(-c)
13 console.log(++a)
14 console.log(++a)
15 console.log(a--)
16 console.log(--a)
17 console.log(a*b)
18 console.log(a**b)
19 console.log(a/b)
20 console.log(a%b)
```

Console Shell

```
7
52
5
5
3
-5
3
-5
5
7
7
5
10
25
2.5
1
Hint: hit control+c anytime to enter REPL.
> 
```

Обратите внимание, как интересно получилось с префиксной и постфиксной формами инкремента и декремента. В ответе не вывелось число 6, а сначала мысленно увеличили 5 на 1, а потом еще раз увеличили второй операцией.

Переставим строки 13, 14 и 15,16 попарно местами:



```
index.js *
1 let a=5
2 let b=2
3 let c="5"
4 let d="2"
5 console.log(a+b)
6 console.log(c+d)
7 console.log(+a)
8 console.log(+c)
9 console.log(a-b)
10 console.log(-a)
11 console.log(c-d)
12 console.log(-c)
13 console.log(++a)
14 console.log(a++)
15 console.log(--a)
16 console.log(a--)
17 console.log(a*b)
18 console.log(a**b)
19 console.log(a/b)
20 console.log(a%b)
```

Console Shell

```
7
52
5
5
3
-5
3
-5
6
6
6
6
10
25
2.5
1
Hint: hit control+c anytime to enter REPL.
> 
```

А вот и наша шестерка. Префикс ее сразу выводит, а постфикс только запоминает.

Логические операторы и операции отношений целесообразно изучить в теме «Организация ветвлений».

### 1.3 Типы данных

В самом начале мы решили, что переменных в объектных языках не существует, тем более, не нужно знать их типы, но первый пример показал, что без них в JavaScript не обойтись.

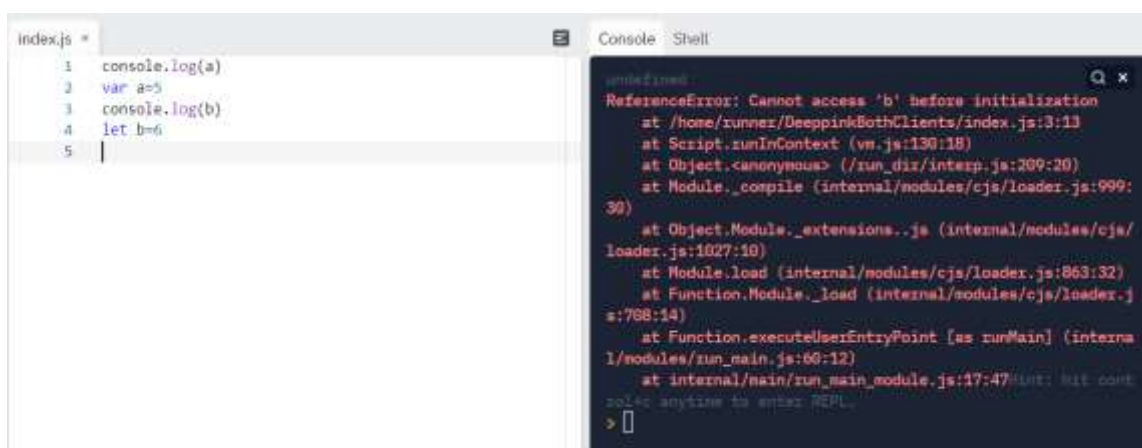
Условно можно поделить данные на следующие виды: number (числа), string (строки), Boolean (логический тип), function (функции) и object (специальные объекты).

Объявить переменную можно с помощью ключевого слова, идентификатора и указания начального значения.

В JavaScript ключевых слов целых три, на выбор – var, let, const. Что у них общего и чем отличаются?

Объявление через var характерно для старых версий браузеров, а для нового стандарта лучше использовать let.

Для простых задач, в которых сначала используется объявление переменной, а затем работа с ней, принципиальной разницы нет. Но если поменять порядок, то разница заметна:



```
index.js *
1 console.log(a)
2 var a=5
3 console.log(b)
4 let b=6
5

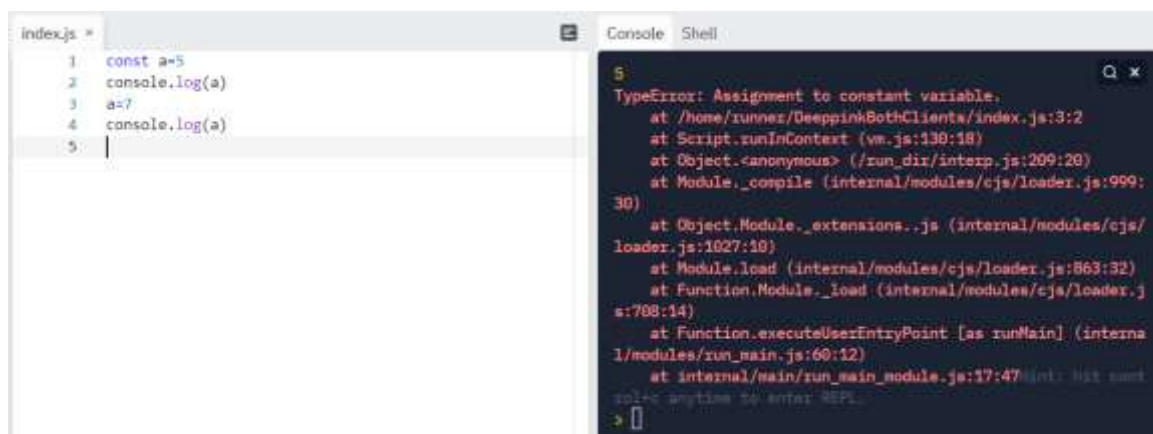
Console Shell
ReferenceError: Cannot access 'b' before initialization
    at /home/runner/DeeppinkBothClients/index.js:3:13
    at Script.runInContext (vm.js:130:18)
    at Object.<anonymous> (/run_dir/interp.js:209:20)
    at Module._compile (internal/modules/cjs/loader.js:999:39)
    at Object.Module._extensions.js (internal/modules/cjs/loader.js:1627:10)
    at Module.load (internal/modules/cjs/loader.js:963:32)
    at Function.Module._load (internal/modules/cjs/loader.js:768:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:60:12)
    at internal/main/run_main_module.js:17:47 [v8]: hit comb
    > |
```

Если в первом случае с var мы получим неопределенное значение undefined, то во втором (с let) – сообщение об ошибке.

Исходя из этих соображений, второй способ предпочтительней, он позволяет контролировать действия программиста при добавлении в скрипт новых переменных.

Ключевое слово `const` используется при объявлении константы – неизменяемого значения (если речь идет о переменной, а не объекте). В новом стандарте для JS роль этого ключевого слова гораздо шире, но в данном контексте это то, что нельзя изменить в процессе выполнения скрипта, как в математике число  $\pi$  или гравитационная постоянная в физике.

Рассмотрим пример:



```
index.js =
1  const a=5
2  console.log(a)
3  a=7
4  console.log(a)
5

Console Shell
5
TypeError: Assignment to constant variable.
    at /home/runner/DeeppinkBothClients/index.js:3:2
    at Script.runInContext (vm.js:139:18)
    at Object.<anonymous> (/run_dir/interp.js:299:28)
    at Module._compile (internal/modules/cjs/loader.js:999:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1027:10)
    at Module.load (internal/modules/cjs/loader.js:863:32)
    at Function.Module._load (internal/modules/cjs/loader.js:788:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:60:12)
    at internal/main/run_main_module.js:17:47
>
```

При первом объявлении константы `a` ее значение благополучно интерпретировалось, но при повторном присваивании значения появилось сообщение об ошибке.

В объявлении `const` происходит ссылка не на значение, а на объект, точнее на адрес объекта.

Идеологически важно, что объявляя

```
const a = array1,
```

мы ссылаемся на область в памяти, где лежит `array1`. Следовательно, мы можем менять содержание объекта, класса, но пересоздать его либо создать новый мы не можем.

Сделаем такой вывод: если мы затрудняемся с выбором ключевого слова, то лучше всегда использовать `let`, оставляя `const` для особых случаев (массивов,

функций), а var может затесаться в наш скрипт при копировании кода из старых учебников.

#### 1.4. Класс Math и математические функции.

Встроенные классы, содержащие основные функции и свойства, облегчают работу программиста. Благодаря Math, можно не изучать долго основы объектно-ориентированного программирования, классы и методы, а сразу приступить к решению популярных математических задач.

В математике неизвестная переменная – это аргумент, при подстановке его значения в функциональное выражение получим значение функции. Пара вида (x,y) однозначно определяет точку на плоскости.

В структурном программировании функция – это фрагмент программы с собственным именем, возвращающий определенное значение. В объектно-ориентированном программировании метод – это группа действий, выполняемых над объектами.

Иными словами,  $y=\sin(x)$  – это метод класса Math, позволяющий вычислить синус числового объекта. Но в результате наши действия сведутся к математике.

метод	описание	пример	результат
abs()	модуль числа	Math.abs(-5)	5
acos()	арккосинус числа	Math.acos(1)	0
asin()	арксинус числа	Math.asin(1)	1.57
atan()	арктангенс числа	Math.atan(1)	0.78
ceil()	округление вверх до наименьшего целого	Math.ceil(3.5)	4
cos()	косинус числа	Math.cos(0.15)	0.99
exp()	$e^x$	Math.exp(1)	2.7

floor()	округление вниз до наибольшего целого	Math.floor(3.5)	3
log()	натуральный логарифм ln(x)	Math.log(2.7)	0.99
max()	значение наибольшего из чисел	Math.max(2,4,1)	4
min()	значение наименьшего из чисел	Math.min(2,4,1)	1
pow()	степень числа	Math.pow(2,4)	16
random()	генерирование случайного числа в диапазоне от 0 до 1	Math.random()	например, 0.44
round()	округление до ближайшего целого числа	Math.round(3.5)	4
sin()	синус числа	Math.sin(0.55)	0.52
sqrt()	квадратный корень из числа	Math.sqrt(36)	6
tan()	тангенс числа	Math.tan(0.8)	1.03

### 1.5. Организация ввода-вывода

Работа в любом приложении заключается в обмене данными между пользователем и системой.

Приведем решение задачи на сложение двух чисел с практической точки зрения. Допустим, заработная плата сотрудника состоит из аванса и



фиксированного оклада. Вычислим полное значение заработка. Чтобы получить итоговую сумму, необходимо знать значения обоих слагаемых.

Можно это сделать совсем просто:

```
//присваиваем 2 значения
let avans=5000
let poluchka=11700
//считаем сумму
let zarplata=avans+poluchka
//выводим результат в консоль разработчика
console.log(zarplata)
```

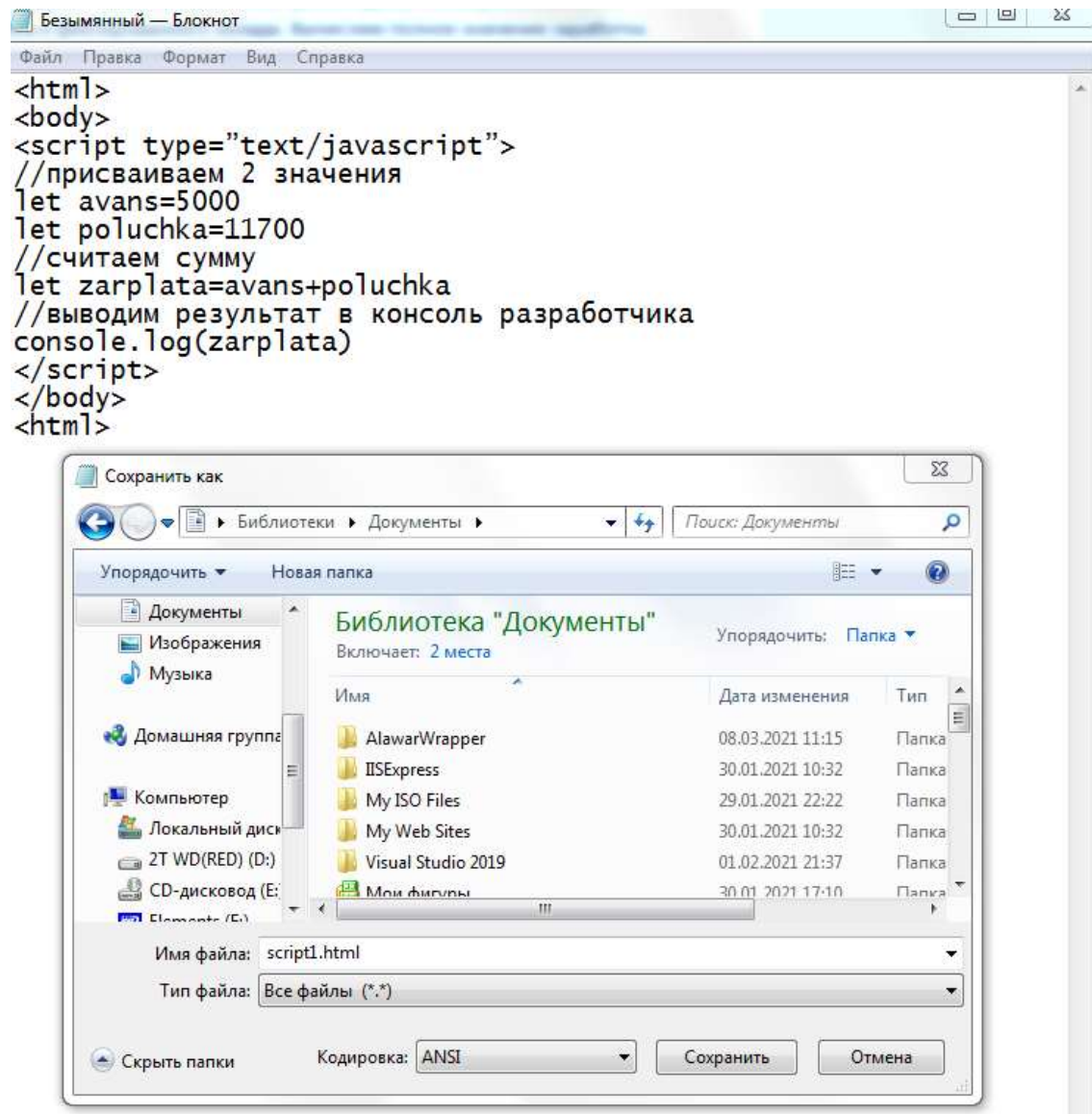
Но такое решение устроит только программиста, а не пользователя. Во-первых, значения переменных могут измениться. Не у всех сотрудников в фирме одинаковые оклады. Во-вторых, пользователь сети интернет имеет дело с готовой веб-страницей, загруженной в браузере. И результат он тоже должен видеть на этой странице.

Покажем это на примере. В обычном Блокноте, встроенном в операционную систему Windows, наберем текст следующего содержания:

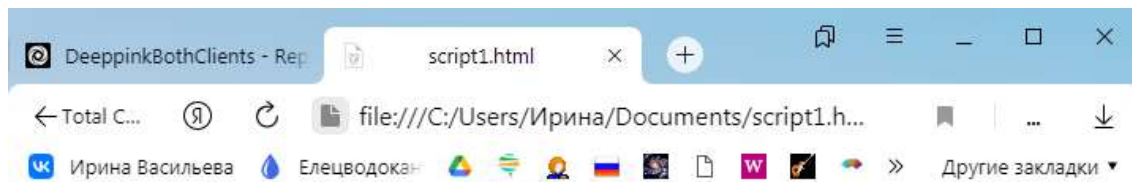
```
<html>
<body>
<script type="text/javascript">
...
</script>
</body>
<html>
```

Это заготовка для будущих скриптов. Вместо многоточия наберем текст предыдущей программы и сохраним полученный файл с расширением .html.

Примечание: в строке `<script type="text/javascript">` для некоторых версий браузера кавычки нужно убрать.



В итоге получим...



... пустую страницу.

Но если в браузере вызвать команду Настройки Яндекс Браузера – Дополнительно – Дополнительные инструменты – Консоль JavaScript, то можно увидеть наш результат:



Но такой вариант не подходит пользователям, которые занимаются веб-серфингом, а не вникают в инструменты разработчика. А задача frontend-разработчика как раз и состоит в предоставлении пользователю готового решения.

В некоторых средах программирования для операционной системы Windows предусмотрен мощный инструмент интерфейса – диалоговые окна, позволяющие обеспечивать ввод данных и выводить сообщения. Подобный механизм предусмотрен и в JavaScript, благодаря объекту Window объектной модели браузера.

Из теории объектно-ориентированного программирования известно, что объект должен иметь свойства и методы. Какие действия можно производить в

окне, представляющем собой страницу браузера? Например, открытие небольшого всплывающего окна с сообщением.

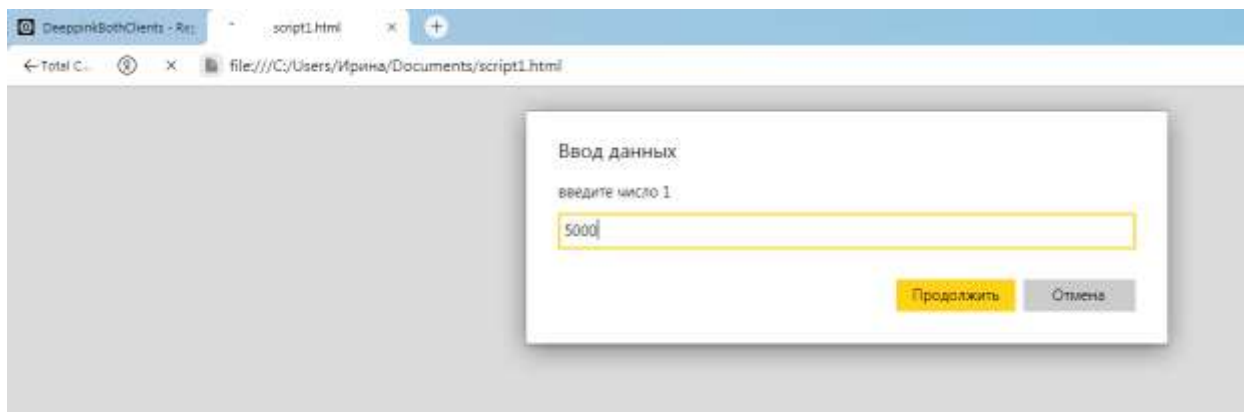
Метод `prompt()` отображает окно с полем ввода. Данные из этого окна поступают на обработку скрипту, т.е. можно присвоить переменной в качестве значения не конкретное число, а записанное в этом окне.

Метод `alert()` отображает окно с сообщением. В качестве сообщения тоже можно использовать значение переменной, только заранее известное, например, посчитанный результат.

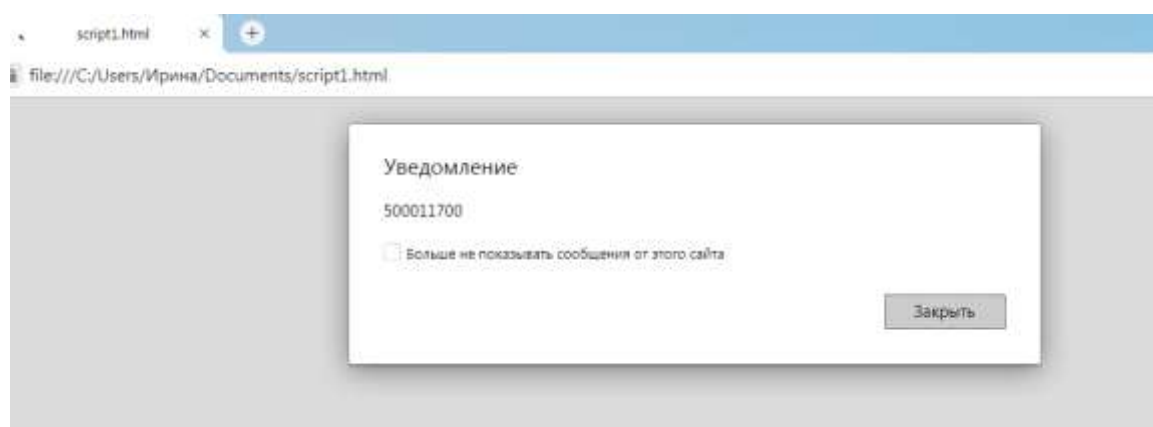
Изменим фрагмент кода с учетом полученной информации:

```
<html>
<body>
<script type=text/javascript>
//присваиваем 2 значения
var avans=prompt("введите число 1","0")
var poluchka=prompt("введите число 2","0")
//считаем сумму
let zarplata=avans+poluchka
//выводим результат в консоль разработчика
alert(zarplata)
</script>
</body>
```

<html>



Аналогично вводим второе число. А вот и результат:



А почему он неправильный? Дело в том, что информация в окне ввода интерпретируется как символьная, поэтому вместо сложения двух чисел интерпретатор JavaScript записал две строки рядом.

То же самое получилось и в Node.js Replit:



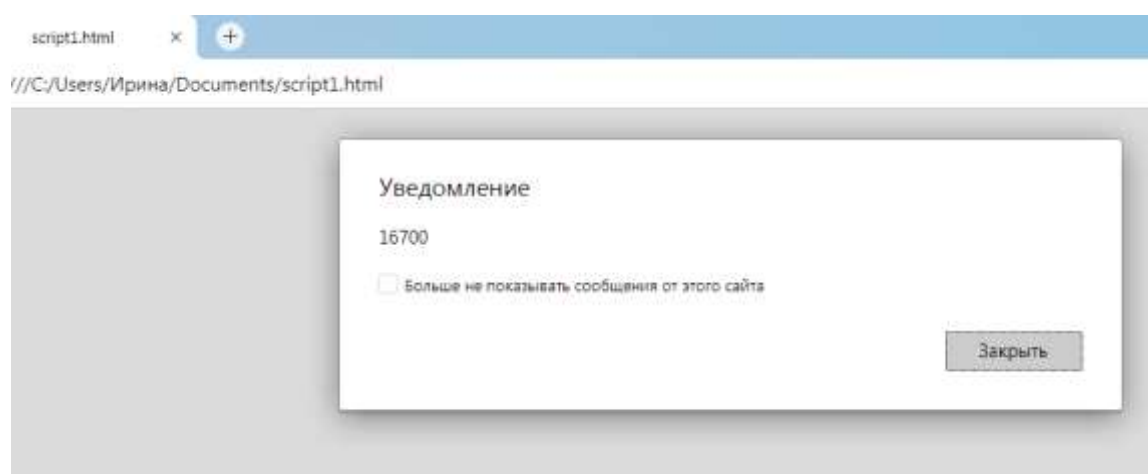
Для отображения корректного результата надо преобразовать строковое значение в числовое, причем, учитывая, какой тип чисел необходим.

Для целого числа это метод `parseInt()`, а для дробного – `parseFloat()`.

Изменим наш скрипт:

```
//присваиваем 2 числовых значения  
var avans=parseInt(prompt("введите число 1","0"))  
var poluchka=parseInt(prompt("введите число 2","0"))  
  
//считаем сумму  
let zarplata=avans+poluchka  
  
//выводим результат  
alert(zarplata)
```

Получим:



## 1.6. Связь функций ввода-вывода с технологией DOM

Каждый из рассмотренных выше методов ввода-вывода имеет свои недостатки. Особенно остро они проявляются при вводе или выводе большого количества информации. Кнопки «Продолжить» и «Закреть» надоест нажимать уже после третьего значения.

Как удобно работать с формой Windows – всё ввёл в текстовые поля и нажал единственную кнопку. И результаты отобразились сразу на форме. Даже простой ввод данных в консольном режиме старого языка программирования под DOS и то казался удобнее.

Но что мешает сделать это в скрипте, а потом вывести информацию прямо на страницу в браузере? Мы же вводим учетные данные для регистрации на сайте социальной сети или интернет-магазина, для других задач тут тоже самое.

В языке HTML есть форма с элементами управления, а в JavaScript - объект `document`, который отвечает за взаимодействие JavaScript с объектом документа (страницей в браузере). Этот объект является свойством объекта `Window` и приходится родителем всех других объектов на веб-странице. Такая технология называется DOM – объектная модель документа.

На данной форме 4 элемента управления: два поля ввода, кнопка и поле вывода. Сами элементы необходимо прописать в виде тегов на языке HTML:

```
<html>
<body>
<input type="text" id="avans">
<input type="text" id="poluchka">
<input id="ab" type="button" value="OK" onclick="press()">
<p id="vivod"></p>
... (код скрипта)
</body>
</html>
```

После ключевого слова `input` указывается вид элемента для ввода: `text` (поле для однострочного текста) или `button` (кнопка). Затем указывается идентификационный номер объекта (`id`). Это внутреннее имя, по которому скрипт определяет, где какое значение нужно брать.

Для кнопки нужно указать два дополнительных параметра: `value` (название самой кнопки, например, «OK») и `onclick` - функции для обработки события.

Что будет являться обработчиком события нажатия левой кнопки мыши? Сама наша программа (код скрипта).

```
<script type=text/javascript>
```

```
var avans1,poluchka1,ab, out //имена глобальных переменных, которые свя-  
заны с элементами управления
```

```
function press() //имя функции – обработчика события
```

```
{
```

```
let avans=parseInt(avans1.value) //связь числовой переменной avans со зна-  
чением, взятым из соответствующего поля ввода avans1
```

```
let poluchka=parseInt(poluchka1.value)
```

```
//считаем сумму
```

```
let zarplata=avans+poluchka
```

```
//выводим результат в специальное поле для вывода
```

```
out.innerHTML=zarplata
```

```
}
```

```
window.onload=function() //функция загрузки формы на веб-страницу
```

```
{
```

```
avans1=document.getElementById("avans"); //получение ссылки на элемент  
управления документа по его id
```

```
poluchka1=document.getElementById("poluchka");
```

```
ab=document.getElementById("ab");
```

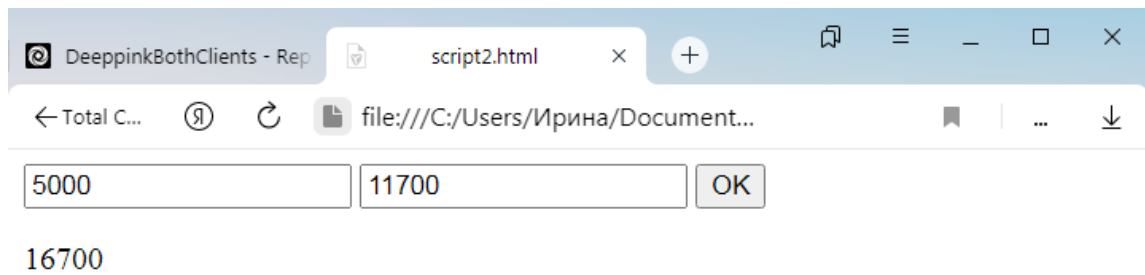
```
out=document.getElementById("vivod");
```

```
}
```

```
</script>
```

В результате получим полноценное веб-приложение:





Кроме `document.getElementById` – обращения к элементу управления документа по его `id` – существует обращение по имени (`document.getElementsByName`) и выбор элемента по запросу (`document.querySelector`). JavaScript метод `document.querySelector()` возвращает первый элемент в документе, соответствующий указанному селектору, или группе селекторов.

Вывод внутри документа тоже можно осуществить другим способом – с помощью метода `write` (`writeln`). Отличие от `innerHTML` (`innerText`) состоит в том, что в последнем случае содержимое объекта изменится при изменении его значения, а в случае с `document.write()` вывод будет окончательным.

В любом случае в примере выше производилась так называемая frontend-разработка. Сценарий выполнялся на стороне клиента. При нажатии на кнопку подгружался обработчик соответствующего события из этого же скрипта.

В языке HTML есть особый вид кнопки – `submit` – для отправки данных из формы. Эту кнопку тоже можно применить при решении той же задачи:

```
<html>
```

```
<body>
```

```
    Введите первое число <input class="avans"><br> //связывание поля  
ввода с запросом на него через переменную
```

```
    Введите второе число <input class="poluchka"><br>
```

```
    <input type="submit" class="button"> //стандартная кнопка Submit -  
отправить
```

```
<script type=text/javascript>
```

```
let avans1 = document.querySelector('.avans'); //связывание переменной с
```

ПОЛЕМ ВВОДА

```
let poluchka1 = document.querySelector('.poluchka');
```

```
let button = document.querySelector('.button');
```

```
// функция-обработчик события при нажатии на кнопку
```

```
button.addEventListener('click', () => {
```

```
    let avans=parseInt(avans1.value); //запрос значения переменной из
```

ПОЛЯ ВВОДА

```
    let poluchka=parseInt(poluchka1.value);
```

```
    let zarplata=avans+poluchka;
```

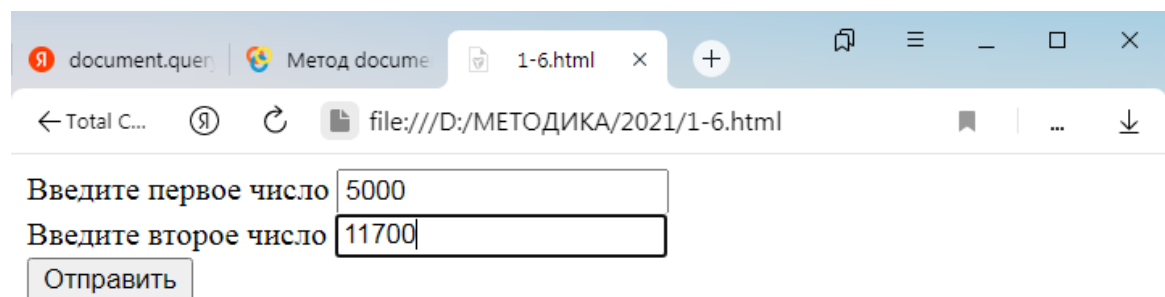
```
    document.write(zarplata) //Вывод результата
```

```
});
```

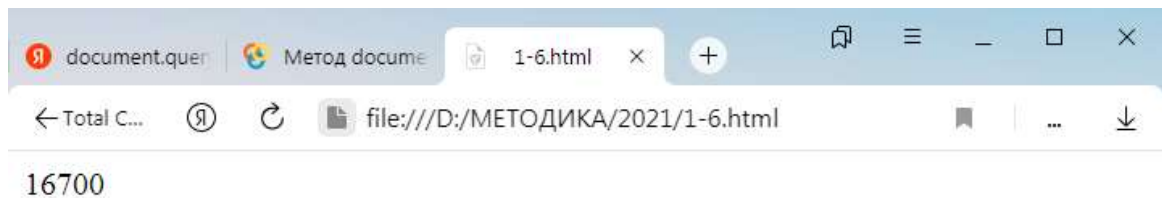
```
</script>
```

```
</body>
```

```
</html>
```



После нажатия на кнопку «Отправить» результат появится в новом окне:



## 1.7. Ввод-вывод с помощью консольной библиотеки Node.js

Строго говоря, Node.js не работает напрямую с веб-интерфейсом, это средство backend-разработки приложения на уровне сервера. Но для выполнения многих функций, в том числе и клиентских, используется большое количество сторонних библиотек.

Readline - это модуль, который реализует консольный ввод и вывод в Node.js, и через интерфейс модуля мы можем достигнуть чтения входного потока построчно.

Метод `question()` выводит то, что передано ему в качестве первого параметра (то есть — вопрос, задаваемый пользователю) и ожидает завершения ввода. После нажатия на Enter он вызывает обратный вызов, переданный ему во втором параметре и обрабатывает то, что было введено. В этом же обратном вызове мы закрываем интерфейс `readline`.

Пример:

```
var readline = require('readline'); //загрузка библиотеки readline
//установка потоков ввода-вывода
var rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});
//ввод данных как обещание ответить на вопрос
```

```

rl.question('Your first number: ', avans => {
    avans = parseInt(avans);
    rl.question('Your second number: ', poluchka => {
        poluchka= parseInt(poluchka);
    let zarplata=avans+poluchka
    console.log(zarplata.toString());
        rl.close(); //заккрытие потока
    });
});
});

```

The screenshot shows a code editor window with the following JavaScript code:

```

1 var readline = require('readline');
2 var rl = readline.createInterface({
3   input: process.stdin,
4   output: process.stdout
5 });
6 rl.question('Your first number: ', avans => {
7   avans = parseInt(avans);
8   rl.question('Your second number: ', poluchka => {
9     poluchka= parseInt(poluchka);
10    let zarplata=avans+poluchka
11    console.log(zarplata.toString());
12    rl.close();
13  });
14 });
15

```

The terminal window shows the execution output:

```

Your first number:  5000
Your second number:  11700
16700
>

```

Когда применять каждый из методов ввода-вывода?

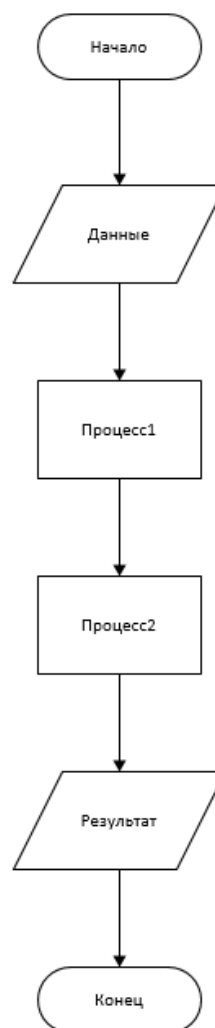
- 1) комбинацию readline/console.log удобно использовать, когда надо проверить в черновом варианте некоторые параметры, особенно, для разработчиков backend, а также сделать прототип приложения без привязки к веб-интерфейсу;
- 2) комбинация prompt/alert незаменима, когда количество входных и выходных данных не более двух;
- 3) методы объекта document используются при создании полноценного веб-приложения (frontend).

На практике чаще всего используются комбинированные способы, например, prompt используют в паре с document.write, а для просмотра промежуточных значений console.log.

## 1.8 Линейные скрипты и их внедрение на веб-страницу

С помощью базовой алгоритмической структуры «следование» можно реализовать линейный алгоритм, в котором действия выполняются друг за другом, не изменяя порядок следования команд. В школьном курсе математики и физики такой вид работы назывался «решением по действиям» или «вычислениями по формулам».

Блок-схема линейного алгоритма:



### Задача №1

Вычислить периметр и площадь прямоугольного треугольника по длинам двух его катетов.		
Скрипт в стиле JavaScript	Скрипт в стиле Node.js	Веб-приложение
<pre>var a=parseFloat(prompt('Введите число 1'));</pre>	<pre>var readline = require('readline'); var rl = readline.createInterface({</pre>	<pre>let aInput = document.querySelector('.a'); let bInput = document.querySelector('.b');</pre>

<pre> var b=parseFloat(prompt('Введите число 2')); let s=(a*b)/2; console.log(s); let c=Math.sqrt(a*a+b*b); let p=a+b+c; alert(p); </pre>	<pre> input: process.stdin, output: process.stdout }); rl.question('Your first number: ', a =&gt; {   a = parseFloat(a);   rl.question('Your second number: ', b =&gt; {     b= parseFloat(b);     let s=(a*b)/2;     console.log(s);     let c=Math.sqrt(a*a+b*b); let p=a+b+c; console.log(p);     rl.close();   });}); </pre>	<pre> let button = docu- ment.querySelector('.but- ton');  button.addEventLis- tener('click', () =&gt; {   let a = aIn- put.value;   let b = bIn- put.value;   let s=(a*b)/2;   let c=Math.sqrt(a*a+b*b);   let p=a+b+c;   document.write(s,p); }); </pre>
---	--	--

Примечания к коду:

- 1) Функция `parseFloat()` позволяет преобразовать строку в вещественное число.
- 2) Для записи строковых констант (пояснительного текста) можно использовать либо апострофы (‘ ‘), либо английские двойные кавычки (“ “).
- 3) В старых версиях JS в конце команды ставилась точка с запятой, а в новом стандарте языка он не обязательна.

Результаты работы веб-приложения с помощью JSEditor.io:

```

1 $(document).ready(function ()
2 {
3     let aInput = document.querySelector('.a');
4     let bInput = document.querySelector('.b');
5     let button = document.querySelector('.button');
6
7     button.addEventListener('click', () => {
8         let a = parseFloat(aInput.value);
9         let b = parseFloat(bInput.value);
10        let s=(a+b)/2;
11        let c=Math.sqrt(a*a+b*b);
12        let p=a+b*c;
13        document.writeIn(s,"<br>",p);
14    });
15 });

```

Введите число 1 3  
Введите число 2 4  
Отправить

6  
12

```

1 <!DOCTYPE html>
2 <html lang="ru">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Document</title>
7 </head>
8 <body>
9     Введите число 1 <input class="a"><br>
10    Введите число 2 <input class="b"><br>
11    <input type="submit" class="button">
12
13 </body>
14 </html>

```

## Задача №2

Поменять местами значения двух переменных		
Скрипт в стиле JavaScript	Скрипт в стиле Node.js	Веб-приложение
<pre> a=par- seInt(prompt('Введите число')); b=par- seInt(prompt('Введите число')); let c=a; a=b; b=c; </pre>	<pre> var readline = re- quire('readline'); var rl = readline.createIn- terface({     input: process.stdin,     output: process.stdout }); rl.question('Your first number: ', a =&gt; {     a = parseInt(a); </pre>	<pre> \$(document).ready(func- tion () { let a1 = docu- ment.querySelector('.a'); let b1 = docu- ment.querySelector('.b'); let button = docu- ment.querySelector('.but- ton'); </pre>

<pre> alert(a.toString() + " " + b.toString()); </pre>	<pre> rl.question('Your second number: ', b =&gt; {     b= parseInt(b); a= a ^ b ^ (b = a); console.log(a, b)     rl.close(); }); }); </pre>	<pre> button.addEventListener('click', () =&gt; {     let a = par- seInt(a1.value);     let b = par- seInt(b1.value);     a = a + b;     b = a - b;     a = a - b; docu- ment.write(a,"&lt;br&gt;",b); }); }); </pre>
--	--	---

Примечания к коду:

1) html-файл для веб-приложения точно такой, как в предыдущей задаче.

2) Чтобы вывести несколько значений в одном сообщении, можно перевести результаты в строковый формат с помощью метода `.ToString()`, а операция «+» соединит получившиеся символы в одну строку с пробелом.

3) Чтобы вывести несколько значений в разных строках, можно использовать тег `<br>` языка HTML. Теги HTML, касающиеся оформления текста, записываются внутри скрипта в кавычках.

4) Эта задача решена тремя равными способами:

а) через вспомогательную переменную `c`;

б) через операцию «^» - исключающее или;

в) через свойства сложения и вычитания чисел.

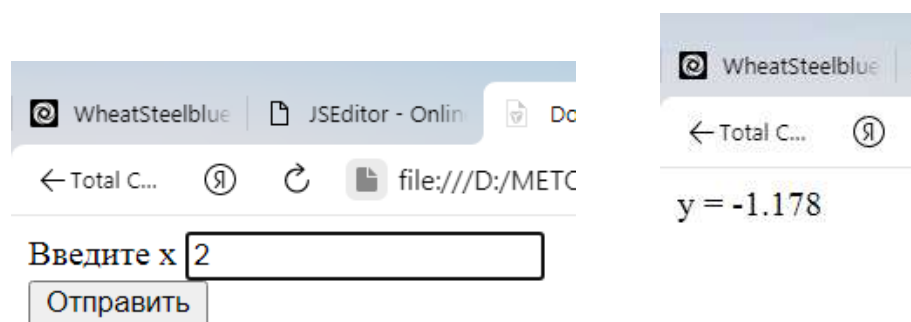
Задача №3



Вычислить значение функции $y = \frac{\sin^2 x - 4}{ x + \ln x }$ при заданном значении $x > 0$ .		
Скрипт в стиле JavaScript	Скрипт в стиле Node.js	Веб-приложение
<pre>x=parseFloat(prompt()); y=(Math.pow(Math.sin(x),2)-4) / Math.abs(x+Math.log(x)); alert(y.toFixed (3));</pre>	<pre>var readline = require('readline'); var rl = readline.createInterface({   input: process.stdin,   output: process.stdout }); rl.question('Your number: ', x =&gt; {   x = parseFloat(x);   let y=(Math.pow(Math.sin(x),2)-4)/Math.abs(x+Math.log(x)); console.log(Math.round(y*1000)/1000)   rl.close(); });</pre>	<pre>&lt;!DOCTYPE html&gt; &lt;html lang="ru"&gt; &lt;head&gt; &lt;meta charset="cp1251"&gt; &lt;title&gt;Document&lt;/title&gt; &lt;/head&gt; &lt;body&gt; Введите x &lt;input class="x"&gt;&lt;br&gt; &lt;input type="submit" class="button"&gt; &lt;script type="text/javascript"&gt; let xInput = document.querySelector('.x'); let button = document.querySelector('.button'); button.addEventListener('click', () =&gt; { let x = parseFloat(xInput.value); let s=((Math.sin(x)**2-4) / Math.abs(x+Math.log</pre>

		<pre> (x))).toFixed( sion(4).toString() document.write("y = ",s); }); &lt;/script&gt; &lt;/body&gt; &lt;/html&gt; </pre>
--	--	--

Результат:



Примечания к коду:

1) Квадрат числа можно записать двумя способами:

- а) с помощью метода `Math.pow()`;
- б) с помощью операции «\*\*».

2) Округление до нужного количества знаков после запятой (N) можно произвести тремя способами:

- а) метод `.toFixed(N)`;
- б) метод `.toFixed(K+N)`, где K – количество цифр в целой части;
- в) округление с последующим делением

`Math.round(число*Math.pow(10,N))/Math.pow(10,N)`.

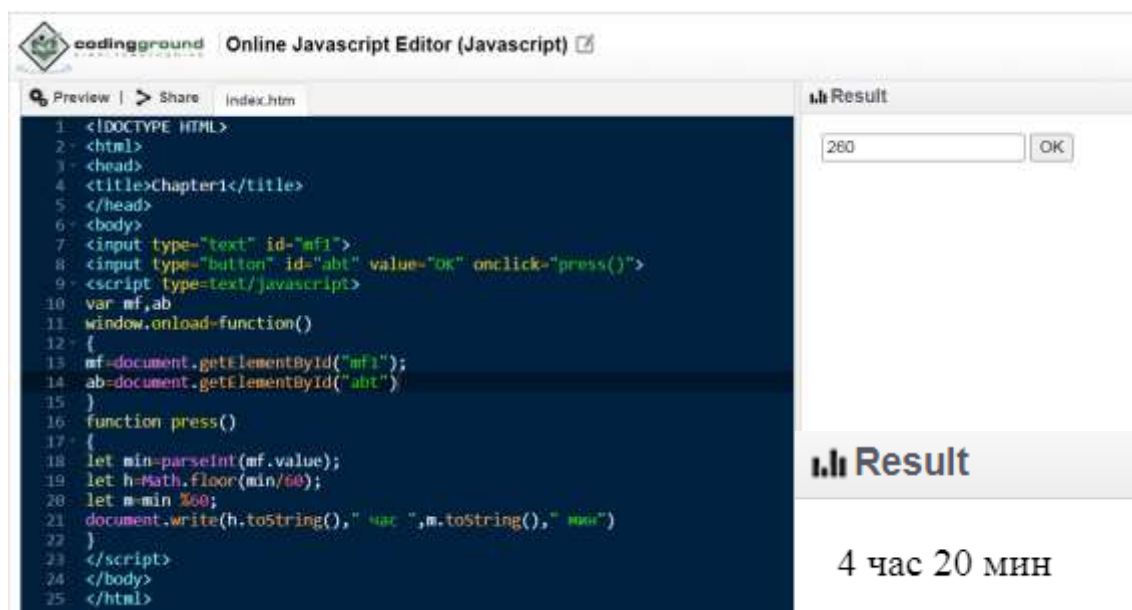
3) Веб-приложение приведено полностью внутри html-документа.

#### Задача №4

Пересчитать величину временного интервала, заданного в минутах, в величину, выраженную в часах и минутах		
Скрипт в стиле JavaScript	Скрипт в стиле Node.js	Веб-приложение
<pre>min=parseInt(prompt()) h=Math.trunc(min/60); m=min %60; alert(`\${h.toString().padStart(2, '0')} час \${m.toString().padStart(2, '0')} МИН `)</pre>	<pre>var readline = require('readline'); var rl = readline.createInterface({   input: process.stdin,   output: process.stdout }); rl.question('Your number: ', min =&gt; {   min = parseInt(min);   let h=Math.round(min/60); let m=min-h*60; console.log(h,"ч",m,"МИН");   rl.close(); });</pre>	<pre>&lt;!DOCTYPE HTML&gt; &lt;html&gt; &lt;head&gt; &lt;title&gt;Chapter1&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;input type="text" id="mf1"&gt; &lt;input type="button" id="abt" value="OK" onclick="press()"&gt; &lt;script type="text/javascript"&gt; var mf,ab window.onload=function() { mf=document.getElementById("mf1"); ab=document.getElementById("abt") } function press() {</pre>

		<pre> let min=parseInt(mf.value); let h=Math.floor(min/60); let m=min % 60; docu- ment.write(h.toString()," час ",m.toString()," мин") } &lt;/script&gt; &lt;/body&gt; &lt;/html&gt; </pre>
--	--	---

Результат:



## 1.9 Контрольные вопросы и тесты

### Дополнительные вопросы

1. Можно ли в JavaScript использовать деструктивное присваивание для обмена значениями переменных?
2. Делятся ли типы данных на целые (int) и вещественные (float)?

3. Чем отличаются встроенные функции для округления числа (round, floor, ceil, trunc)?
4. Что такое объектная модель браузера?
5. Можно ли использовать форматный вывод `$ {}` в консоли JavaScript (Node.js) и для чего он нужен в программировании?
6. Что общего между языками JavaScript и Node.js и в чем разница между ними?
7. Как влияют на вывод результата методы `innerText`, `innerHTML`, `outerText`, `outerHTML`?
8. Чем отличаются методы диалогового окна `prompt()`, `alert()`, `confirm()`?
9. Можно ли для ввода-вывода использовать комбинацию `prompt/document.write`?
10. Как посмотреть результаты в консоли разработчика браузера?

### Тесты с выбором варианта ответа

1. Каждое предложение сценария заканчивается...

- 1) ... новой строкой
- 2) ... точкой с запятой
- 3) ... новой строкой или точкой с запятой
- 4) ... запятой

2. Комментарии заключаются в скобки:

- 1) { };
- 2) /\* \*/;
- 3) [ ];
- 4) /% %/.

3. Идентификатор — это:

1) последовательность латинских букв, цифр и символа «\_», начинающаяся

с буквы или символа «\_»;

2) неизменяемые объекты языка (константы);

3) последовательность латинских и русских букв;

4) способ кодирования, допустимые преобразования над значением данной переменной.

4. Идентификатором в JS будет:

1) schetchikget\_1 i n e al2 Paraml \_ab;

2) %ab 12abc -x schetchik;

3) \ b ab 12abc -x schetchik;

4) \* ab 12abc -x schetchik.

5. Каков будет результат выполнения скрипта:

```
var i , j , s;
```

```
i=j=2;
```

```
s=(i++)+(++j) ;
```

```
console.log(s);
```

1)  $i = 3, j = 2, s = 5$ ;

2)  $i = 3, j = 3, s = 6$ ;

3)  $i = 3, j = 3, s = 5$ ;

4)  $i = 2, j = 3, s = 5$ .

6. Каков будет результат выполнения скрипта:

```
var x,y,a;
```

```
x=5;
```

```
y=x*2+7;
```

```
a=y/4;
```

```
console.log(x,y,a);
```

1)  $x = 5, y = 17, a = 4,25$ ;

2)  $x = 5, y = 17, a = 4$ ;

3)  $x = 5, y = 10, a = 2,25$ ;

4)  $x = 5, y = 32, a = 8$ .

7. Каков будет результат выполнения скрипта:

```
var x,y,a;
```

```
a = ( y = ( x = 5 ) * 2 + 7 ) /4;
```

```
console.log(a);
```

1)  $a = 4,25$ ;

2)  $a = 4$ ;

3)  $a = 2,25$ ;

4) error.

8. Каков будет результат выполнения скрипта:

```
var x,y;
```

```
x=y=5;
```

```
x+=2;
```

```
y-=3;
```

```
x*=y;
```

```
x/=++y;
```

```
console.log(x,y);
```

1)  $y = 3, x = 4$ ;

2)  $y = 4, x = 12$ ;

3)  $y = 12, x = 4.66$ ;

4)  $y = 3, x = 14$ .

9. Константа в JavaScript определяется с помощью директивы:

- 1) namespace
- 2) include
- 3) define
- 4) const

10. Постфиксная форма операции инкремента:

- 1) ++x
- 2) x++
- 3) - -x
- 4) x- -

11. Модуль вещественного числа в JavaScript:

- 1) Math.mod
- 2) Math.abs
- 3) Math.fabs
- 4) Math.pow

12. Префиксная форма операции декремента:

- 1) ++x
- 2) x++
- 3) - -x
- 4) x- -

### Тесты (задания) без выбора варианта ответа

1. Дан скрипт для вычисления объема и площади поверхности цилиндра, содержащий ошибки:

```
console.log(v, " ",s);  
var r=prompt('Введите число');  
const h=alert('Введите число');  
let v=h*Math.PI*Math.pow(2,r);  
let s=h*2*Math.PI*r;
```

Найти и исправить все из них.

2. Дано решение следующей задачи на JavaScript:



Известно, что точки с координатами  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$  являются тремя вершинами некоторого параллелограмма. Найти координаты четвертой вершины.

```
x1=parseFloat(prompt());
y1=parseFloat(prompt());
x2=parseFloat(prompt());
y2=parseFloat(prompt());
x3=parseFloat(prompt());
y3=parseFloat(prompt());
x4=(x2-x1)+x3;
y4=(y2-y1)+y3;
alert(x4, " ",y4);
```

Исправить ввод-вывод на библиотечный вызов Node.js Readline и консоль соответственно.

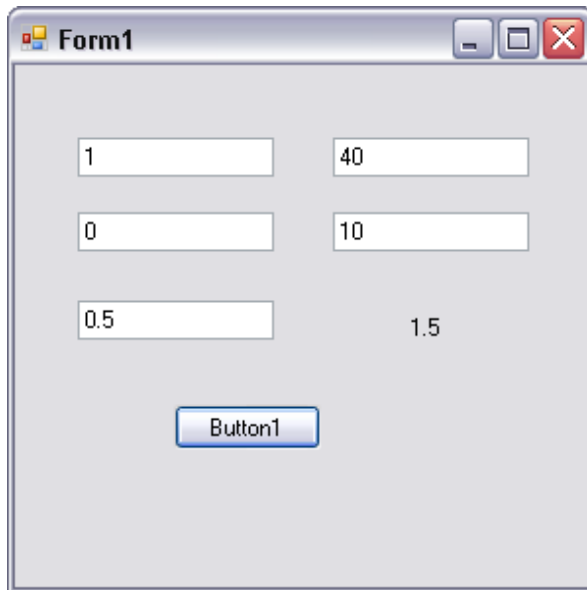
3. Дан скрипт для определения времени свободного падения физического тела с высоты  $H$ :

```
let g=9.8;
h=parseFloat(prompt());
let t=Math.sqrt(2*h/g);
alert(t);
```

Изменить объявление гравитационной постоянной на константное значение.

4. Имеется форма Windows и скрипт на JavaScript для решения следующей задачи:

Дано определенное количество бутылок (b) уксусной кислоты – n-процентной и вода (w=0). Определить, какой объем воды (t) понадобится, чтобы получить нужную крепость раствора – x%.



The image shows a browser window titled "Form1". Inside the window, there are five input fields and one button. The input fields contain the following values: 1, 40, 0, 10, and 0.5. The button is labeled "Button1". The value 1.5 is also visible on the right side of the form area.

```
b=parseFloat(prompt());  
n=parseFloat(prompt());  
w=parseFloat(prompt());  
x=parseFloat(prompt());  
v=parseFloat(prompt());  
let r=Math.abs(n-x)/Math.abs(w-x);  
let t=b*r*v;  
alert(t);
```

С помощью объектной модели браузера сконструировать веб-приложение с аналогичными элементами управления на форме.

5. Дано веб-приложение для вычисления выражения:

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```

<title>5-1</title>
</head>
<body>
<input type="text" id="mf1">
<input type="text" id="mf21">
<button id="abt">OK</button>
<p id="content"></p>
<script type="text/javascript">
var mf,mf2,ab
window.onload=function()
{
mf=document.getElementById("mf1");
mf2=document.getElementById("mf21");
ab=document.getElementById("abt")
}
ab.onclick=function()
{
var s,t,r
s=parseFloat(mf.value);
t=parseFloat(mf2.value);
r=9*t-2*s+1.17+ 2.2*t+s - t;
document.getElementById("content").innerHTML = r;
document.getElementById("content").innerHTML = s+t;
alert("!")
}

```

</script>

</body>

</html>

Почему не выводится результат вычисления? Найти и исправить все ошибки.

### 1.10 Задачи для самостоятельной работы

1) Вычислить длину окружности и площадь круга одного и того же заданного радиуса.

2) Рассчитать площадь треугольника по трем заданным сторонам (формуле Герона).

3) Рассчитать высоты треугольника по заданным сторонам.

4) Рассчитать медианы треугольника по заданным сторонам.

5) Рассчитать биссектрисы треугольника по заданным сторонам.

6) Рассчитать радиус окружности, вписанной в треугольник и описанной вокруг треугольника, зная его стороны.

7) Треугольник задан тремя сторонами. Вычислить биссектрису  $W_a$  и радиус вписанной окружности.

8) Вычислить периметр и площадь равнобедренной трапеции по заданным сторонам.

9) Найти площадь равнобедренной трапеции, если заданы диагональ  $d$  и длина средней линии  $m$ .

10) Найти площадь трапеции, если известны 4 стороны.

11) Даны катеты прямоугольного треугольника. Найти радиусы вписанной и описанной окружностей.

- 12) Дана сторона равностороннего треугольника. Найти его площадь, периметр, радиусы вписанной и описанной окружностей.
- 13) Даны 2 стороны треугольника и угол между ними. Найти площадь и периметр треугольника.
- 14) Даны 3 стороны треугольника. Найти его углы.
- 15) Дан радиус окружности  $R$  и радианная мера центрального угла  $\alpha$ . Найти площадь полученного сегмента и длину стягивающей хорды.
- 16) Дан радиус окружности  $R$  и радианная мера центрального угла  $\alpha$ . Найти площадь сектора и длину дуги, ограничивающей сектор.
- 17) Вычислить периметр и площадь равнобокой трапеции по трем заданным сторонам.
- 18) Заданы две стороны прямоугольника  $a, b$ . Найти его площадь, периметр и радиус вписанной окружности.
- 19) Даны катеты прямоугольного треугольника  $a$  и  $b$ . Найти гипотенузу  $c$  и углы треугольника  $\alpha, \beta, \gamma$ .
- 20) Известна диагональ квадрата  $d$ . Вычислить площадь  $S$  и периметр  $P$  квадрата.
- 21) Известна гипотенуза  $c$  и прилежащий угол  $\alpha$  прямоугольного треугольника. Найти площадь треугольника  $S$  и угол  $\beta$ .
- 22) Известна диагональ ромба  $d$ . Вычислить его площадь  $S$  и периметр  $P$ .
- 23) Известно значение периметра  $P$  равностороннего треугольника. Вычислить его площадь  $S$ , радиусы вписанной и описанной окружностей.
- 24) Задан периметр квадрата  $P$ . Вычислить сторону квадрата  $a$ , диагональ  $d$  и площадь  $S$ .
- 25) Даны смежные стороны параллелограмма  $a, b$ , угол между этими сторонами  $\alpha$  и диагональ  $d_1$ . Найти вторую диагональ  $d_2$ , высоту и площадь параллелограмма.

26) Вода в сосуде цилиндрической формы находится на уровне  $h$ . На каком уровне окажется вода, если её перелить в другой цилиндрический сосуд, у которого радиус основания в  $n$  раз больше, чем у данного?

27) Дан радиус основания, высота и образующая конуса. Найти площадь боковой поверхности и объем.

28) Дан радиус шара. Найти площадь сферической поверхности и объем шара.

29) Дан радиус шара и высота сегмента. Найти объем и площадь сферической поверхности шарового сегмента.

30) Дан радиус шара и высота сегмента. Найти объем шарового сектора.

31) Объем первого цилиндра равен  $V$  м<sup>3</sup>. У второго цилиндра высота в  $n$  раз больше, а радиус основания в  $k$  раз меньше, чем у первого. Найдите объем второго цилиндра (в м<sup>3</sup>).

32) Задано значение площади боковой поверхности цилиндра и диаметр основания. Найти объем цилиндра.

33) Во сколько раз увеличится объем конуса, если его высоту увеличить в  $n$  раз?

34) Дана площадь сферической поверхности  $S$ . Найти объем шара.

35) Дан объем шара  $V$ . Найти площадь его сферической поверхности.

36) Дан объем  $V$  и высота конуса  $h$ . Найти площадь его боковой поверхности.

37) Даны радиусы усеченного конуса  $r_1$ ,  $r_2$  и образующая  $l$ . Найти площадь боковой и полной поверхности конуса.

38) Даны радиусы усеченного конуса  $r_1$ ,  $r_2$  и образующая  $l$ . Найти объем конуса.

39) Дан радиус основания цилиндра и высота. Найти площадь полной поверхности цилиндра.

- 40) Дан объем конуса  $V$  и высота  $h$ . Найти радиус его основания.
- 41) Дан многогранник объемом  $V$ . Найти радиус шара, вписанного в этот многогранник, а также объем этого шара.
- 42) Дан радиус шара, высота сегмента и диаметр основания сегмента. Найти площадь поверхности шарового сегмента.
- 43) Дана площадь боковой поверхности конуса  $S$  и образующая  $l$ . Найти площадь полной поверхности конуса.
- 44) Дан объем шара  $V$ . Найти объем второго шара, если значение радиуса увеличить в  $n$  раз.
- 45) Дана площадь основания цилиндра  $S$  и высота  $h$ . Найти объем и площадь боковой поверхности цилиндра.
- 46) Дан объем  $V$  и образующая конуса  $l$ . Найти площадь его боковой поверхности.
- 47) Дан объем конуса  $V$ . Найти объем второго конуса, если значение радиуса основания увеличить в  $n$  раз.
- 48) Прямоугольный параллелепипед описан около цилиндра, радиус основания которого равен  $r$ . Объем параллелепипеда равен  $V$ . Найдите высоту цилиндра.
- 49) Объем первого конуса равен  $V \text{ м}^3$ . У второго конуса высота в  $n$  раз больше, а радиус основания в  $k$  раз меньше, чем у первого. Найдите объем второго конуса (в  $\text{м}^3$ ).
- 50) Объем первого шарового сегмента равен  $V \text{ м}^3$ . У второго сегмента высота в  $n$  раз больше, а радиус в  $k$  раз меньше, чем у первого. Найдите объем второго шарового сегмента (в  $\text{м}^3$ ).
- 51) Даны переменные  $A, B, C$ . Изменить их значения циклически, переместив их содержимое:  $A$  в  $B$ ,  $B$  - в  $C$ ,  $C$  - в  $A$ , и вывести новые значения переменных  $A, B, C$ .

52) Сдвинуть циклически значения разрядов целого четырехзначного числа на одну позицию влево.

53) Даны два однобайтовых числа. Необходимо сдвинуть влево на  $k$  разрядов как единое двухбайтовое число, каждый байт которого задается данными однобайтовыми числами.

54) Дано длинное (4байта) целое беззнаковое число  $A$  необходимо сдвинуть его циклически на  $B$  разрядов вправо.

55) Дано длинное (4байта) целое знаковое число  $A$  необходимо сдвинуть его циклически на  $B$  разрядов влево.

56) Сформировать шестизначное число из двух трехзначных чисел по алгоритму: в первом числе цифры поменять местами ( $a \rightarrow b, b \rightarrow c, c \rightarrow a$ ) и приписать второе.

57) Поменять местами первую и последнюю цифры четырехзначного числа.

58) Даны 4 переменные. Изменить их значения циклически  $a \rightarrow b, b \rightarrow c, c \rightarrow d, d \rightarrow a$ .

59) Сформировать четырехзначное число из двух двузначных чисел по алгоритму: первая цифра первого числа, первая цифра второго числа, вторая цифра первого числа, вторая цифра второго числа.

60) Сформировать четырехзначное число из двух двузначных чисел по алгоритму: вторая цифра второго числа, первая цифра второго числа, вторая цифра первого числа, первая цифра первого числа.

61) Поменять местами 2 пары трехзначных чисел и записать их цифры в обратном порядке.

62) Сдвинуть циклически значения разрядов целого пятизначного числа на  $k$  позиций влево.

63) Поменять местами вторую и третью цифры четырехзначного числа.

64) Даны 4 целых числа. Поменять местами второе и четвертое число.



## Глава 2. Организация ветвлений

### 2.1. Логические операторы и операции отношений

Во многих математических (и не только) задачах приходится сравнивать числа, величины и другие объекты между собой. Вот классические примеры:

«Если мне хватит денег на покупку, то я куплю компьютер, а иначе придется копить больше».

«Если температура воздуха на улице ниже нуля, то я надену перчатки».

Во всех языках программирования можно использовать стандартные знаки отношений:  $>$ ,  $<$ ,  $=$ .

Проблема в том, что некоторые знаки нельзя напечатать с помощью одной клавиши на клавиатуре:  $\geq$ ,  $\leq$ ,  $\neq$ . Еще одна проблема со знаком равенства, как не спутать сравнение двух выражений с оператором присваивания?

Создатели разных языков программирования решали обе проблемы по-разному. Например, в Паскале знаки " $<$ " означали «не равно», знак " $=$ " применялся для сравнений а " $:=$ " – присваивание значения переменной. В языке C++ по-другому: «не равно» - это " $!=$ ", присваивание – обычный знак равенства " $=$ ", а для сравнения переменных этот знак удваивается " $==$ ".

Разработчики JavaScript в целом придерживаются последней трактовки, но пошли еще дальше. Для сравнения значений разных типов с их приведением используется двойной знак равенства, а для точного сравнения – тройной " $===$ ".

Сведем полученные сведения в одну таблицу:

операция JavaScript	математическая операция	словесное описание
$>$	$>$	больше
$<$	$<$	меньше
$>=$	$\geq$	больше или равно
$<=$	$\leq$	меньше или равно

==	=	равно
===	~	строго равно
!=	≠	не равно

Допустим, мы написали выражение:  $a > b$ . Подставили числовые значения:  $5 > 3$ . Каким будет результат такой операции?

В данном случае выражение записано верно, а значит, в терминах математической логики – истинно.

В JavaScript, как и во многих других языках программирования, есть логический тип данных – `bool`. Его объекты могут принимать только одно из двух значений – истина (`true`), либо ложь (`false`).

Рассмотрим предыдущий пример более подробно:

```
let a=parseInt(prompt())
```

```
let b=parseInt(prompt())
```

```
console.log(a>b)
```

При  $a=5$ ,  $b=3$  в ответе получим истину:

```
index.js *
1 let a=parseInt(prompt())
2 let b=parseInt(prompt())
3 console.log(a>b)

Console Shell
undefined> 5
undefined> 3
true
Hint: hit control+c anytime to enter REPL.
> []
```

А при  $a=3$ ,  $b=5$  наоборот – ложь:

```
Console Shell
undefined> 3
undefined> 5
false
Hint: hit control+c anytime to enter REPL.
> []
```

Что из себя представляет выражение вида:

Операнд1 знак\_отношения Операнд2? (Например, a>b).

В математической логике это – предикат, т.е. функция, которая становится истинным или ложным высказыванием при подстановке вместо неизвестных переменных конкретных значений.

Над логическими функциями можно выполнять следующие операции: отрицание (инверсия), конъюнкция (логическое умножение), дизъюнкция (логическое сложение), импликация (если...то, с помощью формул ее можно заменить на 3 вышеперечисленные операции), эквиваленция (истина при операндах, имеющих одинаковые логические значения), сложение по модулю два ( иное название - исключающее или, истину дают операнды, имеющие разное логическое значение). Некоторые из этих операций применяются в программировании при создании сложных отношений.

#### Логические операции:

обозначение JavaScript	обозначения из алгебры логики	обозначения союзов на русском и английском языке	определение
!	$\neg \bar{A}$	НЕ, NOT	отрицание – истинно тогда, когда операнд ложен
	$\vee +$	ИЛИ, OR	дизъюнкция – истинна тогда, когда хотя бы один операнд истинен
&&	$\wedge \cdot$	И, AND	конъюнкция – истинна тогда, когда все операнды истинны

Пример:

```
index.js *
1 a=true
2 b=false
3 console.log(!a)
4 console.log(a||b)
5 console.log(a&&b) |
```

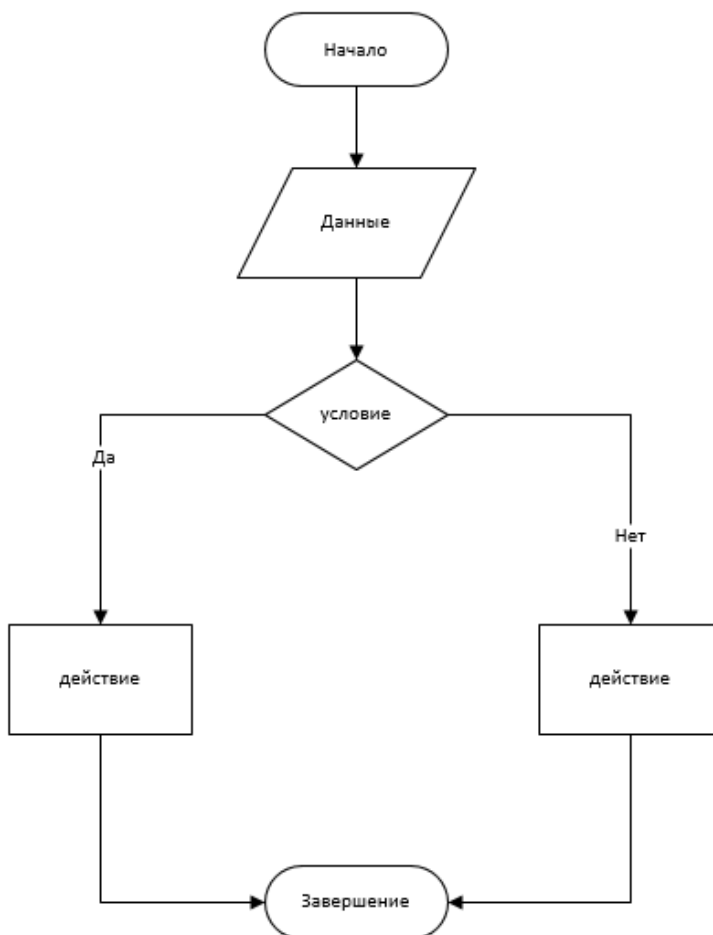
```
false
true
false
Hint: hit control+c anytime to enter REPL.
> |
```

## 2.2. Оператор ветвления

Рассмотрим подробнее задачи, где могут потребоваться операции отношения и логические операции. Пусть бухгалтер рассчитывает заработную плату по схеме: если работник получает менее 10000 рублей, то фирма предоставляет доплату в 10%, в противном случае доплата не предусмотрена.

Алгоритм решения такой задачи называется разветвляющимся, а сама конструкция – развилкой, ветвлением или альтернативой.

Блок-схема ветвления:



Чтобы получить из этой схемы полноценную программу, надо базовые фигуры (или слова в описании) заменить ключевые слова «ЕСЛИ», «ТО» и «ИНАЧЕ» их эквивалентом в языке JavaScript (а фактически, словами на английском языке) – IF, (слово THEN не применяется) и ELSE.

Оператор IF называется условным оператором. Его вид:

IF (условие) действие

Это краткая форма, или неполная развилка.

В полной форме оператор ветвления такой:

IF (условие)

{ операторы, которые выполняются, если условие истинно }

ELSE

{ операторы, которые выполняются, если условие ложно }

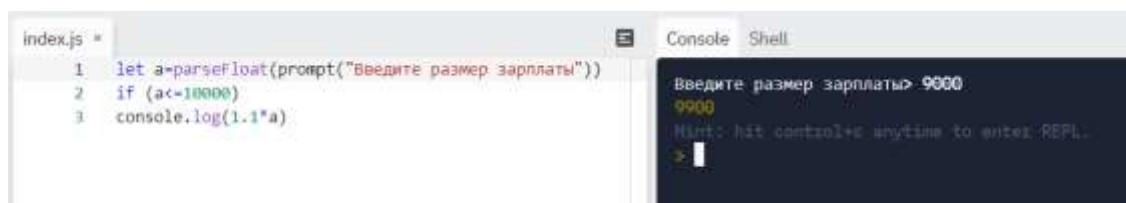
Их различие поясним с помощью скрипта решения задачи про зарплату.

1 вариант:

```
let a=parseFloat(prompt())
```

```
if (a<=10000)
```

```
console.log(1.1*a)
```



```
index.js *
1 let a=parseFloat(prompt("Введите размер зарплаты"))
2 if (a<=10000)
3 console.log(1.1*a)

Console Shell
Введите размер зарплаты: 9000
9900
Hint: hit control+c anytime to enter REPL.
> |
```

Программа работает верно.

Теперь введем новые данные:

```
Console Shell
Введите размер зарплаты> 11000
Hint: hit control+c anytime to enter REPL.
>
```

Если условие ложно, то никаких действий не происходит.

Основное свойство любого алгоритма – это результативность, поэтому необходимо переделать программу так, чтобы в любом случае получить результат:

```
let a=parseFloat(prompt("Введите размер зарплаты"))
if (a<=10000)
  console.log(1.1*a)
else
  console.log("доплата не предусмотрена")
```



```
index.js *
1 let a=parseFloat(prompt("Введите размер зарплаты"))
2 if (a<=10000)
3   console.log(1.1*a)
4 else
5   console.log("доплата не предусмотрена") |
Console Shell
Введите размер зарплаты> 11000
доплата не предусмотрена
Hint: hit control+c anytime to enter REPL.
>
```

При истинном значении условия результат не изменится, а при ложном появится необходимый ответ.

Решить задачу корректно можно и без использования ветки «иначе», для этого придется использовать две сокращенные формы условного оператора:

```
let a=parseFloat(prompt("Введите размер зарплаты"))
if (a<=10000)
  console.log(1.1*a)
if (a>10000)
  console.log("доплата не предусмотрена")
```

Благодаря комбинациям полных и неполных развилочек можно получать условные операторы разной степени вложенности вида: if-else, if-else, ... Единственное замечание – последний if перед последним else можно не писать. Программа сама «догадается», какое условие подходит методом исключений.

Рассмотрим математическую задачу:

Найти значение кусочно-гладкой функции:

$$y = \begin{cases} |x - 1|, & x < 0, \\ \arcsin x, & x = 0, \\ \sqrt{x + 7}, & x > 0. \end{cases}$$

```
let x=parseFloat(prompt('Введите число'));
```

```
let y=0;
```

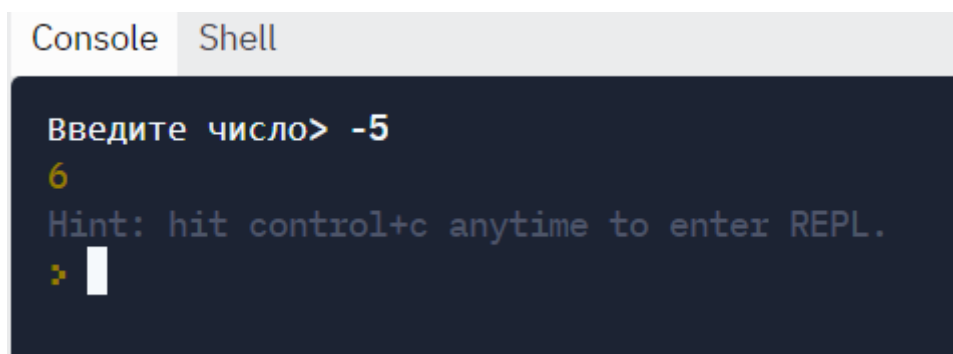
```
if (x<0) y=Math.abs(x-1);
```

```
else if (x==0) y=Math.asin(x);
```

```
else y=Math.sqrt(x+7);
```

```
console.log(y);
```

Тестовое решение №1:



```
Console Shell
Введите число> -5
6
Hint: hit control+c anytime to enter REPL.
> |
```

Тестовое решение №2:

```
Console Shell
Введите число> 0
0
Hint: hit control+c anytime to enter REPL.
> |
```

Тестовое решение №3:

```
Console Shell
Введите число> 2
3
Hint: hit control+c anytime to enter REPL.
> |
```

Последнее условие ( $x > 0$ ) можно не писать, т.к. других вариантов здесь нет.

Рассмотрим еще одну простую задачу: выяснить, какое из двух чисел имеет большее значение, и его вывести.

Решение:

```
let a=parseInt(prompt('Введите 1 число'));
```

```
let b=parseInt(prompt('Введите 2 число'));
```

```
if (a>b) console.log(a);
```

```
else console.log(b);
```



```
Console Shell
Введите 1 число> 3
Введите 2 число> 5
5
Hint: hit control+c anytime to enter REPL.
> 
```

Эту задачу можно решить более коротким способом.

Условный оператор имеет и более упрощенную форму – тернарный оператор. У него три операнда – условие, значение №1 и значение №2.

Форма записи:

Условие ? значение1 : значение2

Тогда скрипт для решения предыдущей задачи имеет вид:

```
console.log((a>b)?a:b);
```

```
index.js x Console Shell
1 let a=parseInt(prompt("Введите 1 число"));
2 let b=parseInt(prompt("Введите 2 число"));
3 console.log((a>b)?a:b); |
Введите 1 число> 3
Введите 2 число> 5
5
Hint: hit control+c anytime to enter REPL.
> 
```

### 2.3. Оператор выбора варианта

Составить программу, которая в зависимости от порядкового номера дня недели (1, 2, ..., 7) выводит на экран его название (понедельник, вторник, ..., воскресенье)

Решим задачу с помощью оператора if:

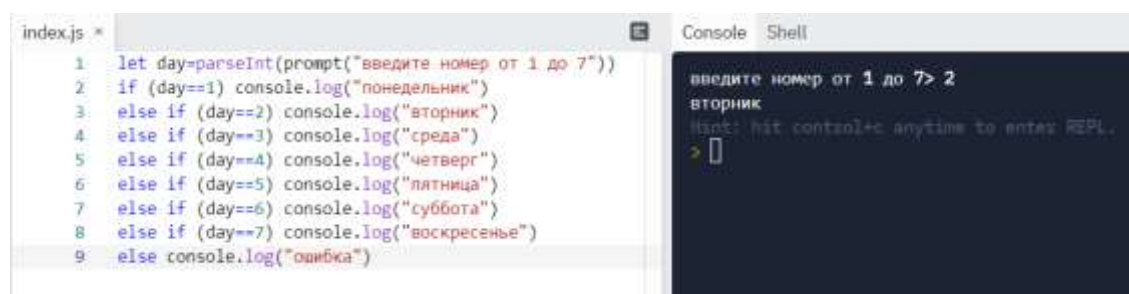
```
let day=parseInt(prompt("введите номер от 1 до 7"))
```

```
if (day==1) console.log("понедельник")
```

```
else if (day==2) console.log("вторник")
```

```
else if (day==3) console.log("среда")
```

```
else if (day==4) console.log("четверг")
else if (day==5) console.log("пятница")
else if (day==6) console.log("суббота")
else if (day==7) console.log("воскресенье")
else console.log("ошибка")
```



```
index.js *
1 let day=parseInt(prompt("введите номер от 1 до 7"))
2 if (day==1) console.log("понедельник")
3 else if (day==2) console.log("вторник")
4 else if (day==3) console.log("среда")
5 else if (day==4) console.log("четверг")
6 else if (day==5) console.log("пятница")
7 else if (day==6) console.log("суббота")
8 else if (day==7) console.log("воскресенье")
9 else console.log("ошибка")

Console Shell
введите номер от 1 до 7> 2
вторник
!hint: hit ctrl+c anytime to enter REPL.
> []
```

Для большого количества значений одного типа существует более удобный способ – оператор выбора SWITCH.

Он необходим в тех случаях, когда в зависимости от значений какой-либо переменной надо выполнить те или иные операторы:

```
switch (выражение)
{
case значение_1: Операторы_1; break;
case значение_2: Операторы_2; break;
case значение_3: Операторы_3; break;
...
case значение_n: Операторы_n; break;
default: Операторы; break;
}
```

Оператор break необходим для того, чтобы осуществить выход из оператора switch. Если оператор break не указан, то будут выполняться следующие операторы из списка несмотря на то, что значение, которым они помечены, не

совпадает со значением выражения. Оператор default является необязательным, он предусматривает защиту от некорректно введенных значений.

Переделаем предыдущий скрипт с учетом оператора switch:

```
let day=parseInt(prompt("введите номер от 1 до 7"))

switch (day)

{

case 1: console.log("понедельник"); break

case 2: console.log("вторник"); break

case 3: console.log("среда"); break

case 4: console.log("четверг"); break

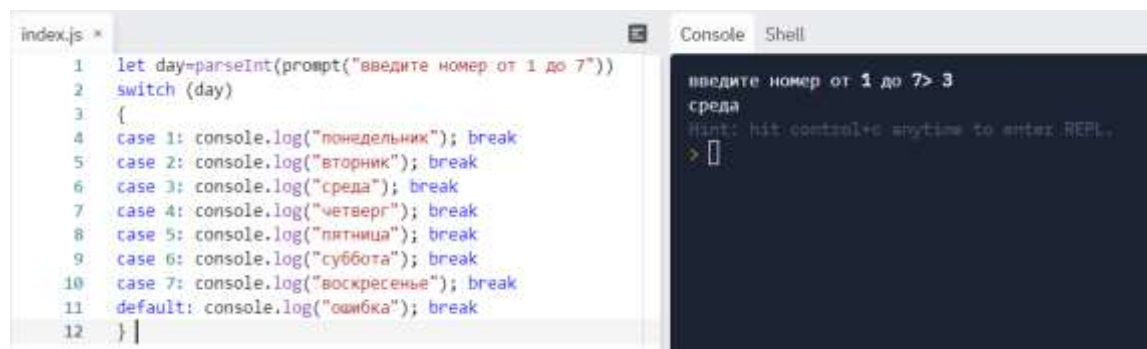
case 5: console.log("пятница"); break

case 6: console.log("суббота"); break

case 7: console.log("воскресенье"); break

default: console.log("ошибка"); break

}
```

The image shows a screenshot of a code editor with a file named 'index.js'. The code in the editor is a JavaScript script that prompts the user for a number between 1 and 7 and then uses a switch statement to log the corresponding day of the week. The console output shows the prompt 'введите номер от 1 до 7>' followed by the user input '3', and the resulting output 'среда'. A hint in the console suggests hitting control+c anytime to enter REPL. The code in the editor is as follows:

```
1 let day=parseInt(prompt("введите номер от 1 до 7"))
2 switch (day)
3 {
4 case 1: console.log("понедельник"); break
5 case 2: console.log("вторник"); break
6 case 3: console.log("среда"); break
7 case 4: console.log("четверг"); break
8 case 5: console.log("пятница"); break
9 case 6: console.log("суббота"); break
10 case 7: console.log("воскресенье"); break
11 default: console.log("ошибка"); break
12 }
```

Если убрать оператор default, а при запуске ввести ошибочные данные, то сообщения об ошибке не будет, но и результат на экране тоже не получим:

```
index.js *
1 let day=parseInt(prompt("введите номер от 1 до 7"))
2 switch (day)
3 {
4 case 1: console.log("понедельник"); break
5 case 2: console.log("вторник"); break
6 case 3: console.log("среда"); break
7 case 4: console.log("четверг"); break
8 case 5: console.log("пятница"); break
9 case 6: console.log("суббота"); break
10 case 7: console.log("воскресенье"); break
11
12 }
```

Console Shell

```
введите номер от 1 до 7> 8
Hint: hit ctrl+c anytime to enter REPL.
>
```

Выражение-селектор может иметь только простой порядковый тип (целый, символьный, логический, значение\_1, ..., значение\_N - константы того же типа, что и селектор (внутри switch)). Выражение может быть представлено в виде константы, числа, строки, переменной, арифметического, логического или строкового выражения.

Оператор Case работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна значению селектора, то выполняется оператор, стоящий за словом default. Если же это слово отсутствует, то активизируется оператор, находящийся за границей switch-case, т.е. после фигурной скобки “}”.

Таким образом, при использовании оператора switch-case должны выполняться следующие правила:

1. Выражение-селектор может иметь только простой порядковый тип (целый, символьный, логический).
2. Все константы, которые предшествуют операторам альтернатив, должны иметь тот же тип, что и селектор.
3. Все константы в альтернативах должны быть уникальны в пределах оператора выбора (значения не повторяются).

## 2.4. Скрипты с разветвляющимися алгоритмами

Задача №1. Проверить, является введенное пользователем число четным или нечетным.

```
let x=parseInt(prompt('Введите число'));  
if (Math.floor(x/2)==x/2)  
console.log("четное");  
else console.log("нечетное");
```



```
index.js =  
1 let x=parseInt(prompt('Введите число'));  
2 if (Math.floor(x/2)==x/2)  
3 console.log("четное");  
4 else console.log("нечетное");
```

Console Shell

```
Введите число> 3  
нечетное  
Hint: hit control+c anytime to enter REPL.  
> []
```

Существуют разные варианты проверки числа на четность. В данном случае округленное до целого частное от деления на два для четного числа должно совпадать с самим частным.

Рассмотрим варианты, связанные с остатком от деления числа на два. Для четного числа он должен равняться нулю, а для нечетного – единице (или проще – не нулю).

<pre>let x=par- seInt(prompt('Введите число')); if (x % 2==0) console.log("четное"); else console.log("нечет- ное");</pre>	<pre>let x=par- seInt(prompt('Введите число')); if (x % 2!=0) console.log("нечетное"); else console.log("чет- ное");</pre>	<pre>let x=par- seInt(prompt('Введите число')); if (x % 2==1) console.log("нечетное"); else console.log("чет- ное");</pre>
--	--	--

Еще один способ - перевести число в двоичную систему. Если младший бит равен 1 - нечетное. если 0 - четное.

<pre>let x=parseInt(prompt('Введите число')); if (x &amp; 1) console.log("нечетное"); else console.log("четное");</pre>	<pre>let x=parseInt(prompt('Введите число')); if (~x &amp; 1) console.log("четное"); else console.log("нечетное");</pre>
---	--

Пояснения к коду:

Если логические операции написать по одному разу (например, вместо `&&` оставить `&`), то они будут применяться не к объекту целиком, а к отдельным битам числа, т.е. к каждой двоичной цифре.

Побитовая операция	Определение
~	двоичное отрицание (инверсия)
&	двоичное И
	двоичное ИЛИ
^	двоичное исключающее ИЛИ
<<	сдвиг разряда влево
>>	сдвиг разряда вправо

Для решения этой задачи подходит не только стандартный оператор ветвления `if`, но и тернарный оператор, и даже оператор выбора:

<pre>let x=parseInt(prompt('Введите число')); console.log(!(x%2) ? "Четное число": "Нечетное число")</pre>	<pre>let x=parseInt(prompt('Введите число')); console.log((Math.round(x/2) === x/2) ? "Четное число": "Нечетное число")</pre>	<pre>let x=parseInt(prompt('Введите число')); switch ( x % 2 ) { case 0 : console.log("Четное число"); break;</pre>
--	---	---

		<pre> case 1 : con- sole.log("Нечетное число"); break; } </pre>
--	--	---

По этим схемам можно устанавливать условия для делимости без остатка числа на заданный делитель, т.е. условие кратности, меняя число 2 на нужное значение.

Задача №2. По длинам трех отрезков, введенных пользователем, определить возможность существования треугольника, составленного из этих отрезков.

Геометрические доказательства – одна из самых распространенных задач в программировании. Допустим, по заданным сторонам треугольника найти его периметр и площадь. Но может оказаться ситуация, когда из трех отрезков с заданными длинами нельзя построить треугольник. Допустим, мы ввели отрицательные или нулевые значения. Второй случай описывается в теореме: длина любой стороны треугольника не может быть больше суммы длин его остальных двух сторон. Именно условие этой теоремы и лежит в условии оператора if при решении данной задачи.

```

let a=parseFloat(prompt('Введите число'));
let b=parseFloat(prompt('Введите число'));
let c=parseFloat(prompt('Введите число'));
if (a+b<=c || a+c<=b || b+c<=a)
console.log("Такой треугольник не существует");
else console.log("Такой треугольник существует");

```

```

index.js *
1 let a=parseFloat(prompt('Введите число'));
2 let b=parseFloat(prompt('Введите число'));
3 let c=parseFloat(prompt('Введите число'));
4 if (a+b<=c || a+c<=b || b+c<=a)
5 console.log("Такой треугольник не существует");
6 else console.log("Такой треугольник существует");

```

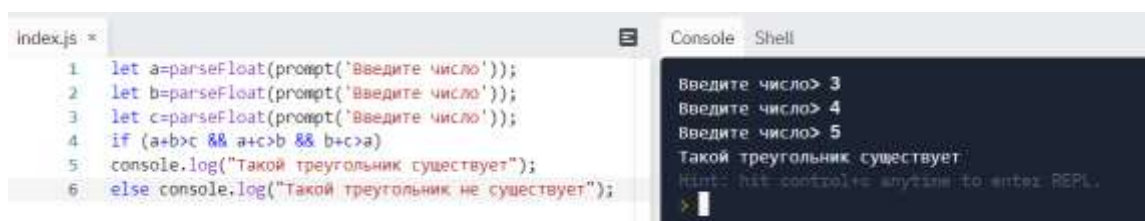
```

Console Shell
Введите число> 1
Введите число> 2
Введите число> 3
Такой треугольник не существует
Hint: hit ctrl+c anytime to enter REPL.
>

```

Условие существования треугольника можно записать и другим способом:

```
let a=parseFloat(prompt('Введите число'));  
let b=parseFloat(prompt('Введите число'));  
let c=parseFloat(prompt('Введите число'));  
if (a+b>c && a+c>b && b+c>a)  
  console.log("Такой треугольник существует");  
else console.log("Такой треугольник не существует");
```



The screenshot shows a code editor with a file named 'index.js' containing the following JavaScript code:

```
1 let a=parseFloat(prompt('Введите число'));  
2 let b=parseFloat(prompt('Введите число'));  
3 let c=parseFloat(prompt('Введите число'));  
4 if (a+b>c && a+c>b && b+c>a)  
5   console.log("Такой треугольник существует");  
6 else console.log("Такой треугольник не существует");
```

To the right of the code editor is a console window with the following output:

```
Введите число> 3  
Введите число> 4  
Введите число> 5  
Такой треугольник существует  
Hint: Hit control-c anytime to enter REPL.  
>
```

Фактически любую геометрическую теорему, в которой сравниваются конкретные значения и делается вывод в форме «если...то», можно положить в основу условия условного оператора для решения геометрической задачи.

Задача №3. Решить степенное уравнение.

Эта задача сводится к трем разным подзадачам вида: решить линейное уравнение  $ax = b$ , квадратное уравнение  $ax^2+bx+c=0$ , кубическое уравнение  $ax^3+bx^2+cx+d=0$ .

1) Линейное уравнение.

В решении линейного уравнения первой степени недостаточно написать  $x=b/a$ , иначе может возникнуть ситуация, когда знаменатель равен нулю.

Например:

```
let a=parseFloat(prompt('Введите число'));  
let b=parseFloat(prompt('Введите число'));  
let x=b/a
```



console.log(x)

Введем первую пару тестовых данных:



```
index.js *
1 let a=parseFloat(prompt('Введите число'));
2 let b=parseFloat(prompt('Введите число'));
3 let x=b/a
4 console.log(x)
```

Console Shell

Введите число> 2  
Введите число> 6  
3  
Hint: hit control+c anytime to enter REPL.  
>

На первый взгляд, скрипт работает корректно. Но для второй пары значений:



```
index.js *
1 let a=parseFloat(prompt('Введите число'));
2 let b=parseFloat(prompt('Введите число'));
3 let x=b/a
4 console.log(x)
```

Console Shell

Введите число> 0  
Введите число> 5  
Infinity  
Hint: hit control+c anytime to enter REPL.  
>

Ответ infinity – бесконечность.

Чтобы не допустить деление на ноль, введем дополнительное условие:

```
let a=parseFloat(prompt('Введите число'));
```

```
let b=parseFloat(prompt('Введите число'));
```

```
if (a!=0)
```

```
{
```

```
let x=b/a
```

```
console.log(x)
```

```
}
```

```
else
```

```
console.log('Нет решений')
```



```
index.js *
1 let a=parseFloat(prompt('Введите число'));
2 let b=parseFloat(prompt('Введите число'));
3 if (a!=0)
4 {
5 let x=b/a
6 console.log(x)
7 }
8 else
9 console.log('Нет решений')
```

Console Shell

Введите число> 0  
Введите число> 5  
Нет решений  
Hint: hit control+c anytime to enter REPL.  
>

Но это ещё не всё. Проверим третий случай:

```
Console Shell
Введите число> 0
Введите число> 0
Нет решений
Hint: hit control+c anytime to enter REPL.
> |
```

Но строго говоря, уравнение вида  $0 \cdot x = 0$  имеет бесконечное множество решений. Поэтому окончательно код скрипта получит ещё одно условие:

```
let a = prompt('Введите значение a', 0);
let b = prompt('Введите значение b', 0);
if (a==0 && b==0)
  console.log('любое число');
else if (a==0)
  console.log('нет решений');
else
  console.log('x= ' + (b / a));
```

<pre>Введите значение a&gt; 2 Введите значение b&gt; 6 x= 3 Hint: hit control+c anytime to enter REPL. &gt;  </pre>	<pre>Введите значение a&gt; 0 Введите значение b&gt; 5 нет решений Hint: hit control+c anytime to enter REPL. &gt;  </pre>	<pre>Введите значение a&gt; 0 Введите значение b&gt; 0 любое число Hint: hit control+c anytime to enter REPL. &gt;  </pre>
---	--	--

## 2) Квадратное уравнение.

В общем случае уравнение второй степени имеет два корня – либо два действительных, либо два совпадающих, либо два комплексно-сопряженных. Последний случай в школьном курсе не рассматривается, поэтому будем находить только действительные корни.

Условие вида корней содержится в знаке дискриминанта:

$$D = b^2 - 4ac$$

Если дискриминант  $D > 0$ , тогда корни вычисляются по формуле:

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$$

Если дискриминант  $D = 0$ , то можно не учитывать этот случай и вывести два одинаковых корня. Но лучше сделать дополнительное условие и вычислить единственный корень по формуле:

$$x = \frac{-b}{2a}$$

При отрицательном дискриминанте ( $D < 0$ ) вывести соответствующее сообщение об отсутствии действительных корней.

Полностью скрипт имеет вид:

```
let a = prompt('Введите значение a', 0);
```

```
let b = prompt('Введите значение b', 0);
```

```
let c = prompt('Введите значение c', 0);
```

```
let D = b * b - 4 * a * c;
```

```
if( D > 0 ){
```

```
let sqrtD = Math.sqrt( D );
```

```
let x1 = ( -b + sqrtD ) / ( 2 * a );
```

```
let x2 = ( -b - sqrtD ) / ( 2 * a );
```

```
console.log( 'x1 = %f', x1 );
```

```
console.log( 'x2 = %f', x2 );
```

```
}
```

```
else if( D == 0 ){
```

```
let x = -b / ( 2 * a );
```

```
console.log( 'x = %f', x );
```

```

}
else
console.log('Действительных корней нет')

```

```

Console Shell
Введите значение a> 1
Введите значение b> 5
Введите значение c> -6
x1 = 1
x2 = -6
Hint: hit control+c anytime to enter REPL.
> 

```

### 3) Кубическое уравнение.

Уравнение третьей степени имеет вид

$$ax^3 + bx^2 + cx + d = 0 \quad (1)$$

После деления на  $a$  уравнение (1) принимает канонический вид:

$$x^3 + rx^2 + sx + t = 0 \quad (2)$$

где  $r = b/a, s = c/a, t = d/a$ . В уравнении (2) сделаем за-

мену  $x = y - r/3$  и получим приведённое уравнение:

$$y^3 + py + q = 0, \quad (3)$$

где

$$p = \frac{3s - r^2}{3}, q = \frac{2r^3}{27} - \frac{rs}{3} + t.$$

$$D = (p/3)^3 + (q/2)^2$$

Число действительных корней приведённого уравнения (3) зависит от знака дискриминанта приведённого кубического уравнения

Количество корней кубического уравнения		
Дискриминант	Количество действительных корней	Количество комплексных корней
$D > 0$	1	2
$D < 0$	3	—

Корни приведённого уравнения могут быть рассчитаны по формулам

Кардано:

$$\begin{aligned}
 y_1 &= u + v \\
 y_2 &= \frac{-u + v}{2} + \frac{u - v}{2}i\sqrt{3} \\
 y_3 &= \frac{-u + v}{2} - \frac{u - v}{2}i\sqrt{3}
 \end{aligned} \quad (3.4)$$

где

$$u = \sqrt[3]{-q/2 + \sqrt{D}}, v = \sqrt[3]{-q/2 - \sqrt{D}}.$$

При отрицательном дискриминанте уравнение (1) имеет три действительных корня, но они будут вычисляться через вспомогательные комплексные величины. Чтобы избавиться от этого, можно воспользоваться формулами:

$$\begin{aligned}
 y_1 &= 2\sqrt[3]{\rho} \cos\left(\frac{\varphi}{3}\right), \\
 y_2 &= 2\sqrt[3]{\rho} \cos\left(\frac{\varphi}{3} + \frac{2\pi}{3}\right), \\
 y_3 &= 2\sqrt[3]{\rho} \cos\left(\frac{\varphi}{3} + \frac{4\pi}{3}\right),
 \end{aligned} \quad (5)$$

где

$$\rho = \sqrt{\frac{-p^3}{27}}, \cos(\varphi) = \frac{-q}{2\rho}.$$

Таким образом, при положительном дискриминанте кубического уравнения (3) расчёт корней будем вести по формулам (4), а при отрицательном — по формулам (5).

После расчёта корней приведённого уравнения (3) по формулам (4) или (5) необходимо по формулам

$$x_k = y_k - \frac{r}{3}, k = 1, 2, 3 \dots$$

перейти к корням заданного кубического уравнения (1).

Код скрипта:

```
let a = parseFloat(prompt('Введите значение a', 0));
let b = parseFloat(prompt('Введите значение b', 0));
let c = parseFloat(prompt('Введите значение c', 0));
let d1 = parseFloat(prompt('Введите значение d', 0));
//Расчёт коэффициентов канонического уравнения по формуле 2.
let r = b/a; let s = c /a; let t = d1/a;
    //Вычисление коэффициентов приведённого уравнения 3.
    let p =(3 * s - r * r ) / 3;
    let q=2* r * r * r /27- r * s/3+ t;
    //Вычисление дискриминанта кубического уравнения.
    let d =( p /3) * Math.pow( p/3, 2)+ Math.pow( q / 2, 2 );
    //Проверка знака дискриминанта,
    //ветка true реализует формулы 5,
    //ветка false — формулы 4
    if (d<0)
    {
        let ro = Math.sqrt (-p * p * p / 27 );
```

```

//Следующие два оператора реализуют расчёт угла fi,
//сначала вычисляется величина косинуса угла,
//затем вычисляется его арккосинус через арктангенс.
let fi =-q /(2 * ro );
fi = Math.PI /2-Math.atan( fi / Math.sqrt (1 - fi * fi ) );
//Вычисление действительных корней уравнения x1, x2 и x3
let x1 =2 * Math.exp (1/3 * Math.log ( ro ) ) * Math.cos ( fi /3) - r
/ 3;

let x2 =2 * Math.exp (1/3 * Math.log ( ro ) ) * Math.cos ( fi /3+2 *
Math.PI /3) - r / 3;

let x3 =2 * Math.exp (1/3 * Math.log ( ro ) ) * Math.cos ( fi /3+4 *
Math.PI /3) - r / 3;

console.log ( ' x1= ', x1, ' x2= ', x2, ' x3= ', x3);

}
else
{
//Вычисление u и v с проверкой знака
// подкоренного выражения.
if (-q/2+ Math.sqrt(d)>0)
u=Math.exp (1/3 * Math.log (-q/2+ Math.sqrt ( d ) ) )
else
if (-q/2+ Math.sqrt(d)<0)
u=-Math.exp (1/3 * Math.log( Math.abs(-q/2+ Math.sqrt ( d
))))
else
u = 0;

```

```

if (-q/2- Math.sqrt(d)>0 )
    v=Math.exp (1/3 * Math.log (-q/2- Math.sqrt ( d ) ) )
else
    if (-q/2- Math.sqrt (d)<0)
        v=-Math.exp (1/3 * Math.log (Math.abs(-q/2- Math.sqrt ( d )
)))
    else
        v = 0;

//Вычисление действительного корня кубического уравнения.
let  x=u+v-r / 3;
    console.log ( ' x= ', x );
}

```

```

Console  Shell
Введите значение a> 1
Введите значение b> 3
Введите значение c> 3
Введите значение d> 1
  x=  -1
Hint: hit control+c anytime to enter REPL.
> 

```

Примечание к коду:

Для вычислений значений  $u$  и  $v$  нельзя записывать выражения через встроенную функцию степени `pow` вида:

```
v=Math.pow (-q/2- Math.sqrt ( d ) ,1/3 )
```



Дело в том, что данный метод некорректно обрабатывает отрицательные основания степени. Лучше воспользоваться известной формулой через логарифмизацию:

$$a^x = e^{x \cdot \ln a}.$$

Задача №4 Ввести 2 числа и вычислить сумму, разность, произведение и частное от деления первого введенного числа на второе.

Эту задачу можно решить при помощи линейного алгоритма, выводя на экран все 4 результата. Но удобнее организовать выбор ветви вычислений в зависимости от знака операции с помощью оператора выбора варианта:

```
let ch=prompt('Введите знак +, -, *, /');
let x=parseFloat(prompt('Введите 1 число'));
let y=parseFloat(prompt('Введите 2 число'));
let f=false;
let z=0;
switch (ch) {
case '+': z=x+y; f=true; break;
case '-': z=x-y; f=true; break;
case '*': z=x*y; f=true; break;
case '/': if (y!=0) {z=x/y; f=true;}
else
console.log('на 0 делить нельзя'); break;
default: console.log('нет такого знака');
}
if (f) console.log(z);
```

```
Console Shell
Введите знак +, -, *, /> +
Введите 1 число> 4
Введите 2 число> 7
11
Hint: hit control+c anytime to enter REPL.
> 
```

Пояснения к коду:

Объект `ch` в этом скрипте – это переменная символьного типа. Она принимает одно из значений – знак арифметической операции. Данный алгоритм предусматривает защиту от неверно введенного знака (оператор `default`) и защиту от деления на ноль. В этом случае вводится логическая переменная `f`, в которой в случае нулевого делителя остается значение `false`. При ложном значении логической переменной выводится соответствующее сообщение вместо частного:

```
Console Shell
Введите знак +, -, *, /> /
Введите 1 число> 2
Введите 2 число> 0
на 0 делить нельзя
Hint: hit control+c anytime to enter REPL.
> 
```

Рассмотрим полноценное веб-приложение – упрощенный калькулятор, выполняющий на выбор одно арифметическое действие.

Для выбора знака операции нам поможет радиокнопка. Ее обозначение в `html`-документе:

```
<input type="radio" name="r1" id="id1">
```

Имя для нескольких радиокнопок (с зависимой фиксацией) должно быть одинаковым, а id разным, т.к. к нему будем применять условие, когда кнопка нажата.

Внутри скрипта должны быть строки вида:

```
if (document.getElementById("id1").checked==true) { выполняемое действие
}
```

Код скрипта полностью:

```
<html><head>
```

```
Введите x <input class="x"><br>
```

```
Введите y <input class="y"><br>
```

```
<input type="radio" name="r1" id="id1"> + <br>
```

```
<input type="radio" name="r1" id="id2"> - <br>
```

```
<input type="radio" name="r1" id="id3"> * <br>
```

```
<input type="radio" name="r1" id="id4"> / <br>
```

```
<input type="submit" class="button"><br>
```

```
<script>
```

```
let button = document.querySelector('.button');
```

```
button.addEventListener('click', () => {
```

```
let z=0.0
```

```
let x = parseFloat(document.querySelector('.x').value);
```

```
let y = parseFloat(document.querySelector('.y').value);
```

```
if (document.getElementById("id1").checked==true) z=x+y
```

```
else
```

```
if (document.getElementById("id2").checked==true) z=x-y
```

```
else
```

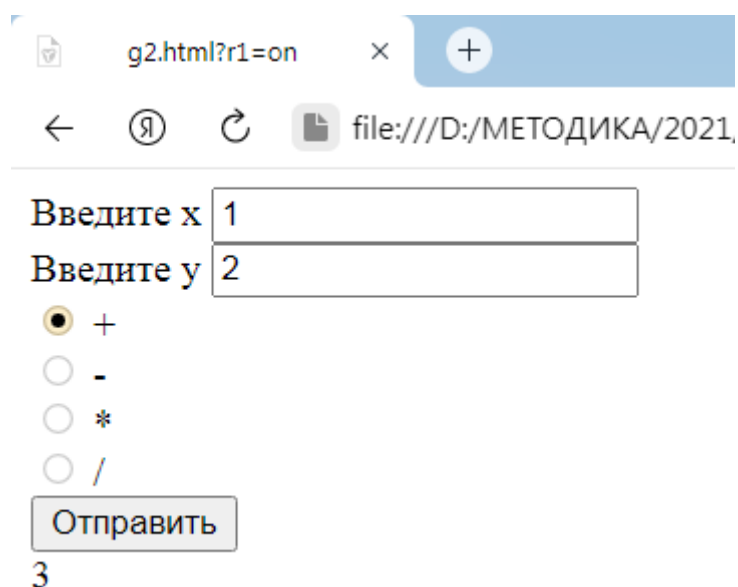
```
if (document.getElementById("id3").checked==true) z=x*y
```

```

else
if (document.getElementById("id4").checked==true) z=x/y
document.getElementById("myblock").innerHTML=z
});
</script>
</head>
<body>

```

Веб-страница с результатом:



Задача №5. Ввести номер семестра и вывести порядковый номер курса студента.

Алгоритм решения следующий: если введено значение 1 или 2, то в ответе – «первый курс», если 3 или 4, то «второй» и т.д. В некоторых языках программирования можно использовать Switch с диапазоном значений (например, в Паскале).

Для решения этой задачи на JavaScript придется перечислять все значения внутри case по отдельности, но вывод результата организовывается только один раз:

```
let arg = prompt("Введите семестр");
switch (arg) {
  case '1':
  case '2':
    console.log( 'Первый курс' );
    break;
  case '3':
  case '4':
    console.log( 'Второй курс' );
    break;
  case '5':
  case '6':
    console.log( 'Третий курс' );
    break;
  case '7':
  case '8':
    console.log( 'Четвертый курс' );
    break;
  default:
    console.log( 'Неизвестное значение' );
}
```

```
Console Shell
Введите семестр> 4
Второй курс
Hint: hit control+c anytime to enter REPL.
> 
```

## 2.5. Контрольные вопросы и тесты

### Дополнительные вопросы

1. Как называется программа, в которой выполнение операторов происходит в зависимости от поставленного условия?
2. Что такое условный оператор?
3. Существует ли в JavaScript оператор перехода GOTO?
4. Куда передается управление, если условие в условном операторе ложно?
5. В каких случаях тернарный оператор не пригоден для использования?
6. Можно ли в условии записывать двойные операции отношений вида  $a < x < b$ ?
7. Как в операторе выбора ввести диапазон значений, например от 0 до 5?
8. В чём заключается смысл оператора ELSE?
9. Чем отличаются логические операции от битовых?
10. Какой приоритет операции отношений имеют по сравнению с логическими операциями?

### Тесты с выбором варианта ответа

1. Какой из операторов отношений в JavaScript неправильный?  
1) ==      2) <>      3) !=      4) ===

2. Составной оператор — это:

1) последовательность операторов, заключенная в фигурные скобки {};

2) последовательность операторов, заключенная квадратные скобки [ ];

3) последовательность операторов, заключенная в операторные скобки  
begin ... end;

4) последовательность операторов, заключенная в круглые скобки ().

3. Логическое «не равно» обозначается:

1) <> ;

2) ||;

3) !;

4) !=;

4. Логическое «и» обозначается:

1) =;

2) ||;

3) &;

4) &&.

5. Логическое «не» обозначается:

1) !;

2) !!;

3) ||;

4) not.

6. Написать предложение в JavaScript, которое сравнивает целую переменную sum с константой 10, и если это так, то печатает строку "Good guess"

1) if sum = 10 then console.log("Good guess");

2) if (sum == 10) then console.log("Good guess");

3) if (sum == 10) console.log("Good guess");





- 2) переменной;
- 3) константой;
- 4) типа Char.

### Тесты без выбора варианта ответа

1. Каков будет результат выполнения скрипта:

```
var a,b  
a=4;  
b=7;  
let m=(a>b)?a:b;  
console.log(m);
```

2. Каков будет результат выполнения скрипта:

```
var x,y  
y=-4;  
x = ( y < 0 ) ? - y : y ;  
console.log(x);
```

3. Перепишите код с использованием одной конструкции switch:

```
const number = +prompt('Введите число между 0 и 3, ');  
if (number === 0) {  
    console.log ('Вы ввели число 0');  
}  
if (number === 1) {  
    console.log ('Вы ввели число 1');
```

```
}  
if (number === 2 || number === 3) {  
    console.log ('Вы ввели число 2, а может и 3!');  
}
```

4. Дан фрагмент скрипта:

```
if(x<0)console.log(x-Math.exp(x)+1/Math.tan(x))  
if(x<=0&& x<3)console.log(x-7*Math.cos(x))  
else if(x>3)console.log((Math.abs(x+1)/Math.log(x)))
```

Найти ошибки в условии и переписать правильно полностью работающий скрипт.

5. Дан скрипт, вычисляющий площадь и периметр треугольника, заданного длинами двух сторон и величиной угла между ними:

```
let a=parseFloat(prompt("Сторона 1"))  
let b=parseFloat(prompt("Сторона 2"))  
let g=parseFloat(prompt("Угол в градусах"))  
g=g*Math.PI/180  
let s=1/2*a*b*Math.sin(g)  
console.log('s='+s)  
let p=Math.sqrt(a*a+b*b-2*a*b*Math.cos(g))+a+b  
    console.log('p='+p)
```

Переписать код с учетом корректности вводимых данных. Стороны треугольника не могут быть отрицательными или равными нулю, а величина угла определяется в диапазоне свыше 0 и меньше 180 градусов.

## 2.6. Задачи для самостоятельной работы

- 1) Дано трехзначное число. Определить, какая из его цифр больше. Равен ли квадрат этого числа сумме кубов его цифр.
- 2) Дано трехзначное число. Определить, больше ли заданного числа  $x$  произведение его цифр.
- 3) Дано трехзначное число. Определить, есть ли среди его цифр одинаковые.
- 4) Даны радиус круга и сторона квадрата. У какой фигуры площадь больше?
- 5) Даны три вещественных числа  $a$ ,  $b$ ,  $c$ . Проверить, выполняется ли неравенство  $a < b < c$ .
- 6) Определить, является ли заданное шестизначное число таким, что сумма его первых трех цифр равна сумме его последних трех цифр.
- 7) Составить программу, которая уменьшает первое введенное число в два раза, если оно больше второго введенного числа по модулю.
- 8) Даны два числа. Если квадратный корень из второго числа меньше первого числа, то увеличить второе число в пять раз.
- 9) Даны три вещественных числа. Возвести в квадрат те из них, значения которых неотрицательны.
- 10) Даны четыре вещественных числа. Определить, сколько из них отрицательных.
- 11) Даны четыре вещественных числа. Найти сумму тех чисел, которые больше заданного  $x$ .
- 12) Даны три различных целых числа. Определить, какое из них является средним (больше наименьшего из данных чисел, но меньше наибольшего).
- 13) Даны четыре целых числа. Определить, сколько из них нечетных.

14) Даны четыре целых числа. Определить сумму тех из них, которые кратны  $x$ .

15) Даны два целых числа:  $A$ ,  $B$ . Проверить истинность высказывания: «Хотя бы одно из чисел  $A$  и  $B$  нечетное».

16) Дано целое положительное число. Проверить истинность высказывания: «Данное число является нечетным трехзначным».

17) Дано целое число, лежащее в диапазоне 1–999. Вывести его строку-описание вида «четное двузначное число».

18) Дано целое число, лежащее в диапазоне 1–999. Вывести его строку-описание вида «нечетное трехзначное число».

19) Даны четыре целых числа. Составьте программу, которая позволяет подсчитать и вывести среднее арифметическое чисел, имеющих нечетное значение.

20) Даны четыре целых числа. Составьте программу, которая позволяет подсчитать и вывести среднее геометрическое чисел, имеющих четное значение.

21) Если треугольник, заданный по трем сторонам, существует, то определить, является ли он разносторонним, равнобедренным или равносторонним.

22) Известны площади круга и квадрата. Определить, уместится ли круг в квадрате.

23) Даны целые числа  $a$ ,  $b$ ,  $c$ , являющиеся сторонами некоторого треугольника. Проверить истинность высказывания: «Треугольник со сторонами  $a$ ,  $b$ ,  $c$  является прямоугольным».

24) Даны стороны выпуклого четырехугольника. Можно ли в него вписать окружность?

25) Даны углы выпуклого четырехугольника. Можно ли около него описать окружность?

26) Даны стороны треугольника. Выяснить, является ли он остроугольным или тупоугольным.

27) Заданы координаты двух точек. Определить, лежат ли они на одной окружности с центром в начале координат.

28) Известны координаты трех точек. Напишите программу, которая определяет, находятся ли точки на одной прямой.

29) Даны два угла треугольника. Определить, является он прямоугольным, тупоугольным или остроугольным.

30) Даны две стороны и угол между ними четырехугольника. Определить фигуру: прямоугольник, квадрат, ромб, параллелограмм.

31) Пересекаются ли параболы  $y=px^2$  и  $y=ax^2+bx+c$ ? Найти точки их пересечения при положительном ответе.

32) Задана окружность с центром в точке  $O(x_0, y_0)$  и радиусом  $R_0$ . Определить пересекается ли заданная окружность с осью абсцисс, если пересекается найти точки пересечения.

33) Задана окружность с центром в точке  $O(x_0, y_0)$  и радиусом  $R_0$ . Определить пересекается ли заданная окружность с осью ординат, если пересекается найти точки пересечения.

34) Задана окружность с центром в точке  $O(0,0)$  и радиусом  $R_0$  и прямая  $y=ax+b$ . Определить, пересекаются ли прямая и окружность. Если пересекаются, найти точки пересечения.

35) Заданы окружности. Первая с центром в точке  $O(x_1, y_1)$  и радиусом  $R_1$ , вторая с центром в точке  $O(x_2, y_2)$  и радиусом  $R_2$ . Определить пересекаются окружности, касаются или не пересекаются.

36) Заданы три точки  $A(x_1, y_1)$ ,  $B(x_2, y_2)$ ,  $C(x_3, y_3)$ . Определить какая из точек наиболее удалена от начала координат.

37) Заданы три точки  $A(x_1, y_1)$ ,  $B(x_2, y_2)$ ,  $C(x_3, y_3)$ . Определить какая из точек  $B$  или  $C$  наименее удалена от точки  $A$ .

38) Определить, пересекаются ли линии  $y=ax+b$  и  $y=kx+m$ . Если пересекаются, найти точку пересечения.

39) Определить, пересекает ли линия  $y=ax+b$  ось абсцисс. Если пересекает, найти точку пересечения.

40) Определить, пересекаются ли линии  $y=ax^3+bx^2+cx+d$  и  $y=kx+m$ . Если пересекаются, найти точки пересечения.

41) Написать программу, которая определяет, лежит ли точка  $A(x_0, y_0)$  внутри треугольной области, ограниченной осями координат и прямой  $y=2-x$ .

42) Написать программу, которая определяет, лежит ли точка  $A(x, y)$  внутри некоторого кольца. Центр кольца находится в начале координат. Для кольца заданы внутренний и внешний радиусы  $r_1, r_2$ ; известно, что  $r_1$  отлично от  $r_2$ , но неизвестно,  $r_1 > r_2$  или  $r_2 > r_1$ .

43) Написать программу, которая определяет, лежит ли точка  $A(x_0, y_0)$  внутри квадратной области, ограниченной прямыми, параллельными осям координат:  $y = -2, y = 2, x = -2, x = 2$ .

44) Написать программу, при выполнении которой с клавиатуры считывается координата точки на прямой ( $x$  – действительное число) и определяется принадлежность этой точки одному из выделенных интервалов  $B$  и  $D$  (включая границы), если  $A(-\infty, a), B(a, b), C(b, c), D(c, d), E(d, +\infty)$ , где  $a, b, c, d$  – введенные целые числа.

45) Определить, в какую из областей — I, II или III — попадает точка с заданными координатами, где область I – выше прямой  $y=b$ , область II – между прямыми  $y=a$  и  $y=b$  ( $a < b$ ), область III – между прямой  $y=a$  и осью абсцисс.

46) Даны координаты точки на плоскости. Если точка совпадает с началом координат, то вывести 0. Если точка не совпадает с началом координат, но лежит на оси  $OX$  или  $OY$ , то вывести соответственно 1 или 2. Если точка не лежит на координатных осях, то вывести 3.

47) Дана окружность  $x^2+y^2=r^2$  (где  $r$  вводится с клавиатуры). Определить, лежат ли на ней точки  $A(x_1, y_1)$  и  $B(x_2, y_2)$  с заданными координатами.

48) Даны вещественные числа  $x$  и  $y$ . Определить принадлежит ли точка с координатами  $(x; y)$  области, если будут выполняться следующие условия:  $x \geq -1$ ,  $x \leq 3$ ,  $y \geq -2$  и  $y \leq 4$ .

49) Даны вещественные числа  $x$  и  $y$ . Определить принадлежит ли точка с координатами  $(x; y)$  области, если будут выполняться следующие условия:  $x^2 + y^2 \geq a$ ,  $x^2 + y^2 \leq b$ ,  $y < 0$ .

50) Задан круг с центром в точке  $O(x_0, y_0)$ , радиусом  $R_0$  и точка  $A(x_1, y_1)$ . Определить, находится ли точка внутри круга.

51) Даны оценки студента по трем предметам. Определить, получит ли он повышенную стипендию, обычную или не получит совсем.

52) Известен вес боксера ( $x$ ), такой, что боксер может быть отнесен к одной из трех весовых категорий: 1) легкий вес — до 60 кг; 2) первый полусредний вес — до 64 кг; 3) полусредний вес — до 69 кг. Определить, в какой категории будет выступать данный боксер.

53) В чемпионате по футболу команде за выигрыш дается 3 очка, за проигрыш — 0, за ничью — 1. Известно количество очков, полученных командой за игру ( $x$ ). Определить словесный результат игры (выигрыш, проигрыш или ничья).

54) Написать программу, которая выводит 3 примера на вычитание, запрашивает ответ пользователя и проверяет результаты. После подсчета количества правильных ответов выводится оценка «отлично» - 3 правильных, «хорошо» - 2 верных, «удовлетворительно» - 1 правильный, «не удовлетворительно» - нет верных ответов.

55) Написать программу, которая выводит пример на умножение двух однозначных чисел, запрашивает ответ пользователя, проверяет его и выводит сообщение «Правильно!» или «Вы ошиблись» и правильный результат.

56) Напишите программу, которая анализирует человека по возрасту и относит к одной из четырех групп: дошкольник, ученик, работник, пенсионер. Возраст человека вводится с клавиатуры.

57) Дан вопрос и три варианта ответа: «Чему равно число  $\pi$ ? а) 3,2; б) 3,14; в) 4,13». При выбранном числовом ответе а) вывести сообщение «это не точно», при б) – «верно», при варианте в) – «неправильно».

58) В психологическом тесте баллы распределяются следующим образом: менее 10 – меланхолик, от 11 до 20 – флегматик, от 21 до 30 – сангвиник, свыше 31 – холерик. По введенному количеству баллов ( $x$ ) определить темперамент.

59) На конкурсе нужно решить  $x$  головоломок и  $y$  задач. За каждое правильное решение присуждалось  $n$  очков. Участник, набравший наибольшее количество очков, является победителем, остальные – призёры. Все числа вводятся с клавиатуры, также вводится  $z$  – количество очков участника. Определить, является ли он победителем или призером.

60) С клавиатуры вводится  $x$  – количество набранных учеником баллов. Вывести оценку ученика, если на «5» необходимо набрать 20 баллов, на «4» необходимо набрать 16 баллов, на «3» необходимо набрать 12 баллов.

61) При судействе соревнований по выездке лошадей присваиваются баллы: 10 - отлично; 9 — очень хорошо; 8 — хорошо; 7 — довольно хорошо; 6 — вполне удовлетворительно; 5 — удовлетворительно; 4 — неудовлетворительно; 3 — довольно плохо; 2 — плохо; 1 — очень плохо; 0 — не выполнено. По числовому значению балла вывести его название.

62) Дано целое число в диапазоне от 20 до 69, определяющее возраст (в годах). Вывести строку-описание указанного возраста, обеспечив правильное согласование числа со словом «год», например: 20 «двадцать лет», 32 «тридцать два года», 41 «сорок один год».

63) Написать программу, которая пересчитывает вес из фунтов в килограммы. Программа учитывает, что в разных странах фунт весит по-разному.



Например, в России фунт равен 409,5 граммов, в Англии - 453,592 грамма, в Австрии – 0,560 г, в Италии – 0,317 г, а в Германии, Дании и Исландии фунт весит 500 граммов.

64) Единицы длины пронумерованы следующим образом: 1 – дециметр, 2 – километр, 3 – метр, 4 – миллиметр, 5 – сантиметр. Дан номер единицы длины и длина отрезка  $L$  в этих единицах (вещественное число). Вывести длину данного отрезка в метрах.

## Глава 3. Циклические конструкции

### 3.1. Цикл с параметром и его особенности

Рассмотрим тип задач, в которых некоторые действия приходится повторять неоднократно. Например, вывести несколько раз одну и ту же фразу:

```
console.log('Привет');
```

```
console.log('Привет');
```

```
...
```

```
console.log('Привет');
```

Даже комбинацию команд «Копировать-Вставить» надоест набирать.

Для облегчения процесса повторения во всех языках программирования имеются циклические конструкции. Самый простой вид цикла из трех – цикл с параметром (цикл-для, цикл с заданным числом повторений, цикл со счетчиком).

Его синтаксис содержится в названиях, «для» по-английски «for», параметр или счетчик – переменная, «пробегающая» все значения по порядку заданное количество раз.

В JavaScript он имеет вид:

```
for (счетчик=начальное значение; условие окончания цикла; следующий шаг счетчика)
```

Условие окончания чаще всего связано с проверкой счетчика на какое либо значение, например, счетчик<=конечное значение.

Рассмотрим решение предыдущего примера, уточнив количество повторений: вывести одну и ту же фразу 10 раз.

```
for (let i=1;i<=10;i++)
```

```
console.log('Привет');
```

```
index.js ×
1 for (let i=1;i<=10;i++)
2 console.log('Привет');
```

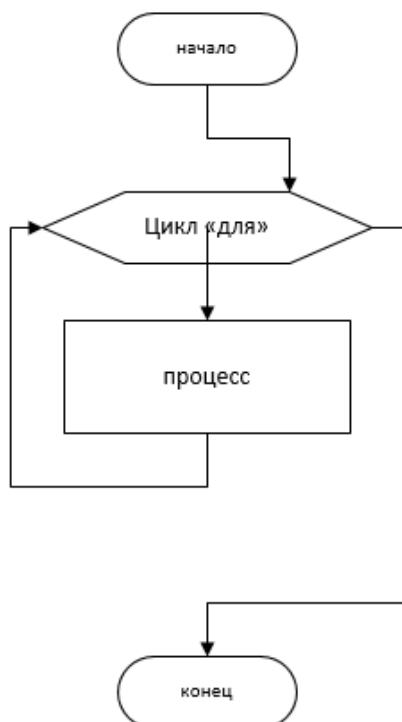
Console Shell

```
Привет
Привет
Привет
Привет
Привет
Привет
Привет
Привет
Привет
Привет
Привет
Hint: hit ctrl+c anytime to enter REPL.
> 
```

В этом цикле объект  $i$  – это переменная-счетчик. Изначально его значение равно единице. Т.к.  $1 < 10$  (истина), то выполнится 1 раз тело цикла. В теле содержится одна команда – вывод на экран фразы «Привет». Затем программа возвращается в цикл и выполняет команду  $i++$ . Следовательно, значение счетчика увеличится на единицу. Теперь  $i=2$ .  $2 < 10$  (истина), следовательно, снова выполнится команда тела цикла – вывод фразы.

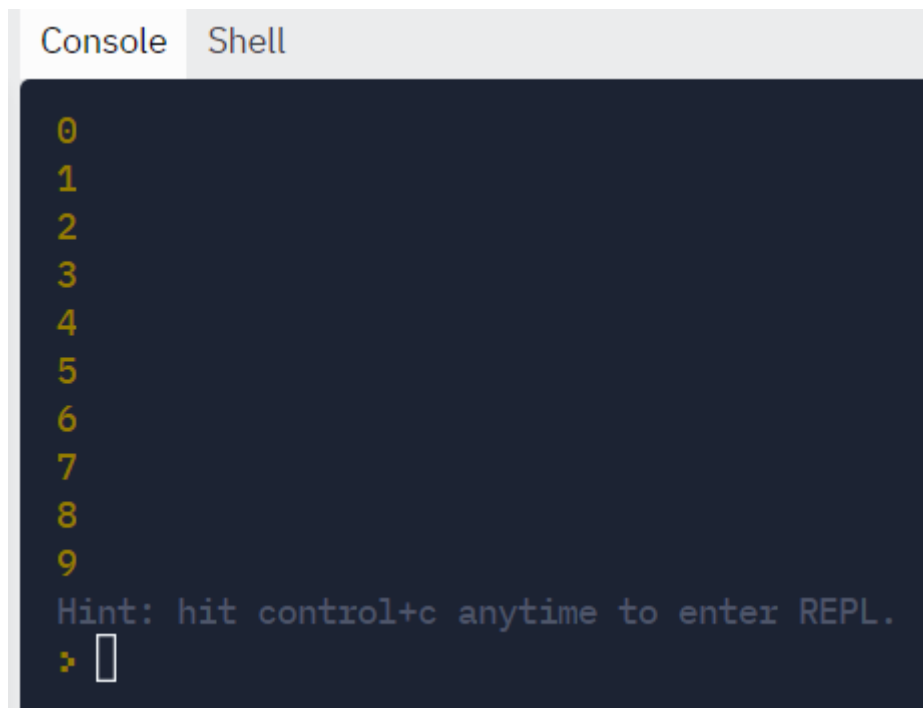
Таким образом, наш цикл будет «крутиться» 10 раз. После этого значение  $i=11$ . В условии  $11 < 10$  (ложь), тогда цикл заканчивает свою работу. Далее будут выполняться команды, которые не относятся к циклу, а идут за ним. В данном примере происходит выход из программы.

Изобразим блок-схему цикла с параметром:



Рассмотрим пример с числами: вывести в столбик числа 0123456789.

```
for (let i=0;i<10;i++)  
console.log(i);
```



```
Console Shell  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
Hint: hit control+c anytime to enter REPL.  
> █
```

Принцип работы программы такой же, как и у предыдущей. Только выводятся те значения, какие в данный момент принимает параметр цикла  $i$ .

Усложним задачу: вывести значения аргумента на отрезке  $[0.5 ; 1.5]$  с шагом  $h=0.1$ .

В некоторых языках программирования, например, Паскале, эту задачу с помощью цикла с параметром решить нельзя, т.к. счетчик цикла должен принимать только целочисленные значения. Но для JavaScript можно использовать и вещественные значения:

```
for (let i=0.5;i<=1.5;i+=0.1)  
console.log(i);
```

```
Console Shell

0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
1.2
1.3
1.4000000000000001
Hint: hit control+c anytime to enter REPL.
> |
```

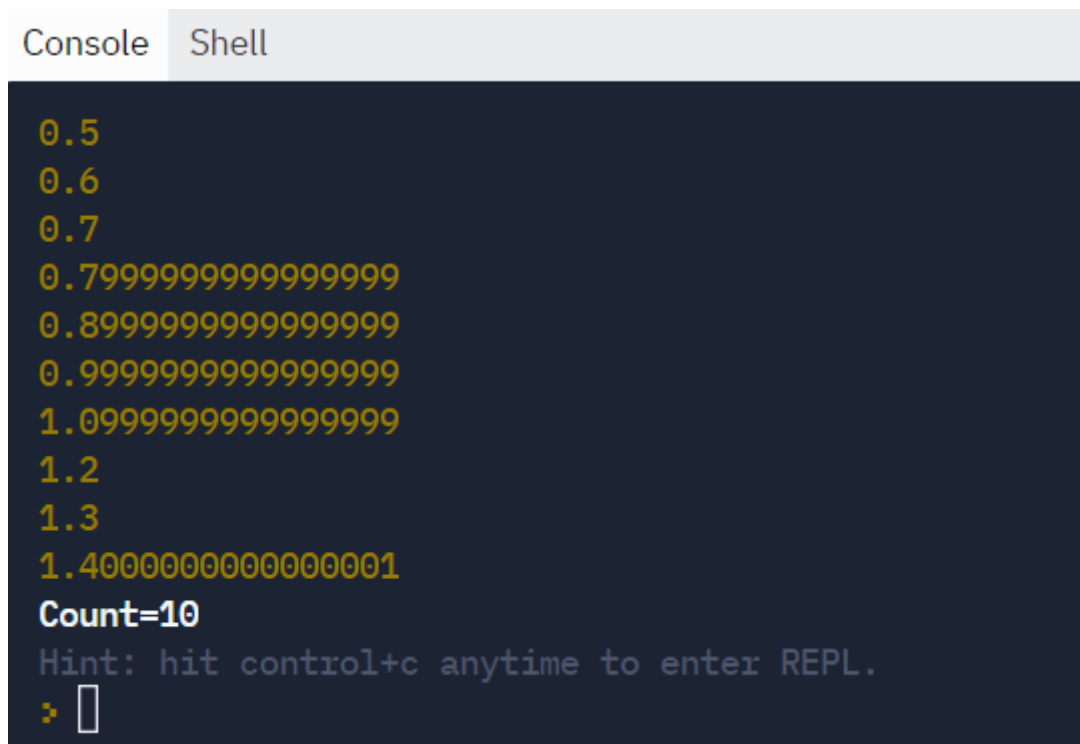
В этом примере команда  $i=i+0.1$  называется шагом цикла.

Но этот алгоритм дает погрешность вычислений. В консоли разработчика браузера результат тот же:

```
Elements Console Sources >> 2 | ⚙️ ⋮ ✕
▶ ⏸ ⏹ top | 👁 Filter Default levels | ⚙️
2 Issues: 2
> for (let i=0.5;i<1.5;i+=0.1)
  console.log(i);
0.5 VM88:2
0.6 VM88:2
0.7 VM88:2
0.7999999999999999 VM88:2
0.8999999999999999 VM88:2
0.9999999999999999 VM88:2
1.0999999999999999 VM88:2
1.2 VM88:2
1.3 VM88:2
1.4000000000000001 VM88:2
< undefined
> |
```

Если в теле цикла надо выполнить несколько команд, то они записываются в фигурных скобках `{}`. Проиллюстрируем на измененной задаче: вывести не только номера счетчика, но и их количество:

```
let count=0
for (let i=0.5;i<=1.5;i+=0.1)
{
console.log(i);
count++
}
console.log('Count='+count)
```



```
Console Shell
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
1.2
1.3
1.4000000000000001
Count=10
Hint: hit control+c anytime to enter REPL.
> █
```

В этом примере внутри тела цикла содержатся две команды – вывод значения и увеличения счетчика цикла на единицу. Вывод значения переменной `count` происходит уже за пределами цикла, иначе мы бы получили счет по каждому проходу по циклу. Передвинем скобку:

```
index.js x:
1 let count=0
2 for (let i=0.5;i<=1.5;i+=0.1)
3 {
4 console.log(i);
5 count++;
6 console.log('Count='+count)
7 }
```

```
Console Shell
0.5
Count=1
0.6
Count=2
0.7
Count=3
0.7999999999999999
Count=4
0.8999999999999999
Count=5
0.9999999999999999
Count=6
1.0999999999999999
Count=7
1.2
Count=8
1.3
Count=9
1.4000000000000001
Count=10
Hint: hit ctrl+c anytime to enter REPL.
>
```

Теперь «нечаянно» оставим одну команду внутри цикла:

```
index.js x:
1 let count=0
2 for (let i=0.5;i<=1.5;i+=0.1)
3 {
4 console.log(i);
5 }
6 count++;
7 console.log('Count='+count)
```

```
Console Shell
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
1.2
1.3
1.4000000000000001
Count=1
Hint: hit ctrl+c anytime to enter REPL.
>
```

Получили неверный результат.

Чтобы не ошибиться с расстановкой скобок, отделяющих тело цикла от заголовка, надо мысленно представить блок-схему, в которой блок «процесс» распадается на несколько блоков.

Внутри тела цикла могут быть блоки ветвления и варианты, другие циклы, а не только линейные процессы.

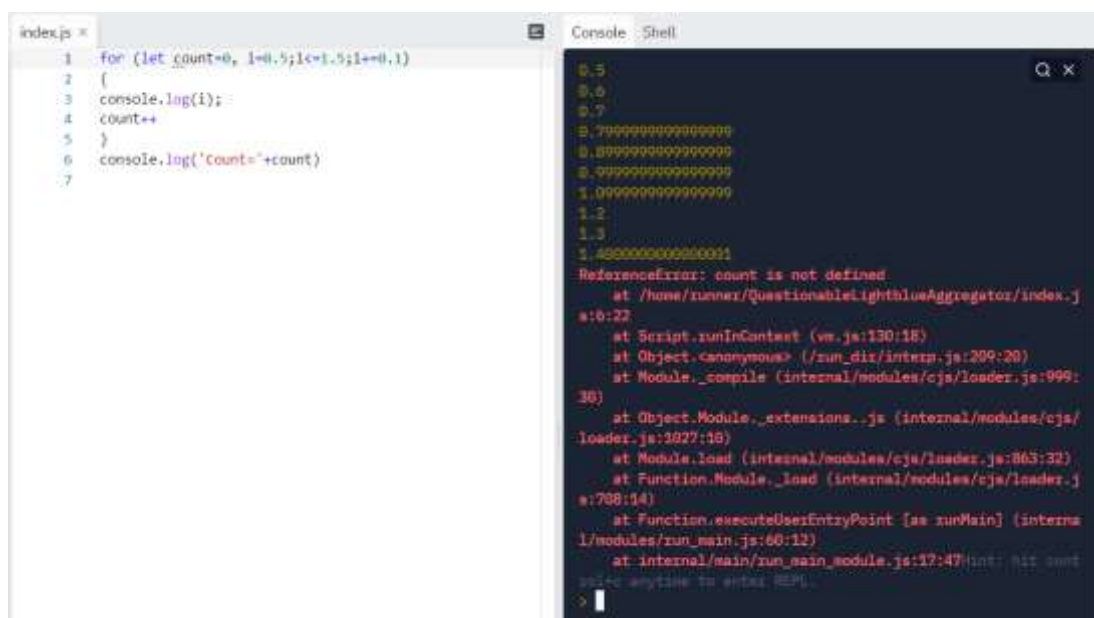
Блок цикла с параметром for (счетчик=начало; счетчик < конец; шаг-приращение) записывается так в общем виде, но его структура не слишком жесткая. Во-первых, можно внутрь заголовка добавить объявление переменных, во-вторых, убрать часть блоков. В-третьих, изменить условие окончания цикла.

Проиллюстрируем варианты на предыдущем примере:

1. В данном скрипте объявление переменной count заносится в заголовок цикла первым блоком.

```
for (let count=0, i=0.5;i<=1.5;i+=0.1)
{
console.log(i);
count++
}
console.log('Count='+count)
```

Но если работа с этой переменной происходит за циклом, то JavaScript вернет сообщение об ошибке:



Поэтому объявление внутри цикла лучше использовать, когда переменная больше нигде не будет встречаться, т.е. следить за областью видимости.



```
index.js x Console Shell
1 for (let count=0, i=0.5;i<=1.5;i+=0.1)
2 {
3   console.log(i);
4   count++
5   console.log('Count='+count)
6 }
```

```
0.5
Count=1
0.6
Count=2
0.7
Count=3
0.7999999999999999
Count=4
0.8999999999999999
Count=5
0.9999999999999999
Count=6
1.0999999999999999
Count=7
1.2
Count=8
1.3
Count=9
1.4000000000000001
Count=10
Hint: hit control+c anytime to enter REPL.
> |
```

## 2. Оператор цикла с двумя пустыми блоками.

```
index.js x Console Shell
1 let count=0, i=0.5
2 for (;i<=1.5;)
3 {
4   console.log(i);
5   count++
6   i+=0.1
7 }
8 console.log('Count='+count)
9
```

```
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
1.2
1.3
1.4000000000000001
Count=10
Hint: hit control+c anytime to enter REPL.
> |
```

Если блок №1 – начало цикла и блок №3 – шаг цикла пустые, то инициация переменной (установка начального значения) должна осуществляться перед циклом, а приращение значения добавляется в тело цикла. В противном случае цикл будет выполняться бесконечно, т.е. код «защелкивается».

3. Если тело цикла содержит одну команду (или небольшое количество), то его можно через запятую записать в заголовок цикла:

```
for (i=0.5;i<=1.5; console.log(i), i+=0.1);
```

Если тело цикла пустое, то в конце оператора for обязательно ставится точка с запятой “;”. В противном случае следующая за циклом команда будет автоматически считаться телом цикла.

4. Все три блока заголовка цикла пустые, а условие окончания использует ключевое слово `break` – досрочный выход из цикла:

```
i=0.5
for (;;)
{
  console.log(i)
  i+=0.1
  if (i>1.5) break
}
```

Результат получим тот же, но такой вариант цикла используется редко. Особенность цикла с параметром как раз и состоит в том, чтобы не следить дополнительно за изменением параметра и выходом из цикла. В теле цикла должно быть только решение задачи и ничего лишнего.

Если необходимо отслеживать параметры изменения цикла, особенно в тех случаях, когда условие окончания задано в неявном виде, то используют один из двух вариантов цикла с неизвестным числом повторений – цикл с предусловием («пока») или цикл с постусловием («до»).

### 3.2. Цикл с предусловием

Этот цикл содержит любое условие в явном виде (как и в условном операторе `if`), но действия при истинном значении этого условия могут повторяться многократно.

Синтаксис оператора «пока» почти одинаков во всех языках программирования, в JavaScript он имеет вид:

```
while (условие)
{
  Выполняемые команды
```

}

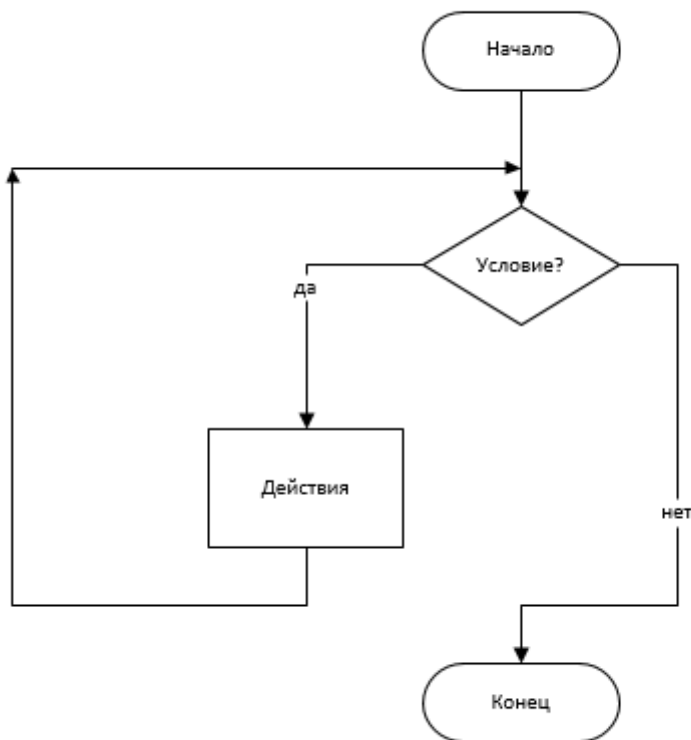
После проверки условия возможны следующие варианты:

1) Условие истинно, значит, первый раз выполняются команды тела цикла. Затем программа заново возвращается к проверке условия. На одном из шагов условие станет ложным, цикл останавливается, и выполняется следующий оператор, который идет после тела цикла.

2) Условие ложно, а значит, тело цикла не выполнится ни разу. Такой скрипт не является синтаксически ошибочным, но на деле он не имеет смысла.

3) Условие всегда истинно – программа зациклилась. Это ошибка, исправить которую можно заменой условия или оператора присваивания в том объекте, от которого зависит условие.

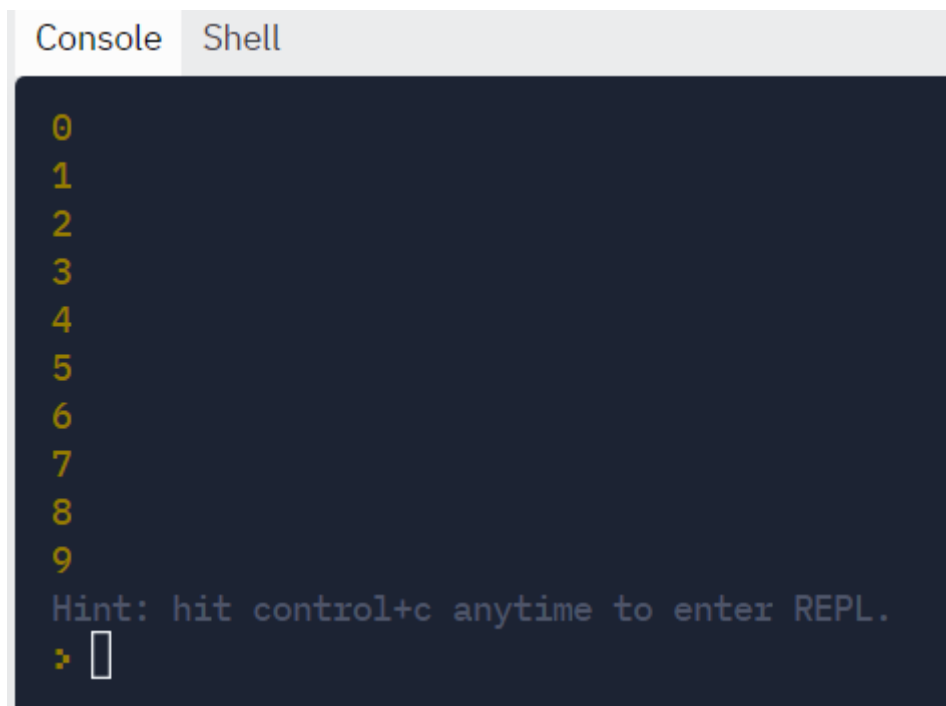
На блок-схеме циклического процесса с предусловием показана петля возврата в очередную проверку условия:



Рассмотрим тот же пример с числами: вывести в столбик числа 0123456789.

```
let i=0
while (i<9)
{
  console.log(i)
  ++i
}
```

Результат:



```
Console Shell
0
1
2
3
4
5
6
7
8
9
Hint: hit control+c anytime to enter REPL.
> █
```

Если менять условие ( $i \leq 9$ ,  $i < 10$ ) и оператор инкремента, то можно получить другие результаты, хотя логически всё верно.

Например, в следующей задаче цикл выполнится 10 раз, и в результате не захватится последнее значение  $i=1.5$  из-за проблем с точностью хранения вещественных чисел.

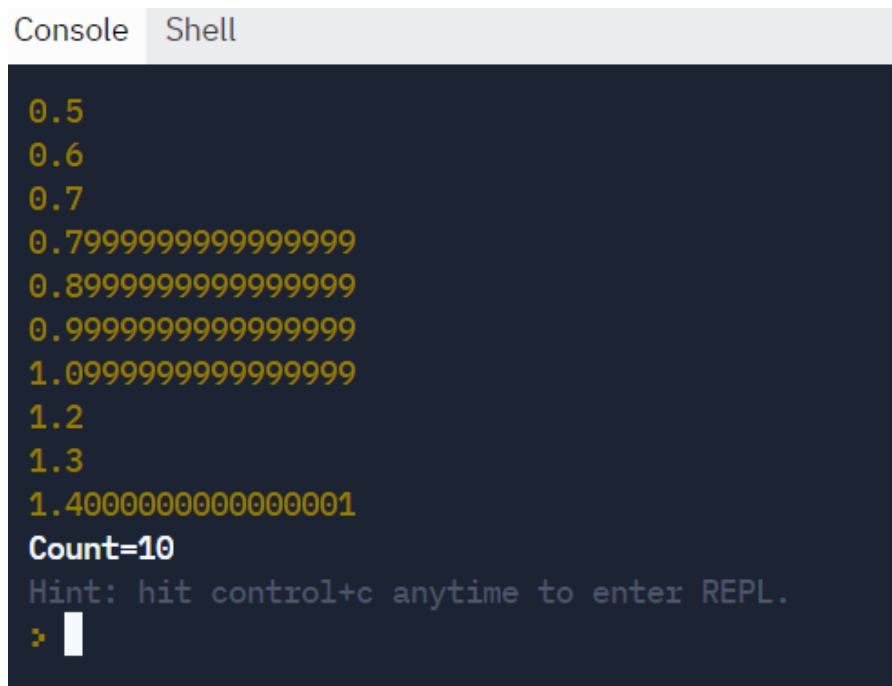
```
let count=0
let i=0.5;
```

```
while (i<=1.5)
{
console.log(i);

count++

i+=0.1
}

console.log('Count='+count)
```



```
Console Shell
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
1.2
1.3
1.4000000000000001
Count=10
Hint: hit control+c anytime to enter REPL.
> |
```

Этот вид цикла удобен для расчетов сумм последовательностей (прогрессий и рекуррентных соотношений) или количества членов последовательности при достижении заданной суммы (произведения). При многократном вводе значения переменной, пока не достигнуто останавливающее значение (например, 0 или пустая строка).

### 3.3. Цикл с постусловием

Цикл с постусловием (или цикл «до») – обратный к циклу с предусловием.

В классическом варианте старых языков программирования (Basic, Pascal) его операторы содержали слово `until` (до тех пор, пока), за которым следовало условие. Пока условие было ложно, операторы тела цикла повторялись, а при истинном значении происходил выход из цикла.

Но в языке JavaScript (как в наследии языка C) этот оператор видоизменился в `do-while`. Условие точно так же проверяется после выполнения действий, а проверка истинности, как и в цикле «пока» - повторение команд, пока истинно.

Полный синтаксис имеет вид:

```
do
{
  Команды тела цикла
}
while (условие)
```

Команды в теле этого цикла выполняются хотя бы один раз, даже если условие сразу ложно. При работе с циклом «до» важно просчитывать количество повторений – на каком шаге они прервутся, чтобы не выполнить лишнее действие.

Цикл с постусловием можно использовать для всех предыдущих задач, но удобнее всего в вычислениях последующих членов или сумм бесконечных рядов, когда предыдущее значение сравнивается с текущим по заданному значению точности.

Блок-схема цикла с постусловием:



Пример первой программы с выводом чисел 0123456789 в цикле «до»:

```

let i=0
do
{
console.log(i)
++i
}
while (i<9)
  
```

Результат будет тот же, что и для предыдущих видов цикла.

Проиллюстрируем примером подсчет суммы бесконечного ряда и количества шагов, при котором достигается заданная точность.

Определить число членов ряда, необходимых для расчета с заданной погрешностью суммы членов ряда:

$$1 / (1*3) + 1 / (3*5) + 1 / (5*7) + \dots + 1 / ((2*N-1)*(2*N+1)) + \dots$$

```

let s=0;
let s0=0
    var eps=parseFloat(prompt("Введите погрешность"));
    let i=1;
    do
    {
    s0= 1/((2*i-1)*(2*i+1));
        s+=s0;
        i++;
    }
    while (s0>eps)
    console.log("Число членов ряда =",i,"\nСумма ряда =",s,"\nПоследний
член ряда =",s0)

```

Результат:

```

Console Shell
Введите погрешность> 0.00001
Число членов ряда = 160
Сумма ряда = 0.4984326018808773
Последний член ряда = 0.00000988894712379973
Hint: hit control+c anytime to enter REPL.
> █

```

### 3.4. Условия досрочного выхода из цикла

1) Оператор break прерывает выполнение цикла по заданному условию.

Например, в скрипте:

```

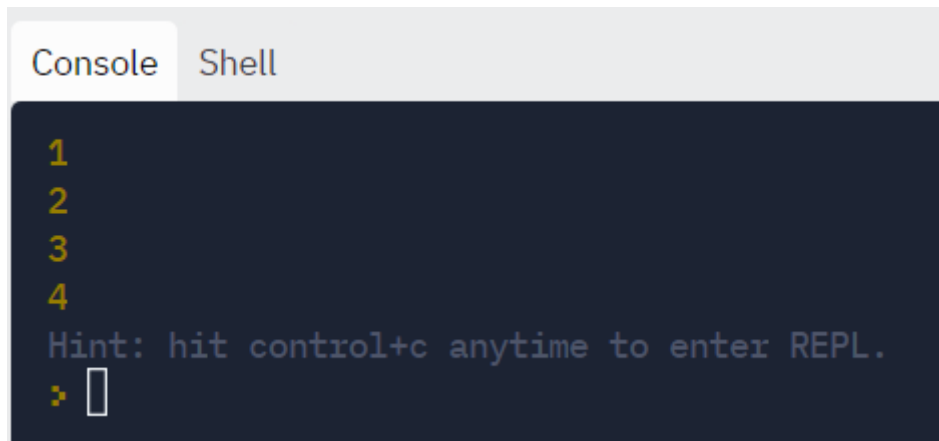
for (i=1;i<11;i++)
{

```



```
if (i==5) break
console.log(i)
}
```

выведутся только первые четыре значения из десяти:



```
Console Shell
1
2
3
4
Hint: hit control+c anytime to enter REPL.
> 
```

Такой оператор удобно применять, когда условие может выполниться несколько раз, а нам достаточно одного первого раза, чтобы завершить программу.

Например, решим задачу: даны 5 чисел последовательности и отдельное число  $x$ . На каком шаге элемент последовательности совпадет с  $x$ ? При этом известно, что в последовательности есть 2 одинаковых числа.

Вариант 1 – без досрочного прерывания цикла:

```
let count =0
let x=prompt()
for (i=1;i<6;i++)
{
  a=prompt()
  if (a==x)
  count=i
}
console.log(count)
```

Результат:

```
Console Shell
undefined> 3
undefined> 1
undefined> 3
undefined> 2
undefined> 3
undefined> 4
4
Hint: hit control+c anytime to enter REPL.
> 
```

Число 3 встретится на 2 и 4 шагах цикла, а в ответе будет 4 – номер выполнения последней подходящей итерации.

Вариант 2 – с оператором break:

```
let count =0
let x=prompt()
for (i=1;i<6;i++)
{
  a=prompt()
  if (a==x)
  {
    count=i
    break
  }
}
console.log(count)
```

Результат:

```
Console Shell
undefined> 3
undefined> 1
undefined> 3
2
Hint: hit control+c anytime to enter REPL.
> 
```

Цикл работает до первого совпадения с числом 3 на шаге 2. Ввод остальных данных не понадобился.

2) Оператор `continue` прерывает выполнение текущего шага цикла (итерации) и переходит на следующий шаг.

Например, вывести числа от 1 до 9 за исключением числа 5:

```
for (i=1;i<10;i++)
{
  if (i==5)
  continue
  console.log(i)
}
```

Результат:

```
Console Shell
1
2
3
4
6
7
8
9
Hint: hit control+c anytime to enter REPL.
> █
```

Этот оператор удобно использовать, когда не все значения из промежутка подходят для области допустимых значений функции при подстановке в выражение.

Решим задачу вывода значений функции  $y = \frac{1}{x}$  на отрезке  $[-3;3]$ , исключая  $x=0$ .

Вариант 1 – без использования дополнительных условий:

```
for (i=-3;i<=3;i++)
{
  console.log(1/i)
}
```

В ответе при  $x=0$  получим бесконечность:

```
Console Shell
-0.3333333333333333
-0.5
-1
Infinity
1
0.5
0.3333333333333333
Hint: hit control+c anytime to enter REPL.
> 
```

Вариант 2 – с использованием оператора continue:

```
for (i=-3;i<=3;i++)
{
if (i==0)
continue
console.log(1/i)
}
```

Результат:

```
Console Shell
-0.3333333333333333
-0.5
-1
1
0.5
0.3333333333333333
Hint: hit control+c anytime to enter REPL.
> 
```

Вариант 3 – с использованием истинного условного оператора:

```
for (i=-3;i<=3;i++)
```

```
{  
if (i!=0)  
console.log(1/i)  
}
```

Результат будет, как и в предыдущем случае.

### 3.5. Вложенные операторы цикла

Циклы не могут перекрываться между собой, но можно использовать вложенные циклы, т.е. в теле одного цикла может полностью помещаться другой цикл.

Классический пример вложенного цикла – таблица умножения двух чисел:

```
for (i=1;i<=9;i++)  
{  
for (j=1;j<=9;j++)  
console.log(i*j)  
console.log()  
}
```

Фрагмент консоли вывода:

```
Console Shell
1
2
3
4
5
6
7
8
9

2
4
6
8
10
12
14
16
18

3
6
9
12
15
18
21
24
27
```

В этом примере сначала выполняется первый шаг первого цикла, затем полностью второй (внутренний) цикл. Т.е. счетчик  $j$  «пробежал» все 9 значений с умножением на  $i=1$ . Затем вернулся назад и счетчик  $j$  снова прошел все 9 значений с умножением на  $i=2$ . Процесс повторяется 9 раз, т.е. полностью серия циклов выполнится 81 раз:

```
9
18
27
36
45
54
63
72
81

Hint: hit control+c anytime to enter REPL.
> 
```

Для вложенного цикла `while` или `do-while` недостаточно записать просто 2 условия через оператор логического И - `&&`. Например:

```
let i=1, j=1
while (i<3 && j<3)
{
  console.log(i*j)
  ++i
  ++j
}
```

Результат:

```
Console Shell
1
4
3
Hint: hit control+c anytime to enter REPL.
> 
```

Необходимо «вложить» один цикл в другой по тем же правилам:



```
let i=1
while (i<=3)
{
  j=1
  while (j<=3)
  {
    console.log(i*j)
    j++
  }
  console.log()
  i++
}
```



```
Console Shell
1
2
3

2
4
6

3
6
9

3
Hint: hit control+c anytime to enter REPL.
➤
```

В результате получилось лишнее число, поэтому цикл for в данном случае надежнее.

В скрипте JS можно комбинировать разные виды циклов:

```
let i=1
while (i<=3)
{
  for (j=1;j<=3;j++)
  {
    console.log(i*j)
  }
  console.log()
  i++
}
```

Или:

```
for (i=1;i<=3;i++)
{
  j=1
  while (j<=3)
  {
    console.log(i*j)
    j++
  }
  console.log()
}
```

### 3.6. Скрипты с циклическими конструкциями

Задача №1. Рассчитать среднее арифметическое положительных чисел, вводя последовательность до тех пор, пока не введен 0.

```
let SP=0

let N=0

var x=parseInt(prompt('Следующее число: '));

while (x != 0)

{

  if (x > 0)

    { SP=SP+x

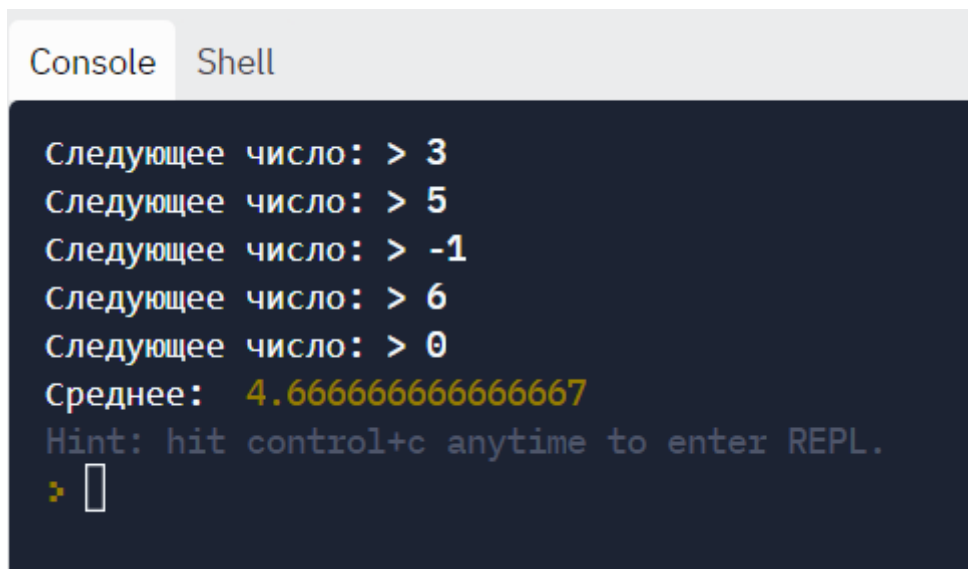
      N=N+1}

  x=parseInt(prompt('Следующее число: '));

}

console.log('Среднее: ',SP/N)
```

Результат:



```
Console Shell

Следующее число: > 3
Следующее число: > 5
Следующее число: > -1
Следующее число: > 6
Следующее число: > 0
Среднее: 4.666666666666667
Hint: hit control+c anytime to enter REPL.
> █
```

Примечание к коду:

В этом скрипте использовался цикл с предусловием. Команду ввода значения придется применять два раза, чтобы сравнить первое введенное значение с нулем.

Использование цикла с постусловием упростит задачу:

```
let SP=0
let N=0
let x=0
do
{
x=parseInt(prompt('Следующее число: '));
if (x > 0)
{
SP=SP+x
N=N+1
}
}
while (x != 0)
console.log('Среднее: ',SP/N)
```

Результат мы получим тот же.

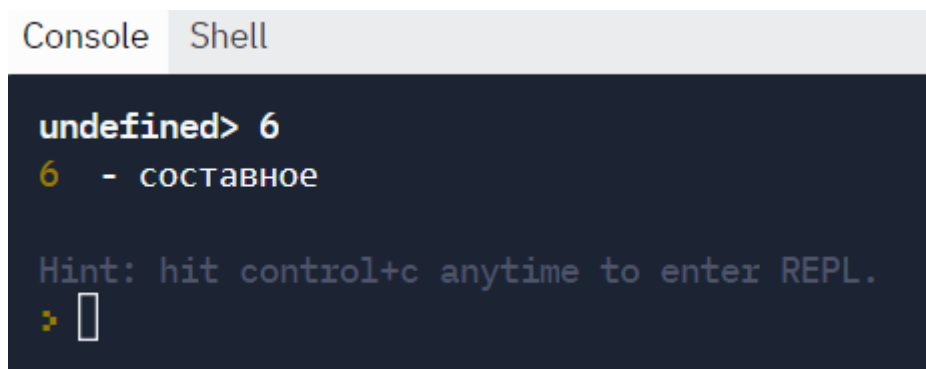
А классический цикл с параметром (for) при решении этой задачи применять нельзя, т.к. заранее неизвестно число шагов цикла – на каком шаге мы введем ноль?

Задача №2. Определить, является ли число простым

```
var N=parseInt(prompt());
let Pr=true;
```

```
for (i=2; i<=N/2; i++)
if (N%i==0)
{Pr=false;
break;}
if (Pr)
console.log(N," - простое \n");
else
console.log(N," - составное \n");
```

Результат:



```
Console Shell
undefined> 6
6 - составное

Hint: hit control+c anytime to enter REPL.
➤
```

Примечания к коду:

Простым называется число, которое делится на единицу и на само себя, поэтому в параметрах цикла эти значения не учитываются. Кроме того, нет необходимости проверять, делится ли число на делитель, больший его половины. Например, 24 не будет делиться на число, большее 12 (его половины). Данный алгоритм проводит проверку от противного с помощью флага – логической переменной Pr. Если число N делится хотя бы на одно из чисел 2,3,... N/2, то оно не является простым. Чтобы алгоритм стал эффективным по времени выполнения, в цикл добавлен оператор break – досрочный выход из цикла. Например, четное число уже на первом шаге деления на 2 окажется составным, и нет смысла проверять делимость дальше.

Этот алгоритм можно видоизменить для нахождения количества делителей, самих делителей, наибольшего общего делителя и наименьшего общего кратного.

Второй способ решения этой задачи – с помощью цикла с предусловием:

```
do
{
x=parseInt(prompt())
if (x<=0) console.log('введите положительное число')
}
while (x<=0)
let h=x / 2
let delitel=2;
let i=0;
while (delitel<=h)
{
if (x % delitel==0)
i=i+1;
delitel=delitel+1;
}
if (i==0) console.log('простое')
else console.log('не простое');
```

Примечания к коду:

В первом цикле (с постусловием do-while) производится корректировка ввода – защита от ввода неправильного значения. Во втором цикле (с предусловием while) производится расчет, как и в предыдущем случае. Флаговая переменная *i* принимает целочисленное значение, а не булево.

Эта программа не эффективна по времени – если число слишком большое или во внешнем цикле обрабатывается много чисел, то скрипт может не работать. Для корректной работы необходимо поменять условие последнего делителя на  $\sqrt{x}$  и добавить оператор досрочного выхода. В итоге получим:

```
do
{
x=parseInt(prompt())
if (x<=0) console.log('введите положительное число')
}
while (x<=0)
let delitel=2;
let i=0;
while (delitel<=Math.sqrt(x))
{
if (x % delitel==0)
{
i=i+1;
break
}
delitel=delitel+1;
}
if (i==0) console.log('простое')
else console.log('не простое');
```

Задача №3. В арифметической прогрессии с начальным значением  $a_1 = 5$  и разностью  $d = 2$  вывести первые  $n$  членов прогрессии и найти сумму  $n$  членов.

```

var n=parseInt(prompt("введите количество"))
let k=5; let d=2; let s=0; let i=0
let an=k+d*(n-1);
while (k<=an)
{
    s=s+k;
    k=k+d;
    i++
    alert(i,"-й член = ",k, " ");
}
alert("Сумма = ",s);

```

Результат:

```

Console Shell
введите количество> 5
1 -й член = 7
2 -й член = 9
3 -й член = 11
4 -й член = 13
5 -й член = 15
Сумма = 45
Hint: hit control+c anytime to enter REPL.
> 

```

Примечания к коду:

Этот алгоритм напрямую производит расчеты по формуле, увеличивая каждый член на разность прогрессии и добавлением суммы к предыдущему значению. Вычисления заканчиваются, когда член прогрессии равен последнему члену, подсчитанному по формуле.



Этот алгоритм можно упростить, если цикл повторять до нужного количества  $n$ , а сумму прогрессии рассчитать линейно по формуле  $S_n = \frac{2a_1 + d(n-1)}{2} \cdot n$

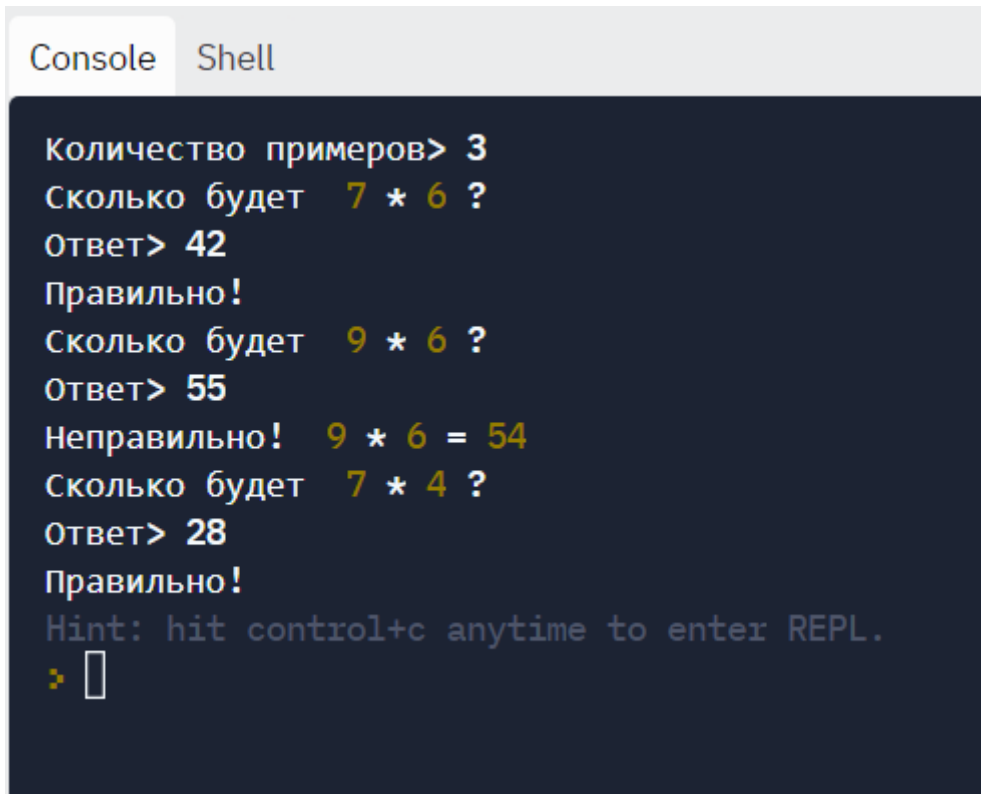
```
var n=parseInt(prompt("введите количество"))
let k=5; let d=2; let s=0; let a1=k
for (i=1;i<=n;i++)
{
    k=k+d;
    alert(i,"-й член = ",k, " ");
}
s=(2*a1+d*(n-1))*n/2
alert("Сумма = ",s);
```

Задача №4. Составить программу проверки знаний учеником таблицы умножения. Компьютер задает ученику  $n$  вопросов на умножение чисел от 2 до 9. На каждое задание ученик вводит свой ответ, компьютер сообщает, верный ответ или нет.

```
var n=parseInt(prompt("Количество примеров"))
for (i=1;i<=n;i++){
    let x=Math.round(Math.random()*8+2);
    let y=Math.round(Math.random()*8+2);
    alert('Сколько будет ',x,'*',y,'?');
    let z=parseInt(prompt("Ответ"))
    if (z==x*y)
        alert('Правильно!');
    else
        alert('Неправильно! ',x,'*',y,'!=' ,x*y);
```

```
}
```

Результат:



```
Console Shell
Количество примеров> 3
Сколько будет 7 * 6 ?
Ответ> 42
Правильно!
Сколько будет 9 * 6 ?
Ответ> 55
Неправильно! 9 * 6 = 54
Сколько будет 7 * 4 ?
Ответ> 28
Правильно!
Hint: hit control+c anytime to enter REPL.
> █
```

Пояснения к коду:

Циклы часто используются для определения количества попыток в играх или тестах, например, угадать число или слово, решить нужное количество заданий и т.д. Часто при этом используется генератор псевдослучайных чисел из заданного промежутка.

Задача №5. Рассчитать значения функции  $y = \sin(x)/(x+A)^2$  при изменении аргумента "x" в диапазоне от 0 до  $\pi/2$  с шагом  $\pi/80$  и при изменении параметра "A" в диапазоне от 1 до 2 с шагом 0, 2.

```
let x=0
let A=1
while (x<=Math.PI/2 && A<=2)
{
```

```

y=Math.sin(x)/Math.pow((x+A),2)

x=x+Math.PI/80

A=A+0.2

alert(y)

}

```

Результат:

```

Console Shell
0
0.02556327243961025
0.0358903133190519
0.039831435310644814
0.04084279427331655
0.040442017271212834
Hint: hit control+c anytime to enter REPL.
>

```

Оба параметра меняются в одном цикле одновременно.

Второй вариант решения задачи:

```

for (x = 0;x< Math.PI / 2;x+= Math.PI / 80)
{
for (A = 1; A< 2; A+= 0.2)
{
y=Math.sin(x)/Math.pow((x+A),2)
alert(y)
} }

```

Результат:

Console

Shell

```
0
0
0
0
0
0
0.03634892338388165
0.02556327243961025
0.018952380475436962
0.014609902707867396
0.011605325508425272
0.009440583452937924
0.06744828840738325
0.047997076312719204
0.0358903133190519
0.027847101857036958
0.02223322634573244
0.018160449840631976
0.09406765187219862
0.06768160581097273
0.051020112069640945
0.039831435310644814
0.03195692896787919
0.02620608793166628
0.11684387279747303
0.0849418618727665
0.06452244163182384
0.050669842239526065
0.04084279427331655
0.03362018392197578
0.13630743677437018
0.10005698966392985
0.07655608962904777
0.06045803545705079
0.04895111145797096
0.040442017271212834
```

В результате мы получим гораздо больше значений, т.к. количество итераций возрастет в 20 раз за счет того, что параметр A изменяется в отдельном цикле. Как правильно, решать пользователю-математику.

### 3.7. Контрольные вопросы и тесты

#### Дополнительные вопросы

1. Какими должны быть параметры цикла for, чтобы значения вывелись от последнего к первому?
2. Что общего у циклов с предусловием и постусловием?
3. Являются ли циклы с предусловием и постусловием взаимозаменяемыми?
4. Может ли тело цикла состоять из пустого оператора?
5. Как избежать «заикливания» программы?
6. Как посчитать количество итераций (шагов) в цикле с заданным числом повторений?
7. Можно ли для счетчика цикла с параметром использовать дробные значения?
8. Сколько раз выполнится тело цикла с постусловием, если условие всегда ложно?
9. С помощью какой команды можно прервать выполнение цикла и перейти к следующему шагу?
10. Для решения каких задач не подходит цикл с параметром?

#### Тесты с выбором варианта ответа

1. Оператор цикла DO/WHILE является:
  - 1) конструкцией цикла с предусловием;

- 2) конструкцией цикла с постусловием;
- 3) конструкцией цикла с выбором варианта;
- 4) конструкцией цикла с перебором значений параметра.

2. Написать фрагмент скрипта, выводящий: 1 2 3 4 5 6 7 8 9 10:

- 1) for i=1 to 10 do console.log(i);
- 2) for (i=1; i<=10; i++) console.log(i);
- 3) for (i=0,i<10; ++i) console.log(i);
- 4) for i=1; i<10; i=i+1; console.log(i);

3. Какое слово не относится к циклической программе?

- 1) while
- 2) for
- 3) repeat
- 4) do

4. Оператор цикла с предусловием?

- 1) while
- 2) do while
- 3) for
- 4) if

5. Конструкция вида: for (;i<n;s+=i\*i,i++);

- 1) ошибочна, нельзя начинать цикл с точки с запятой
- 2) верна
- 3) ошибочна, нельзя выполнять действия в заголовке цикла
- 4) ошибочна, нельзя использовать 3 и более переменных в цикле

6. Чем отличается цикл «пока» от цикла «до»?

- 1) «пока» – с параметром, «до» – с условием
- 2) «пока» – с условием, «до» – с параметром
- 3) «пока» – с постусловием, «до» – с предусловием
- 4) «пока» – с предусловием, «до» – с постусловием.

7. Два и более цикла в одной конструкции называются:

1) рекурсивными

2) последовательными

3) вложенными

4) зависимыми

8. Написать фрагмент программы, выводящий следующее:

1

22

333

4444

55555

1) `var s = "";`

`for (loop = 1; loop < 5; loop++)`

`{`

`for (loop1 = 1; loop1 < loop; loop1++)`

`{`

`s += loop1 + " ";`

`}`

`console.log(s);`

2) `s = "";`

`}`

`var s = "";`

`for (loop = 1; loop <= 5; loop++)`

`{`

`for (loop1 = 1; loop1 <= loop; loop1++)`

```
{  
    s += loop + " ";  
}  
console.log(s);  
s = "";  
}
```

```
3) var s = "";  
for (loop = 1; loop <= 5; loop++)  
    {  
        for (loop1 = 1; loop1 <= loop; loop1++)  
            {  
                s += loop + " ";  
            }  
        console.log(s);  
    }  
s = "";  
}
```

9. Конструкция вида: `for (var n=100,s=0,i=1;i<=n; i++)`;

- 1) ошибочна, нельзя использовать 3 переменные в цикле
- 2) ошибочна, нельзя выполнять присваивание в цикле
- 3) верна
- 4) ошибочна, нельзя начинать цикл с 1

10. Какой оператор цикла с предпроверкой условия?

- 1) Repeat
- 2) While
- 3) For
- 4) Until



11. Чем отличается цикл «для» от цикла «до»?

- 1) «для» – с параметром, «до» – с постусловием
- 2) «для» – с условием, «до» – с параметром
- 3) «для» – с постусловием, «до» – с предусловием
- 4) «для» – с предусловием, «до» – с постусловием.

12. Какое ключевое слово не относится к циклической программе?

- 1) If          2) While          3) For          4) Do

### Тесты без выбора варианта ответа

1. Найти сумму чисел от 1 до заданного N

```
var n=parseInt(prompt())
```

```
let s=0;
```

```
for (i=1;i<=n;i++)
```

```
s=s+i;
```

```
alert(s);
```

Переписать скрипт, используя цикл: а) с предусловием; б) с постусловием.

2. Дан скрипт:

```
For (k=1;k<100;k++)
```

```
{
```

```
If (k==5) break;
```

```
Console.log(k)
```

```
}
```

Сколько раз выполнится этот цикл и что выведет скрипт в результате его выполнения?

3. Дан скрипт с ошибкой:

```
Let k=5
While (k>0)
{
Console.log(k)
}
```

Какую строку надо добавить, чтобы цикл перестал работать бесконечно?

4. Исправьте ошибки в данном скрипте:

```
do
k=5
while (k>0)
{
console.log(k)
k--
}
```

5. Какие значения будут выведены в результате выполнения скрипта?

```
for (i=1;i<11;i++)
{
if (i==5) continue
if (i==9) continue
if (i==7) break
console.log(i) }
```

### 3.8. Задачи для самостоятельной работы

- 1) Найти сумму всех удвоенных четных целых чисел от  $a$  до 100 (значение  $a$  вводится с клавиатуры;  $a < 100$ ).
- 2) Найти произведение всех нечетных целых чисел от  $a$  до  $b$ , не делящихся на 11 (значения  $a$  и  $b$  вводятся с клавиатуры;  $b > a$ ).
- 3) Найти среднее арифметическое всех целых чисел от  $a$  до 200 (значение  $a$  вводится с клавиатуры;  $a < 200$ ).
- 4) Найти сумму кубов всех целых чисел от 20 до 40 и разность кубов целых чисел от 50 до 90.
- 5) Найти сумму квадратов всех целых отрицательных чисел и произведение положительных чисел из промежутка  $(-N; N)$ .
- 6) Найти среднее арифметическое и среднее геометрическое всех целых чисел из промежутка от 10 до 90, кратных трем.
- 7) Найти сумму и количество целых чисел, принадлежащих числовому отрезку  $[1016; 7937]$ , которые делятся на 3 и не делятся на 7 и 17.
- 8) В промежутке от 15 до 115 найти сумму квадратов чисел, кратных трем, и сумму кубов чисел, кратных пяти.
- 9) Если в промежутке  $(-a; b)$  больше отрицательных чисел, то найти сумму их модулей, а если положительных – то найти их произведение. Числа  $a, b > 0$ , вводятся с клавиатуры.
- 10) Найти среднее арифметическое целых чисел, которые делятся на 7 и не делятся на 14, из промежутка от 12 до 250.
- 11) Вывести на экран таблицу значений функции  $y = 5,4 \cdot x^3 - 2,8 \cdot x^2 - x + 1,6$  в диапазоне от  $a$  до  $b$  с шагом  $h$ . Значения  $a$ ,  $b$  и  $h$  вводятся с клавиатуры.
- 12) Составить программу вычисления значений функции  $\arctg(x/2)$  на отрезке  $[A, B]$  в точках  $X_i = A + iH$ , где  $H = (B - A)/M$ , где  $A, B, M$  — заданное целое число. Значение шага  $H$  должно вычисляться один раз.

13) Получить таблицу значений функции  $y=x^2+0,5x-1$  при изменении  $x$  в пределах от  $xh = -2,5$  до  $xk = 2,5$  с шагом  $h = 0,5$ .

14) Табуляция функции  $y = \sqrt{x+4} - \frac{1}{x^2}$  с известным шагом, концы отрезка  $a, b$  и шаг  $h$  вводятся с клавиатуры.

15) Написать программу, которая выводит таблицу значений функции  $y = -2,4x^2+5x-3$  в диапазоне от  $-2$  до  $2$ , с шагом  $0,5$ .

16) Написать программу, которая выводит таблицу значений функции  $y=|x-2|+|x+1|$ . Диапазон изменения аргумента от  $-4$  до  $4$ , шаг приращения аргумента  $0,5$ .

17) Написать программу, которая выводит таблицу значений функции  $y = \frac{x}{x + \frac{1}{\sqrt{x^2+10}}}$ . Диапазон изменения аргумента от  $-a$  до  $a$ , шаг приращения аргумента  $h$ . Значения  $a, h$  вводятся с клавиатуры.

18) Написать программу, которая выводит таблицу значений функции  $y = \sqrt{\left|x - \frac{3}{x}\right|^2 + 5}$ . Диапазон изменения аргумента от  $-a$  до  $a$ , шаг приращения аргумента  $h$ . Значения  $a, h$  вводятся с клавиатуры.

19) Составить программу для вычисления значения функции  $f(x)$  с шагом  $0,3$ . Натуральное число  $k$  вводится с клавиатуры:

$$f = \begin{cases} kx^2, & \text{если } k < x \leq 3k, \\ \sqrt{k-x}, & \text{если } 3k < x \leq 4k. \end{cases}$$

20) Составить программу для вычисления значения функции  $y(x)$  с шагом  $h$  и значением  $p$  ( $h$  и  $p$  вводятся с клавиатуры):

$$y = \begin{cases} 3x + x^2, & \text{если } x \in [-2p, 0], \\ |x - p|, & \text{если } x \in (0, 2p]. \end{cases}$$

21) Вводится последовательность ненулевых чисел,  $0$  – конец последовательности. Определить сумму и среднее арифметическое положительных четных элементов последовательности.

22) Вводится последовательность ненулевых чисел, 0 – конец последовательности. Определить наибольшее число в последовательности.

23) Вводится последовательность ненулевых чисел, 0 – конец последовательности. Подсчитать процент положительных и отрицательных чисел.

24) Вводится последовательность произвольных чисел, 0 – конец последовательности. Определить является ли эта последовательность строго убывающей (то есть, каждый следующий элемент меньше предыдущего).

25) Вводится последовательность ненулевых целых чисел, 0 – конец последовательности. Определить произведение и среднее значение четных элементов последовательности.

26) Вводится последовательность ненулевых целых чисел, 0 – конец последовательности. Получить сумму тех чисел последовательности, которые нечетны, отрицательны и по модулю кратны 5.

27) Вводится последовательность ненулевых целых чисел, 0 – конец последовательности. Найти те члены последовательности, которые при делении на 7 дают остатки 1, 2, 3.

28) Вводится последовательность ненулевых целых чисел, 0 – конец последовательности. Верно ли, что количество положительных чисел в последовательности больше, чем отрицательных?

29) Посчитать приближенное значение функции  $y = \sin(x)$  с помощью разложения в ряд:

$$x - x^3/3! + \dots + (-1)^{(N+1)} * x^{(2*N+1)}/(2*N+1)! + \dots;$$

30) Найти сумму ряда с точностью  $\varepsilon = 10^{-4}$ , общий член которого

$$a_n = \frac{1}{2^n} + \frac{1}{3^n}.$$

31) Найти сумму ряда с точностью  $\varepsilon = 10^{-4}$ , общий член которого

$$a_n = \frac{1}{((3n - 2)(3n + 1))}.$$

32) Найти сумму ряда с точностью  $\varepsilon=10^{-4}$ , общий член которого  $a_n = \frac{10^n}{n!}$

33) Найти сумму ряда с точностью  $\varepsilon=10^{-4}$ , общий член которого  $a_n = 10^{-n}(n-1)!$

34) Найти сумму ряда с точностью  $\varepsilon=10^{-4}$ , общий член которого  $a_n = \frac{n^3}{(3n-3)!}$

35) Найти сумму ряда  $s = \frac{1}{1+1^2} + \frac{2}{1+2^2} + \frac{3}{1+3^2} + \dots$ ;  $a_i = \frac{i}{1+i^2}$ ;  $s = \sum_i a_i$

с заданной точностью  $\varepsilon$ .

36) Найти сумму ряда с заданной точностью.

$$s = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \dots$$

37) Вводится последовательность из  $N$  положительных целых чисел. Вывести на экран количество делителей каждого числа.

38) Найти наибольший общий делитель двух чисел с помощью алгоритма Евклида.

39) Найти наименьшее общее кратное двух чисел.

40) Найти количество делителей заданного числа  $N$ . Есть ли среди делителей число 3?

41) Найти простые множители заданного натурального числа  $N$ .

42) Найти простые числа из промежутка от 1 до заданного  $N$ . Может ли сумма двух последовательных натуральных чисел быть простым числом? Вывести такие суммы, если есть.

43) Вывести простые числа из промежутка от 1 до 14, рассчитав их по теореме Вильсона (число  $p$  — простое, если  $(p-1)! + 1$  делится на  $p$ ).

44) Найти все целые числа  $n$ , для которых модуль значения трёхчлена  $n^2 - 7n + 10$  будет простым числом.

45) Пусть  $p > 5$  — простое число. Доказать, что  $p^2 - 1$  делится нацело на 24.

46) Среди четырехзначных чисел выбрать те, у которых все четыре цифры различны, и найти их количество.

47) Вводится целое положительное число. Определить количество четных и нечетных цифр в числе.

48) Найти четырехзначные числа, которые равны сумме кубов цифр исходного числа.

49) Определить среди трехзначных чисел числа Армстронга. Число Армстронга — натуральное число, которое равно сумме своих цифр, возведенных в степень, равную количеству его цифр. Например, число 153 — число Армстронга, потому что:  $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$ .

50) Найти трехзначные числа, которые равны сумме удвоенных цифр исходного числа.

51) Рассчитать все трехзначные числа, сумма цифр которых делится нацело на 12. Рассмотреть в трех циклах цифры, а при выводе собрать число.

52) Рассчитать все трехзначные числа, сумма цифр которых делится нацело на 15. Рассмотреть в одном цикле трехзначное число, а в условии разбить его на цифры.

53) Если приписать цифру 1 впереди некоего 5-значного числа, то получится число в 3 раза меньше, чем если приписать цифру 1 в конце этого же числа. Найти это число.

54) Дано натуральное  $k$ . Напечатать  $k$ -ю цифру последовательности, в которой выписаны подряд все числа Фибоначчи.

55) Дана арифметическая прогрессия с первым членом  $a_1=0.4$  и разностью  $-d=0.5$ . Записать программу определения суммы и количества членов прогрессии с нечетными номерами.

56) Какое наибольшее число последовательных нечётных чисел, начиная с 1, можно сложить, чтобы получившаяся сумма осталась меньше 400?

57) Вывести  $n$  членов геометрической прогрессии и найти ее сумму, если первый член  $b_1=1$ , а знаменатель  $q=3$ .

58) Последовательность задана формулой  $a_n = \frac{11}{n+1}$ . Сколько членов в этой последовательности больше 1?

59) Сколько натуральных чисел  $n$  удовлетворяет неравенству  $\frac{20}{n+2} > 3$ ?

60) Дана арифметическая прогрессия: 33; 25; 17; ... Найти первый отрицательный член этой прогрессии и ее номер.

61) Найти сумму всех отрицательных членов арифметической прогрессии: -8,6; -8,4; ...

62) Нарисовать равнобедренный треугольник из символов «\*». Высоту выбирает пользователь.

63) Составить программу для нахождения всех Пифагоровых чисел, не превышающих заданного  $n$ . Пифагоровы числа удовлетворяют соотношению  $x^2+y^2=z^2$  Результат оформить в виде таблицы из трех столбцов –  $x$ ,  $y$ ,  $z$ .

64) Вывести на экран квадраты и кубы нечётных целых чисел от 1 до заданного  $n$  в виде таблицы.



## Глава 4. Массивы

### 4.1. Одномерные массивы

Программирование становится по-настоящему интересным, когда появляется возможность работать с наборами (коллекциями) элементов. Вот лишь некоторые примеры того, где они встречаются:

- Постраничный вывод данных на сайте
- Подсчёт общей суммы в заказе на основании каждой из позиций
- Вывод списка друзей, сообщений, фильмов и тому подобное
- Обработка набора DOM-узлов (HTML, фронтенд-разработка)

Любые списки, которые окружают нас в реальном или виртуальном мире, являются коллекциями элементов с точки зрения программирования. В JavaScript для их хранения используется *массив* – структура данных, позволяющая работать с набором как с единым целым.

```
// Определение массива друзей  
const friends = ['petya', 'vasya', 'ivan'];
```

В отличие от примитивных типов данных, массивы в JavaScript могут изменяться. Причем как по содержимому, так и по размеру самого массива. Это сильно влияет на работу с ними и добавляет с одной стороны больше возможностей, а с другой – ответственности. Используя массивы, одну и ту же задачу можно решить множеством разных способов. Только некоторые из них будут хорошими, остальные же, не эффективными, сложными в отладке и анализе.

Массивом в программировании представляют любые упорядоченные наборы (или коллекции) элементов. Задача массива представить такие коллекции в виде единой структуры, которая позволяет работать с ними как с единым целым.

## Определение массива

```
// Создание пустого массива
const items = [];

// Создание массива с тремя элементами
const animals = ['cats', 'dogs', 'birds'];
```

В примере происходит определение массива ['cats', 'dogs', 'birds'], который затем присваивается константе `animals`.

## Получение данных

Элементы в массиве упорядочены слева направо. Каждый элемент имеет порядковый номер, называемый **индексом**. Индексация массива начинается с нуля. То есть первый элемент массива доступен по индексу 0, второй — по индексу 1 и так далее... Для извлечения элемента из массива по индексу используется особый синтаксис:

```
const animals = ['cats', 'dogs', 'birds'];
animals[0]; // 'cats'
animals[1]; // 'dogs'

// Последний индекс в массиве всегда меньше размера массива на единицу.
// В этом массиве три элемента, но последний индекс равен двум
animals[2]; // 'birds'
```

Узнать размер массива можно, обратившись к его свойству `length`.

```
const animals = ['cats', 'dogs', 'birds'];
animals.length; // 3
```

В реальных задачах индекс часто вычисляется динамически, поэтому обращение к конкретному элементу происходит с использованием переменных:

```
let i = 1;
const animals = ['cats', 'dogs', 'birds'];
animals[i]; // 'dogs'
```

И даже так:

```
let i = 1;
let j = 1;
const animals = ['cats', 'dogs', 'birds'];
animals[i + j]; // 'birds'
```

Такой вызов возможен по одной простой причине — внутри скобок ожидается *выражение*. А там, где ожидается выражение, можно подставлять всё, что вычисляется. В том числе вызовы функций:

```
const getIndexOfSecondElement = () => 1;
const animals = ['cats', 'dogs', 'birds'];
animals[getIndexOfSecondElement()]; // 'dogs'
```

Довольно часто, в задачах с использованием массивов, нужно взять последний элемент. Для этого вычисляется последний индекс массива по формуле *размер\_массива - 1*, по которому и можно обратиться к последнему элементу:

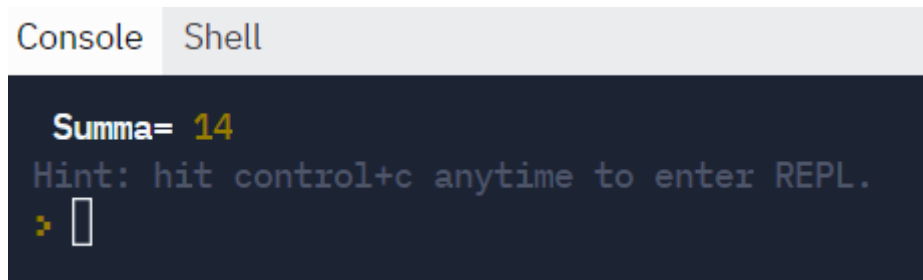
```
const animals = ['cats', 'dogs', 'birds'];
animals[animals.length - 1]; // 'birds'
```

Пример использования массива - посчитать сумму элементов массива:

```
let a=[2,5,7];
let s=0;
```

```
s+=a[0]+a[1]+a[2];  
console.log(" Summa=",s);
```

Результат:



```
Console Shell  
Summa= 14  
Hint: hit control+c anytime to enter REPL.  
> █
```

## Модификация

Примитивные типы данных, с которыми мы работали до сих пор, невозможно изменить. Любые функции и методы над ними возвращают новые значения, но не могут ничего сделать со старым.

```
const name = 'elsu';  
name.toUpperCase(); // 'ELSU'  
// Значение name не поменялось  
console.log(name); // 'elsu'
```

С массивами это правило не работает. Массивы могут меняться: увеличиваться, уменьшаться, изменять значения по индексам. Ниже мы разберем все эти операции.

## Изменение элементов массива

Синтаксис изменения элемента массива, практически такой же, как и при обращении к элементу массива. Разница лишь в наличии присваивания:

```
const animals = ['cats', 'dogs', 'birds'];  
// Меняется первый элемент массива  
animals[0] = 'horses';  
console.log(animals); // => ['horses', 'dogs', 'birds']
```

Самое неожиданное в данном коде – изменение константы. Константы в JavaScript не совсем то, как мы себе это представляли раньше. Константы хранят ссылку на данные, а не сами данные. Это значит, что менять данные можно, но нельзя заменить ссылку. Технически это значит, что мы не можем заменить все значение константы целиком:

```
const animals = ['cats', 'dogs', 'birds'];  
  
// Меняем данные, а сам массив остался тем же  
  
// Такой код работает  
animals[2] = 'fish';  
  
// Произойдет ошибка, так как здесь идет замена константы  
animals = ['fish', 'cats'];  
  
// Uncaught TypeError: Assignment to constant variable.
```

### Методы и свойства класса Array

Для создания и обработки массивов можно использовать класс Array, в котором находятся методы работы с ними.

### Количество элементов

Свойство `length` содержит количество элементов массива. Нумерация элементов производится с нуля, поэтому свойство `length` удобно использовать внутри цикла для обхода элементов с нулевого до последнего.

Рассмотрим простой пример:

```
let M=new Array(1,2,3)  
  
console.log(M.length)  
  
//=>3
```

## Добавление элемента в массив

Метод `push()` добавляет элемент в *конец* массива:

```
const animals = ['cats', 'dogs', 'birds'];
animals.push('horses');
// массив animals изменён — стал больше
console.log(animals); // => ['cats', 'dogs', 'birds', 'horses']
// строка 'horses' была добавлена в конец массива (индекс = 3)
console.log(animals[3]); // => 'horses'
```

Метод `array.unshift()` добавляет элемент в *начало* массива:

```
const animals = ['cats', 'dogs', 'birds'];
animals.unshift('horses');
// массив animals изменён — стал больше
console.log(animals); // => ['horses', 'cats', 'dogs', 'birds']
// строка 'horses' была добавлена в начало массива (индекс = 0)
console.log(animals[0]); // => 'horses'
```

Иногда индекс добавления известен сразу и в таком случае добавление работает так же, как и изменение:

```
const animals = ['cats', 'dogs', 'birds'];
animals[3] = 'horses';
console.log(animals); // => ['cats', 'dogs', 'birds', 'horses']
```

## Удаление элемента из массива

Удалить элемент из массива можно с помощью специальной конструкции `delete: delete arr[index]`.

Пример:

```
const animals = ['cats', 'dogs', 'birds'];  
delete animals[1]; // удаляем элемент под индексом 1  
console.log(animals); // => ['cats', <1 empty item>, 'birds']
```

Этот способ обладает рядом недостатков, завязанных на особенности внутренней организации языка JavaScript. Например, после такого удаления, можно с удивлением заметить, что размер массива не изменился:

```
animals.length; // 3
```

### Сортировка массива

Метод `sort()` способен выполнить сортировку массива по возрастанию элементов:

```
let M=new Array(3,2,1)  
console.log(M.sort())  
// => [ 1, 2, 3 ]
```

Для более сложных случаев – сортировка по убыванию, массив большого размера, массив с повторяющимися элементами – используются различные алгоритмы сортировки.

### Проверка существования значения

При работе с массивами, часто допускается ситуация, называемая "выход за границу массива". Она возникает при обращении к несуществующему индексу:

```
const animals = ['cats', 'dogs', 'birds'];  
  
// Элемента с индексом 5 не существует
```

```
animals[5]; // undefined
```

В разных языках программирования поведение в случае выхода за границу реализовано совершенно по-разному. Иногда возникает ошибка, иногда нет, а иногда подобный выход возвращает случайные данные из соседнего блока памяти, как в Си, что может привести к катастрофе.

В JavaScript свой путь. Здесь дана большая свобода, допускающая почти любые вольности. Обращение по несуществующему индексу возвращает значение `undefined`. При этом никаких ошибок не возникает, это рассматривается как нормальная ситуация:

```
const animals = ['cats', 'dogs', 'birds'];
```

```
// Выход за границы массива
```

```
animals[5]; // undefined
```

```
animals[4]; // undefined
```

```
animals[3]; // undefined
```

```
// мы попали в границы массива :)
```

```
animals[2]; // 'birds'
```

В подавляющем большинстве ситуаций, выход за границу массива является нежелательным поведением. Он происходит из-за логических ошибок в программе. Программа, при этом, продолжает работать и, даже, иногда выдавать правильный результат.

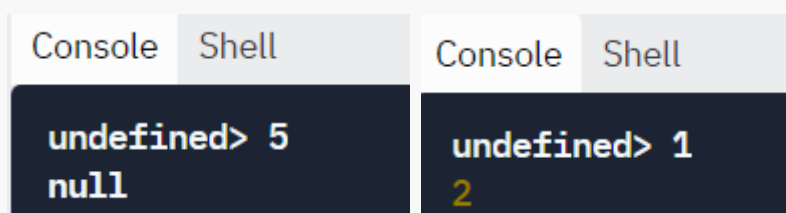
### Пример

Извлечь из массива элемент по указанному индексу, если индекс существует, либо вывести значение по умолчанию. Значение по умолчанию (равно `null`)



```
let arr=[1,2,3]
let def = null
let index=parseInt(prompt())
if (arr[index] === undefined) {
  console.log(def);
}
else console.log(arr[index])
```

Результат:



### Ссылки

Переменные (и константы) в JavaScript могут хранить два вида данных: примитивные и ссылочные. К примитивным относятся все примитивные типы: числа, строки, булевский и так далее. К ссылочным – объекты.

```
typeof []; // 'object'
```

В чем разница между ссылочными и примитивными типами данных и почему об этом нужно знать?

С точки зрения прикладного программиста, разница проявляется при изменении данных, их передаче и возврате из функций. Мы уже знаем, что массив можно менять даже если он записан в константу. Здесь как раз и проявляется ссылочная природа. Константа хранит ссылку на массив, а не сам массив и эта ссылка не меняется. А вот массив поменяться может. С примитивными типами такой трюк не пройдет.

Другой способ убедиться в том, что массивы ссылки – создать несколько констант, содержащих один массив и посмотреть, как они меняются:

```
const items = [1, 2];  
  
// Ссылаются на один и тот же массив  
  
const items2 = items;  
  
items2.push(3);  
  
console.log(items2); // => [1, 2, 3]  
  
console.log(items); // => [1, 2, 3]  
  
items2 === items; // true
```

Сравнение массивов тоже происходит по ссылке. Это может быть очень неожиданно с непривычки. Одинаковые массивы по структуре имеют разные ссылки и не равны друг другу:

```
[1,2,3] === [1,2,3]; // false
```

// Проверить два массива на равенство их значений можно с помощью библиотеки lodash

## 4.2 Циклы при работе с массивами

Работа с массивами, почти всегда, завязана на одновременную обработку всех его элементов. Это нужно при выводе списков на экран, при выполнении различных расчетов или проверке данных. Во всех этих случаях нужен механизм для перебора элементов массива. Самый простой способ сделать это – использовать цикл.

### Обход с помощью for

Циклы напрямую с массивами не связаны, но у циклов есть счетчик, который может выступать в качестве индекса массива. Поэтому соединить их не составляет никакого труда:

```
// Создаем массив
```

```

const userNames = ['petya', 'vasya', 'evgeny'];

// Определяем цикл

// Начальное значение счетчика i = 0 – вычисляется один раз перед нача-
лом выполнения

// Условие остановки i < userNames.length – выполняется перед каждой
итерацией

// Изменение счетчика i += 1 – выполняется после каждой итерации
for (let i = 0; i < userNames.length; i += 1) {

  // Этот код выполняется для каждого элемента

  console.log(userNames[i]);

}

// => 'petya'

// => 'vasya'

// => 'evgeny'

```

В данном коде создаём массив из трёх элементов — имён. Далее в цикле обходим массив и выводим на экран все имена так, что каждое имя оказывается на новой строке (`console.log` автоматически делает перевод строки).

Рассмотрим этот этап подробнее. При обходе массива циклом `for` счётчик, как правило, играет роль индекса в массиве. Он инициализируется нулём и увеличивается до `userNames.length - 1`, что соответствует индексу последнего элемента. Именно поэтому мы используем строгое сравнение `<` (*меньше*) в условном выражении `i < userNames.length`, а не `<=` (*меньше либо равно*).

А что, если нам нужно вывести значения в обратном порядке? Для этого есть два способа. Один — идти в прямом порядке, то есть от нулевого индекса

до последнего, и каждый раз вычислять нужный индекс по такой формуле размер массива - 1 - текущее значение счётчика.

```
const userNames = ['petya', 'vasya', 'evgeny'];

for (let i = 0; i < userNames.length; i += 1) {
  const index = (userNames.length - 1) - i;
  console.log(userNames[index]);
}
```

Другой способ подразумевает обход в обратном порядке, от верхней границы до нижней, то есть от последнего индекса массива к первому (нулю, так как индексирование начинается с нуля). В такой ситуации код меняется на следующий:

```
const userNames = ['petya', 'vasya', 'evgeny'];
// Начальное значение i соответствует последнему индексу в массиве
for (let i = userNames.length - 1; i >= 0; i -= 1) {
  console.log(userNames[i]);
}
```

При таком обходе проверка остановки должна быть именно на  $\geq$ , иначе элемент с индексом 0 не попадет в цикл.

### Изменение

Во время обхода массива его можно не только читать, но и модифицировать. Предположим, что перед нами стоит задача нормализации списка электронных адресов — например, приведение их к нижнему регистру. Тогда код будет выглядеть так:

```
const emails = ['VASYA@gmAil.com', 'iGoR@yandex.RU',
'netiD@hot.CoM'];
```

```
console.log(emails);

// => [ 'VASYA@gmAil.com', 'iGoR@yandex.RU', 'netiD@hot.CoM' ]

for (let i = 0; i < emails.length; i += 1) {
  const email = emails[i];

  // toLowerCase() — стандартный метод js,
  // преобразующий строку в нижний регистр
  const normalizedEmail = email.toLowerCase();

  // Заменяем значение
  emails[i] = normalizedEmail;
}

console.log(emails);

// => [ 'vasya@gmail.com', 'igor@yandex.ru', 'netid@hot.com' ]
```

Ключевая строчка: `emails[i] = normalizedEmail;`. В ней происходит перезапись элемента под индексом `i`.

Итак, цикл `for` можно комбинировать с массивами в любых вариантах. Массив не обязательно перебирать полностью и от начала до конца. Можно, например, смотреть только каждый второй элемент или двигаться до половины. Все это зависит от конкретной задачи.

Точно так же массивы сочетаются с `while`. Единственное что нужно массивам — индекс.

Распространённый вариант использования циклов с массивами — **агрегация**. Агрегацией называются любые вычисления, которые, как правило, строятся на основе всего набора данных, например, поиск максимального, среднего, суммы и так далее. Процесс агрегации не требует знания нового синтаксиса, но влияет на алгоритм решения задач. Поэтому имеет смысл рассмотреть его отдельно. Начнем с поиска максимального.

```
const coll=[3, 2, -10, 38, 0]

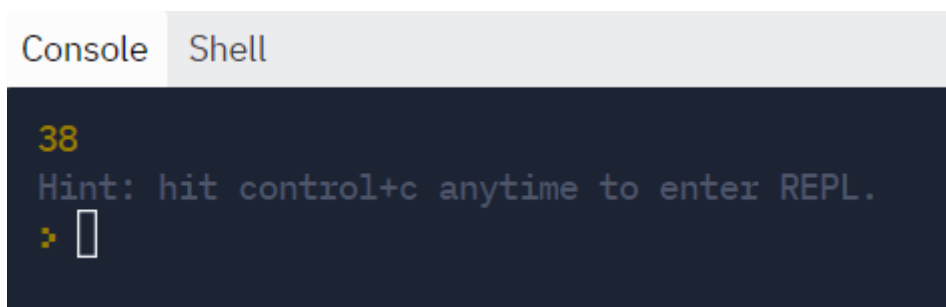
// Сравнение элементов нужно начать с какого-то первого элемента
let max = coll[0]; // Принимаем за максимальное первый элемент

// Обход начинаем со второго элемента
for (let i = 1; i < coll.length; i += 1) {
  const currentElement = coll[i];

  // Если текущий элемент больше максимального,
  // то он становится максимальным
  if (currentElement > max) {
    max = currentElement;
  }
}

// Не забываем вывести максимальное число
console.log(max); // => 38
```

Результат:



```
Console Shell
38
Hint: hit control+c anytime to enter REPL.
➤
```

Почему это пример агрегации? Здесь мы видим *вычисление*, которое включает в себя сравнение всех элементов для поиска одного, которое станет результатом этой операции.

Обратите внимание, что начальным значением `max` взят первый элемент, а не, скажем, число 0. Ведь может оказаться так, что все числа в массиве меньше 0, и тогда мы получим неверный ответ.

Изменим задачу нахождения суммы элементов массива с помощью цикла

```
let a=[];

let s=0;

var n=parseInt(prompt('Количество'));

for( i=0;i<n;i++)

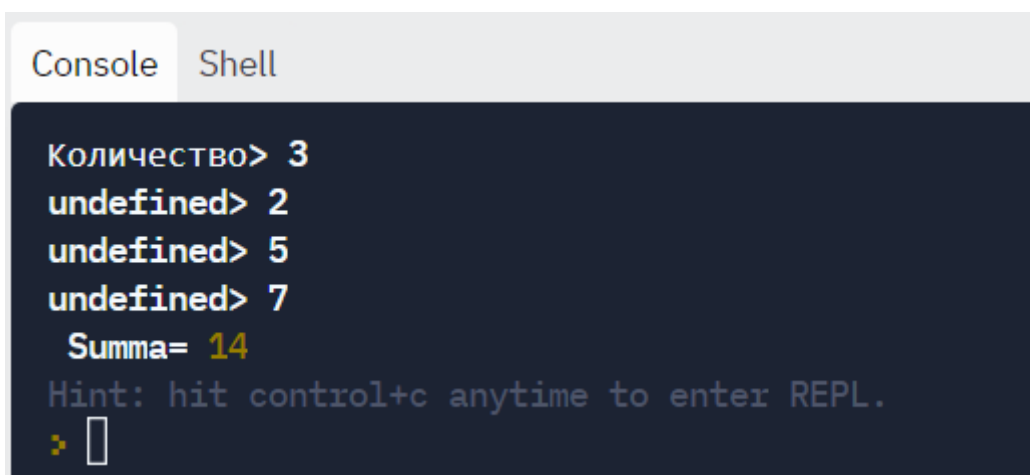
  a[i]=parseInt(prompt());

for( i=0;i<n;i++)

  s+=a[i];

console.log(" Summa=",s);
```

Результат:



```
Console Shell
Количество> 3
undefined> 2
undefined> 5
undefined> 7
Summa= 14
Hint: hit control+c anytime to enter REPL.
> █
```

Сумма элементов всегда возвращает какое-то число. Если массив пустой, то сумма его элементов 0

Алгоритм поиска суммы обладает парой важных нюансов.

Чему равна сумма элементов пустого массива? С точки зрения математики такая сумма равна 0. Что в принципе совпадает со здравым смыслом. Если у нас нет яблок, значит у нас есть 0 яблок (количество яблок равно нулю). Функции в программировании работают по этой логике.

Второй момент связан с начальным элементом суммы. У переменной `s` есть начальное значение равное 0. Зачем вообще задавать значение? Любая повторяющаяся операция начинается с какого-то значения. Нельзя просто так объявить переменную и начать с ней работать внутри цикла. Это приведет к неверному результату:

```
// начальное значение не задано
// js автоматически делает его равным undefined
let sum;

// первая итерация цикла
sum = sum + 2; // ?
```

В результате такого вызова, внутри `sum` окажется NaN, то есть не-число. Оно возникает из-за попытки сложить 2 и `undefined`. Значит, какое-то значение все же нужно. Почему в коде выше выбран 0? Очень легко проверить, что все остальные варианты приведут к неверному результату. Если начальное значение будет равно 1, то результат получится на 1 больше, чем нужно.

В математике существует понятие **нейтральный элемент операции** (у каждой операции свой элемент). Это понятие имеет очень простой смысл. Операция с этим элементом не изменяет то значение, над которым проводится



операция. В сложении любое число плюс ноль дает само число. При вычитании тоже самое.

Агрегация далеко не всегда означает, что коллекция элементов сводится к некоторому простому значению. Результатом агрегации может быть сколь угодно сложная структура, например, массив. Подобные примеры часто встречаются в реальной жизни.

### Цикл for...of

FOR относится к низкоуровневым циклам. Он требует задания счетчика, правил его изменения и условия остановки. Было бы значительно удобнее обходить элементы коллекции напрямую, без счетчика. Многие языки программирования решают это введением специального вида цикла. В JavaScript тоже есть такой: for...of.

```
const userNames = ['petya', 'vasya', 'evgeny'];
```

```
// name на каждой итерации свой собственный (локальный), поэтому используется const
```

```
for (const name of userNames) {  
  console.log(name);  
}
```

```
// => "petya"
```

```
// => "vasya"
```

```
// => "evgeny"
```

Как видно из примера, код использующий for...of получается значительно чище, чем с использованием цикла for. Цикл for...of знает о том, как перебирать элементы и знает о том, когда они закончатся.

Этот цикл отлично подходит для задач агрегации. Вычислим сумму вторым способом, используя `for...of` для перебора элементов:

```
let a=[];

let s=0;

var n=parseInt(prompt('Количество'));

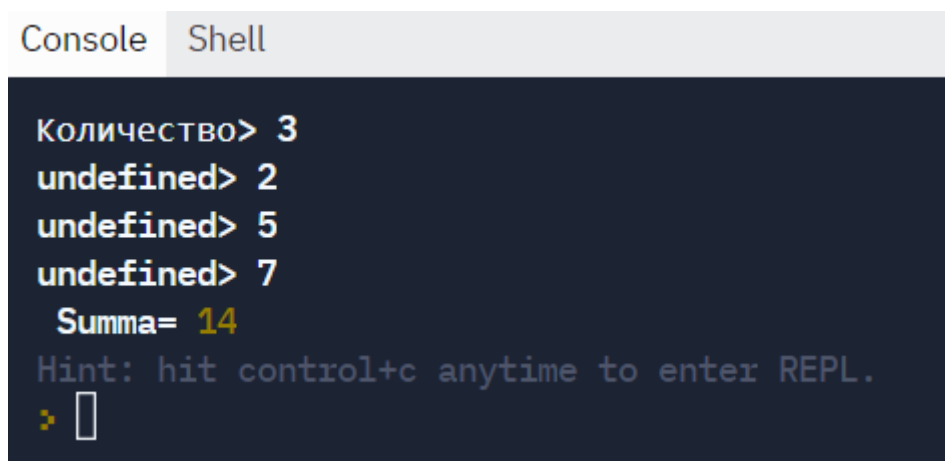
for( i=0;i<n;i++)

    a.push(parseInt(prompt()))

for( i of a)

    s+=i;

console.log(" Summa=",s);
```



```
Console Shell
Количество> 3
undefined> 2
undefined> 5
undefined> 7
Summa= 14
Hint: hit control+c anytime to enter REPL.
> █
```

### Применимость

В большинстве задач, использующих цикл, предпочтительнее `for...of`. Иногда его бывает недостаточно, и требуется ручное управление обходом. В таких случаях можно возвращаться к использованию `for`. Например, когда нужно идти не по каждому элементу массива, а через один:

```
for (let i = 0; i < items.length; i += 2) {

    // какой-то код

}
```

Иногда нужно обходить массив в обратном порядке. `for...of` здесь бессилён, и снова нужен `for`:

```
for (let i = items.length - 1; i >= 0; i -= 1) {  
  // какой-то код  
}
```

Другие задачи вообще с массивами напрямую не связаны. К последним относятся ситуации, когда нужно перебирать числа в определённом диапазоне. В этом случае нет массива, по которому можно было бы пройти с помощью `for...of`.

```
for (let i = 5; i < 10; i += 1) {  
  // какой-то код  
}
```

Ну и наконец, встречаются задачи, в которых нужно во время обхода менять исходный массив:

```
for (let i = 0; i < items.length; i += 1) {  
  items[i] = /* что-то делаем */  
}
```

### 4.3. Основные функции работы с массивами

#### Проектирование функций

Проектируя функции, работающие с массивами, есть два пути: менять исходный массив или формировать внутри новый и возвращать его наружу. Какой лучше? В подавляющем большинстве стоит предпочитать второй. Это безопасно. Функции, возвращающие новые значения, удобнее в работе, а поведение программы становится в целом более предсказуемым, так как отсутствуют неконтролируемые изменения данных.

Изменение массива может повлечь за собой неожиданные эффекты.

В каких же случаях стоит менять сам массив? Есть ровно одна причина по которой так делают – производительность. Именно поэтому некоторые встроенные методы массивов меняют их, например `reverse()` – переписывание элементов массива в обратном порядке или `sort()` – сортировка массива по возрастанию:

```
const items = [3, 8, 1];  
// Нет присвоения результата, массив изменяется напрямую  
items.sort();  
console.log(items); // => [1, 3, 8]  
items.reverse();  
console.log(items); // => [8, 3, 1]
```

Обычно в документации каждой функции отдельно подчёркивают, изменяет ли она исходный массив или возвращает результатом новый массив, не модифицируя исходный. Например, метод `concat()`, в отличие от `sort()`, возвращает новый массив, о чём написано в документации.

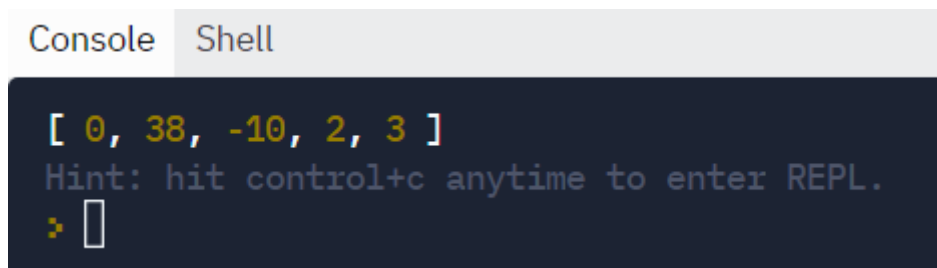
Несмотря на то, что подход, меняющий массивы напрямую, сложнее в отладке, его используют в некоторых языках для увеличения эффективности работы. Если массив достаточно большой, то полное копирование окажется дорогой операцией. В реальной жизни (веб-разработчика) это почти никогда не является проблемой, но знать об этом полезно.

Пример. Реализуем вариант скрипта, который принимает на вход массив и располагает элементы внутри него в обратном порядке. Для решения этой задачи, проще всего брать и менять местами элементы, стоящие на зеркальных местах: первый и последний, второй и предпоследний и так далее до середины.

```
const arr=[3, 2, -10, 38, 0]  
  
let l = Math.floor(arr.length/2);  
for (let i = 0; i < l; i+=1) {
```

```
let buffer = arr[i];  
arr[i] = arr[(arr.length - 1) - i];  
arr[(arr.length - 1) - i] = buffer;  
}  
console.log(arr);
```

Результат:



```
Console Shell  
[ 0, 38, -10, 2, 3 ]  
Hint: hit control+c anytime to enter REPL.  
> []
```

### Удаление элементов массива

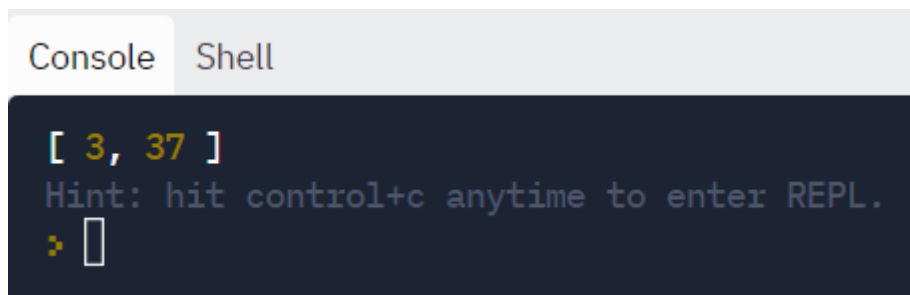
В JavaScript не существует настоящего способа удалить элемент из массива. Инструкция `delete` лишь очищает значение, но сама ячейка никуда не девается:

```
const numbers = [1, 10];  
delete numbers[0];  
console.log(numbers);  
// => [ <1 empty item>, 10 ]
```

При этом задача удаления возникает регулярно. Причем, обычно, удаляется не один элемент, а набор элементов по определенным правилам. Например, удалить четные элементы массива. При этом изменение массива должно трансформироваться в создание нового массива, в котором отсутствуют удаляемые элементы:

```
const coll=[3, 2, -10, 37, 0]  
const result=[]
```

```
for (const item of coll) {  
  if ((item %2) !==0) {  
    result.push(item);  
  }  
}  
  
console.log(result);
```



```
Console Shell  
[ 3, 37 ]  
Hint: hit control+c anytime to enter REPL.  
> █
```

Главное, на что нужно обратить внимание, — не происходит модификаций исходного массива `coll`. Вместо этого создаётся новый массив `result`, который наполняется только подходящими под условие значениями. Именно так нужно воспринимать фразу "удалить из массива что-то". Код, использующий новый массив, меньше подвержен ошибкам, проще в отладке и оставляет больше возможностей для анализа. Вы всегда можете посмотреть исходный массив, если что-то пошло не так. Вы всегда можете наблюдать за процессом наполнения результирующего массива, что позволит чётко отследить правильность поставленных условий.

По сути, код выше — пример агрегации. Только в отличие от предыдущих примеров, в которых результатом был примитивный тип, здесь результат — массив. Результат может быть и более сложной структурой. Сама операция прореживания (удаления элементов по определенным условиям) массива обычно называется **фильтрацией**.

#### 4.4. Двумерные массивы

Двумерный массив – это массив (совокупность элементов), элементами которого являются одномерные массивы. Он имеет два индекса и, соответственно, размерность, равную двум.

Первый индекс отвечает за элементы строки, а второй – за элементы столбца. На их пересечении находится искомый элемент.

Двумерные массивы так же называются прямоугольными таблицами, а в математике – матрицами. Если количество строк и столбцов одинаково. То матрица является квадратной.

Алгоритм создания двумерного массива следующий:

1) Создается одномерный массив по правилам для одномерных массивов:

```
let A=[]
```

Или

```
let B= new Array(m)
```

2) Открывается внешний цикл для создания второго (внутреннего) массива, например:

```
for (i=0;i<B.Length;i++)
```

```
{
```

```
B[i]=new Array(n)
```

```
...
```

3) Открывается внутренний цикл для заполнения внутреннего массива:

```
for (j=0;j<B[i].length;j++)
```

```
{
```

```
B[i][j]=prompt()
```

```
}
```

```
}
```

Для отображения элементов массива двойной цикл, как в других языках программирования, можно не использовать. Достаточно команды `console.log(B)`.

Переменные `m`, `n` для обозначения количества строк и столбцов, определяются заранее. При дальнейшем обращении к элементу двумерного массива указывается два индекса – каждый в отдельных квадратных скобках.

Пример:

Вычислить определитель матрицы 3x3.

```
let m=3;
```

```
let n=3;
```

```
let d=0;
```

```
var a = [];
```

```
for (var i=0;i<m;i++)
```

```
{
```

```
  a.push([]);
```

```
  a[i].push( new Array(n));
```

```
  for(var j=0; j < n; j++)
```

```
  {
```

```
    // Initializes:
```

```
    a[i][j] = parseInt(prompt());
```

```
  }
```

```
}
```

```
d=a[0][0]*a[1][1]*a[2][2]+a[0][1]*a[1][2]*a[2][0]+a[0][2]*a[1][0]*a[2][1]-  
a[2][0]*a[1][1]*a[0][2]-a[2][1]*a[1][2]*a[0][0]-a[2][2]*a[1][0]*a[0][1]
```

```
console.log(d);
```

Результат:



```
Console Shell
undefined> 1
undefined> 2
undefined> 3
undefined> 2
undefined> 3
undefined> 1
undefined> 3
undefined> 1
undefined> 2
-18
Hint: hit control+c anytime to enter REPL.
> |
```

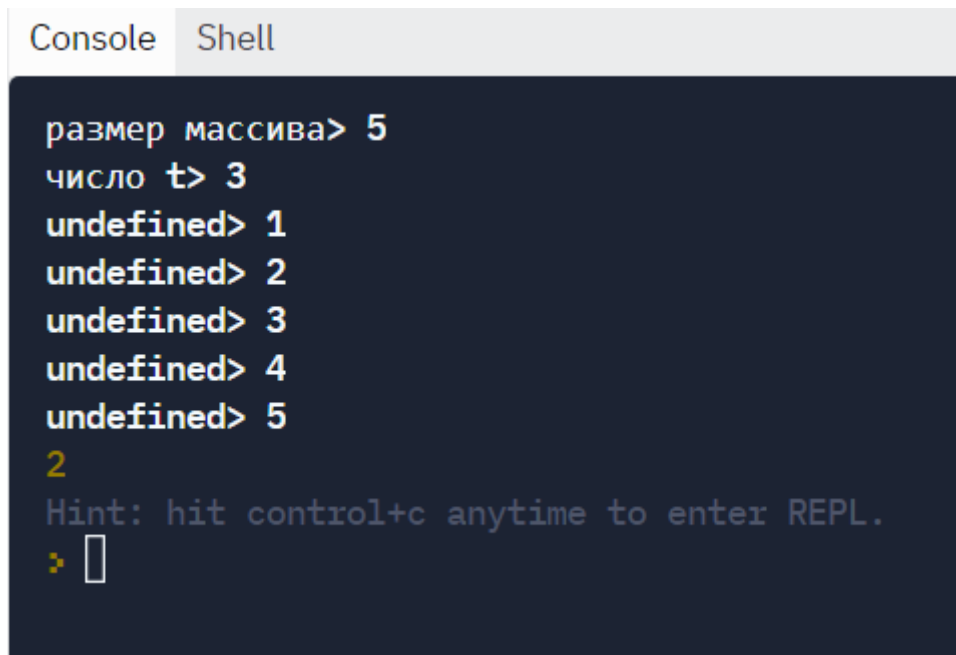
#### 4.5. Скрипты, содержащие массивы

Задача №1. Определить количество элементов одномерного массива, значения элементов которых больше заданного действительного числа  $t$ .

```
let a=[];
let p=0;
var n=parseInt(prompt('размер массива'));
var t=parseInt(prompt('число t'));
for( i=0;i<n;i++)
    a[i]=parseInt(prompt());
for( i=0;i<n;i++)
    if(a[i]>t)
        {
            p++;
        }
```

```
console.log(p);
```

Результат:



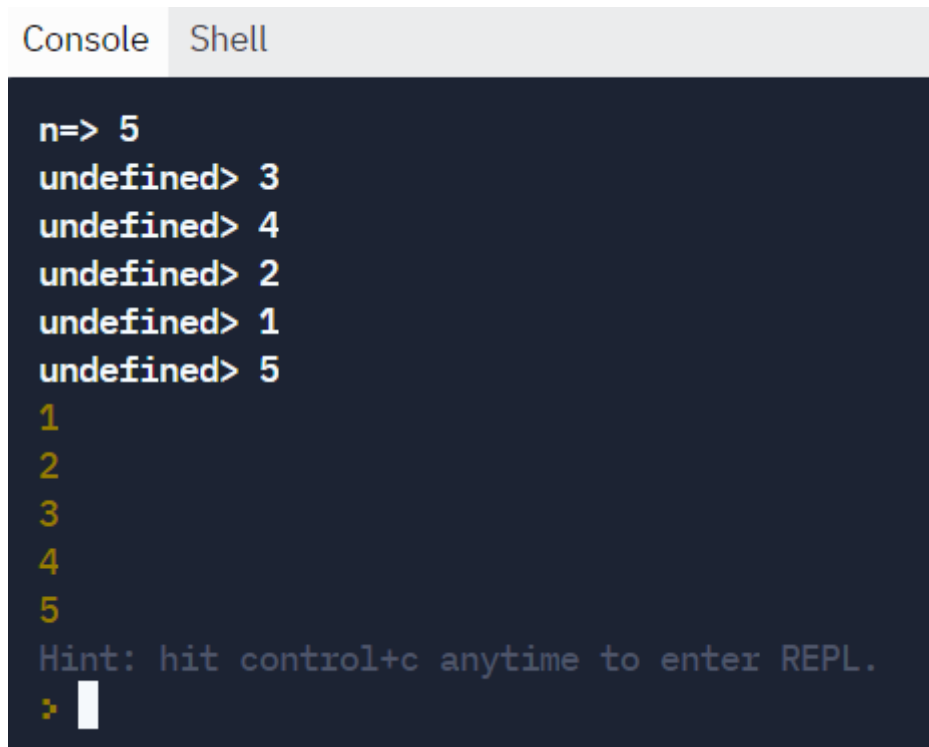
```
Console Shell
размер массива> 5
число t> 3
undefined> 1
undefined> 2
undefined> 3
undefined> 4
undefined> 5
2
Hint: hit control+c anytime to enter REPL.
> █
```

Задача №2. Отсортировать массив по возрастанию методом прямого выбора

```
let a=[];
var n=parseInt(prompt('n='));
for( i=0;i<n;i++)
  a[i]=parseInt(prompt());
let j = n-1;
while (j > 0) {
  let id = 0;
  for (i=1; i<=j; i++)
    if (a[i] > a[id]) id = i;
  let b = a[id];
  a[id] = a[j];
```

```
        a[j] = b;
        j -= 1;
    }
for( i=0;i<n;i++)
    console.log(a[i]);
```

Результат:



```
Console Shell
n=> 5
undefined> 3
undefined> 4
undefined> 2
undefined> 1
undefined> 5
1
2
3
4
5
Hint: hit control+c anytime to enter REPL.
> |
```

Пояснения к коду:

1) Алгоритм сортировки имеет вид:

а) находится наибольший элемент;

б) найденный максимум меняется местами с первым элементом;

в) в оставшейся части массива снова повторяются предыдущие пункты до тех пор, пока весь массив не будет отсортирован.

2) В алгоритме можно выбирать не только максимальный, но и наименьший элемент. Поменять местами можно не только с первым, но и с последним элементом. Процесс продолжается во втором (вложенном) цикле, уменьшая каждый раз количество неотсортированных элементов.

3) Для вывода массива цикл использовать необязательно.

Исходя из вышеперечисленного, приведем второй вариант скрипта:

```
let num=[];

var n=parseInt(prompt('n='));

for( i=0;i<n;i++)

    num[i]=parseInt(prompt());

    let min, temp; // для поиска минимального элемента и для обмена
for (i = 0; i < num.length; i++)
{
    min = i; // запоминаем индекс текущего элемента
    // ищем минимальный элемент чтобы поместить на место i-ого
    for (j = i + 1; j < num.length; j++) // для остальных элементов после i-ого
    {
        if (num[j] < num[min]) // если элемент меньше минимального,
            min = j; // запоминаем его индекс в min
    }
    temp = num[i]; // меняем местами i-ый и минимальный элементы
    num[i] = num[min];
    num[min] = temp;
}

console.log(num);
```

Результат:

```
n=> 5
undefined> 3
undefined> 4
undefined> 2
undefined> 1
undefined> 5
[ 1, 2, 3, 4, 5 ]
Hint: hit control+c anytime to enter REPL.
> []
```

Задача №3. Дана матрица  $A(N,M)$ . Найти количество элементов этой матрицы, больших среднего арифметического всех её элементов.

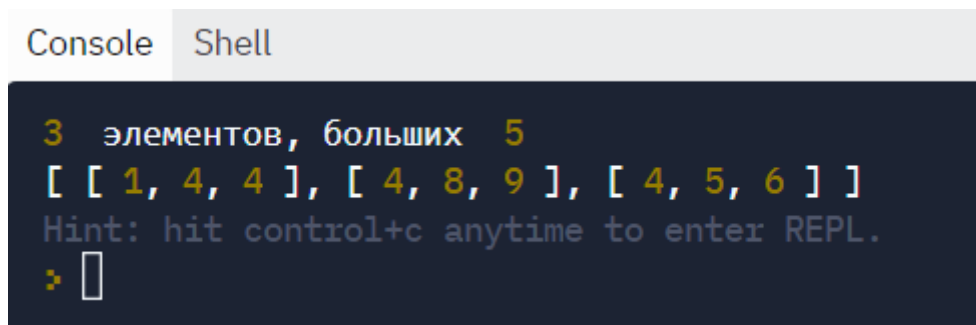
```
let m=3;
let n=3;
let c=0;
let sum=0;
let avg=0;
var a = [];
for (var i=0;i<m;i++)
{
  a.push([]);
  a[i].push( new Array(n));
  for(var j=0; j < n; j++)
  {
    // заполнение массива:
    a[i][j] = Math.round(Math.random(10)*10);
    sum =sum+ a[i][j];
  }
}
```

```

    }
  }
  avg = sum / (n * m);
  for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      if (a[i][j] > avg)
        {
          c++;
        }
  console.log(c, " элементов, больших ", avg);
  console.log(a)

```

Результат:



```

Console Shell
3 элементов, больших 5
[ [ 1, 4, 4 ], [ 4, 8, 9 ], [ 4, 5, 6 ] ]
Hint: hit control+c anytime to enter REPL.
> █

```

Пояснение к коду:

Массив заполняется псевдослучайными числами из промежутка целых чисел от 0 до 10 с помощью метода `random` библиотеки `Math`.

Задача №4. Дан целочисленный квадратный массив 3 x 3. Найти сумму максимальных элементов из каждой строки.

```

let m=3;

let n=3;

let sum=0;

```

```

var a = [];

for (var i=0;i<m;i++)
{
  a.push([]);
  a[i].push( new Array(n));
  for(var j=0; j < n; j++)
  {
    // заполнение массива
    a[i][j] = Math.round(Math.random(10)*10);
  }
}

for (i=0;i<m;i++)
{
  max = a[i][1];
  for (j=1;j<n;j++)
    if (a[i][j] > max) max = a[i][j];
  sum = sum + max;
}

console.log(sum);
console.log(a);

```

Результат:

```

Console Shell
22
[[ 9, 5, 7 ], [ 2, 1, 8 ], [ 3, 6, 7 ]]
Hint: hit control+c anytime to enter REPL.
➤ 

```

Задача №5. Написать скрипт, который принимает на вход массив и число, которое задает размер фрагмента подмассива. Вывести массив, состоящий из подмассивов указанной размерности.

```
let arr=[1,2,3,4]

let chunkSize=parseInt(prompt())

const chunkCount = ( arr.length % chunkSize === 0) ? arr.length / chunkSize :
Math.floor(arr.length / chunkSize) + 1;

const chunkedArray = [];

for (let i = 0; i < chunkCount; i += 1) {

  chunkedArray.push([]);

  for (let j = 0 + chunkSize*i; j < chunkSize + chunkSize*i; j += 1){

    if (arr[j] !== undefined){

      chunkedArray[i].push(arr[j]);

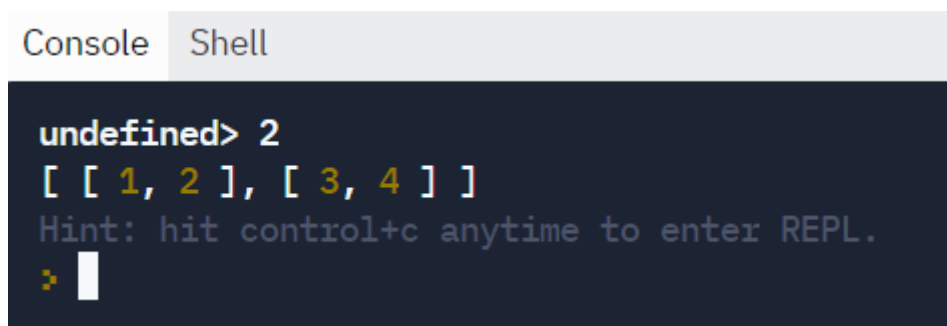
    }

  }

}

console.log(chunkedArray);
```

Результат:



```
Console Shell
undefined> 2
[[ 1, 2 ], [ 3, 4 ] ]
Hint: hit control+c anytime to enter REPL.
> |
```

Задача №6. Дан двумерный массив (матрица) размером 4x4. Вывести массив, изменённый таким образом, что правая половина матрицы становится



зеркальной копией левой половины, симметричной относительно вертикальной оси матрицы.

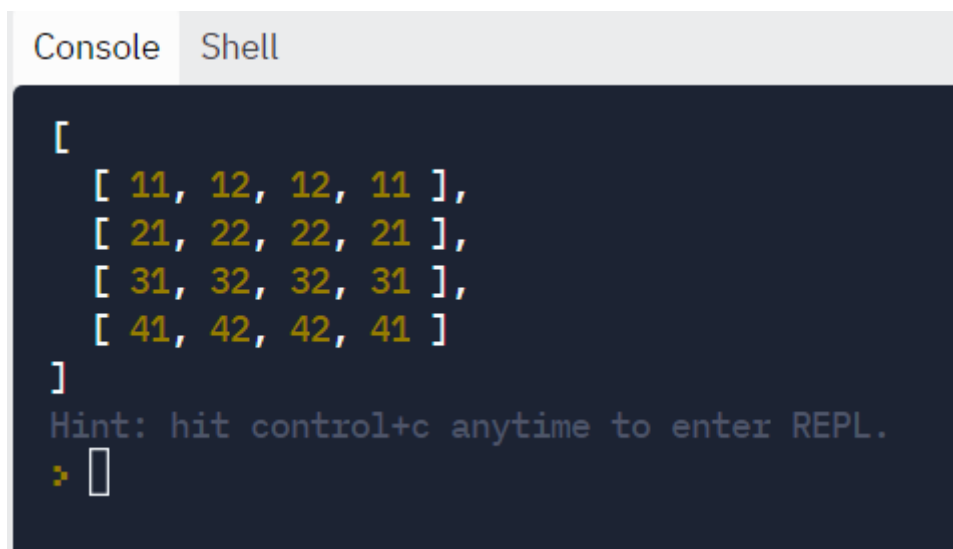
```
let matrix=[
  [11, 12, 13, 14],
  [21, 22, 23, 24],
  [31, 32, 33, 34],
  [41, 42, 43, 44]
]

const matrixlength = matrix.length;

for ( const row of matrix) {
  for (let i=0; i<matrixlength/2;i++){
    row[matrixlength-1-i] = row[i];
  }
}

console.log(matrix);
```

Результат:



```
Console Shell
[
  [ 11, 12, 12, 11 ],
  [ 21, 22, 22, 21 ],
  [ 31, 32, 32, 31 ],
  [ 41, 42, 42, 41 ]
]
Hint: hit control+c anytime to enter REPL.
> █
```

## 4.6. Контрольные вопросы и тесты

### Дополнительные вопросы

1. Какое максимальное количество размерностей может иметь массив?
2. Является ли массив в JavaScript динамическим?
3. Каким математическим понятиям соответствуют одномерный и двумерный массивы?
4. Можно ли удалить элемент массива с удалением количества его индексов?
5. Можно ли среди элементов одного массива использовать числа и символы?
6. Каков результат работы скрипта при нахождении суммы элементов пустого массива?
7. Сколько методов сортировки массивов существует?
8. Сколько ячеек памяти необходимо для хранения элементов двумерного массива размером  $m \times n$ ?
9. Можно ли нумеровать элементы массива с единицы?
10. Как называется операция удаления элементов массива по определенным условиям?

### Тесты с выбором варианта ответа

1. Определить статический массив из 20 целых чисел:
  - а) `let a=[];`
  - б) `int x[20];`
  - в) `dim x(20);`
  - г) `int[] x=20;`
2. Присвоить число 20 четвертому элементу массива x:
  - а) `x[20]=4;`
  - б) `x[4]=20;`
  - в) `x[3]=20;`
  - г) `x[4]:={20};`



г) ассоциацией.

9. Массив в JavaScript – это объект...

а) примитивного типа

б) переменного типа

в) неопределенного типа

г) ссылочного типа

10. Элементы двумерного массива находятся на пересечении...

а) диагоналей

б) строки и столбца

в) индексов и ячеек

г) записи и поля

11. Метод, для добавления нового элемента в конец массива –

а) pop()

б) join()

в) push()

г) unshift()

12. Массив можно создать с помощью конструктора

а) Array()

б) Dim()

в) Massive()

г) Matrix()

### Тесты без выбора варианта ответа

1. В скрипте описан одномерный целочисленный массив с индексами от 0 до 10:

```
let n=10;
```

```

let sum=0;
var a = [];
for (var i=0;i<n;i++)
{
    a[i] = Math.round(Math.random(999)*900+100);
}
for (i=0;i<n-2;i++)
sum=sum+a[i]-a[i+2]
console.log(sum);
console.log(a);

```

Какое наибольшее значение может иметь переменная `sum` после выполнения данного скрипта?

2. В скрипте используется одномерный целочисленный массив `A` с индексами от 0 до 9:

```

let A=[6, 9, 7, 2, 1, 5, 0, 3, 4, 8]
let c = 0;
for (i=1;i<=9;i++)
    if (A[i-1] < A[i]) {
        c ++
        t = A[i];
        A[i] = A[i-1];
        A[i-1] = t
    }
console.log(c);

```

Определить значение переменной  $s$  после выполнения скрипта. Как изменится массив после выполнения?

3. В скрипте используется одномерный целочисленный массив  $A$  с индексами от 0 до 24. Ниже представлен фрагмент кода, в котором задаются значения элементов:

```
let A=[]  
let n= 25;  
A[0]= 2;  
for (i= 1;i<n;i++)  
    A[i]= 2*A[i-1] % 10;
```

Чему будет равно значение  $A[24]$  после выполнения фрагмента скрипта?

4. В скрипте используется одномерный целочисленный массив  $A$  с индексами от 0 до 9. Ниже представлен фрагмент кода, в котором значения элементов сначала задаются, а затем меняются:

```
for (i= 0;i<=9;i++)  
    A[i]=9-i;  
for (i=0;i<=4;i++) {  
    k=A[i];  
    A[i]=A[9-i];  
    A[9-i]=k;  
}
```

Чему будут равны элементы этого массива после выполнения фрагмента скрипта?

5. Дан фрагмент скрипта, обрабатывающий двумерный массив  $A$  размера  $n \times n$ :

```
let k=0;
for (i= 0;i<3;i++)
{
  c = A[i][i];
  A[i][i] = A[k][i];
  A[k][i] = c;
}
```

Представим массив в виде квадратной матрицы, в которой для элемента массива  $A[i][j]$  величина  $i$  является номером строки, а величина  $j$  – номером столбца, в котором расположен элемент. Тогда какие элементы матрицы данный алгоритм меняет местами?

#### 4.7. Задачи для самостоятельной работы

- 1) В заданном массиве  $A(N)$  вычислить среднее геометрическое и среднее арифметическое значения для положительных элементов.
- 2) Подсчитать число и произведение отрицательных элементов заданного массива  $A(N)$ .
- 3) В одномерном массиве размера  $N$  найти сумму элементов, стоящих на четных местах.
- 4) Определить частное от деления суммы положительных элементов массива размера  $N$  на модуль суммы отрицательных элементов.
- 5) Дан массив целых чисел размера  $N$ . Верно ли, что сумма элементов, которые больше 20, превышает 100?
- 6) Дан массив целых чисел размера  $N$ . Верно ли, что сумма элементов, которые меньше 50, это четное число?

7) Дан массив размера  $N$ . Определить сумму и количество неотрицательных элементов, отличных от последнего элемента.

8) Дан массив размера  $N$ . Найти произведение и среднее арифметическое элементов массива, меньших некоторого числа  $m$ .

9) Дан массив вещественных чисел размера  $N$  и числа  $k_1, k_2$ . Из всех положительных элементов вычесть число  $k_1$ , из остальных — число  $k_2$ .

10) Дан массив вещественных чисел размера  $N$  и числа  $m_1, m_2 < N$ . Ко всем отрицательным элементам прибавить элемент с номером  $m_1$ , к остальным — элемент с номером  $m_2$ .

11) Дан массив вещественных чисел размера  $N$  и число  $b$ . Ко всем отрицательным элементам прибавить элемент с номером 1, из всех нулевых вычесть число  $b$ . Положительные элементы оставить без изменения.

12) Задан массив вещественных чисел  $A(n)$ . Найти сумму четных чисел, расположенных до заданного элемента массива  $x$ .

13) Задан массив целых чисел  $X(n)$ . Найти количество элементов массива, расположенных после первого нулевого элемента.

14) Дан массив  $X(N)$ . Вычислить сумму элементов массива с  $k_1$ -го по  $k_2$ -й (значения  $k_1$  и  $k_2$  вводятся с клавиатуры;  $k_2 > k_1, k_1, k_2 < N$ ).

15) Дан массив  $X(N)$ . Вычислить среднее арифметическое элементов массива с  $s_1$ -го по  $s_2$ -й (значения  $s_1$  и  $s_2$  вводятся с клавиатуры;  $s_2 > s_1, s_1, s_2 < N$ ).

16) Дан массив  $X(N)$ . Вычислить произведение элементов массива с  $k_1$ -го по  $k_2$ -й (значения  $k_1$  и  $k_2$  вводятся с клавиатуры;  $k_2 > k_1, k_1, k_2 < N$ ).

17) Даны массивы  $A(10)$  и  $B(10)$ . Сформировать массив  $C$ , элементы которого вычисляются по формуле:  $C_i = 3 \cdot a_i + 7 \cdot b_i$ .

18) Даны массивы  $A$  и  $B$  размера  $N$ . Сформировать новый массив  $C$  того же размера  $N$ , элементы которого определяются следующим образом:

$$C_i = \begin{cases} A_i + A_i^2 - B_i^3 + 42, & \text{если } i < 5 \\ A_i - B_i - 5, & \text{если } i \geq 5 \end{cases}.$$



19) Даны массивы A и B размера N. Сформировать новый массив C того же размера N, элементы которого определяются следующим образом:  $C_i = \begin{cases} \sin(A_i - B_i^2) + 4, & \text{если } i < 4 \\ |A_i - B_i| - 2, & \text{если } i \geq 4 \end{cases}$ .

20) Даны массивы A и B размера N. Сформировать новый массив C того же размера N, элементы которого определяются следующим образом:  $C_i = \begin{cases} 2 \cdot A_i + B_i^2 - 2, & \text{если } i < 7 \\ A_i + 3 \cdot B_i + \sqrt{A_i}, & \text{если } i \geq 7 \end{cases}$ .

21) Ввести массив A размера N. Сформировать массив B следующим образом:  $B_i = \begin{cases} 2 \cdot A_i + 12, & \text{если } A_i < 0 \\ A_i - 5\sqrt{A_i}, & \text{если } A_i \geq 0 \end{cases}$ .

22) Ввести массив A размера N. Сформировать массив B следующим образом:  $B_i = \begin{cases} 3 \cdot A_i + A_i^2, & \text{если } A_i < 1 \\ A_i - \left|7 - \frac{A_i}{2}\right|, & \text{если } A_i \geq 1 \end{cases}$ .

23) Ввести массив A размера N. Сформировать массив B следующим образом:  $B_i = \begin{cases} \sin A_i + \pi, & \text{если } A_i < \pi \\ A_i - \sqrt{\cos A_i}, & \text{если } A_i \geq \pi \end{cases}$ .

24) В одномерном массиве A(15), элементы которого подсчитываются по формуле:  $A(I) = (0.5 \cdot \sin(I-1)) / 3$  определить среднее арифметическое элементов массива.

25) Дан массив целых чисел размера N. Все элементы, оканчивающиеся цифрой 4, уменьшить вдвое. Все остальные четные элементы заменить на их квадраты.

26) Задан массив целых чисел B(n). Найти среднее геометрическое элементов с четными индексами, кратных 3.

27) Дан целочисленный массив из 20 элементов. Найти и вывести значения среди элементов массива, которые имеют чётное значение и не делятся на три, а также их номера.

28) Дан целочисленный массив из 30 элементов, принимающих значения от 0 до 999. Найти среднее арифметическое всех двузначных элементов массива, оканчивающихся цифрой 5.

29) Дан целочисленный массив из 40 элементов, все элементы которого – целые числа в интервале от -500 до 500. Найти среднее арифметическое всех положительных элементов массива, которые кратны модулю первого элемента.

30) Дан целочисленный массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 1000. Найти и вывести количество и сумму элементов массива, кратных тринадцати.

31) Дан целочисленный массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 100. Найти и вывести количество и произведение элементов массива, которые имеют чётное значение и не оканчиваются на 0.

32) Дан целочисленный массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 100. Найти и вывести количество и произведение двузначных элементов массива, которые не делятся на 6.

33) Дан массив целых чисел, состоящий из  $N$  элементов. Упорядочить в порядке убывания модулей элементы, расположенные между первым и последним отрицательным значениями массива.

34) Дан массив целых чисел, состоящий из  $N$  элементов. Поменять местами наибольший и наименьший элементы.

35) Дан массив целых чисел, состоящий из  $N$  элементов. Найти сумму элементов, расположенных до минимального элемента массива.

36) Дан массив целых чисел, состоящий из  $N$  элементов. Найти разность между наибольшим и наименьшим значениями массива. Вывести значения этих элементов и их индексы.

37) Дан массив целых чисел, состоящий из  $N$  элементов. Обнулить все элементы массива, расположенные за максимальным элементом массива.

38) Дан массив целых чисел, состоящий из  $N$  элементов. Подсчитать количество положительных элементов, расположенных до максимального элемента массива, и количество отрицательных элементов, расположенных после минимального.

39) Дан массив целых чисел, состоящий из  $N$  элементов. Найти и вывести минимальное значение среди элементов массива, которые имеют чётное значение и не делятся на три.

40) Дан массив целых чисел, состоящий из  $N$  элементов. Проверить, является ли он возрастающей последовательностью.

41) Для заданной целочисленной матрицы  $A(N,M)$  определить, является ли сумма её элементов чётным числом.

42) Проверить, является ли заданная квадратная матрица  $A(N,N)$  единичной.

43) Задана матрица  $A(n,m)$ . Поменять местами ее максимальный и минимальный элементы.

44) В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить наименьший элемент и сумму положительных элементов.

45) В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить количество и произведение нечетных элементов.

46) В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить номер наибольшего элемента, т.е. номера строки и столбца его расположения.

47) В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить произведение ненулевых элементов, кратных 3.

48) В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить среднее геометрическое элементов и элемент, наименее отличающийся от среднего геометрического.

49) Для заданной матрицы  $A(n \times m)$  и матрицы того же типа и размерности  $C(n \times m)$  найти значение выражения  $B=2 \cdot A+3 \cdot C$

50) Для заданной матрицы  $B(n \times m)$  и матрицы того же типа, но другой размерности  $C(k \times n)$  найти значение выражения  $A=B \cdot C+E$ , где  $E$  – единичная матрица того же размера.

51) Для заданной матрицы  $B(n \times m)$  и матрицы того же типа, но другой размерности  $C(k \times n)$  найти значение выражения  $A=3 \cdot B \cdot C$ .

52) Для заданной матрицы  $B(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $A=2 \cdot B+C^2$ .

53) Для заданной матрицы  $C(n \times n)$  найти значение выражения  $A=C+C^T$ , где  $C^T$  – транспонированная матрица (строки и столбцы меняются местами).

54) Для заданной матрицы  $A(n \times n)$  найти значение выражения  $B=5 \cdot A^2$ .

55) Для матрицы  $B(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $A=2 \cdot B-C^T$ , где  $C^T$  – транспонированная матрица (строки и столбцы меняются местами).

56) Для заданной матрицы  $A(n \times n)$  и матрицы того же типа и размерности  $C(n \times n)$  найти значение выражения  $B=A^2-C^2$ .

57) Найти сумму элементов матрицы  $A(N,N)$ , лежащих выше главной диагонали.

58) Вычислить количество положительных элементов квадратной матрицы  $A(N,N)$ , расположенных на диагоналях.

59) Преобразовать исходную матрицу  $N \times M$  так, чтобы нулевой элемент каждой строки был заменен средним арифметическим элементов этой строки.

60) Поменять местами элементы главной и побочной диагонали матрицы  $A(k,k)$ .

61) В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить количество простых чисел, расположенных на диагоналях матрицы.

62) В двумерном массиве  $A$ , состоящем из  $n \times n$  целых чисел вычислить минимальное простое число среди элементов, расположенных на главной диагонали.

63) Задана матрица целых чисел  $A(n \times m)$ . Вычислить среднее арифметическое элементов каждого столбца заданной матрицы.

64) Задана матрица целых чисел  $A(n \times m)$ . Вывести номера строк, в которых находится более двух простых чисел.

## Глава 5. Функции пользователя

### 5.1. Синтаксис функций пользователя в JavaScript

Функция в JavaScript – это фрагмент кода, который можно вызывать через имя и параметры из любого места основного сценария.

Функции можно описать двумя способами – по аналогии с классическими языками (Pascal, C++) и по новому стандарту ES6 в стрелочном виде.

В классическом виде функция описывается с помощью ключевого слова `function`, за которым следует уникальное имя. Затем в круглых скобках указываются параметры, разделяемые запятыми. Они называются формальными параметрами или аргументами. Функция может не иметь параметров вообще.

Команды и операторы, позволяющие выполнять действия над параметрами функции, записываются в фигурных скобках.

Условно функции можно поделить на два вида – невозвращающие результат (подпрограммы) и возвращающие результат с помощью ключевого слова `return`.

После команды `return` происходит выход из функции. В основном скрипте вызов функции происходит по ее имени с заменой параметров на фактические значения. Они называются фактическими параметрами.

Рассмотрим работу с функциями на примере сложения двух аргументов:

```
function sum(num1, num2)
{
  let s=num1+num2
  return s
}
console.log(sum(3,5))
```

```
Console Shell
8
Hint: hit control+c anytime to enter REPL.
> 
```

В стрелочном виде при создании функции не используется слово `function`, а сразу записывается ее имя. Перед именем нужно указать тип объекта, лучше, `const` – неизменяемый объект. На самом деле здесь имя не самой функции, а переменной, позволяющей обращаться к этой функции.

При коллективной разработке мы можем создать функцию

```
let myfun = (param) => result;
```

а затем сказать

```
myfun = 7.
```

И потеряем функцию.

Поэтому критически важно с помощью `const` объявлять объектные переменные и функции.

```
const myfun = (param) => result;
```

Это не даёт возможности переопределения и в большом приложении убеждает от краша приложения по непонятным причинам.

Используя стрелочную функцию, можно, как показано в следующем примере с суммой, существенно упростить синтаксис:

```
const sum = (num1, num2) => {
  const result = num1 + num2;
  return result;
};
console.log(sum(4, 11)); // => 15
```

Сокращенный вариант:

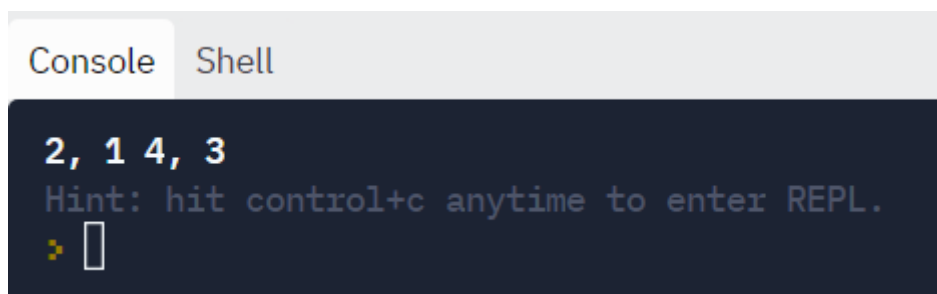
```
const sum = (num1, num2) => num1 + num2;  
console.log(sum(4, 11)); // => 15
```

В сокращенном варианте удалено и слово `return`, поскольку стрелка указывает на то, что должно быть возвращено. Еще одно преимущество заключается в том, что, если функция принимает только один аргумент, можно убрать и круглые скобки, окружающие аргументы.

Если тело функции состоит более чем из одной строки, то его следует заключить в фигурные скобки.

Пример. Составить скрипт, в результате которого величина `a` меняется значением `c` величиной `b`, а величина `c` — с величиной `d`. (Определить функцию, осуществляющую обмен значениями двух переменных величин):

```
const dd1=(x,y)=>  
{  
  let temp=x  
  x=y  
  y=temp  
  return(x+', '+y)  
}  
console.log(dd1(1,2),dd1(3,4))
```



```
Console Shell  
2, 1 4, 3  
Hint: hit control+c anytime to enter REPL.  
> █
```



## 5.2. Область видимости объектов и переменных

Переменные классифицируются как глобальные или локальные в зависимости от области видимости (доступности) переменной.

Глобальная переменная доступна во всём сценарии – как и в основном коде, так и во всех функциях. Она объявляется вне тела функции. Переменная, которая объявляется в теле конкретной функции, называется локальной, и доступна только внутри этой функции. По завершении выполнения функции локальные переменные удаляются из памяти, поэтому одно и то же имя можно использовать для разных локальных переменных в разных функциях или для глобальной переменной. Но во избежание путаницы лучше использовать разные имена.

Проиллюстрируем на примере:

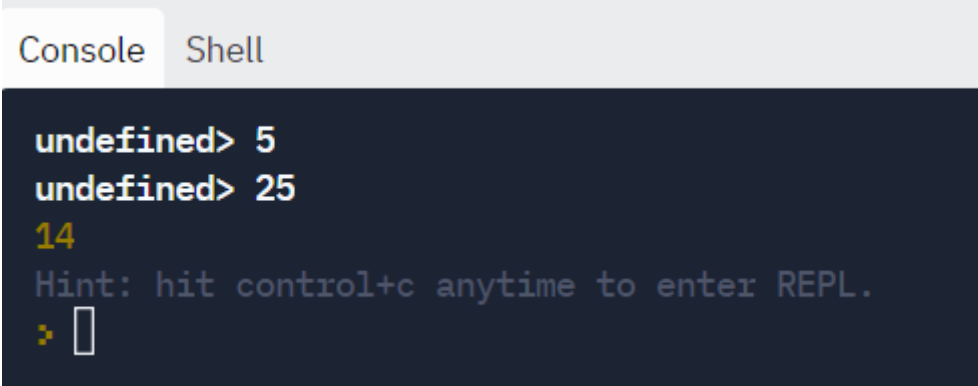
Найти среднее арифметическое целых чисел, которые делятся на 7 и не делятся на 14, из промежутка [a,b]:

```
const srzn=(a,b)=>
{
  num=0;
  res=0;
  for(i=a;i<b;i++)
  {if(i%7===0&& i%14>0)
  {res+=i;
  num+=1 } }
  return(res/num)
}
let res=parseInt(prompt())
```

```
let num=parseInt(prompt())
```

```
console.log(srzn(res,num))
```

Результат:



```
Console Shell  
undefined> 5  
undefined> 25  
14  
Hint: hit control+c anytime to enter REPL.  
> █
```

Все глобальные переменные являются свойствами объекта `window`. Если имя локальной переменной совпадает с именем глобальной, то все операции внутри функции осуществляются с локальной переменной, а значение глобальной не изменится. Если нужно обратиться к глобальной переменной с тем же именем внутри функции, то ее записывают через объект `window`, например, `window.var1`.

### 5.3. Характеристики функций

#### 1) Детерминированность

Если функция при одних и тех же входных параметрах принимает каждый раз разное значение, то она называется недетерминированной, в противном случае – детерминированной.

Примерами недетерминированных функций являются те, которые содержат среди выражений датчики псевдослучайных чисел, методы расчета времени (изменения происходят каждую миллисекунду).

Функция `Math.random()` возвращает случайное число от 0 до 1:

```
Math.random(); // 0.9337432365797949
```

```
Math.random(); // 0.5550694016887598
```

функция `Date.now()` каждый раз возвращает новое значение:

```
// Возвращает текущее время в миллисекундах
```

```
Date.now(); // 1571909874844
```

```
Date.now(); // 1571909876648
```

Детерминированные функции, напротив, ведут себя предсказуемо. Для одних и тех же входных данных они всегда выдают один и тот же результат. Именно такими являются функции в математике. Для одного и того же  $x$  результат работы функции  $y = f(x)$  будет один и тот же.

## 2) Побочные эффекты

Вторая ключевая характеристика функций — наличие побочных эффектов. Побочным эффектом называют любые действия, изменяющие среду выполнения. К ним относятся любые файловые операции, такие как запись в файл, чтение файла, отправка или приём данных по сети и даже вывод в консоль. Кроме того, побочными эффектами считаются обращения к глобальным переменным (как на чтение, так и запись) и изменение входных аргументов в случае, когда они передаются по ссылке. Вызов функции с побочными эффектами также считается побочным эффектом.

```
const sayHiTo = (name) => {  
  const greeting = `Hi, ${name}!`;   
  console.log(greeting);  
};
```

С другой стороны, любые вычислительные операции не являются побочными эффектами.

Побочные эффекты составляют одну из самых больших сложностей при разработке. Их наличие значительно затрудняет тестирование и отладку. Приводит к возникновению огромного числа ошибок (только при работе с файлами количество возможных ошибок измеряется сотней: начиная с того, что закончилось место на диске, заканчивая попыткой читать данные из несуществующего файла). Для их предотвращения код обрастает большим числом проверок и защитных механизмов.

Не существует способа избавиться от побочных эффектов совсем, но их влияние на программу можно минимизировать. Как правило, в типичной программе побочных эффектов не так много по отношению к остальному коду, и происходят они лишь в самом начале и в конце.

### 3) Чистые функции

Идеальная функция с точки зрения удобства работы с ней называется чистой. Чистая функция — это детерминированная функция, которая не производит побочных эффектов. Такая функция зависит только от своих входных аргументов и всегда ведёт себя предсказуемо. Такие функции на 100% соответствуют своим математическим аналогам и могут рассматриваться как математические функции.

Чистые функции обладают рядом ключевых достоинств:

- Их просто тестировать. Достаточно передать на вход функции нужные параметры и посмотреть ожидаемый выход.
- Их безопасно запускать повторно, что особенно актуально в асинхронном коде или в случае многопоточного кода.
- Их легко комбинировать, получая новое поведение без необходимости переписывать скрипт.

В хорошо спроектированных программах побочные эффекты стараются изолировать в небольшой части приложения так, чтобы большая часть кода была чистой.

#### 4) Рекурсии

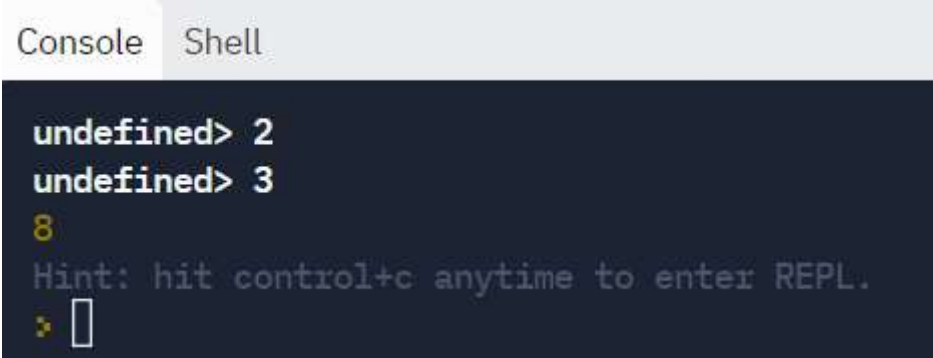
Рекурсиями или рекурсивными (возвратными) функциями называются те функции, которые вызываются через свои предыдущие вызовы, т.е. возможность функций вызывать самих себя.

Рассмотрим пример вычисления  $x^n$ :

```
const RecursivePow=(base, power)=>
{
  return power == 0 ? 1 : base * RecursivePow(base, power - 1);
}

let x=parseInt(prompt())
let n=parseInt(prompt())
console.log(RecursivePow(x,n))
```

Результат:



```
Console Shell
undefined> 2
undefined> 3
8
Hint: hit control+c anytime to enter REPL.
* □
```

Алгоритм работы такой программы следующий. Подставим начальные значения  $base=2$ ,  $power=3$ .

1 шаг: 3 не равно 0, поэтому выполняется вторая ветка условного оператора  $2 * RecursivePow(2, 2)$ ;

2 шаг: теперь `power=2`, 2 не равно 0, значит, выполняется заново: `2*RecursivePow(2, 1)`;

3 шаг: значение `power=1`, снова 1 не равно 0, значит еще раз вызов функции: `2*2*2* RecursivePow(0)`;

4 шаг: теперь `power=0` – истинное условие, выполняем первую ветку – `return 1`. Тогда целиком получим значение `2*2*2*1=8`.

Если остановить рекурсию шагом раньше, т.е. изменить строку `return power == 1 ? base : base * RecursivePow(base, power - 1)`; то для `power>0` потребуется на один вызов меньше, а для `power=0` будет некорректным результатом (ошибка).

Рекурсии часто используются для организации возвратных последовательностей (прогрессий), нахождения произведения чисел (факториала), чисел Фибоначчи и т.д. Главным соглашением при создании рекурсий является наличие условия выхода из рекурсии во избежание ее заикленности.

#### 5.4. Упаковка и распаковка аргументов

Давайте снова определим функцию `sum`, принимающую на вход два числа и возвращающую их сумму:

```
const sum = (a, b) => a + b;
```

```
sum(1, 2); // 3
```

```
sum(-3, 10); // 7
```

Пока всё просто и понятно. Сложности возникают при дополнительных требованиях: что, если захотим суммировать не два, а три числа? Или пять, или даже десять? Писать для обработки каждого случая отдельную функцию — очевидно плохой, непрактичный вариант:

```
const sumOfTwo = (a, b) => a + b;
```

```
const sumOfTree = (a, b, c) => a + b + c;
```

```
const sumOfTen = (a, b, c, d, e, f, g, h, i, j) => a + b + c + d + e + f + g + h + i +  
j;
```

Надо, чтобы единая функция могла работать с разным количеством аргументов. Как это сделать?

Можно заметить, что в стандартной библиотеке JavaScript существуют функции, которые могут принимать разное количество аргументов. Например, сигнатура функции `Math.max` определяется так:

```
Math.max([value1[, value2[, ...]])
```

Она говорит нам о том, что в `Math.max` можно передать любое количество элементов:

```
Math.max(10, 20); // 20
```

```
Math.max(10, 20, 30); // 30
```

```
Math.max(10, 20, 30, 40, 50); // 50
```

```
Math.max(-10, -20, -30); // -10
```

С точки зрения вызова — ничего необычного, просто разное число аргументов. А вот определение функции с переменным числом аргументов выглядит необычно и использует незнакомый для нас синтаксис:

```
const func = (...params) => {  
  // params — это массив, содержащий все  
  // переданные при вызове функции аргументы  
  console.log(params);  
};  
  
func(); // => []  
func(9); // => [9]  
func(9, 4); // => [9, 4]
```

```
func(9, 4, 1); // => [9, 4, 1]
func(9, 4, 1, -3); // => [9, 4, 1, -3]
```

Символ троеточия ... перед именем формального параметра в определении функции обозначает rest-оператор. Запись ...params в определении func из примера выше означает буквально следующее: "все переданные при вызове функции аргументы поместить в массив params".

Если вовсе не передать аргументов, то rest-массив params будет пустым:

```
func(); // => []
```

В функцию можно передать любое количество аргументов — все они попадут в rest-массив params:

```
func(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
// => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Аргументы могут быть любого типа — числа, строки, массивы и др.:

```
func(1, 2, 'hello', [3, 4, 5], true);
// => [1, 2, 'hello', [3, 4, 5 ], true]
```

Основная сложность для понимания оператора ... состоит в том, что он выполняет различные действия в зависимости от того, где применяется. В определении функции он выполняет "упаковку" параметров, а при вызове — наоборот, "распаковку". Теперь у нас достаточно знаний, чтобы с помощью rest-оператора переписать нашу функцию sum так, чтобы она умела суммировать любое количество чисел (а не только два числа, как сейчас):

```
const sum = (...numbers) => {
  let result = 0;
  for (let num of numbers) {
    result += num;
  }
}
```



```

    return result;
};
sum();    // 0
sum(10);  // 10
sum(10, 4); // 14
sum(8, 10, 4); // 22

```

Наша реализация `sum` имеет интересную особенность: мы решили сделать так, что при вызове без аргументов возвращается значение 0 ("Ничего"), при вызове с одним числом возвращается это же число. Хотя можно было обработать эти ситуации как-то по-другому (например, вернуть специальное значение `null`), это вопрос удобного выбора, подходящего для решения конкретных практических задач.

В таком контексте `rest`-массив можно считать, как "необязательные аргументы", которые можно либо вовсе не передавать, либо передавать столько, сколько хочешь. А что, если мы захотим, чтобы функция имела два обычных ("обязательных") именованных параметра, а остальные были необязательными и сохранялись в `rest`-массиве? Всё просто: при определении функции сначала указываем стандартные именованные формальные параметры (например, `a` и `b`) и в конце добавляем `rest`-массив:

```

const func = (a, b, ...params) => {
  // параметр 'a' содержит первый аргумент
  console.log(`a -> ${a}`);

  // параметр 'b' содержит второй аргумент
  console.log(`b -> ${b}`);

  // params содержит все остальные аргументы
  console.log(params);
}

```

```
};
```

```
func(9, 4);
```

```
// => a -> 9
```

```
// => b -> 4
```

```
// => []
```

```
func(9, 4, 1);
```

```
// => a -> 9
```

```
// => b -> 4
```

```
// => [1]
```

```
func(9, 4, 1, -3);
```

```
// => a -> 9
```

```
// => b -> 4
```

```
// => [1, -3]
```

```
func(9, 4, 1, -3, -5);
```

```
// => a -> 9
```

```
// => b -> 4
```

```
// => [1, -3, -5]
```

## Распаковка аргументов

spread-оператор распаковки аргументов в вызовах функций синтаксически идентичен rest-оператору в определениях, но выполняет обратное действие. Посмотрим на примере функции `sum`, напомним её определение:

```
const sum = (...numbers) => {
```

```
  let result = 0;
```

```
  for (let num of numbers) {
```

```
    result += num;
  }
  return result;
};
```

Вызовем `sum`, применив `spread`-оператор к массиву аргументов:

```
const arrayOfNumbers = [1, 7, 4];
sum(...arrayOfNumbers); // 12
```

`spread`-оператор раскладывает массив на аргументы. Количество аргументов, полученных `spread`-оператором, равно количеству элементов массива. По сути, код выше преобразуется в вызов:

```
sum(arrayOfNumbers[0], arrayOfNumbers[1], arrayOfNumbers[2]);
// sum(1, 7, 4);
```

Как и в случае с определением функций, `spread`-оператор может использоваться совместно с позиционными аргументами:

```
const arrayOfNumbers = [1, 7, 4];
sum(8, ...arrayOfNumbers); // 20
sum(8, 10, ...arrayOfNumbers); // 30
sum(8, 10, 70, ...arrayOfNumbers); // 100
```

В отличие от `rest`-оператора в определении функций, `spread`-оператор не обязательно должен быть последним, он может располагаться в любой позиции:

```
const arrayOfNumbers = [1, 7, 4];
sum(8, 10, ...arrayOfNumbers); // 30
sum(8, ...arrayOfNumbers, 10); // 30
sum(...arrayOfNumbers, 8, 10); // 30
```

Более того, может быть любое количество spread-операторов и в любом порядке:

```
const arr1 = [1, 7, 4];
const arr2 = [5, 5];
// sum(1, 7, 4, 5, 5);
sum(...arr1, ...arr2); // 22
// sum(5, 5, 1, 7, 4);
sum(...arr2, ...arr1); // 22
// sum(8, 1, 7, 4, 10, 5, 5);
sum(8, ...arr1, 10, ...arr2); // 40
```

## 5.5. Юнит-тесты

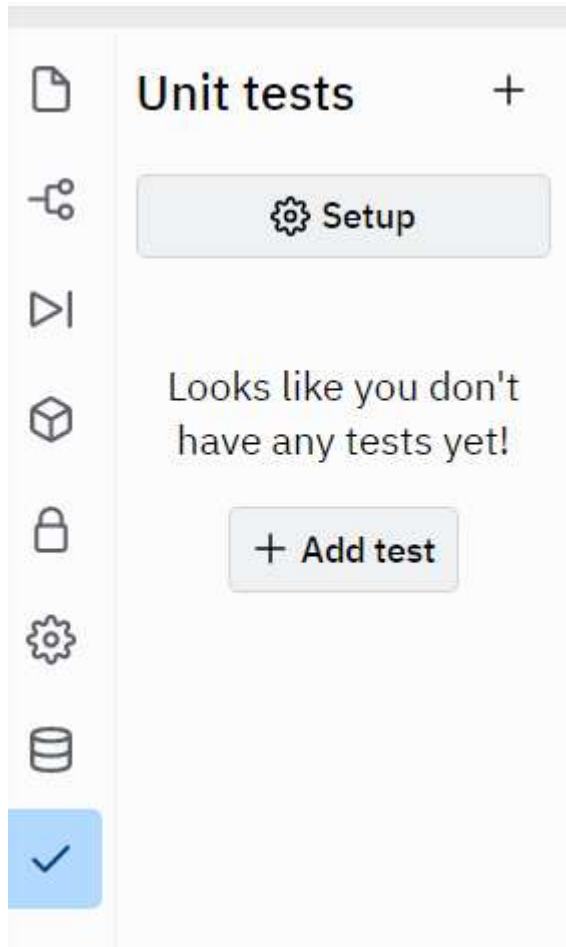
Каждый, даже самый простейший скрипт, можно оформить в виде отдельной функции, а в основной части сделать ее вызов. Такой вид позволяет удобно тестировать функцию на наличие ошибок.

Согласно правилам модульного программирования, задача разбивается на отдельные подзадачи, из которых формируются пользовательские функции. Каждую из этих функций можно отдельно проверить на наличие ошибок с помощью юнит-тестов.

Создадим в repl.it новый node.js проект и наберем там следующий код:

```
const mysum = (a,b) => {
  return a + b;
}
module.exports = {mysum};
```

Затем перейдем во вкладку Unit tests:



Нажмем кнопку Add test (добавить тест) и напишем там следующий код:

```
test("mysum", async function() {  
  expect(index.mysum(1, 2)).toBe(3);  
});
```

**Name**

**Code** <> Show me an example

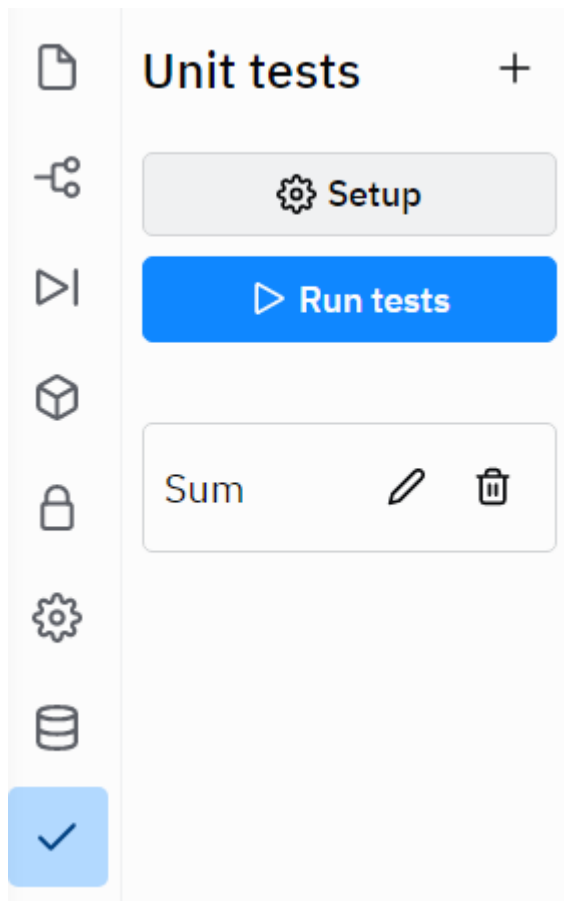
```
1 test("mysum", async function() {
2   expect(index.mysum(1, 2)).toBe(3);
3 });
```

**Failure message**

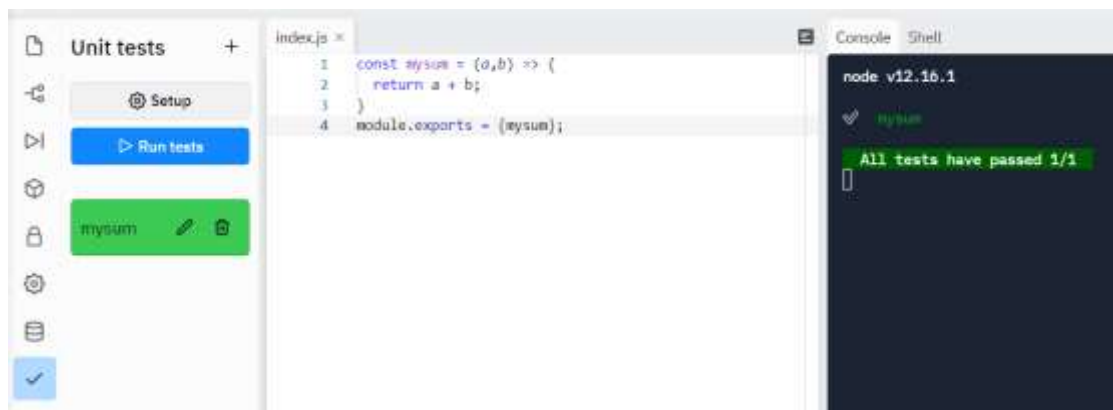
Close Save

Этот код составляется по следующему правилу: записывается имя функции с тестовыми аргументами, значение которых легко посчитать, например, `mysum(1, 2)`. После точки записывается метод `toBe` с известным результатом. В нашем примере  $1+2=3$ , поэтому `mysum(1, 2).toBe(3)`.

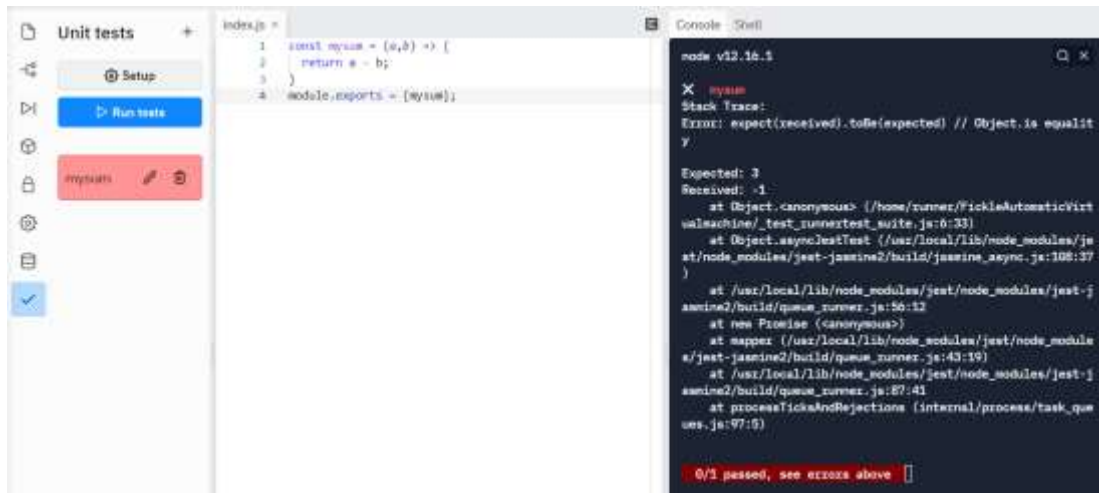
После сохранения `Save` всегда можно вернуться к текстовой функции для исправления. Для вызова теста нужно нажать на кнопку `Run tests`:



Если тест не содержит ошибок, то мы увидим следующий результат:



Допустим. В нашей функции закралась ошибка:  $a-b$ . Тогда в результате несовпадения значения функции с числом 3 получим:



Ожидалось число 3, а получено -1.

## 5.6. Скрипты, содержащие функции пользователя

Задача №1. Даны действительные числа  $s, t$ . Составить программу вычисления выражения  $f(t, -2s, 1.17) + f(2.2, t, s-t)$ , где  $f(a,b,c) = (2a - b - \sin(c)) / (5 + |c|)$ .

Способ 1:

$f=(a,b,c) \Rightarrow$

{

return  $(2 * a - b - \text{Math.sin}(c)) / (5 + \text{Math.abs}(c))$ ;

}

let  $s=\text{parseFloat}(\text{prompt}())$ ;

let  $t=\text{parseFloat}(\text{prompt}())$ ;

let  $r=f(t, -2 * s, 1.17) + f(2.2, t, s - t)$ ;

console.log(r)

Результат:



```
Console Shell
undefined> 4
undefined> 3
2.2129016865834266
Hint: hit control+c anytime to enter REPL.
> █
```

Способ 2:

HTML:

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<title>5-1</title>
```

```
</head>
```

```
<body>
```

```
<input type="text" id="mf1">
```

```
<input type="text" id="mf21">
```

```
<input id="abt" type="button" value="Нажми меня" onclick="press()>
```

```
<p id="content"></p>
```

```
<script type="text/javascript" src="p5-1.js">
```

```
</script>
```

```
</body>
```

```
</html>
```

JS (Файл p5-1.js):

```
function f(a,b,c)
```

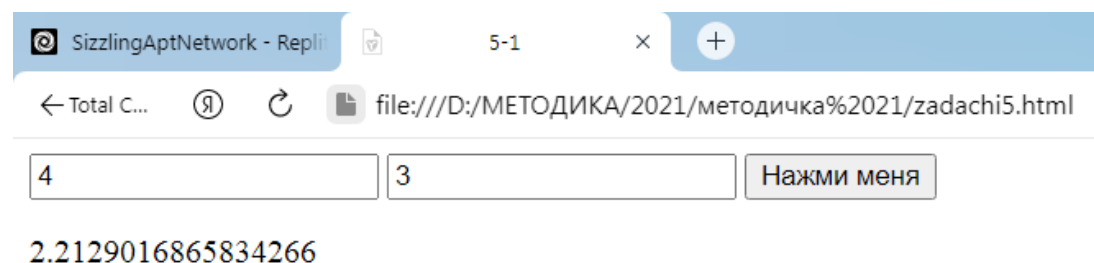
```
{
```

```

return (2 * a - b - Math.sin(c)) / (5 + Math.abs(c));
}
var mf,mf2,ab, out
function press()
{
var s,t,r
s=parseFloat(mf.value);
t=parseFloat(mf2.value);
r=f(t, -2 * s, 1.17) + f(2.2, t, s - t);
out.innerHTML = r;
}
window.onload=function()
{
mf=document.getElementById("mf1");
mf2=document.getElementById("mf2");
ab=document.getElementById("ab");
out=document.getElementById("content");
}

```

Результат:

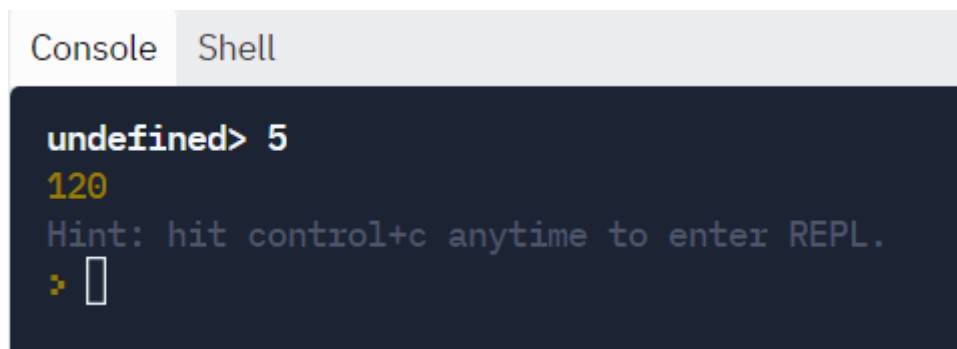


Задача №2. Вычислить факториал числа

Способ 1 (рекурсия)

```
const fact=(n)=>
{
if (n==1)
{ return 1 }
else
{ return n*fact(n-1)
}
}
var n=parseInt(prompt())
console.log(fact(n))
```

Результат:



```
Console Shell
undefined> 5
120
Hint: hit control+c anytime to enter REPL.
undefined> █
```

Способ 2 – циклическое умножение

```
const fact=(n)=>
{
var f=n
for (var i=1; i<n; i++)
f=i*f
return f
}
var n=parseInt(prompt())
```

```
console.log(fact(n))
```

Задача №3. В массиве A(12, 15) найти произведение столбцов: второго на десятый, третьего на девятый. Вычисление произведения двух столбцов матрицы оформить в виде функции.

```
const product=(q,k)=>
{
for (i=0;i<b.length;i++)
b[i]=a[i][q]*a[i][k]
return b
}

let a=new Array(12)
for (i=0;i<a.length; i++)
{
a[i]=new Array(15)
for (j=0;j<a[i].length;j++)
a[i][j]=Math.floor(10*Math.random()+1)
}

var b=new Array(12)

n=2

m=10

b=product(n,m)

console.log(b)

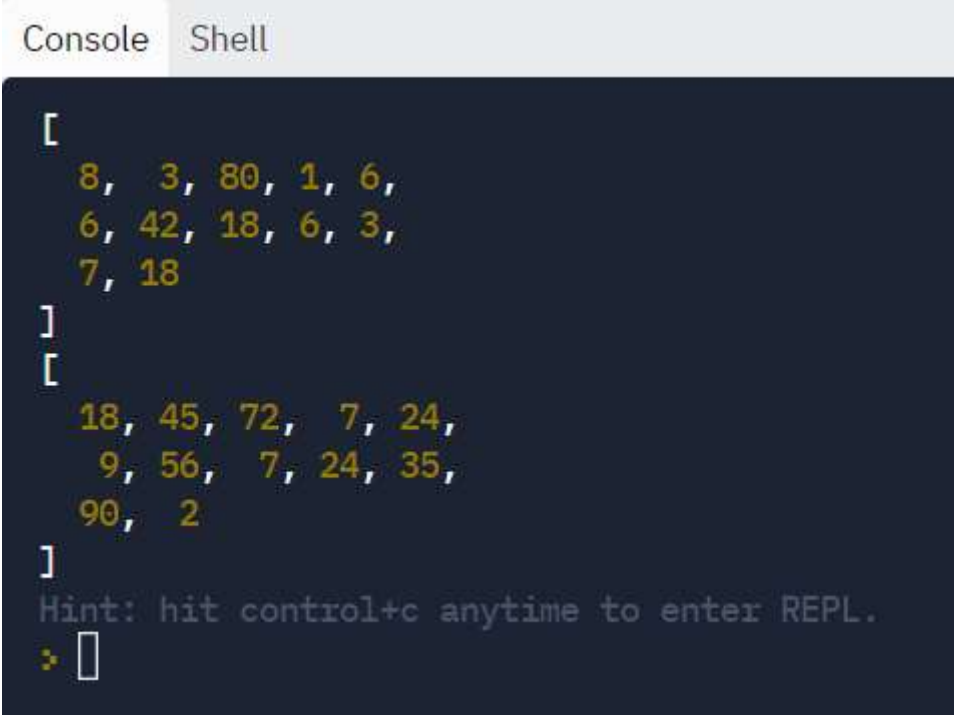
n=3

m=9

b=product(n,m)
```

console.log(b)

Результат:



```
Console Shell
[
  8, 3, 80, 1, 6,
  6, 42, 18, 6, 3,
  7, 18
]
[
  18, 45, 72, 7, 24,
  9, 56, 7, 24, 35,
  90, 2
]
Hint: hit control+c anytime to enter REPL.
> [
```

Задача №4. Найти количество и сумму делителей числа.

```
const ksd=x1=>
```

```
{
```

```
count=0
```

```
sum=0
```

```
for (i=2;i<=x1/2;i++)
```

```
{
```

```
if (x1%i==0)
```

```
{
```

```
count++
```

```
sum=sum+i
```

```
}
```

```

}
console.log('Количество='+count+"\n" + 'Сумма='+sum)
}
let x=parseInt(prompt())
ksd(x)

```

Результат:

```

Console Shell
undefined> 12
Количество=4
Сумма=15
Hint: hit control+c anytime to enter REPL.
➤ █

```

Задача №5. Перевод натурального числа  $n$  из десятичной системы счисления в

двоичную.

```

function bin(n)
{
  let s=""
  if (n>1) bin(Math.floor(n/2))
  s+=n%2
  console.log(s)
}
var n1
  n1=parseInt(prompt())
bin(n1)

```

Результат:

```
Console Shell
undefined> 12
1
1
0
0
Hint: hit control+c anytime to enter REPL.
> 
```

2 способ:

```
function bin(n)
{
  let s=""
  while (n>0) {
    s=n%2+s
    n=Math.floor(n/2)
  }
  console.log(s)
}
var n1
  n1=parseInt(prompt())
bin(n1)
```

Результат:

```
undefined> 12
1100
Hint: hit control+c anytime to enter REPL.
> 
```

## 5.7. Контрольные вопросы и тесты

### Дополнительные вопросы

1. Что позволяет функция пользователя улучшить в коде?
2. Можно ли изменить число параметров функции при ее вызове?
3. Что регулирует область видимости функции?
4. В каких случаях в стрелочной записи функций можно не использовать круглые и фигурные скобки?
5. Что произойдет с кодом, если в рекурсии не предусмотреть условие выхода?
6. Можно ли избавиться от побочных эффектов для чистоты функции?
7. Является ли функция, не возвращающая значение, детерминированной?
8. Для чего используется юнит-тестирование?
9. Как получить доступ к глобальной переменной внутри функции?
10. Как объявить и вызвать функцию без параметров?

### Тесты с выбором варианта ответа

1. Заголовок функции не содержит:
  - а) тип возвращаемого значения
  - б) тело функции
  - в) имя функции
  - г) список переменных



2. Переменные, описанные внутри функции, называются:
- а) глобальными
  - б) локальными
  - в) формальными
  - г) фактическими
3. Функция, вызывающая саму себя через предыдущее значение:
- а) рекурсия
  - б) процедура
  - в) подпрограмма
  - г) факториал
4. Тело функции содержит:
- а) тип возвращаемого значения
  - б) возвращаемое значение
  - в) имя функции
  - г) список переменных
5. Переменные, описанные перед всеми функциями, называются:
- а) глобальными
  - б) локальными
  - в) формальными
  - г) фактическими
6. Параметры указанные в заголовке функции называются:
- а) глобальными
  - б) локальными
  - в) формальными
  - г) фактическими
7. К рекурсивным алгоритмам не относится подсчет:
- а) возведения в степень
  - б) числа пи
  - в) числа Фибоначчи
  - г) факториала
8. Возврат результата из функции в программу:
- а) void
  - б) return
  - в) goto
  - г) function
9. Выберите пример недетерминированной функции:
- а) функция, которая всегда возвращает число 42

б) функция подсчета площади на основе длины и ширины

в) функция, возвращающая произведение переданного аргумента и случайного числа от 1 до 3

г) функция, которая вычисляет степень натурального числа

10. Какой тип данных будет у значения параметра something:

```
const foo = (...something) => {
```

```
  // some code
```

```
};
```

а) Массив

б) Число

в) undefined

г) Строка

11. Выберите правильное определение функции:

а) `const foo = (...first, ...rest) => {`

б) `const foo = (...rest, last) => {`

в) `const foo = (...arguments) => {`

г) `const foo = (first, ...rest, last) => {`

12. Что напечатает функция на экран?

```
const args = [3, 2];
```

```
// Функция Math.pow возводит первый аргумент в степень, переданную
```

вторым аргументом

```
console.log(Math.pow(...args));
```

а) 6            б) 9

в) 5            г) 1

1. Ниже приведено определение функции, высчитывающей квадрат суммы двух чисел:

```
const squareOfSum = (a, b) => {  
  const result = (a * a) + (2 * a * b) + (b * b);  
  console.log(result);  
  return result;  
};
```

Что можно сказать о такой реализации функции? Функция является или не является чистой, детерминированная или не является детерминированной, обладает или нет побочными эффектами? Объяснить, почему выбраны соответственные характеристики.

2. Что напечатает функция на экран?

```
const foo = (...arguments) => {  
  console.log(arguments);  
};
```

```
const arg = [[], 3, 10];  
foo(arg);
```

3. Напишите в ответе число, которое будет напечатано в результате выполнения следующего скрипта.

```
function F(x)  
{  
  return (x*x-4)*(x*x-4)+6  
}
```

```

var a,b,t,M,R
a=-10; b=10;
M=a; R=F(a);
for (t=a;t<=b;t++)
{
  if (F(t)<R){
    M=t;
    R=F(t);
  }
}
console.log(M)

```

4. Дан скрипт для вычисления  $P$  по формуле:  $P = \begin{cases} (n + 4)^2, & \text{если } n < 5 \\ (n + 1)!, & \text{если } n \geq 5 \end{cases}$  где

$n$  - заданное натуральное число. Факториал и степень числа оформляются в виде функций.

```

function fa(n)
{
  let f=1
  for (var i=2; i<=n;i++)
  f=f*i
  return f
}
function sc(n)
{

```

```
return Math.pow((n+4),2)
}
var x=parseInt(prompt())
var y
if (x<5) y=sc(x)
else y=fa(x+1)
console.log(y)
```

Изменить эти функции так, чтобы степень числа высчитывалась циклически, а факториал сразу  $(n+1)!$  Переделать заголовки функций под стрелочную нотацию.

5. Дан скрипт, решающий задачу:

Задано двузначное число. Определить, какая из его цифр больше: первая или вторая, или одинаковы ли его цифры.

```
var x=parseInt(prompt('Введите число'));
let b=x % 10;
let a=Math.floor(x/10);
if (a>b) alert("первая");
else if (a<b) alert("вторая");
else alert("равны");
```

Переписать его так, чтобы можно определить максимум из цифр трехзначного и четырехзначного числа, используя функцию для нахождения наибольшей из двух цифр.

## 5.8. Задачи для самостоятельной работы

1) Даны координаты вершин многоугольника  $(x_1, y_1, x_2, y_2, \dots, x_{10}, y_{10})$ . Написать программу для вычисления его периметра (вычисление расстояния между вершинами оформить подпрограммой).

2) Определить полусумму длин двух векторов:  $A(1,5; 2,5; -0,3)$  и  $B(-11,7; 9,3; 2,5; 3,7; -1,2)$ . Вычисление длины вектора оформить в виде функции.

3) Написать программу, которая выбирает максимальное из четырех заданных чисел, используя функцию, которая выбирает максимальное из двух чисел.

4) Даны отрезки  $A, B, C$  и  $D$ . Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника (определить процедуру, печатающую площадь треугольника со сторонами  $x, y$  и  $z$ , если такой треугольник существует).

5) Определить значение  $z = (\max(a, 2b)) \cdot (\max(2a - b, b))$ , где  $\max(x, y)$  — максимальное из чисел  $x, y$ .

6) Определить значение  $z = (\min(a, 3b) + \min(2a - b, 2b))$ , где  $\min(x, y)$  — минимальное из чисел  $x, y$ .

7) Даны три квадратных уравнения:  $ax^2 + bx + c = 0$ ,  $bx^2 + ax + c = 0$ ,  $cx^2 + ax + b = 0$ . Сколько из них имеют вещественные корни? (Определить функцию, позволяющую распознавать наличие вещественных корней в квадратном уравнении.)

8) Найти периметр треугольника, заданного координатами своих вершин. (Определить функцию для расчета длины отрезка по координатам его вершин.)

9) Алгоритм вычисления значения функции  $F(n)$ , где  $n$  — натуральное число, задан следующими соотношениями:

$$F(1) = 1;$$

$$F(n) = F(n-1) * n, \text{ при } n > 1.$$

Чему равно значение функции  $F(5)$ ?

10) Алгоритм вычисления значения функции  $F(n)$ , где  $n$  – натуральное число, задан следующими соотношениями:

$$F(1) = 1;$$

$$F(n) = F(n-1) * (n + 1), \text{ при } n > 1.$$

Чему равно значение функции  $F(7)$ ?

11) Алгоритм вычисления значения функции  $F(n)$ , где  $n$  – натуральное число, задан следующими соотношениями:

$$F(1) = 1;$$

$$F(n) = F(n-1) * (2*n^2 + 1), \text{ при } n > 1.$$

Чему равно значение функции  $F(4)$ ?

12) Алгоритм вычисления значения функции  $F(n)$ , где  $n$  – натуральное число, задан следующими соотношениями:

$$F(0) = 1, F(1) = 1,$$

$$F(n) = F(n-1) + F(n-2), \text{ при } n > 1.$$

Чему равно значение функции  $F(8)$ ?

13) Алгоритм вычисления значения функции  $F(n)$ , где  $n$  – натуральное число, задан следующими соотношениями:

$$F(0) = 1, F(1) = 1,$$

$$F(n) = F(n-1)*F(n-2)+1, \text{ при } n > 1.$$

Чему равно значение функции  $F(6)$ ?

14) Алгоритм вычисления значения функции  $F(n)$ , где  $n$  – натуральное число, задан следующими соотношениями:

$$F(1) = 1, F(2) = 1,$$

$$F(n) = F(n-2)*n + 2, \text{ при } n > 2.$$

Чему равно значение функции  $F(8)$ ?

15) Алгоритм вычисления значения функции  $F(n)$ , где  $n$  – натуральное число, задан следующими соотношениями:

$$F(1) = 1, F(2) = 1,$$

$$F(n) = F(n-2) \cdot (n-1) + 2, \text{ при } n > 2.$$

Чему равно значение функции  $F(7)$ ?

16) Вычислить  $\frac{A_{x+1}^{y+1} P_{x-y}}{P_{x-1}}$  при заданных значениях  $x$  и  $y$ . Где функции  $A_n^k = \frac{n!}{(n-k)!}$ ,  $P_n = n!$ .

17) Вычислить  $\frac{A_{x+y}^{y-3} P_{2+y}}{P_{x-y}}$  при заданных значениях  $x$  и  $y$ . Где функции  $A_n^k = \frac{n!}{(n-k)!}$ ,  $P_n = n!$ .

18) Вычислить  $\frac{A_y^x}{P_{x-1}} + C_y^{y-x}$  при заданных значениях  $x$  и  $y$ . Где функции  $C_n^k = \frac{n!}{k!(n-k)!}$ ,  $A_n^k = \frac{n!}{(n-k)!}$ ,  $P_n = n!$ .

19) Вычислить  $\frac{A_{y-1}^{x+1}}{P_{x+y}} + C_x^{y-1}$  при заданных значениях  $x$  и  $y$ . Где функции  $C_n^k = \frac{n!}{k!(n-k)!}$ ,  $A_n^k = \frac{n!}{(n-k)!}$ ,  $P_n = n!$ .

20) Найти значение выражения  $\frac{3 \cdot 5! - 7 \cdot 4!}{5! + 6!}$ , где  $n!$  означает факториал числа  $n$  ( $n! = 1 \cdot 2 \cdot \dots \cdot n$ ). (Определить функцию для расчета факториала натурального числа.)

21) Вычислить при заданном  $x$ :  $7C_{x-1}^{x+2} - 6A_{x+3}^{x-2}$ , где функции  $C_n^k = \frac{n!}{k!(n-k)!}$  и  $A_n^k = \frac{n!}{(n-k)!}$ .

22) В заданной матрице из отрицательных целых чисел поменять местами первую строку со строкой, содержащей максимальный элемент матрицы. Нахождение максимального элемента оформить в виде функции.



23) В заданной матрице из вещественных чисел поменять местами последний столбец и столбец, содержащий минимальный по абсолютной величине элемент матрицы. Нахождение минимального элемента оформить в виде функции.

24) В заданной матрице из целых чисел поменять местами столбец, содержащий максимальный отрицательный элемент матрицы, и столбец, содержащий минимальный положительный элемент матрицы. Нахождение минимального и максимального элемента оформить в виде функций.

25) В заданной матрице из целых чисел поменять местами минимальный элемент главной диагонали и максимальный элемент побочной диагонали. Нахождение минимального и максимального элемента оформить в виде функций.

26) Задана матрица  $A(4, 4)$ . Найти строку с наименьшей и наибольшей суммой элементов. Суммы вычисляются в отдельной функции.

27) Вычислить количество элементов каждого столбца, делящихся без остатка на 2 для матрицы  $A(5, 5)$ .

28) Задана матрица  $A(6, 4)$ . Вычислить сумму элементов заданных столбцов с помощью функции.

29) Описать функцию  $\text{MinElem}(A, N)$  целого типа, находящую минимальный элемент целочисленного массива  $A$  размера  $N$ . С помощью этой функции найти минимальные элементы массивов  $A, B, C$  размера  $NA, NB, NC$  соответственно.

30) Описать функцию  $\text{SumRow}(A, M, N, K)$  вещественного типа, вычисляющую сумму элементов вещественной матрицы  $A$  размера  $M \times N$ , расположенных в  $K$ -й строке.

31) Найти наибольший общий делитель трех натуральных чисел, имея в виду, что  $\text{НОД}(a, b, c) = \text{НОД}(\text{НОД}(a, b), c)$ . (Определить функцию для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида.)

32) Найти все трехзначные простые числа. (Определить функцию, позволяющую распознавать простые числа.)

33) Два простых числа называются "близнецами", если они отличаются друг от друга на 2 (таковы, например, числа 41 и 43). Напечатать все пары чисел "близнецов", не превышающих число 200. (Определить функцию, позволяющую распознавать простые числа.)

34) Даны два натуральных числа. Выяснить, в каком из них сумма цифр больше. (Определить функцию для расчета суммы цифр натурального числа.)

35) Даны натуральные числа  $a$  и  $b$ . Найти их наименьшее общее кратное. (Определить функцию для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида.)

36) Даны натуральные числа  $a$  и  $b$ , обозначающие соответственно числитель и знаменатель дроби. Сократить дробь, т. е. найти такие натуральные числа  $p$  и  $q$ , не имеющие общих делителей, что  $p/q = a/b$ . (Определить функцию для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида.)

37) Вводится последовательность из  $N$  целых чисел, найти среднее арифметическое совершенных чисел последовательности. Число называется совершенным, если сумма всех делителей, меньших его самого равна этому числу.

38) Вводится последовательность из  $N$  целых чисел, найти среднее геометрическое простых чисел последовательности. (Определить функцию, позволяющую распознавать простые числа.)

39) Найти значение  $x$ , при котором функция  $F(x) = 4(x - 1)(x - 3) = 4(x^2 - 4x + 3)$  принимает минимальное значение на интервале от  $a$  до  $b$ .

40) Найти наибольшее значение функции  $F(x) = x^2 + 4x + 8$  на интервале от  $a$  до  $b$ . (Числа  $a$ ,  $b$  вводятся с клавиатуры).

41) Пусть задана функция  $y = \frac{x^2}{a} + \frac{1}{b}x + c$  на интервале  $x \in [a_1; a_2]$ . Найти максимальное значение функции и минимальное значение функции  $y$  на заданном интервале  $[a_1; a_2]$ .

42) Найти наибольшее и наименьшее значения функции  $f(x) = 3x - x^3$  на отрезке  $[-2; 3]$ .

43) Найти наибольшее и наименьшее значения функции  $f(x) = x^4 - 2x^2 + 3$  на отрезке  $[-3; 2]$ .

44) Найти точку на отрезке  $[-5; 5]$ , в которой функция  $y = (x-1)^4$  имеет минимальное значение.

45) Найти точку на отрезке  $[-1; 1]$ , в которой функция  $y = x\sqrt{1-x^2}$  имеет максимальное значение.

46) Найти точки экстремума функции (в которых функция принимает наибольшее или наименьшее значения)  $f(x) = \frac{(x-2)^2}{x^2}$  на отрезке  $[1; 9]$ .

47) Заданы два числа —  $A$  и  $B$ , первое в системе счисления с основанием  $p=4$ , второе в системе счисления с основанием  $q=16$ . Вычислить значение  $C = 2 \cdot (A^2 + B^2)$  и вывести его на экран в десятичной системе счисления.

48) Заданы два числа —  $A$  и  $B$ , первое в системе счисления с основанием  $p=3$ , второе в системе счисления с основанием  $q=6$ . Вычислить значение  $C = 2 \cdot B^2 \cdot (A+B)$  и вывести его на экран в десятичной системе счисления.

49) Заданы два числа —  $A$  и  $B$ , первое в системе счисления с основанием  $p=12$ , второе в системе счисления с основанием  $q=9$ . Вычислить значение  $C = 3 \cdot A^2 \cdot (A-B)$  и вывести его на экран в десятичной системе счисления.

50) Заданы два числа —  $A$  и  $B$ , первое в системе счисления с основанием  $p=5$ , второе в системе счисления с основанием  $q=7$ . Вычислить значение  $C = (A-B)^2 + 3 \cdot A$  и вывести его на экран в десятичной системе счисления.

51) Заданы два числа —  $A$  и  $B$ , первое в системе счисления с основанием  $p=8$ , второе в системе счисления с основанием  $q=6$ . Вычислить значение  $C = A^2 + A \cdot B$  и вывести его на экран в десятичной системе счисления.

52) Заданы два числа —  $A$  и  $B$ , первое в системе счисления с основанием  $p=3$ , второе в системе счисления с основанием  $q=11$ . Вычислить значение  $C = (5 \cdot B - 2 \cdot A)^2$  и вывести его на экран в десятичной системе счисления.

53) Заданы два числа —  $A$  и  $B$ , первое в системе счисления с основанием  $p=16$ , второе в системе счисления с основанием  $q=8$ . Вычислить значение  $C = (B - A)^2 + 2 \cdot A$  и вывести его на экран в десятичной системе счисления.

54) Заданы два числа —  $A$  и  $B$ , первое в системе счисления с основанием  $p=7$ , второе в системе счисления с основанием  $q=14$ . Вычислить значение  $C = A^2 + 2 \cdot A + B^2$  и вывести его на экран в десятичной системе счисления.

55) Чему равна сумма чисел  $57_8$  и  $46_{16}$ ? Ответ записать в двоичной системе счисления.

56) Даны 4 числа, они записаны с использованием различных систем счисления. Указать среди этих чисел те, в двоичной записи которых содержится ровно 6 единиц.

57) Написать функцию определения в одномерном массиве максимального числа. Определить, встречается ли заданное число  $x$  среди максимальных элементов столбцов матрицы  $A(n, m)$ .

58) Задана матрица  $A(n, n)$ . Найти столбец с наименьшим и наибольшим произведением элементов. Произведения вычисляются в отдельной функции.

59) Вычислить сумму и количество элементов каждого столбца, делящихся без остатка на 5 для матрицы  $A(5, 5)$ .

60) В заданной матрице из вещественных чисел поменять местами первый столбец и столбец, содержащий минимальный элемент матрицы, кратный трем. Нахождение минимального элемента оформить в виде функции.

61) Найти среднее арифметическое четных положительных элементов каждой строки матрицы  $A(6,6)$ .

62) Написать функцию определения в одномерном массиве самого часто встречающегося элемента. В матрице  $A(n,m)$  найти количество столбцов, содержащих такой элемент.

63) Построить функцию для сравнения двух числовых массивов: если они равны, то вернуть 1, а если не равны – вернуть 0.

64) Построить функцию  $\text{swar}(x,y)$  для обмена значениями между переменными. Поменять значения соседних элементов в массиве.

## Глава 6. Работа с символами и строками

### 6.1. Организация работы со строковыми объектами в JS

- Строка — это упорядоченная последовательность символов
- Пустая строка — это тоже строка (последовательность нуля символов)
- Обозначается одинарными или двойными кавычками

Создание строки с константой:

```
const str1 = "Hello";
```

```
const str2 = 'Hello';
```

Возможно включить кавычку одного типа внутрь строки, окружив её кавычками другого типа:

```
const str1 = "They call him "Harry", and he likes it";
```

```
const str2 = "They call him 'Harry', and he likes it";
```

Если в строке используются кавычки того же типа, они должны быть экранированы с помощью обратного слеша \:

```
const str1 = "They call her \'Ann\', and she likes it";
```

```
const str2 = "They call her \"Ann\", and she likes it";
```

Если строка включает обратный слеш (именно как символ, который хочется иметь в строке), он должен быть экранирован другим обратным слешем:

```
const str = "This is a backslash \\ here"
```

```
// This is a backslash \ here
```

Так же существуют управляющие символы — специальные комбинации, которые генерируют невидимые детали:

```
const str = "There is a tab \t and here \ncomes the new line!"  
  
// Here is a tab   and here  
  
// comes the new line!
```

Таблица специальных символов не отличается от таблиц в других языках программирования:

<code>\n</code>	перевод строки (на новую строку)
<code>\r</code>	возврат каретки (на первый символ строки)
<code>\f</code>	перевод страницы
<code>\t</code>	знак табуляции (по умолчанию. Перевод на расстояние 8 символов)
<code>\v</code>	знак вертикальной табуляции
<code>\'</code>	апостроф
<code>\"</code>	кавычка
<code>\\</code>	обратная косая черта (обратный слеш)
<code>\uNNNN</code>	символ с номером NNNN в кодировке Unicode

### Конкатенация строк

Строки могут склеиваться друг с другом. Такой процесс называется конкатенацией и задаётся символом `+`:

```
const name = "Alex";  
  
const age = 22;  
  
console.log("His name is " + name + " and his age is " + age);  
  
// His name is Alex and his age is 22
```

Строки будут склеены в том порядке, в котором они указаны: "mos" + "cow" → "moscow", а "cow" + "mos" → "cowmos"

Второй вариант – использование метода `concat()`.

### Доступ к индивидуальным символам

`str[i]` это *i*-ый символ строки `str`, начинающейся с 0. Например, `"elets"[0]` это `e`, а `"elets"[3]` это `t`.

Функция, которая принимает строку и возвращает копию этой строки без каждой второй буквы. Например, `"elets"` становится `"ees"`.

```
const skip = (str) => {  
  let i = 0;  
  let result = "";  
  while (i < str.length) {  
    result = result + str[i];  
    i = i + 2;  
  }  
  return result;  
}
```

`str.length` это длина `str`, то есть количество символов. Это просто количество, поэтому мы не начинаем отсчёт от 0. Например, `"food".length` это 4.

### Свойства строк:

#### 1) Неизменяемость

В JavaScript строки являются неизменяемыми, так же говорят "immutable". Это означает, что какие бы вы к ним не применяли функции, они не производят in-place замены (то есть не производят изменения самой строки). Любые



строковые функции, примененные к строкам, возвращают новую строку. Это верно и в том случае, когда мы обращаемся к конкретному символу в строке.

Пример:

```
const str = 'hello';  
  
str.toUpperCase(); // HELLO  
  
console.log(str); // hello  
  
str[0].toUpperCase(); // H  
  
console.log(str); // hello  
  
str[0] = 'W';  
  
console.log(str); // hello
```

Таким образом, к символу строки можно обратиться как к элементу массива, но изменить символ по индексу нельзя.

## 2) Лексикографический порядок

Лексикографический порядок — это, другими словами, алфавитный порядок. Такой порядок используется в словарях, телефонных справочниках, записных книжках и так далее.

В JavaScript вы можете сравнивать строки с помощью знаков  $>$  и  $<$ , и сравнение будет происходить именно лексикографически.

Сравнение зависит от расположения символа в кодировочной таблице. Т.е. сначала идут цифры (как символы), затем прописные буквы, а потом — строчные. Например, '8' это не число, а строка.

## 3) Интерполяция

Кроме одиночных " и двойных кавычек "", современный JavaScript содержит обратные тики (backticks):

..

С обратными тиками вы можете использовать интерполяцию, вместо конкатенации. Например:

```
const name = "Alex";  
const a = 10;  
const b = 12;  
console.log(`His name was ${name} and his age was ${a + b}`);
```

Такой код выведет на экран His name was Alex and his age was 22. Внутри `${}` вы можете поместить любое выражение.

Интерполяция предпочтительнее конкатенации. Мы советуем не использовать конкатенацию вообще. Вот некоторые из причин:

- Такой код заставляет больше думать, потому что синтаксически + больше похож на сложение.
- Из-за слабой типизации можно легко получить не тот результат. Конкатенация может породить ошибки.
- Сложные строки при использовании конкатенации невозможно нормально разобрать в голове и понять, как они устроены.

## 6.2. Класс String

Создать строку можно с помощью класса String в следующей форме:

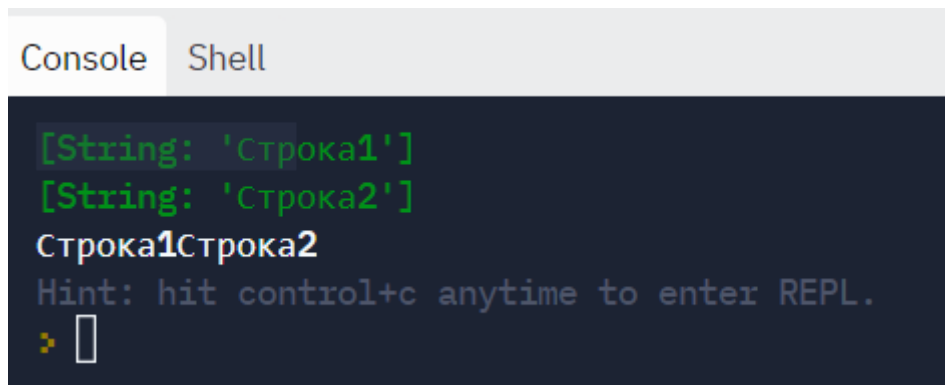
```
let <Объект> = new String(<Строка>)
```

Рассмотрим пример с созданием и выводом строк:

```
let str1=new String('Строка1')  
let str2=new String("Строка2")  
console.log(str1)
```

```
console.log(str2)
```

```
console.log(str1+str2)
```



The screenshot shows a REPL window with two tabs: 'Console' and 'Shell'. The 'Console' tab is active. The output is as follows:

```
[String: 'Строка1']  
[String: 'Строка2']  
Строка1Строка2  
Hint: hit control+c anytime to enter REPL.  
: 
```

## Методы класса String

1) Свойство `length` – содержит длину строки (как с массивами). Символы в строке тоже нумеруются с нуля.

Например:

```
let str1=new String("Строка1")
```

```
console.log(str1.length)
```

```
//=>7
```

2) Метод `toString()` – преобразует объект в строку. Применяется при выводе числовых значений внутри строкового выражения.

Например:

```
let x=parseInt(prompt())
```

```
let y=parseInt(prompt())
```

```
console.log(x+y)
```

```
console.log(x.toString()+y.toString())
```

Результат:

```
Console Shell
undefined> 3
undefined> 5
8
35
Hint: hit control+c anytime to enter REPL.
> █
```

В первом случае выведется сумма двух числовых значений, а во втором произойдет конкатенация двух строк '3' и '5'.

3) Метод `charAt(N)` – возвращает символ строки с указанным номером. Нумерация начинается с нуля. Например:

```
let s='Строка'
console.log(s[0])
console.log(s.charAt(0))
```

В обоих способах записи результат получится одинаковым:

```
Console Shell
С
С
Hint: hit control+c anytime to enter REPL.
> █
```

4) Противоположный предыдущему метод `charCodeAt(N)` – возвращает код символа строки с указанным номером в кодировке Unicode (если не указана кодировка в настройках веб-страницы). Пример.

```
...
console.log(s.charCodeAt(0))
```

//=>1057 – код буквы С

5) Следующие два метода используются для изменения регистра символов:

`toLowerCase()` – преобразует символы в нижний регистр (строчные буквы);

`toUpperCase()` – преобразует символы в верхний регистр (прописные буквы).

Пример:

```
let s='JavaScript'
```

```
console.log(s.toLowerCase()) // javascript
```

```
console.log(s.toUpperCase()) // JAVASCRIPT
```

Методы извлечения подстроки из строки:

6) Метод `substring(N,K)` – извлекает фрагмент из заданной строки, начиная с символа с номером N и заканчивая символом с номером K-1.

Например:

```
let s='JavaScript'
```

```
console.log(s.substring(0,4)) // Java
```

```
console.log(s.substring(4)) // Script
```

В первом случае результатом станет вывод символов с нулевого по третий.

Во втором случае начальное значение можно не ставить, тогда выведутся символы, начиная с заданного K (в примере K=4) и до конца строки.

7) Похожий метод `slice(N,K)` также извлекает фрагмент строки, заданный начальным и конечным индексами.

Различие между методами состоит в использовании отрицательных индексов. Метод `substring()` заменяет отрицательное значение нулем, а метод `slice()` вычитает это значение из длины строки.

Пример:

```
let s='JavaScript'  
  
console.log(s.substring(-2)) // JavaScript  
console.log(s.substring(-2,4)) // Java  
console.log(s.slice(0,-2)) // JavaScri  
console.log(s.slice(-2)) //pt
```

8) Метод `substr(N, D)` – извлекает фрагмент строки, начиная с символа с номером `N`, заданной длины `D`. Если второй параметр пропущен, то символы извлекаются до конца строки.

Пример:

```
let s='JavaScript'  
  
console.log(s.substr(-2)) // pt  
console.log(s.substr(0,4)) // Java  
console.log(s.substr(4)) // Script
```

Методы поиска и замены символов в строке:

9) Метод `indexOf(uns,N)` – возвращает номер позиции первого вхождения подстроки `uns` в текущей строке, начиная с позиции поиска `N`. Если подстрока не найдена, то метод возвращает `-1`. Если второй параметр не задан, то поиск начинается с начала строки.

10) Метод `lastIndexOf(uns,N)` возвращает номер последнего вхождения подстроки `uns` в строку. Поиск ведется от конца к началу. Если подстрока не найдена, то метод возвращает `-1`.

Пример:

```
let s='JavaScriptJavaScript'  
console.log(s.indexOf("Java")) // 0  
console.log(s.indexOf("Java",2)) // 10  
console.log(s.lastIndexOf("Java")) // 10  
console.log(s.lastIndexOf("Java",2)) // 0  
console.log(s.lastIndexOf("Yawa")) // -1
```

11) Методы, возвращающие логические значения, если строка начинается с заданной подстроки (`startsWith(uns, N)`) или заканчивается подстрокой (`endsWith(uns, N)`).

12) Методы удаления пробельных символов:

`trim()` – слева и справа;  
`trimLeft()` – в начале строки;  
`trimRight()` – в конце строки.

Пример:

```
s=' JavaScript '  
console.log(s.startsWith('Java'))  
s=s.trimLeft()  
console.log(s.startsWith('Java'))
```

```
Console Shell
false
true
Hint: hit control+c anytime to enter REPL.
> █
```

В первом случае строка начинается с пробела, поэтому результат метода `startsWith()` равен `false`, а во втором случае убрали первый пробел (слева). Теперь результат предыдущего метода равен `true`.

### 6.3. Регулярные выражения

Регулярные выражения предназначены для сложного поиска и замены в строках. Создать шаблон для регулярного выражения можно двумя способами – как параметры строки со знаками слеш /, так и как экземпляр класса `RegExp`:

Имя шаблона = / регулярное выражение/ параметры;

Имя шаблона = `new RegExp(“регулярное выражение”, Параметры)`.

В обоих случаях параметры использовать необязательно, а их наличие задает дополнительные возможности поиска:

`i` – поиск без учета регистра;

`g` – глобальный поиск всех вхождений регулярного выражения в строку;

`m` – многострочный режим, при котором символом `^` выделяется начало подстроки, а символом `$` - конец подстроки.

Внутри регулярного выражения используются метасимволы, которые управляют проверкой строк. Если необходимо указать несколько символов. То они заключаются в квадратные скобки `[]`.

Примеры метасимволов:

`[09]` – соответствует числу 0 или 9;



[0-9] – соответствует диапазону, т.е. любому числу от 0 до 9;

[^0-9] – любой символ, кроме цифр от 0 до 9;

[аб] – соответствует буквам а и б;

[а-я] – соответствует буквам от а до я без буквы ё;

[а-яё] – соответствует любой букве от а до я;

Аналогично для прописных букв.

[0-9а-яА-ЯёЁа-zA-Z] – любая цифра или буква русского или латинского алфавита.

Для работы с регулярными выражениями в классе String используются следующие методы:

1) search(R) – возвращает индекс позиции первого вхождения регулярного выражения R в качестве подстроки. Если совпадений нет, то метод возвращает -1.

2) match(R) – возвращает массив с результатами поиска, совпадающими с регулярным выражением R. Если совпадений нет, то возвращается null.

3) replace(R, T) – возвращает строку, в которой происходит поиск и замена подстроки в исходной строке на строку T в соответствии с регулярным выражением R.

Пример. Найдем и заменим в строке все заглавные буквы латинского алфавита:

```
s=' JavaScript '  
let R=/[A-Z]/  
console.log(s.search(R)) //=> 1  
let R1=new RegExp("[A-Z]","g")  
console.log(s.search(R1)) //=> 1
```

```
console.log(s.match(R)) //=> J
console.log(s.match(R1)) //=> J, S
let str1=s.replace(R,"Y")
let str2=s.replace(R1,"Y")
console.log(str1) //=> YavaScript
console.log(str2) //=> YavaYcript
```

Результат в консоли:



```
Console Shell
1
1
[ 'J', index: 1, input: ' JavaScript ', groups: undefined ]
[ 'J', 'S' ]
YavaScript
YavaYcript
Hint: hit control+c anytime to enter REPL.
> 
```

В этом примере регулярное выражение R создано первым способом без учета параметров, а регулярное выражение R1 – вторым способом с параметром глобального поиска. Первый метод search() выдал одинаковые результаты, т.к. он рассчитывает только первые вхождения совпадений.

Регулярные выражения удобно использовать в веб-приложениях:

- при создании онлайн-версий текстовых редакторов;
- для парсинга (поиска веб-страниц по заданному шаблону)

## 6.4. Работа с числами и массивами с помощью строк

Многие задачи при работе с разбиением многозначного числа на отдельные цифры, вводом элементов числового массива в одну строку и т.д., удобно решать через символьные методы.

Для преобразования строки в массив используется метод класса `String` – `split(D, M)`.

Этот метод возвращает массив, полученный в результате разбиения строки на подстроки по разделителю `D`. Второй параметр `M` указывает максимальное количество элементов в полученном массиве. Если первый параметр равен пустой строке, т.е. `D=""`, то каждый элемент массива будет содержать по одному символу. Если второй параметр отсутствует, то в разбиении участвуют все символы строки.

Для обратного преобразования массива в строку используется метод класса `Array` – `join(D)`.

Этот метод соединяет все элементы массива в одну строку с помощью разделителя `D`. Если параметр `D` не указан, то в качестве разделителя используется запятая.

Например:

```
let arr=[1,2,3,4,5]
```

```
let str=arr.join()
```

```
console.log(str)
```

```
let arr2=str.split(",")
```

```
console.log(arr2)
```

```
let sum=0
```

```
for (i of arr2)
```

```
sum+=i
```

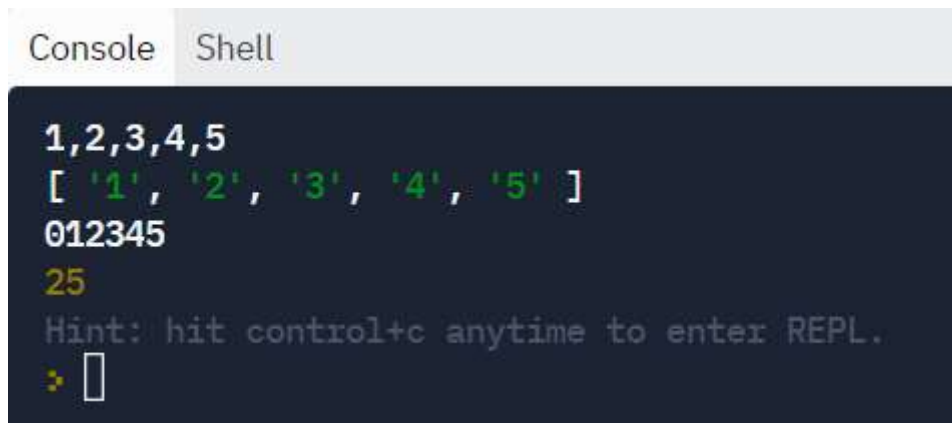
```
console.log(sum)

let sum1=0

for (j of arr2)

sum1+=parseInt(i)

console.log(sum1)
```



```
Console Shell
1,2,3,4,5
[ '1', '2', '3', '4', '5' ]
012345
25
Hint: hit control+c anytime to enter REPL.
> 
```

В этом примере из массива arr создали строку. Затем обратно из этой строки создали массив arr2. По умолчанию элементы этого массива – символы, поэтому в объекте sum хранится конкатенация (соединение) символов в строку. Посимвольное преобразование в число даст в итоге числовой результат суммы sum1.

## 6.5. Скрипты, содержащие символы и строки

Задача №1. Подсчитать количество запятых в заданной строке.

Вариант 1 (классический алгоритм):

```
var s=prompt()

var count=0

for (i=0;i<s.length;i++)

{

    if (s[i]==",")

        count++

}
```

```
}  
console.log(count)
```

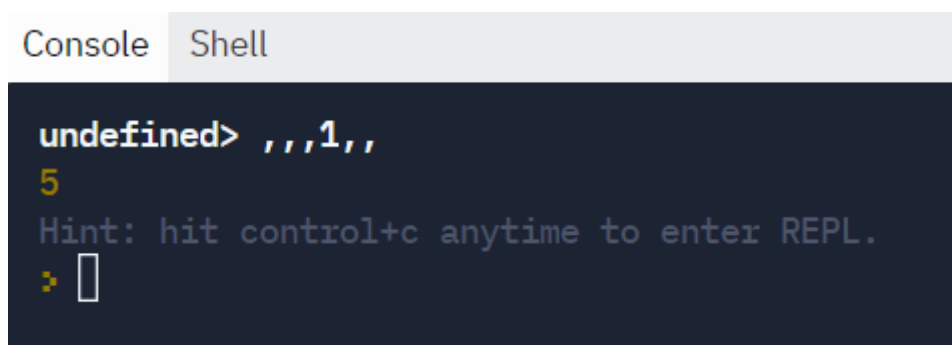
Вариант 2 (в цикле for..of):

```
let s=prompt()  
let count=0  
for (i of s)  
{  
  if (i=="")  
    count++  
}  
console.log(count)
```

Вариант 3 (через регулярное выражение):

```
let s=prompt()  
let R=/[,]/g  
let count=(s.match(R))  
console.log(count.length)
```

Результат:

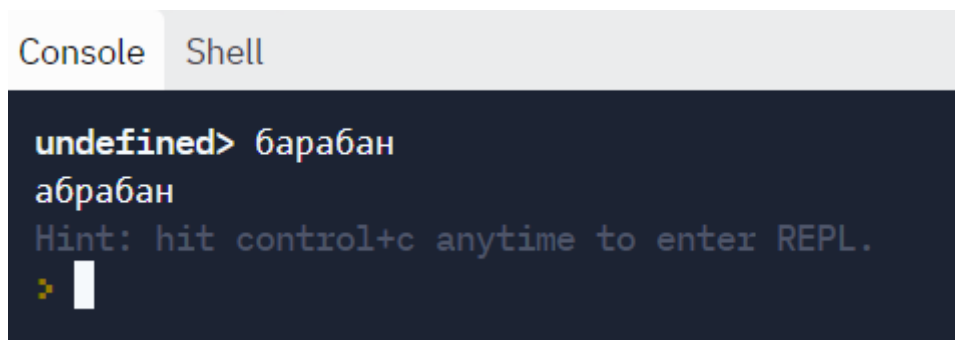


The screenshot shows a terminal window with two tabs: 'Console' and 'Shell'. The 'Console' tab is active. The terminal output is as follows:  
undefined> ,,,1,,  
5  
Hint: hit control+c anytime to enter REPL.  
> |

Задача №2. В заданном тексте везде букву "а" заменить на букву "б", а букву "б" — на букву "а".

```
var s=prompt()
s=s.replace("а", "*")
s=s.replace("б", "а")
s=s.replace("*", "б")
console.log(s)
```

Результат:



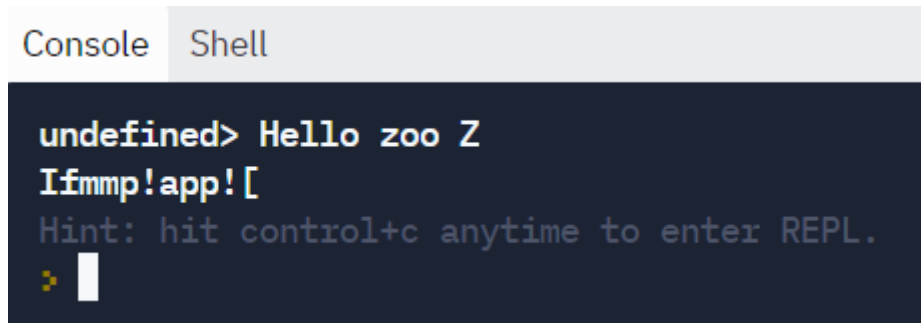
```
Console Shell
undefined> барабан
абрабан
Hint: hit control+c anytime to enter REPL.
> |
```

Задача №3. Дана строка из букв латинского текста. Составить программу шифровки текста, пользуясь правилом: каждая буква заменяется на следующую по алфавиту, при этом буква 'z' заменяется на букву 'a'.

```
let s=prompt()
let n=s.length
let s1=""
for (i=0;i<=n;i++)
{
  k=s.charCodeAt(i)+1
  s1+=String.fromCharCode(k)
  s1=s1.replace("{","a")
}
```

```
console.log(s1)
```

Результат:



```
Console Shell
undefined> Hello zoo Z
Ifmmp!app![
Hint: hit control+c anytime to enter REPL.
> |
```

Пояснения к коду:

В этом скрипте есть недочеты: пробелы заменяются на следующий по коду символ «!» и не учитывается регистр символа «z».

Исправим эти недочеты:

```
let s=prompt()
let n=s.length
let s1=""
for (i=0;i<=n;i++)
{
  k=s.charCodeAt(i)+1
  s1+=String.fromCharCode(k)
  s1=s1.replace(" ","a")
  s1=s1.replace("!"," ")
  s1=s1.replace("[","A")
}
console.log(s1)
```

```
Console Shell
undefined> Hello zoo Z
Ifmmp app A
Hint: hit control+c anytime to enter REPL.
> []
```

Задача №4. Из слова «килобайт» путем "вырезок" и "склеек" его букв получить слова: «байт», «лик», «кот», «блок», «колба», «бита».

```
s='килобайт';
console.log(s.substr(4, 4));
console.log(s[2]+s[1]+s[0]);
console.log(s[2]+s[3]+s[7]);
console.log(s[4]+s.substr(2,2)+s[0]);
console.log(s[0]+s[3]+s[2]+s.substr(4,2));
console.log(s[4]+s[1]+s[7]+s[5]);
```

Задача №5. Игра «Угадай слово, заданное компьютером».

```
const words = ['информатика','программа','образование','занятие','технология'];
// выбираем случайное слово
const word = words[Math.floor(Math.random()*words.length)];
// создаем итоговый массив и заполняем неотгаданные буквы прочерками
let answerArray = [];
for (let i = 0; i < word.length; i++){
    answerArray[i] = "_";
}
```



```

// Задаем количество букв, которое надо угадать
let remainigLetters = word.length;
// Создаем доп переменную
let newRemainigLetters = remainigLetters;
//Создадим переменную с количеством жизней
let lives = 5;
// Основной цикл игры
while ((remainigLetters > 0) && (lives !== 0)){
    // Показываем состояние игры
    alert("Попытки: " + lives + "\n" + answerArray.join(" ") + "\n" + remainig-
Letters + " осталось букв");
    // Запрашиваем букву
    let guess = prompt('Угадайте букву, или нажмите кнопку "Отмена" для
выхода из игры');
    // Проверяем подходят ли введенные данные
    if (guess === null){
        break;
    } else if (guess.length !== 1) {
        alert('Введите одиночную букву!')
    } else {
guess = guess.toLowerCase();
        for (let j = 0; j < word.length; j++) {
            // Проверяем есть ли буква среди уже угаданных
            if (answerArray[j] === guess) {
                alert("Буква уже названа")
                break;

```

```

    }
    // Проверяем угадали ли букву
    if(word[j] === guess){
        answerArray[j] = guess;
        remainigLetters--;
    }
}
}

// Проверяем, уменьшилось ли количество оставшихся букв, которые
надо отгадать

if (remainigLetters === newRemainigLetters) {
    lives--;
}

// Обновляем доп переменную с количеством оставшихся букв
newRemainigLetters = remainigLetters;
}

// развязка после цикла
if (lives === 0) {
    alert("Не угадали! Загадано слово: " + word);
} else if(remainigLetters === 0){
    alert("Угадали! Это слово " + word);
} else {
    alert("Пока!");
}

Результат:

```

```
Попытки: 4
т е х н о л о г _ _
2 осталось букв
Угадайте букву, или нажмите кнопку "Отмена" для выхода из и
гры> г
Буква уже названа
Попытки: 3
т е х н о л о г _ _
2 осталось букв
Угадайте букву, или нажмите кнопку "Отмена" для выхода из и
гры> и
Попытки: 3
т е х н о л о г и _
1 осталось букв
Угадайте букву, или нажмите кнопку "Отмена" для выхода из и
гры> я
Угадали! Это слово технология
Hint: hit control+c anytime to enter REPL.
```

## 6.6. Контрольные вопросы и тесты

### Дополнительные вопросы

1. Можно ли сформировать строку путем соединения символа и числа?
2. Чему равна длина пустой строки?
3. Чем отличаются методы `substr()`, `substring()`, `slice()`?
4. Какой метод позволяет заменить подстроку в строке в соответствии с шаблоном поиска?
5. В каких случаях удобнее работать с числами через преобразование их к строковому типу?
6. Есть ли в JavaScript ограничение на длину строки?
7. Можно ли изменить символ в строке `s` путем прямого присваивания `s[i]="..."`?

8. Как сохранять изменения в строке после выполнения различных методов?
9. Как называются символы, используемые в шаблоне регулярных выражений?
10. Что такое «возврат каретки»?

### Тесты с выбором варианта ответа

1. Символьные величины заключаются в:

- а) кавычки
- б) фигурные скобки
- в) круглые скобки
- г) апострофы

2. Результат выполнения программы:

```
s1="паро";s2="воз"; s3="";
```

```
s3=s3.concat(s1,s2);
```

```
console.log(s3);
```

- а) пар и воз
- б) возпаро
- в) зоворап
- г) паровоз

3. Перевод символов в нижний регистр:

- а) islower(c);
- б) isupper(c);
- в) tolower(c);
- г) s.toLowerCase()

4. Строковые величины заключаются в:

- а) кавычки
- б) фигурные скобки
- в) круглые скобки
- г) апострофы

5. Результат выполнения программы:

```
s1="паро";s2="ход"; s3="";
```



а) [1..4] of 20/;

б) /\*1 || 4/;

в) /200[1,4]/;

г) /20[14]/.

12. Номер позиции первого вхождения подстроки в строке определяется методом:


а) match();

б) search();

в) indexOf();

г) lastIndexOf().

### Тесты без выбора варианта ответа

1. Что нужно вставить в команду `console.log("\uABCD")` вместо букв А, В, С, D, чтобы получить символ  («ножницы»)?

2. Какие слова образуются при выполнении следующего скрипта:

```
let s='вертикаль'
```

```
console.log(s.substr(3,3))
```

```
console.log(s.substring(6)+s[3])
```

```
console.log(s.slice(0,3)+s[6])
```

3. Каков результат выполнения этой функции?

```
let p=/([0-9]+)/g
```

```
let s='a-1 b-2 c-3 d-4'
```

```
let str2=s.replace(p, function(s)
```

```
{  
  let n=parseInt(s,10)  
  n+=10  
  return n+""  
})  
console.log(str2)
```

Переделайте скрипт так, чтобы для поиска чисел не использовать регулярное выражение.

4. Дан скрипт, позволяющий удалить букву «q» из строки:

```
let s = prompt("Введите строку") || ""  
let new_s = ""  
for(let i = 0;i < s.length;i++)  
  if(!(s[i] == 'q'))  
    new_s = new_s + s[i]  
console.log("Результат: ", new_s)
```

Переписать его, используя метод `replace()` с регулярным выражением.

5. Что будет выведено в результате работы скрипта:

```
let s = 'a\tb\tc\td'  
let arr=s.split('\t',3)  
console.log(arr)
```

Что получится в результате – массив или строка?

## 6.7. Задачи для самостоятельной работы

- 1) Написать программу, определяющую, какая из букв первая или последняя встречаются в заданном слове чаще.
- 2) Вводится строка, состоящая из слов, разделенных пробелами. Требуется посчитать количество слов в ней.
- 3) Дано предложение. Определить долю (в %) букв «а» в нем.
- 4) Дано предложение. Определить число вхождений в него буквосочетания «ро».
- 5) Дан текст. Подсчитать общее число вхождений в него символов "+" и "-".
- 6) Дано предложение. Определить, сколько в нем гласных букв.
- 7) Дан текст. Верно ли, что в нем есть N идущих подряд одинаковых символов?
- 8) Дано предложение, в котором имеются буквы «С» и «Т». Определить, какая из них встречается позже (при просмотре слова слева направо).
- 9) Дано предложение. Определить, есть ли буква «а» в нем.
- 10) Дан текст из нескольких предложений. Определить количество букв «и» в первом предложении.
- 11) Дано слово. Если его длина нечетная, то удалить среднюю букву, в противном случае — две средних буквы.
- 12) Дано предложение. Удалить из него все буквы «у», стоящие на нечетных местах.
- 13) Дано предложение. Удалить из него все согласные буквы.
- 14) Дан текст. Все гласные буквы в нем заменить символом «\_».
- 15) Поменять местами первую и последнюю строки текста.
- 16) Дан текст. Поставить перед каждым восклицательным знаком вопросительный.



17) Удалить из теста все восклицательные и вопросительные знаки.

18) Добавить в начало каждой строки текста ее номер и пробел.

19) Заменить в строке все цифры на пробел. Вывести количество замен.

20) Удалить из строки все слова, начинающиеся на букву «я».

21) В записке слова зашифрованы — каждое из них записано наоборот. Расшифровать сообщение.

22) Дана строка-предложение. Зашифровать ее, поместив вначале все символы, расположенные на четных местах, а затем, в обратном порядке, все символы, расположенные на нечетных местах.

23) Вводится текст, заканчивающийся точкой. Зашифровать его следующим образом: заменить каждую английскую букву в английском алфавите, стоящую на 2 буквы ранее. Другие символы оставить без изменений.

24) Написать программу, которая кодирует английский текст шифром Цезаря. Шифр Цезаря реализует кодирование фразы путем “сдвига” вперед всех букв фразы на определенное число  $k$ .

25) Написать программу шифровки 4-буквенного однословного сообщения. Для получения 4 букв нужно ввести 3 строки: из первой строки прочесть только первую букву; из второй строки прочесть только первую букву; из третьей строки прочесть первую и вторую буквы.

26) Составить программу шифрования текстового сообщения. Пользователь задает ключ шифровки - целое число, которое определяет величину смещения назад букв русского алфавита. Например, ключ =3, тогда в тексте буква “Г” заменяется на “а” и т.д. Используются все буквы русского алфавита.

27) Зашифровать текст методом Атбаш. Правило шифрования состоит в замене  $i$ -й буквы алфавита буквой с номером  $n - i + 1$ , где  $n$  - число букв в алфавите. (Первая буква меняется с последней, вторая с предпоследней и т.д.)

28) Зашифровать текст: удалить пробелы и поменять местами каждые две буквы.

29) Написать программу, которая кодирует английский текст путем “сдвига” вперед или назад всех букв фразы на псевдослучайное целое число.

30) Написать программу, которая кодирует русский текст путем “сдвига” вперед или назад гласных букв на  $k=2$ , а согласных букв на  $k=3$ .

31) Посчитать количество строчных (маленьких) и прописных (больших) букв в введенной строке. Учитывать только английские буквы.

32) Дана строка - предложение на английском языке. Преобразовать строку так, чтобы каждое слово начиналось с заглавной буквы и заканчивалось строчной.

33) Дан текст. Определите процентное отношение строчных и прописных букв к общему числу символов в нем.

34) Пользователь нечаянно нажал кнопку переключения регистра и перепутал строчные и прописные буквы. Восстановить правильный вариант, поменяв регистр каждой буквы.

35) Дана строка - предложение на русском языке. Преобразовать строку так, чтобы каждое слово начиналось с заглавной буквы, а остальные буквы были строчными.

36) Ввести строку из прописных и строчных букв. Если длина такой строки четна, то перевести ее к нижнему регистру, а если нечетна, то к верхнему.

37) Ввести строку из прописных и строчных букв. Вывести для каждой буквы результат – строчная буква или прописная.

38) Ввести строку из прописных и строчных букв. Прописные буквы удалить, а строчные перевести в верхний регистр.

39) Дан текст из нескольких предложений. Заменить точки (кроме последней) на запятые, а прописные буквы в начале предложений – на строчные.

40) Дан текст, состоящий из предложений, разделяемых точками. Написать программу, производящую следующее форматирование: после каждой точки в

конец предложения должен стоять хотя бы один пробел; первое слово в предложении должно начинаться с прописной буквы.

41) С помощью датчика случайных чисел заполнить массив  $\text{Sim}[0..10]$  строчными английскими буквами. Затем массив отсортировать в алфавитном порядке.

42) Рассматривая строку как массив символов, в тексте длины не более 255 знаков, если  $i$ -ый символ является пробелом, а  $(i+1)$ -ый точкой, то поменять их местами.

43) Дан массив символов – русских букв  $X(33)$ . Сформировать из него массив  $Y(33)$ , в котором первыми располагаются буквы, стоящие на четных местах массива  $X$ , а затем – буквы на нечетных местах. Вывести оба массива.

44) Ввести массив символов из 10 элементов. Заменить символы-цифры на символ ' \* '.

45) Дана матрица символов размером  $2 \times 6$ . Сколько раз среди данных символов встречаются символы «#, \$, &»?

46) Дан массив символов, среди которых есть символ двоеточие ':'. Определить, сколько символов ему предшествует.

47) Дан массив символов, среди которых есть символ '#'. Определить, сколько символов стоит после него.

48) Дан массив символов. Определить, сколько раз входит в него группа букв a, b, c.

49) Ввести массив символов из 12 элементов. Удвоить все коды символов массива и вывести полученный результат.

50) Дана матрица символов размером  $5 \times 5$ . Упорядочить символы строк по возрастанию их кода.

51) Проверить, является ли введенная строка двоичным числом.

52) Проверить, является ли введенная строка шестнадцатеричным числом.

53) Проверить, является ли введенная строка дробным числом.

54) Дан текст. Напечатать все имеющиеся в нем цифры. Найти их сумму.

55) Дан текст, в котором имеются цифры. Найти порядковый номер максимальной цифры.

56) Дан текст, в котором имеется несколько идущих подряд цифр. Получить число, образованное этими цифрами.

57) Введена строка. Проверить, является ли она правильным идентификатором в JavaScript. Идентификатор считается правильным, если он состоит только из латинских букв, цифр и знака “\_” и не начинается с цифры.

58) Дан символ С, изображающий цифру или букву (латинскую или русскую). Если С изображает цифру, то вывести строку «digit», если латинскую букву — вывести строку «lat», если русскую — вывести строку «rus».

59) Дана строка. Если она представляет собой запись целого числа, то вывести 1, если вещественного (с дробной частью) — вывести 2; если строку нельзя преобразовать в число, то вывести 0.

60) Дана строка, изображающая целое положительное число. Вывести сумму и произведение цифр этого числа.

61) Ввести две строки. Определить, являются ли они анаграммами, то есть одна строка получена из другой перестановкой букв. Например, строки "БУК" и "КУБ" или "СОЛЬ" и "ЛОСЬ" являются анаграммами.

62) Даны две строки: А и В. Составить программу, проверяющую, можно ли из букв, входящих в А, составить В (буквы можно использовать не более одного раза и можно переставлять). Например, А: ИНТЕГРАЛ; В: АГЕНТ – составить можно; В: ГРАФ – составить нельзя.

63) Тест по русскому языку. Определить правильность написания ЖИ-ШИ, ЧА-ЩА и ЧУ-ЩУ в предложении и сообщить процент ошибок.

64) Из слова «информатика» путем "вырезок" и "склеек" его букв получить слова: «форма», «тик», «кит», «фора», «фирма», «форт».

## Глава 7. Файловая система и динамические структуры данных

### 7.1. Работа с файлами с помощью языка Node.js

#### Запись в текстовый файл

Когда мы составляем программу на каком-либо языке программирования и запускаем на выбранной платформе (внутри операционной системы), то зачастую эта операционная система предоставляет нам доступ к файловой системе компьютера. Благодаря этому мы можем вводить данные не только вручную с клавиатуры, а получать их из текстового (числового) или даже двоичного файла. Результаты, полученные в программе, тоже можно записать в файл (текстовый или графический, например).

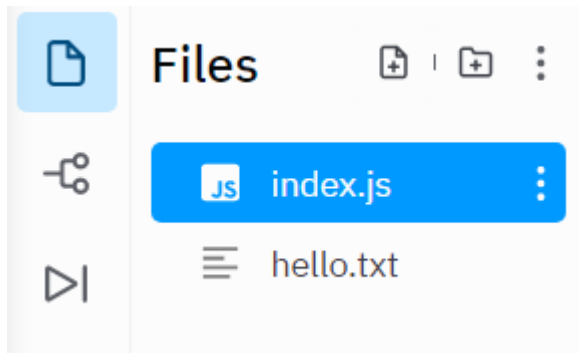
Но у браузера напрямую нет доступа к файловой системе компьютера клиента. Данные можно загрузить или из другой веб-страницы, или прибегнуть к помощи программирования для серверов.

Все задачи, рассматриваемые в этом пункте, не будут запускаться и работать на JavaScript, язык не предназначен для этих целей, но можно воспользоваться возможностями Node.js.

Чтение и запись в файл производятся в двух режимах, синхронном и асинхронном, но чаще используется асинхронный режим.

Рассмотрим конкретную задачу: записать фразу «Hello, world!» в текстовый файл `hello.txt`.

Для простоты будем использовать имитацию файловой системы в сеансе работы Replit. После запуска программы в левом поле появится дополнительный файл:



Теперь сам скрипт:

```
var fs = require('fs');  
  
fs.writeFile('hello.txt', 'Hello, world!', function(err) {  
  if(err) return console.error(err);  
});
```

В первой строке происходит подключение модуля для работы с файловой системой – fs. После подключения этого модуля с помощью команды require будут доступны методы работы с файлами: открытие файла, чтение и запись, копирование, переименование и многие другие.

В данном случае нас интересует запись текста из строковой переменной в файл. Соответствующий метод имеет вид:

```
writeFile('имя файла с расширением, 'Строка, которую надо записать')
```

Но почему такой скрипт:

```
var fs = require('fs');  
  
fs.writeFile('hello.txt', 'Hello, world!');  
  
не запускается?
```

Дело в том, что данный метод требует обратного вызова. Вдруг файловая система защищена от записи или недоступна? Необходимо предусмотреть способы защиты от ошибок. Вот почему в описание метода добавлена функция:

```
function(err)  
{
```

```
if(err) return console.error(err);  
});
```

Она возвращает код ошибки при проблемах создания файла.

Это упрощенный пример. Во всех языках программирования работа с файлами строится по определенному алгоритму:

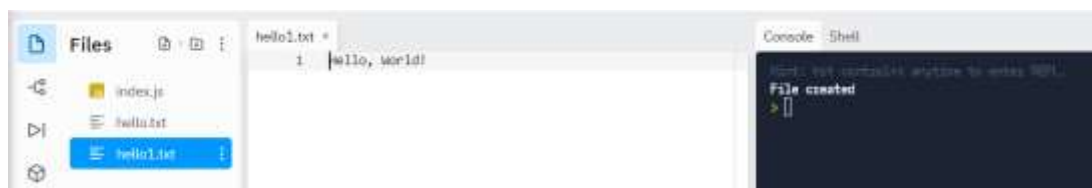
- 1) открытие файла с параметрами доступа для чтения, для записи, для чтения или перезаписи;
- 2) непосредственно чтение или запись данных;
- 3) завершение работы с файлом.

Благодаря «сборщикам мусора», последний пункт стал необязательным.

Приведем новый скрипт с записью в файл `hello1.txt` того же текста:

```
const fs = require("fs");  
fs.open("hello1.txt",'w', (error) => {  
    if(error) throw error;  
    console.log('File created');  
});  
fs.writeFile("hello1.txt", "Hello, World!", function(error){  
    if(error) throw error; // если возникла ошибка  
} )
```

Результатом работы этого скрипта станет сообщение «File created» в консоли и сам текстовый файл:



Содержимое этого текстового файла можно менять вручную, как в текстовом редакторе, но при повторном выполнении скрипта информация снова перезапишется.

В этом скрипте использовалась инструкция `throw`, которая позволяет генерировать исключения, определяемые пользователем.

Есть и синхронная версия того же метода для записи файла — `fs.writeFileSync()`, а работу с исключениями можно произвести через операторы `try/catch`:

```
const fs = require('fs')

const content = 'Some content!'

try {
  const data = fs.writeFileSync('test.txt', content)
  //файл записан успешно
} catch (err) {
  console.error(err)
}
```

Эти методы записи, по умолчанию, заменяют содержимое существующих файлов. Изменить их стандартное поведение можно, воспользовавшись соответствующим флагом:

```
fs.writeFile('test.txt', content, { flag: 'a+' }, (err) => {})
```

В этом случае новая информация добавится в конец существующего файла.

Запишем с помощью этого флага числа циклической последовательности:

```
const fs = require("fs");

fs.open("hello1.txt", 'w', (error) => {
  if(error) throw error;
  console.log('File created');
```



```

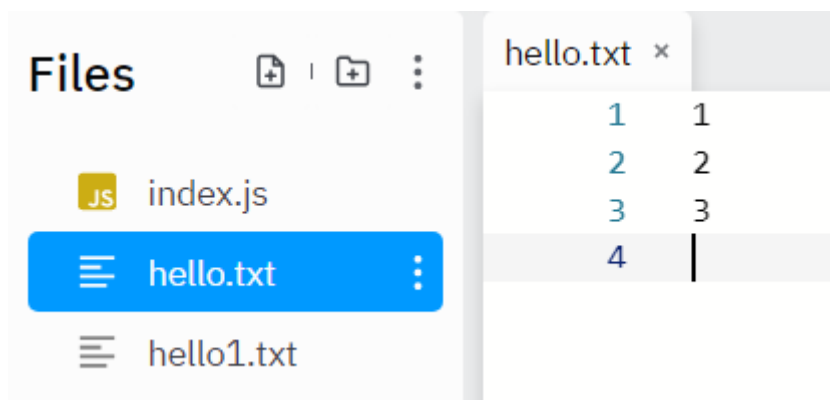
});
for (let i=1;i<=3;i++)
{
fs.writeFile("hello1.txt", i, { flag: 'a+' }, function(error){
  if(error) throw error; // если возникла ошибка
} )
}

```

Без флага 'a+' в файл запишется только последнее число, а предыдущие значения сотрутся.

### Чтение из текстового файла

Допустим, у нас уже есть текстовый файл с нужными числовыми значениями:



Выведем эти значения на экран:

```

const fs = require("fs");
fs.readFile("hello1.txt", "utf8",
function(error,data){
  if(error) throw error; // если возникла ошибка
  console.log(data); // выводим считанные данные
});

```

```
Console Shell
Hint: hit control+c anytime to enter REPL.
1
2
3
> 
```

Усложним задачу. Пусть необходимо посчитать сумму полученных чисел и вывести на экран консоли:

```
const fs = require("fs");

let s=0

let fileContent = fs.readFileSync("hello.txt", "utf8");

s=s+parseInt(fileContent)

console.log(s)
```

В этой задаче меняется только метод на синхронный, позволяющий считать информацию из файла в переменную:

```
имя переменной = fs.readFileSync("имя файла", "кодировка");
```

Проверять на ошибки файл не обязательно. Если он создан, то скрипт считывает нужную информацию, а если нет, то возникнет сообщение об ошибке.

Несмотря на то, что в нашем файле записаны верные данные, сумма посчиталась неправильно, и в ответе получилась «единица», а не «шестерка».

Дело в том, что файлы обрабатываются целиком, функция `parseInt()` считывает только первое число, игнорируя пробелы, знаки абзаца и другие символы.

Для решения задачи либо придется разбивать считанную строку на отдельные элементы массива, либо использовать модуль `Readline`.

`Readline` - это собственный модуль Node.js, он был разработан специально для чтения содержимого построчно из любого читаемого потока. Его можно использовать для чтения данных из командной строки.

Первый вариант решения:

```
const fs = require("fs");  
  
let s=0  
  
let fileContent = fs.readFileSync("hello.txt", "utf8");  
  
var arr = fileContent.split(' ')  
  
for (var i in arr)  
  
{s=s+parseInt(arr[i])}  
  
console.log(s)
```

Второй вариант решения:

```
const fs = require( 'fs' );  
  
const readline = require( 'readline' );  
  
const file = readline.createInterface({  
  
input: fs.createReadStream( 'hello.txt' ),  
  
output: process.stdout,  
  
terminal: false  
  
});  
  
let s=0;  
  
file.on('line' , (line) => {  
  
s=s+parseInt(line);  
  
console.log(s);  
  
} );
```

### Методы модуля fs

`fs.access()`: проверяет существование файла и возможность доступа к нему с учётом разрешений.

`fs.appendFile()`: присоединяет данные к файлу. Если файл не существует — он будет создан.

`fs.chmod()`: изменяет разрешения для заданного файла. Похожие методы: `fs.lchmod()`, `fs.fchmod()`.

`fs.chown()`: изменяет владельца и группу для заданного файла. Похожие методы: `fs.fchown()`, `fs.lchown()`.

`fs.close()`: закрывает дескриптор файла.

`fs.copyFile()`: копирует файл.

`fs.createReadStream()`: создаёт поток чтения файла.

`fs.createWriteStream()`: создаёт поток записи файла.

`fs.link()`: создаёт новую жёсткую ссылку на файл.

`fs.mkdir()`: создаёт новую директорию.

`fs.mkdtemp()`: создаёт временную директорию.

`fs.open()`: открывает файл.

`fs.readdir()`: читает содержимое директории.

`fs.readFile()`: считывает содержимое файла. Похожий метод: `fs.read()`.

`fs.readlink()`: считывает значение символической ссылки.

`fs.realpath()`: разрешает относительный путь к файлу, построенный с использованием символов «.» и «..», в полный путь.

`fs.rename()`: переименовывает файл или папку.

`fs.rmdir()`: удаляет папку.

`fs.stat()`: возвращает сведения о файле. Похожие методы: `fs.fstat()`, `fs.lstat()`.

`fs.symlink()`: создаёт новую символическую ссылку на файл.

`fs.truncate()`: обрезает файл до заданной длины. Похожий метод: `fs.ftruncate()`.

`fs.unlink()`: удаляет файл или символическую ссылку.

`fs.unwatchFile()`: отключает наблюдение за изменениями файла.

`fs.utimes()`: изменяет временную отметку файла. Похожий метод: `fs.futimes()`.

`fs.watchFile()`: включает наблюдение за изменениями файла. Похожий метод: `fs.watch()`.

`fs.writeFile()`: записывает данные в файл. Похожий метод: `fs.write()`.

Интересной особенностью модуля `fs` является тот факт, что все его методы, по умолчанию, являются асинхронными, но существуют и их синхронные версии, имена которых получаются путём добавления слова `Sync` к именам асинхронных методов.

Например:

`fs.rename()` - `fs.renameSync()`

`fs.write()` - `fs.writeSync()`

Рассмотрим пример переименования файлов с помощью асинхронной версии с использованием `callback` (возвратов):

```
const fs = require('fs')
fs.rename('test.txt', 'after.txt', (err) => {
  if (err) {
    return console.error(err)
  }
})
```

При использовании его синхронной версии для обработки ошибок используется конструкция `try/catch`:

```
const fs = require('fs')
try {
  fs.renameSync('after.txt', 'test.txt')
```

```
} catch (err) {  
    console.error(err)  
}
```

Основное различие между этими вариантами использования данного метода заключается в том, что во втором случае выполнение скрипта будет заблокировано до завершения файловой операции.

## 7.2. Коллекции данных

### Ассоциативный массив

В настоящее время среди документов получили распространение базы данных, а программы, в которых можно производить действия над данными, называются СУБД – системы управления базами данных.

Для работы в сетях можно напрямую подключаться к нужной базе данных, а языки веб-программирования тесно взаимодействуют с СУБД, работающими в сетевом режиме. Например, Node.js поддерживает работу с MySQL, а также со многими сторонними СУБД.

Но для небольших задач гораздо удобнее было бы пользоваться массивами данных, если не одно но... Допустим, у нас есть сведения о студентах некоторой группы. В одном массиве хранятся фамилии, а в другом – оценка каждого студента в том же порядке следования элементов. Если массив со списком отсортировать в алфавитном порядке, то оценки не будут соответствовать их получателям. В этом случае придется производить обмен данными при сортировке обоих массивов. А если у нас и третий параметр, например, даты рождения?

В языках программирования высокого уровня содержатся более удобные средства для работы со сложными данными – записи (Паскаль) или структуры (C++). В объектно-ориентированных средах программирования можно создавать более сложные структурированные данные, например, словари, в которых каждому значению противопоставляется его ключ.

Аналогичный объект используется и в JavaScript. Он называется ассоциативным массивом.

Имя и ключ разделяются двоеточием:

```
let student=  
{  
  name: "Иванов", mark: "5"  
}
```

```
console.log(student.name)
```

```
console.log(student.mark)
```

На экране мы получим следующее:



The screenshot shows a code editor with the following code in index.js:

```
1 let student=  
2 {  
3   name: "Иванов", mark: "5"  
4 }  
5 console.log(student.name)  
6 console.log(student.mark)
```

The console output shows:

```
Иванов  
5  
Hint: hit ctrl+c anytime to enter REPL.  
> []
```

Основное отличие ассоциативных массивов от обычных состоит в том, что в качестве индексов используются не числа, а строки.

Добавим в этот массив новое свойство – дату рождения, дописав соответствующую строку:



The screenshot shows a code editor with the following code in index.js:

```
1 let student=  
2 {  
3   name: "Иванов", mark: "5"  
4 }  
5 console.log(student.name)  
6 console.log(student.mark)  
7 student["date"]="01/01/2005"  
8 console.log(student.date)
```

The console output shows:

```
Иванов  
5  
01/01/2005  
Hint: hit contro  
> []
```

Вывести все элементы ассоциативного массива можно двумя способами:

1) написать имя массива; 2) использовать цикл for..in:

```
index.js = Console Shell
1 let student=
2 {
3   name: "Иванов", mark: "5"
4 }
5 console.log(student.name)
6 console.log(student.mark)
7 student["date"]="01/01/2005"
8 console.log(student.date)
9 console.log(student)
10 for (var key in student)
11 console.log(key+"="+student[key])
```

```
Иванов
5
01/01/2005
{ name: 'Иванов', mark: '5', date: '01/01/2005' }
name=Иванов
mark=5
date=01/01/2005
Hint: hit control+c anytime to enter REPL.
>
```

Для нахождения длины массива, т.е. определения количества элементов, метод `length` напрямую не действует. Но можно использовать метод `Object.keys()`, который возвращает массив из собственных перечисляемых свойств переданного объекта, в том же порядке, в котором они бы обходились циклом `for...in`:

```
9 console.log(student)
10 for (var key in student)
11 console.log(key+"="+student[key])
12 console.log(Object.keys(student.name).length );
13 console.log(Object.keys(student).length)
```

```
6
3
Hint: hit co
>
```

В первом случае (строка 12) метод возвращает длину строки – значения элемента, а во втором (строка 13) – количество всех элементов.

Студенческая группа состоит. Как правило, более, чем из одного студента, поэтому для описания всех студентов можно использовать массив, элементами которого являются ассоциативные массивы. Над такими структурами можно выполнять различные операции: суммирование элементов, сортировка, нахождение элемента по определенному параметру. Для выполнения этих действий в цикле необходимо после имени массива через точку записывать его ключ.

Элементами ассоциативных массивов могут быть и числа, но индексы являются строками. Например:

```
let m=new Array()
m["one"]=1
m["two"]=2
m["three"]=3
for (let i in m)
```



```
console.log(m[i])
```

## Множество

Множества (sets) представляют структуру данных, которая может хранить только уникальные значения. В JavaScript функционал множества определяет объект Set. Для создания множества применяется конструктор этого объекта.

Также можно передать в конструктор массив значений, которыми будет инициализировано множество.

Для добавления применяется метод `add()`. Его результатом является измененное множество. Для удаления элементов применяется метод `delete()`, а для удаления всех элементов - `clear()`.

Если нужно проверить, есть ли элемент в множестве, то используется метод `has()`. Если элемент есть, то метод возвращает `true`, иначе возвращает `false`. Для перебора элементов множества применяется метод `forEach()`. Также для перебора множества можно использовать цикл `for...of`.

Рассмотрим множество первых целых чисел на примере:

1 способ	2 способ
<pre>const mySet = new Set(); mySet.add(1); mySet.add(2).add(3); console.log(mySet)</pre>	<pre>const arr = [1, 1, 2, 3, 2]; const numbers = new Set(arr); console.log(numbers);</pre>

```
index.js =
1  const mySet = new Set();
2  mySet.add(1);
3  mySet.add(2).add(3);
4  console.log(mySet)
5
6  const arr = [1, 1, 2, 3, 2];
7  const numbers = new Set(arr);
8  console.log(numbers);
```

Console Shell

```
Set {1, 2, 3}
Set {1, 2, 3}
Hint: hit control+c anytime to enter REPL.
> []
```

Два способа создания множества дают одинаковый результат.

Рассмотрим операции над элементами:

...

```
console.log(numbers.size);
```

```
numbers.delete(3);
```

```
console.log(numbers);
```

```
let isDeleted = numbers.delete(3);
```

```
console.log(isDeleted);
```

```
console.log(numbers.has(3));
```

```
numbers.forEach(function(value1, value2, set){
```

```
    console.log(value1);
```

```
})
```

```
for(n of numbers){
```

```
    console.log(n);
```

```
}
```

Результат:

```
3
Set { 1, 2 }
false
false
1
2
1
2
Hint: hit control+c anytime to enter REPL.
> 
```

Стандартные операции над множествами:

- Union (Объединение). Объединяет все элементы из двух разных множеств и возвращает результат, как новый набор (без дубликатов).

- Intersection (Пересечение). Если заданы два множества, эта функция вернет другое множество, содержащее элементы, которые имеются и в первом и во втором множестве.

- Difference (Разница). Вернет список элементов, которые находятся в одном множестве, но НЕ повторяются в другом.

- Subset(Подмножество) - возвращает булево значение, показывающее, содержит ли одно множество все элементы другого множества.

Рассмотрим пример с предыдущими множествами:

```
const mySet = new Set();
mySet.add(1);
mySet.add(2).add(3);
const arr = [1, 1, 2, 3, 2];
const numbers = new Set(arr);
numbers.delete(3);
flag=false
```

```
const unionSet=new Set();
const interSet=new Set();
const subSet=new Set();
for (element of mySet)
unionSet.add(element)
for (element of numbers)
{
if (!unionSet.has(element))
unionSet.add(element)
}
console.log(unionSet)
for (element of mySet)
{
for (element2 of numbers)
if (element==element2)
interSet.add(element)
}
console.log(interSet)
for (element of mySet)
subSet.add(element)
for (element of numbers)
{
if (subSet.has(element))
subSet.delete(element)
}
```

```

console.log(subSet)
for (element of mySet)
{
  for (element2 of numbers)
  if (element==element2 && numbers.size<mySet.size)
flag=true
}
if (flag) console.log("numbers underset mySet")

```

```

Console Shell
Set { 1, 2, 3 }
Set { 1, 2 }
Set { 3 }
numbers underset mySet
Hint: hit control+c anytime to enter REPL.
➤ █

```

## Словарь

Словарь Map – это коллекция ключ/значение, как и ассоциативный массив. Но основное отличие в том, что Map позволяет использовать ключи любого типа.

Методы и свойства:

`new Map()` – создаёт коллекцию.

`map.set(key, value)` – записывает по ключу `key` значение `value`.

`map.get(key)` – возвращает значение по ключу или `undefined`, если ключ `key` отсутствует.

`map.has(key)` – возвращает `true`, если ключ `key` присутствует в коллекции, иначе `false`.

`map.delete(key)` – удаляет элемент по ключу `key`.

`map.clear()` – очищает коллекцию от всех элементов.

`map.size` – возвращает текущее количество элементов.

Для перебора коллекции `Map` есть 3 метода:

`map.keys()` – возвращает итерируемый объект по ключам,

`map.values()` – возвращает итерируемый объект по значениям,

`map.entries()` – возвращает итерируемый объект по парам вида [ключ, значение], этот вариант используется по умолчанию в `for..of`.

Рассмотрим пример:

```
let map = new Map();
```

```
map.set("1", "str1"); // строка в качестве ключа
```

```
map.set(1, "num1"); // цифра как ключ
```

```
map.set(2, "num1")
```

```
map.set(3, "num1")
```

```
console.log(map.get(1));
```

```
console.log(map.get("1"));
```

```
map.delete("1")
```

```
console.log(map.size);
```

```
console.log(map.values())
```

```
console.log(map.keys())
```

```
Console Shell
num1
str1
3
[Map Iterator] { 'num1', 'num1', 'num1' }
[Map Iterator] { 1, 2, 3 }
Hint: hit control+c anytime to enter REPL.
> █
```

### 7.3. Классы и объекты

В отличие от структуры C++ ассоциативный массив в JavaScript – это объект, поэтому рассмотрим создание любого типа объектов подробнее.

Любой объект можно воспринимать как совокупность данных и методов (функций для их обработки). Рассмотрим тот же пример с базой данных, но установим, что объект – это студент. У этого объекта в нашем примере будут две характеристики – имя и оценка. Они называются свойствами объекта. Фактически, свойство – это имя переменной, указывающее на его характеристики. Обратиться к свойству можно, указав его через точку:

Объект.свойство

Например, `student.name`

Метод – это функция для выполнения действий над данными. Например, можно вывести все свойства объекта или произвести расчеты по каким-либо признакам.

Обратиться к методу можно так же, как и к свойству, но с указанием параметров функции:

Объект.метод(параметры)

Например, `student.getInfo()`

С методом можно связать саму функцию, которая и будет обрабатываться.

Поясним на конкретном примере. Создадим функцию и соотнесем ее в качестве метода `getInfo`. Она будет определять, в каком порядке выводить значения данного объекта `student`:

```
var student=new Object()
student.name="Иванов"
student.mark="5"
student.getInfo=function()
{
  let info=this.name+" - "+this.mark
  return info
}
console.log(student.getInfo())
```

В первой строке этого скрипта мы создали объект `student` с помощью встроенного класса `Object`. После того, как объект создан, в переменной `student` сохраняется на него ссылка. В качестве значения свойства объекта можно использовать любой тип данных, например, число, массив, другой объект. А если в качестве значения указана ссылка на функцию, то такое свойство становится методом объекта. Внутри этого метода доступен указатель на текущий объект – `this`.

Создать подобный объект можно другим способом, определив свойства и методы в фигурных скобках, как в языке C++:

```
const student={
  name:"Иванов",
  mark:"5",
  getInfo:function()
{
  let info=this.name+" - "+this.mark
  return info
}
```



```
} }  
console.log(student.getInfo())
```

Объекты можно группировать в классы. Класс – это шаблон (тип) объекта, включающий в себя свойства (переменные) и методы (функции). Каждый объект в классе называется экземпляром.

Создание экземпляра класса имеет вид:

```
Экземпляр = new имя_класса(параметры)
```

Как и в примере с ассоциативным массивом, в студенческой группе обучается более одного человека. Поэтому класс – это вся группа, а объект – это один конкретный студент со своей фамилией и оценкой.

В JavaScript классы создаются с помощью функции-конструктора, например:

```
function student(name, mark)  
{  
  this.name=name  
  this.mark=mark  
  this.getInfo=function(){  
    let info=this.name+" - "+this.mark  
    return info  
  } }  
  
let student1=new student("Иванов","5")  
let student2=new student("Петров","4")  
console.log(student1.getInfo())
```

Внутри функции-конструктора могут располагаться методы – внутренние функции класса, манипулирующие свойствами объекта. С помощью них можно

изменять и выводить значения свойств, например, входящих переменных и других объектов.

Если свойства объектов будут содержать несколько значений, а метод один и тот же, то можно определить метод вне конструктора с помощью прототипа.

Для добавления метода в прототип используется свойство `prototype`.  
Например:

```
function chislo(m)
{
  this.znachenie=m;
}
chislo.prototype.getZnach=function(){
  return this.znachenie
}
let num1=new chislo(1)
let num2=new chislo(2)
let num3=new chislo(3)
console.log(num1.getZnach())
console.log(num1.znachenie)
```

В этом примере значение «1» выведется и при прямом обращении к свойству, и при использовании функции-прототипа.

#### **7.4. Итераторы и генераторы последовательностей**

Итераторы и генераторы пригодятся, когда вы хотите постепенно обрабатывать коллекцию. Вы получаете прирост в производительности, отслеживая только состояние коллекции, а не загружая все элементы. Элементы в коллекции

оцениваются по одному, а оценка остальной коллекции откладывается до следующего обращения.

Итераторы обеспечивают эффективный способ перемещения и управления большими списками. Генераторы обеспечивают эффективный способ создания списков.

В JavaScript итератор - это объект, который предоставляет метод `next()`, возвращающий следующий элемент последовательности. Этот метод возвращает объект с двумя свойствами: `done` и `value`.

Допустим, у нас есть объект, в котором надо перечислить элементы. Рассмотрим последовательность первых трех чисел с помощью итератора:

```
let range = {
  from: 1,
  to: 3
}
// сделаем объект range итерируемым
range[Symbol.iterator] = function() {
  let current = this.from;
  let last = this.to;
  // метод должен вернуть объект с методом next()
  return {
    next() {
      if (current <= last) {
        return {
          done: false,
          value: current++
        };
      }
    }
  };
};
```

```

    } else {
      return {
        done: true
      };
    }
  }
};

for (let num of range) {
  console.log(num); // 1, затем 2, 3
}

```

Перебираемый объект `range` сам не реализует методы для своего перебора. Для этого создаётся другой объект, который хранит текущее состояние перебора и возвращает значение. Этот объект называется итератором и возвращается при вызове метода `range[Symbol.iterator]`.

У итератора должен быть метод `next()`, который при каждом вызове возвращает объект со свойствами:

`value` – очередное значение,

`done` – равно `false` если есть ещё значения, и `true` – в конце.

Конструкция `for..of` в начале своего выполнения автоматически вызывает `Symbol.iterator()`, получает итератор и далее вызывает метод `next()` до получения `done: true`. Внешний код при переборе через `for..of` видит только значения.

Генераторы — это особый класс функций, которые упрощают задачу написания итераторов.

Генератор — это процесс, который может быть остановлен и возобновлен, и может выдать несколько значений. Генератор в JavaScript состоит из функции генераторов, которая возвращает элемент `Generator`, поддерживающий итерации.

Генераторы могут поддерживать состояние и обеспечивать эффективный способ создания итераторов, а также позволяют работать с бесконечным потоком данных. Оператор `yield` может приостановить функцию генератора и вернуть значение, которое следует за `yield`, тем самым обеспечивая более простой способ итерации значений.

Для демонстрации генератора рассмотрим аналогичный пример с последовательностью из трех чисел:

```
function* idGenerator() {  
  let i = 1;  
  while (i<=3) {  
    yield i++;  
  }  
}  
  
const ids = idGenerator();  
  
for (let i of idGenerator())  
  console.log(ids.next(i).value);
```

## 7.5. Динамические структуры

### Стек

Стек (`stack`) – это линейная структура данных, в которой элементы располагаются по принципу: первым зашел, последним вышел.

Стек имеет следующие методы:

`Push` - добавить новый элемент

`Pop` - удалить верхний элемент, вернуть его

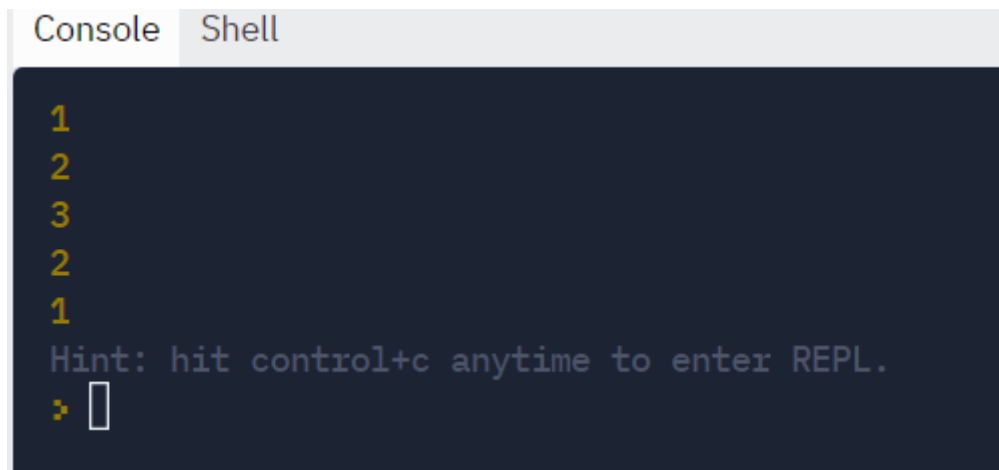
`Peek` - вернуть верхний элемент

`Length` - вернуть количество элементов в стеке

Рассмотрим пример, в котором в стек добавляются последовательно три элемента, а затем они поочередно удаляются:

```
var stack = [];  
for (let i=1;i<=3;i++)  
  console.log(stack.push(i))  
  stack.pop()  
  console.log(stack.push())  
  stack.pop()  
  console.log(stack.push())
```

Результат будет таким:



```
Console Shell  
1  
2  
3  
2  
1  
Hint: hit control+c anytime to enter REPL.  
js
```

В первом выводе отобразилась последовательность 1, 2, 3. Но внутри стека элементы расположены в обратном порядке: 3, 2, 1 (первый элемент внизу).

Затем удалили первый элемент, т.е. удалили 3, а первым стал 2.

Снова удалили первый элемент, остался последний – 1.

Фактически, стек представляет собой стопку – пока дотянулись до нижней вещи, надо снять все верхние.

Очередь

Очередь (Queue) – аналогичная стеку структура, но расположение элементов – первым зашел, первым и вышел.

Это означает, что после добавления нового элемента все элементы, которые были добавлены до этого, должны быть удалены до того, как новый элемент будет удален.

Очередь имеет следующие методы:

enqueue: войти в очередь, добавить элемент в конец

dequeue: покинуть очередь, удалить первый элемент и вернуть его

front: получить первый элемент

isEmpty: проверить, пуста ли очередь

size: получить количество элементов в очереди

Создать очередь можно вручную как объект и описать прототипы функций для работы с ней:

```
function Node(data) {
  this.data = data;
  this.next = null;
}
function Queue() {
  this.head = null;
  this.tail = null;
}
Queue.prototype.enqueue = function(data) {
  var newNode = new Node(data);
  if (this.head === null) {
    this.head = newNode;
    this.tail = newNode;
```

```

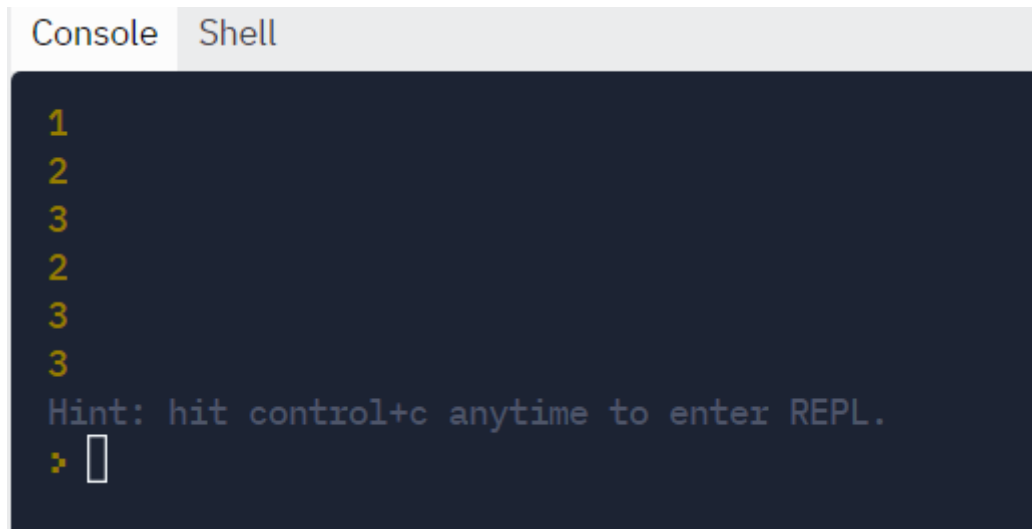
    } else {
        this.tail.next = newNode;
        this.tail = newNode;
    }
}
Queue.prototype.dequeue = function() {
    var newNode;
    if (this.head !== null) {
        newNode = this.head.data;
        this.head = this.head.next;
    }
    return newNode;
}
Queue.prototype.print = function() {
    var curr = this.head;
    while (curr) {
        console.log(curr.data);
        curr = curr.next;
    }
}
var q = new Queue();
q.enqueue(1);
q.enqueue(2);
q.enqueue(3);
q.print();

```



```
q.dequeue();
q.print();
q.dequeue();
q.print();
```

Результат:



```
1
2
3
2
3
3
Hint: hit control+c anytime to enter REPL.
> 
```

Числа удалились в том же порядке, в каком и следовали.

## 7.6. Скрипты, содержащие структуры и объекты

### Задача №1

Создать текстовый файл, записать в него построчно данные, которые вводит пользователь. Окончанием ввода служит пустая строка.	
Node.js Replit	Node.js Tutorialspoint
<pre>const fs = require("fs"); fs.open("hello.txt", 'w', (error) =&gt; {   if(error) throw error;   console.log('File created'); }); var x=prompt()</pre>	<pre>const fs = require("fs"); let rl = require("readline-sync"); function readline() {   return rl.question(); } let x=readline()</pre>

<pre> let y=x while (x!=="") { x=prompt() y=y+x fs.writeFile("hello.txt", y, function(er- ror){   if(error) throw error; // если воз- никала ошибка } ) } </pre>	<pre> y=x while (+(x = readline()) !== 0)   y += x; fs.writeFile("hello.txt", y, function(er- ror){   if(error) throw error; // если воз- никала ошибка } ) </pre>
--	--

Строго говоря, метод `prompt()` не поддерживается в Node.js, но разработчики онлайн-среды для программирования Replit сделали некую гибридную версию JavaScript/Node.js, благодаря установке дополнительных пакетов. А в онлайн-среде [Tutorialspoint.com](https://www.tutorialspoint.com) мы получим сообщение об ошибке:

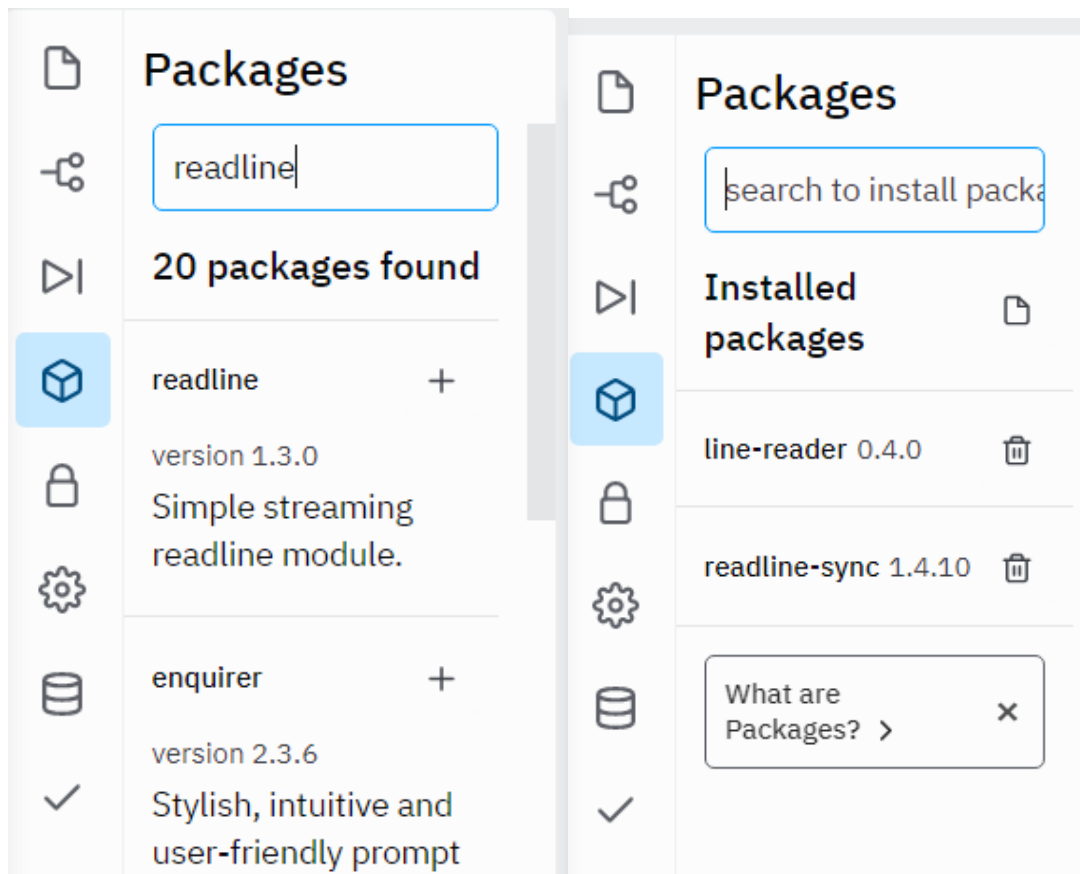
```

Result

$node main.js
/home/cg/root/4182458/main.js:6
var x=prompt()
      ^
ReferenceError: prompt is not defined

```

Поэтому для решения задачи в классическом Node.js необходимо подключить модуль `Readline-sync`. Для подключения этого модуля в среде Replit необходимо зайти в раздел `Packages`, найти нужный модуль и нажать кнопку «+»:



## Задача №2

Составить программу с использованием записи «Студент», включая следующие поля: ФИО студента, дату рождения, адрес, курс, группа.

Ассоциативный массив	Класс
<pre>var stud=[] var n=parseInt(prompt()) for (i=0;i&lt;n;i++) { stud[i]={ name:prompt("ФИО"), age:prompt("Дата"),addr: prompt("Адрес"), kurs:prompt("Курс"),gr:p rompt("Группа")</pre>	<pre>function student(name, data, addr, course, group) { this.name=name this.data=data this.addr=addr this.course=course this.group=group this.getInfo=function(){ let info=this.name+" "+this.data+" "+this.addr+" "+this.course+" "+this.group</pre>

<pre> }} for (i=0;i&lt;n;i++) con- sole.log(stud[i].name,stud [i].age, stud[i].addr,stud[i].kurs,st ud[i].gr) </pre>	<pre> return info } } let stud=[] var n=parseInt(prompt("Количество")) for (i=0;i&lt;n;i++) stud[i]=new stu- dent(prompt("ФИО"),prompt("Дата"),prompt("Адрес" ),prompt("Курс"),prompt("Группа")) for (i=0;i&lt;n;i++) console.log(stud[i].getInfo()) </pre>
--	---

Результат:

```

Console Shell
Количество> 2
ФИО> Иванов
Дата> 01.01.2005
Адрес> ул.Джавы, д.10
Курс> 2
Группа> ПКс-21
ФИО> Петров
Дата> 02.02.2005
Адрес> ул.Скрипта, д.5
Курс> 2
Группа> ПКс-22
Иванов 01.01.2005 ул.Джавы, д.10 2 ПКс-21
Петров 02.02.2005 ул.Скрипта, д.5 2 ПКс-22
Hint: hit control+c anytime to enter REPL.
> █

```

Примечания к коду:

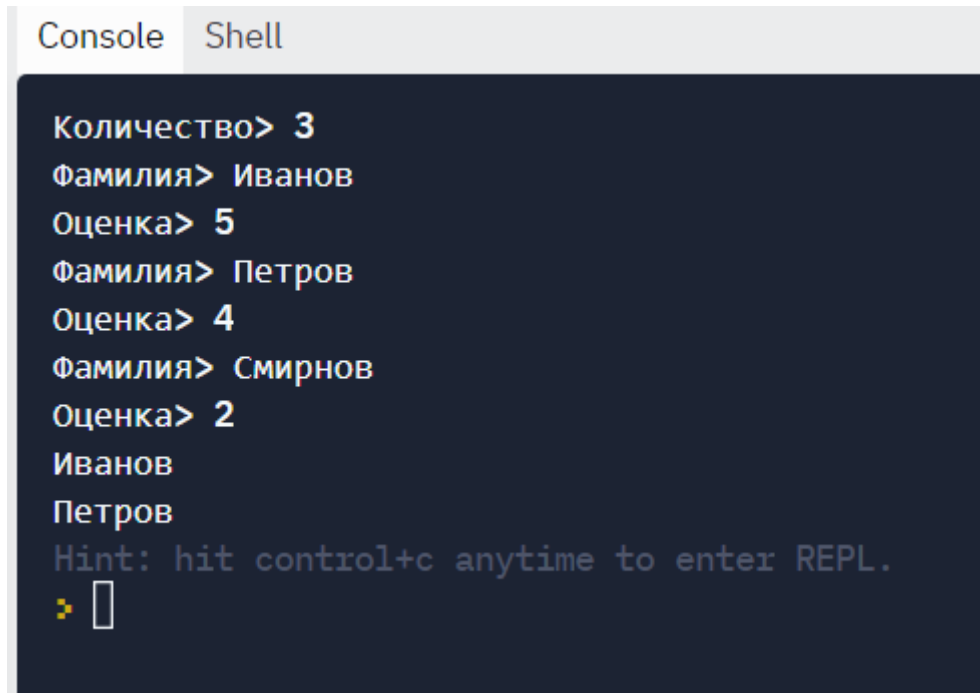
При решении первым способом код получился короче, но при втором способе скрипт легче редактировать, не теряя наглядности.

### Задача №3

Создать базу данных студентов. Определить студентов с баллом выше среднего.	
Ассоциативный массив	Класс
<pre> var stud=[] var n=parseInt(prompt()) for (i=0;i&lt;n;i++) { stud[i]={ name:prompt("fio"), ball:par- seInt(prompt("ball"))} } let sum = 0.0; for (i=0;i&lt;n;i++){ sum += stud[i].ball; } let avg=sum/n for (i=0;i&lt;n;i++){ if (stud[i].ball&gt;avg) console.log(stud[i].name) } </pre>	<pre> function student(name, mark) { this.name=name this.mark=mark this.getInfo=function(){ let info=this.name return info } this.Summa=function(){ let sum = 0.0; sum+=parseInt(this.mark) return sum } } let stud=[] var n=parseInt(prompt("Количество")) for (i=0;i&lt;n;i++) stud[i]=new stu- dent(prompt("Фамилия"),prompt("Оценка")) let sum1=0 for (i=0;i&lt;n;i++) sum1+=stud[i].Summa() let avg=sum1/n for (i=0;i&lt;n;i++) if (parseInt(stud[i].mark)&gt;avg) </pre>

```
console.log(stud[i].getInfo())
```

Результат:



```
Console Shell
Количество> 3
Фамилия> Иванов
Оценка> 5
Фамилия> Петров
Оценка> 4
Фамилия> Смирнов
Оценка> 2
Иванов
Петров
Hint: hit control+c anytime to enter REPL.
>
```

Примечания к коду:

Внутри объявления класса можно использовать как несколько свойств, так и несколько методов. Минус второго способа состоит в том, что нельзя посчитать количество экземпляров класса внутри объявления метода.

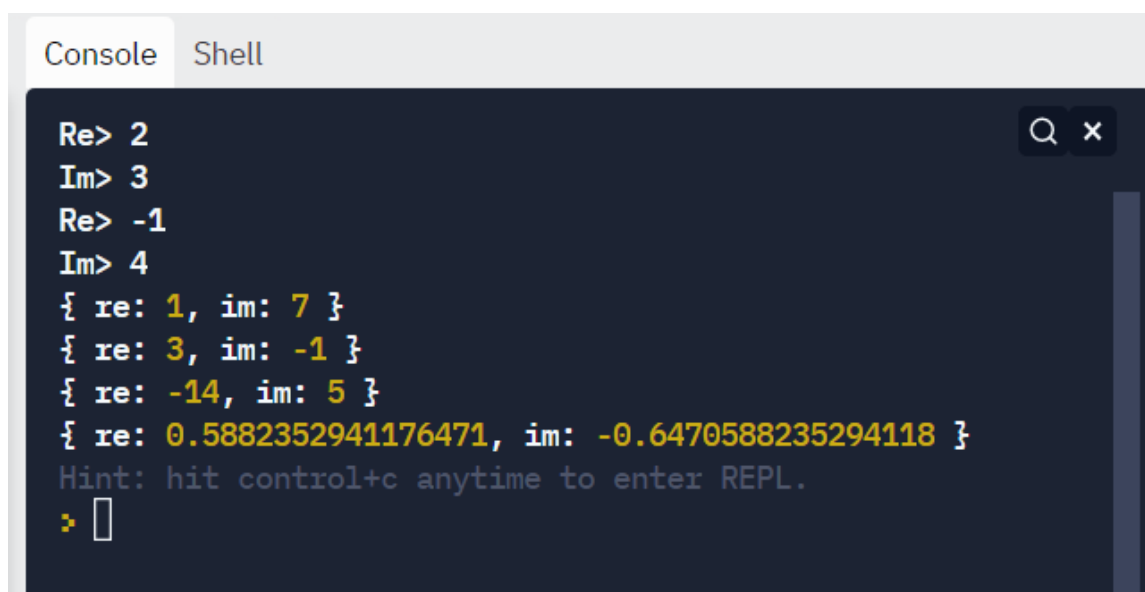
#### Задача №4

Выполнить операции над комплексными числами: сумма, разность и произведение.	
Класс	Модуль complex.js
<pre>function Complex(real, imaginary) {   this.x = real; // Вещественная часть числа   this.y = imaginary; // Мнимая часть числа } Complex.prototype.magnitude = function() {</pre>	<pre>let Complex=require("complex.js") let a=parseInt(prompt("Re")) let b=parseInt(prompt("Im")) let c=parseInt(prompt("Re")) let d=parseInt(prompt("Im"))</pre>

<pre> return Math.sqrt(this.x*this.x + this.y*this.y); } Complex.prototype.negative = function() { return new Complex( -this.x, -this.y); }; Complex.prototype.add = function(that) { return new Complex(this.x + that.x, this.y + that.y); } Complex.prototype.multiply = function(that) { return new Complex(this.x * that.x, this.y * that.y, this.x*that.y + this.y * that.x); } Complex.add = function (a, b) { return new Complex(a.x + b.x, a.y + b.y); }; Complex.multiply = function(a, b) { return new Complex(a.x * b.x, a.y * b.y, a.x * b.y + a.y * b.x)}; Complex.prototype.toString = function() { return "{" + this.x + "," + this.y + "}"; }; var x1=parseFloat(prompt()) var x2=parseFloat(prompt()) var y1=parseFloat(prompt()) var y2=parseFloat(prompt()) let a=new Complex(x1,x2) </pre>	<pre> let x=new Complex(a,b) let y=new Complex(c,d) console.log(x.add(y)) console.log(x.sub(y)) console.log(x.mul(y)) console.log(x.div(y)) </pre>
--	--

```
let b=new Complex(y1,y2)
let s=a.add(b)
let z=b.negative()
let r=a.add(z)
let m=a.multiply(b)
con-
sole.log(s.toString(),r.toString(),m.toString())
```

Результат:



Примечания к коду:

1) Для работы над комплексными числами вида  $a+bi$ , где  $a$  – действительная часть числа,  $b$  – мнимая часть числа, можно написать собственный класс `Complex` с помощью функции-конструктора. Но существуют готовые модули, которые достаточно подключить к сценарию. Второй вариант решения гораздо короче благодаря использованию модуля `complex.js`.

2) Для второго варианта задачи в решение добавлен метод деления комплексных чисел.

3) Рассмотрим третий вариант скрипта с использованием модуля `math.js`. Подключим этот модуль по прямой ссылке с сервера



<https://cdnjs.cloudflare.com/ajax/libs/mathjs/10.0.0/math.js> и реализуем функции add, subtract, multiply, divide:

```
<!DOCTYPE HTML>

<html>

<head>

  <script

src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/10.0.0/math.js"

type="text/javascript"></script>

</head>

<body>

Введите реальную часть 1-ого числа<input type="text" class="real1">

Введите мнимую часть 1-ого числа<input type="text" class="imag-
ine1"><br>

Введите реальную часть 2-ого числа    <input type="text" class="real2">

Введите мнимую часть 2-ого числа<input type="text" class="imag-
ine2"><br>

<input type="submit" class="button">

<div id=id_div>Ответ: <br></div>

  <script type="text/javascript">

let real1Input = document.querySelector('.real1');

let imagine1Input = document.querySelector('.imagine1');

let real2Input = document.querySelector('.real2');

let imagine2Input = document.querySelector('.imagine2');

let out=document.querySelector("#id_div");

let button = document.querySelector('.button');

button.addEventListener('click', () => {
```

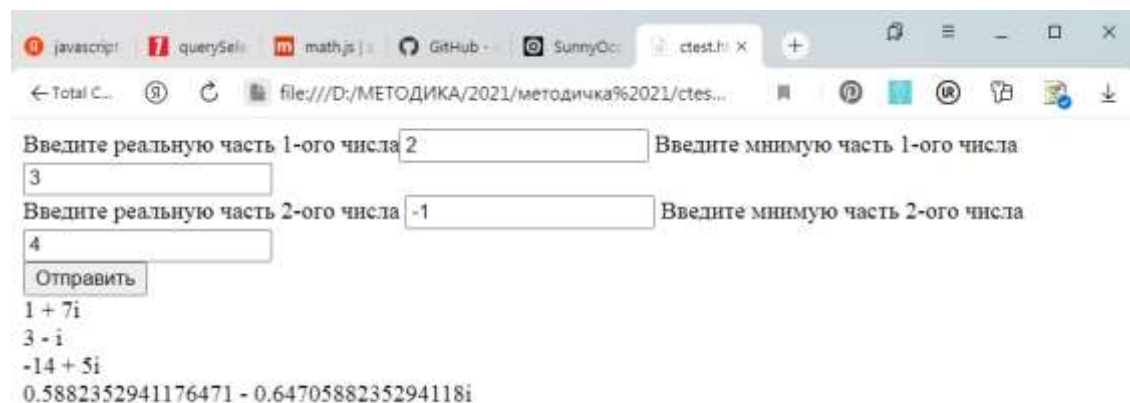
```

let real1 = Number(real1Input.value);
let imagine1 = Number(imagine1Input.value);
let real2 = Number(real2Input.value);
let imagine2 = Number(imagine2Input.value);

let a=math.complex(real1,imagine1)
let b=math.complex(real2,imagine2)
let s1=math.add(a,b).toString()+"<br>"
let s2=math.subtract(a,b).toString()+"<br>"
let s3=math.multiply(a,b).toString()+"<br>"
let s4=math.divide(a,b).toString()
out.innerHTML=s1+s2+s3+s4
});
</script>
</body>
</html>

```

Результат:



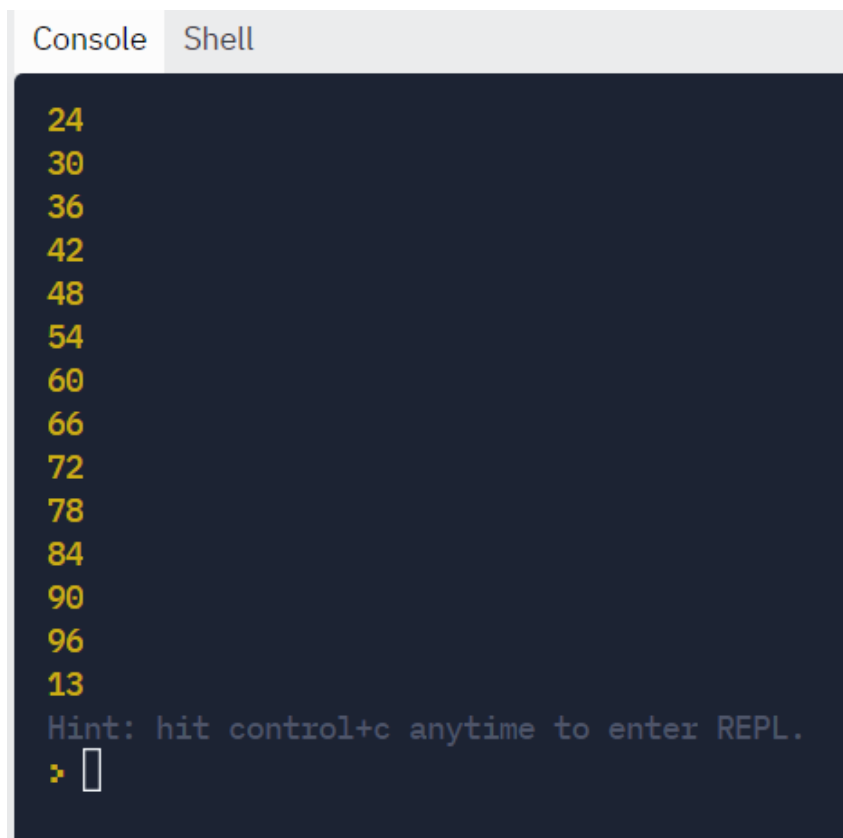
Задача №5

В заданном числовом промежутке  $[20;100]$  найти числа, кратные 6, и посчитать их количество.

Функция последовательности	Итератор	Генератор	Множество
<pre>function linSeq(x0, xN) {   return lin- space(x0, xN, Math.abs(xN- x0)+1); } function lin- space(x0, xN, n){   dx = (xN - x0)/(n- 1);   var x = [];   for(var i =0; i &lt; n; ++i){     x.push(x0 + i*dx);   }   return x; } var mn = linSeq(20,100).fil- ter(function(num- ber) {   return number % 6===0; });</pre>	<pre>const iterable = { [Symbol.iterator]() {   return {     current: 20, end: 100, next() {   if (this.current &lt;= this.end ) {     return { value: this.current++, done: false }; } return { done: true }; } }; }; let k=0 for (const value of iterable)   if (value%6===0)   {     con- sole.log(value);     k++   } console.log(k)</pre>	<pre>function* natu- ralRowGenera- tor() {   let current = 20;   while (current &lt;= 100) {     if (cur- rent%6===0)       yield cur- rent;     current++;   } let k=0 for (let num of naturalRowGen- erator()) {   con- sole.log(num);   k++ } console.log(k)</pre>	<pre>const arr=[] let j=0 for (i=20;i&lt;=100;i++) if (i%6===0) {   arr[j]=i   j++ } const numbers = new Set(arr) console.log(num- bers) console.log(num- bers.size)</pre>

con- sole.log(mn.length)			
-----------------------------	--	--	--

Результат:



```
Console Shell
24
30
36
42
48
54
60
66
72
78
84
90
96
13
Hint: hit control+c anytime to enter REPL.
> █
```

Примечания к коду:

1) Функцию-генератор можно упростить с помощью цикла, сделав аналог метода range, используемого в языках PHP и Python:

```
function* range(start, end) {
  for (let i = start; i <= end; i++) {
    if (i%6===0)
      yield i;
  }
}
a=[...range(20,100)]
```

```
console.log(a)
```

```
console.log([...range(20,100)].length)
```

### Задача №6

С клавиатуры вводится последовательность целых чисел, ограниченная нулем (ноль не входит в последовательность). Получить и вывести квадраты нечетных чисел, не превышающих 15

Фильтрация массива	Итератор	Генератор
<pre>arr=[] var n=1 do {   var x=parseInt(prompt())   arr[n]=x   n++ } while (x!==0) function condition(value, index, array) {   var result = false;   if (value %2 == 1 &amp;&amp; value&lt;=15) {     result = true;   }   return result; }; var filteredNumbers = arr.filter(condition);</pre>	<pre>const iterable = {   [Symbol.iterator]() {     return {       current: 2,       next() {         if (this.current !==0 )         {           return { value: this.current=parseInt(prompt()), done: false };         }         return { done: true };       }     };   } }; let k="" for (const value of iterable)   if (value%2!=0 &amp;&amp;value&lt;=15)   {</pre>	<pre>function* naturalRowGenerator() {   let current = parseInt(prompt());   while (current !=0)   {     if (current%2!=0&amp;&amp;current&lt;=15)       yield current**2;     current= parseInt(prompt());   } } let k="" for (let num of naturalRowGenerator()) {   k+=num.toString()+" "</pre>

for(var i=0; i < filteredNumbers.length; i++) console.log(filteredNumbers[i]*filteredNumbers[i]);	k+=(value**2).toString()+'; } console.log(k)	console.log(k);
--	--	-----------------

Результат:

```

Console Shell
undefined> 3
undefined> 5
undefined> 6
undefined> 11
undefined> 16
undefined> 0
9 25 121
Hint: hit control+c anytime to enter REPL.
>

```

## 7.7. Контрольные вопросы и тесты

### Дополнительные вопросы

1. Может ли хакер, написав скрипт на JS, проникнуть в файловую систему клиентского компьютера?
2. Всегда ли необходимо обрабатывать исключения с кодом ошибки при чтении и записи файла?
3. Для чего служит модуль fs в Node.js?
4. Чем отличается ассоциативный массив от словаря?
5. Каким образом можно создать объект в классе объектов?
6. Чем по своей сути являются свойства объектов?

7. Могут ли описания классов содержать более одного метода?
8. Можно ли в задачах использовать генератор без итератора?
9. Чем отличаются структуры queue и stack?
10. Какая реализация стека и очереди удобней, через массив или через класс?

### Тесты с выбором варианта ответа

1. Модуль работы с файлами подключается командой:

- 1) `require fs = const ('fs');`
- 2) `var fs = require('fs');`
- 3) `const fs = require('writeFile');`
- 4) `fs.writeFile();`

2. Методы чтения из файла делятся на:

- 1) прямой и обратный;
- 2) текстовый и двоичный;
- 3) синхронный и асинхронный;
- 4) последовательный и параллельный.

3. Массив, в котором индекс является строковым ключом, называется:

- 1) объектным;
- 2) множественным;
- 3) ассоциативным;
- 4) текстовым.

4. Функции для обработки свойств объектов называются:

- 1) методы;
- 2) классы;
- 3) конструкторы;

4) данные.

5. Для возврата следующего элемента последовательности используется:

1) генератор;

2) итератор;

3) нумератор;

4) конструктор.

6. Возврат значения функции-генератора осуществляется с помощью оператора:

1) next;

2) count;

3) yield;

4) range.

7. Структура, в которой элементы располагаются по принципу LIFO, называется:

1) стек;

2) очередь;

3) множество;

4) список.

8. Добавление нового элемента в стек осуществляется методом:

1) pop();

2) pop();

3) push();

4) peek().

9. Для добавления элемента в множество применяется метод:

1) add();

2) push();



3) set();

4) has().

10. Для возврата значения по ключу в коллекции map используется метод:

1) map.set(key);

2) map.get(key);

3) set.map(key);

4) get.add(key).

11. Создание экземпляра класса имеет вид:

1) имя класса = new экземпляр();

2) класс.экземпляр.функция(параметры);

3) экземпляр = new имя\_класса(параметры);

4) объект = new (параметры).класс.

12. Методы сложения, вычитания, умножения и деления комплексных чисел в модуле math.js:

1) add, negative, multy, divisor;

2) add, subtract, multiply, divide;

3) add, neg, mul, div;

4) plus +, minus -, asterix \*, slash /.

### Тесты без выбора варианта ответа

1. Дан скрипт:

```
var fs = require('fs');
```

```
fs.writeFile('hello.txt', 'Hello, world!', function(err) {
```

```
if(err) return console.error(err);
```

```
});
```

Переделать вывод информации в текстовый файл с помощью синхронного метода.

2. Дан скрипт:

```
const student={  
  name:"Иванов",  
  mark:"5",  
}  
console.log(student)
```

Какая структура (коллекция) называется student – объект или ассоциативный массив?

3. Дан скрипт, содержащий ошибки:

```
range[Symbol.iterator] = function() {  
  let current = 1;  
  let last = 3;  
  return {  
    next() {  
      if (current <= last) {  
        return {  
          done: false,  
          yield current++  
        }  
      }  
    }  
  }  
}  
const ids = Generator();  
for (let i of Generator())
```

```
console.log(next().value);  
  
};
```

Выяснить, с помощью чего была создана последовательность – итератора или генератора. Исправить ошибки.

4. Укажите верные утверждения:

А) словарь `map` может использовать только строковые ключи, а ассоциативный массив также и числовые.

Б) в массиве элементы не повторяются, а в множестве могут повторяться.

В) ассоциативный массив является объектом.

Г) объект в классе можно создать с помощью конструктора.

5. Укажите неверные утверждения:

А) элементы в очереди расположены по принципу FIFO.

Б) работа с файловой системой поддерживается в JavaScript.

В) из стека нельзя удалить элемент.

Г) для работы с комплексными числами разработаны сторонние библиотеки.

### **7.8. Задачи для самостоятельной работы**

1) Найти значение наибольшего модуля среди заданных комплексных чисел.

2) Дано комплексное число  $z=x+iy$ . Возвести его в заданную степень  $N$ .

3) Дано комплексное число  $z=x+iy$ . Извлечь корень  $N$ -ой степени из этого числа.

4) Дано комплексное число  $z=x+iy$ . Найти комплексно-сопряженное число  $z'=x-iy$  и вычислить произведение  $z \cdot z'$ .

5) Дано комплексное число  $z=x+iy$ . Найти главное значение аргумента комплексного числа  $z$ .

6) Дано комплексное число  $z=x+iy$ . Найти  $\operatorname{arcsin}$  комплексного числа  $z$ .

7) Дано комплексное число  $z=x+iy$ . Найти  $\operatorname{arctan}$  комплексного числа  $z$ .

8) Дано комплексное число  $z=x+iy$ . Найти косинус числа  $z$ .

9) Составить скрипт с использованием записи «Отдел кадров», включая поля: фамилия сотрудника, имя, отчество, должность, стаж работы, оклад.

10) Составить скрипт с использованием записи «Красная книга», включая поля: вид животного, род, семейство, место обитания, численность популяции.

11) Составить скрипт с использованием записи «Производство», включая поля: обозначение изделия, группа к которой оно относится, год выпуска, объем выпуска, расход металла.

12) Составить скрипт с использованием записи «Компьютеры», включая поля: фирма-изготовитель, тип процессора, тактовая частота, емкость ОЗУ, емкость жесткого диска.

13) Составить скрипт с использованием записи «Библиотека», включая поля: автор книги, название, год издания, цена, количество в библиотеке.

14) Составить скрипт с использованием записи «Радиодетали», включая поля: обозначение, тип, номинал, количество на схеме, обозначение возможного заменителя

15) Составить скрипт с использованием записи «Текстовые редакторы», включая поля: название программы, фирма-изготовитель, количество пунктов меню, количество шрифтов, расширение файла документа.

16) Составить скрипт с использованием записи «Телефонная станция», включая поля: номер абонента, фамилия, адрес, наличие АОН, задолженность.

17) Известны данные об  $N$  учениках класса: фамилия, имя, отчество, адрес и домашний телефон, если он есть. Вывести на экран фамилию, имя и адрес учеников, у которых нет домашнего телефона.

18) Известны фамилии  $N$  человек, их семейное положение: женат (замужем) или нет, и сведения о наличии детей (есть или нет). Определить фамилии женатых (замужних) людей, имеющих детей.

19) Известны данные об  $N$  учениках: фамилия, класс и оценки по математике и информатике. Определить фамилии учеников 9-х классов, имеющих оценку "5" по информатике.

20) Известна информация об  $N$  сотрудниках фирмы: фамилия, имя, отчество, адрес и дата поступления на работу (месяц, год). Вывести фамилию, имя, отчество и адрес сотрудников, которые на сегодняшний день проработали в фирме не менее трех полных лет.

21) Известны данные о стоимости каждого из  $N$  ( $N > 2$ ) наименований товаров: число рублей и число копеек. Составить программу, сравнивающую стоимость двух любых наименований товаров (определяющую, какой из товаров стоит дороже).

22) Известны фамилии  $N$  сотрудников фирмы и их адреса. Определить, работают ли в фирме люди с фамилией, начинающейся на букву «К». В случае положительного ответа вывести их адреса.

23) Даны названия  $N$  стран и частей света, в которых они находятся. Определить, есть ли среди них страны, находящиеся в Африке или в Азии. В случае положительного ответа вывести их названия.

24) Известны данные об  $N$  студентах: фамилии, имена, отчества, даты рождения (год, номер месяца и число). Определить, есть ли студенты, у которых сегодня день рождения, и если да, то вывести их имя, фамилию и возраст.

25) Найти объединение, пересечение и разность двух заданных множеств.

26) Дано натуральное число  $n$ . Вывести все цифры, не входящие в десятичную запись этого числа в порядке возрастания.

27) Найти простые числа в промежутке  $[1..n]$ . Число  $n$  вводится с клавиатуры. (Решето Эратосфена).

28) Дана непустая последовательность символов. Вывести множества, элементами которых являются: цифры от «0» до «9» и знаки арифметических операций.

29) Дана непустая последовательность символов. Вывести множества, элементами которых являются: буквы от «A» до «F» и от «X» до «Z».

30) Дана непустая последовательность символов. Вывести множества, элементами которых являются: знаки препинания и буквы от «E» до «N».

31) Подсчитать общее количество цифр и знаков «+», «-», «\*» в множестве, введенном с клавиатуры.

32) Сформировать множество строчных латинских букв, входящих в введенную строку, и подсчитать количество знаков препинания в ней.

33) Дан массив чисел. Записать их в файл, расположив каждый элемент массива на отдельной строке с сохранением порядка.

34) Дано имя файла и вещественные числа  $A$  и  $D$ . Создать файл вещественных чисел с данным именем и записать в него 10 первых членов арифметической прогрессии с начальным членом  $A$  и разностью  $D$ :  $A, A + D, A + 2 \cdot D, \dots$

35) Дано имя файла и целое число  $N (> 1)$ . Создать файл целых чисел с данным именем и записать в него  $N$  первых положительных четных чисел (2, 4, ...).

36) Дано имя файла и целые положительные числа  $N$  и  $K$ . Создать текстовый файл с указанным именем и записать в него  $N$  строк, каждая из которых состоит из  $K$  символов «\*».

37) Даны вещественные числа  $A$ ,  $B$  и целое число  $N$ . Создать текстовый файл, содержащий значения функции  $y=x^2$  на промежутке  $[A, B]$  с шагом  $(B - A)/N$ .

38) Даны вещественные числа  $A$ ,  $B$  и целое число  $N$ . Создать текстовый файл, содержащий значения функции  $y=\sin x$  на промежутке  $[A, B]$  с шагом  $(B - A)/N$ .

39) Дан массив из  $N$  целых чисел. Все его компоненты удвоить и переписать в текстовый файл.

40) Дан массив из  $N$  целых чисел. Записать его в текстовый файл так, чтобы каждое отрицательное значение было заменено на максимальное значение из этого массива.

41) Имеется текстовый файл с числами. Вывести все его элементы с нечетным порядковым номером.

42) Имеется текстовый файл с числами. Напечатать все его элементы, большие числа  $x$ .

43) Имеется текстовый файл с числами. Найти сумму первого и второго чисел файла.

44) Имеется текстовый файл с числами и два числа, введенные с клавиатуры  $k_1$ ,  $k_2$ . Найти сумму  $k_1$ -го и  $k_2$ -го чисел файла.

45) Имеется текстовый файл с числами. Найти произведение первого и последнего чисел файла.

46) Имеется текстовый файл с числами. Найти среднее арифметическое положительных чисел файла.

47) Имеется текстовый файл с числами. Найти первое число, большее числа  $b$ . Если таких чисел нет, то сообщить об этом.

48) Имеется текстовый файл с числами. Найти максимальное число, имеющееся в файле.

49)  $N$  детей располагаются по кругу. Начав отсчет от первого, удаляют каждого  $k$ -го, смыкая при этом круг. Определить порядок удаления детей из круга.

50) Ввести последовательность чисел, оканчивающуюся нулем, в стек.

51) Задана последовательность слов. Определить частоту вхождения каждого из слов в последовательность.

52) Создать стек из целых чисел. Вычислить произведение нечётных значений элементов стека.

53) Дано число  $N (> 0)$  и набор из  $N$  чисел. Создать стек, содержащий исходные числа (последнее число будет вершиной стека), и вывести указатель на его вершину.

54) Создать стек из вещественных чисел. Определить максимальный элемент в стеке.

55) Создать стек из целых чисел. Вычислить среднее арифметическое чётных значений элементов стека.

56) Создать стек из целых чисел. Определить сумму значений элементов стека, кратных 5.

57) С клавиатуры вводятся целые числа, ограниченные нулем. Сам ноль в набор чисел не входит. Найти и вывести значения максимального и минимального из введенных чисел.

58) Даны  $n$  целых чисел, принадлежащих отрезку  $[-99; 99]$ . Найти и вывести значение последнего четного числа, максимального по абсолютной величине.

59) Даны  $n$  случайных чисел, находящихся в диапазоне от  $-100$  до  $100$ . Найти и вывести значения суммы и произведения его элементов, не кратных трем. Количество чисел не превышает 50.

60) С клавиатуры вводится последовательность натуральных чисел, не превышающих 100 и ограниченная нулем, который в последовательность не входит.



Найти среднее значение для последовательности квадратных корней введенных значений.

61) Заполнить массив из  $n$  элементов целочисленными значениями, вводимыми с клавиатуры. Вывести элементы массива, принадлежащие отрезку  $[\min+10; \max-12]$ , где  $\min$  и  $\max$  – значения минимального и максимального элементов массива соответственно.

62) Среди первой тысячи натуральных чисел найти сумму тех из них, которые оканчиваются цифрой 7. Ввод осуществить с помощью функции последовательности.

63) Для 15 случайных целых чисел на отрезке  $[-55; 92]$  найти минимальное, максимальное и среднее значение.

64) Массив предназначен для хранения значений роста двенадцати человек. С помощью датчика случайных чисел заполнить массив целыми значениями, лежащими на отрезке  $[163; 190]$ .

## Глава 8. Работа с графикой в JS

### 8.1. Возможности изображения фигур на холсте

Холст (canvas) – это один из инструментов работы с веб-страницей. Прежде, чем создать изображение с помощью скрипта, необходимо установить размеры холста в теле (тег <body>) html-документа:

```
<canvas id="canvas" width="x" height="y"></canvas>
```

Параметры тега <canvas>:

- 1) id - идентификатор, по которому обращаться к данному холсту;
- 2) x,y – ширина и высота холста прямоугольной формы.

В основном коде скрипта обязательными являются команды:

```
var canvas = document.getElementById('canvas');  
var ctx = canvas.getContext('2d');
```

В первой строке объект canvas необходимо связать с id холста – идентификатором, который мы определили заранее на веб-странице.

Во второй строке подключаем контекст устройства для создания двумерной графики, чтобы связать логические координаты холста с физическими координатами изображения.

Кроме ввода команд рисования фигур и линий используется еще три термина: путь, обводка и заливка.

Путь – это последовательность линий. Если разные фигуры принадлежат одному пути, то они могут соединиться линией. Начало пути задается методом `beginPath()`. Конец пути – `closePath()` – не только завершает процесс рисования, но и замыкает фигуру, например, достраивает четвертую сторону четырехугольника, если изображены уже 3 стороны.

Обводка (stroke) – это контуры фигуры. Свойство `strokeStyle` определяет цвет, которым будет нарисована обводка. Толщина линии определяется свойством `lineWidth`, которое может содержать любое положительное число.

Заливка (fill) зависит от типа фигуры. Например, метод fillRect заливает прямоугольник. Он принимает координаты левого верхнего угла x,y, затем ширину и высоту. Метод fillStyle используется для изменения цвета заливки.

Рисование прямых линий, как и во многих других языках, включает в себя две команды:

moveTo(x1,y1) – перемещение позиции в точку с заданными координатами;

lineTo(x2,y2) – проведение линии от предыдущей точки до точки с заданными координатами.

Для рисования окружностей существует метод arc (дуга), которому надо передать шесть параметров:

- 1) Горизонтальную координату центра окружности на canvas (в пикселях).
- 2) Вертикальную координату центра окружности на canvas (в пикселях).
- 3) Радиус окружности (в пикселях).
- 4) Начальный угол окружности (в радианах).
- 5) Конечный угол окружности (в радианах).
- 6) Рисовать по часовой стрелке или против (если против, то значение "false", а если по часовой, то "true"). По умолчанию рисует против часовой стрелки.

Например, нарисовать полную окружность:

```
Context.arc(x0,y0,r,0,2*Math.PI);
```

Изменять толщину линии можно с помощью известного метода lineWidth, а цвет с помощью strokeStyle.

Теперь рассмотрим три функции рисования прямоугольников в canvas:

fillRect(x, y, width, height) - рисование заполненного прямоугольника.

strokeRect(x, y, width, height) - рисование прямоугольного контура.

`clearRect(x, y, width, height)` - очистка прямоугольной области, делая содержимое прозрачным.

Каждая из приведенных функций принимает несколько параметров:

1) `x`, `y` устанавливают положение верхнего левого угла прямоугольника в `canvas` (относительно начала координат);

2) `width` (ширина) и `height` (высота) определяют размеры прямоугольника.

Рассмотрим пример рисования домика, состоящего из линий (крыши), прямоугольников (стен, окон, дверей) и окружности (чердачного окна):

```
<!DOCTYPE html>

<html><head>

<title>Web Page Design</title>

</head>

<body>

<canvas id="canvas" width="500" height="500"></canvas>

<script>

var canvas = document.getElementById('canvas');

var ctx = canvas.getContext('2d');

ctx.beginPath();

ctx.moveTo(50, 150);

ctx.lineTo(100, 100);

ctx.lineTo(150, 150);

ctx.lineTo(50,150);

ctx.stroke();

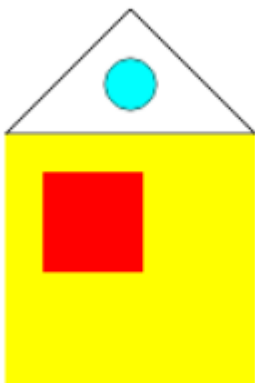
ctx.fillStyle = 'yellow';

ctx.fillRect(50, 150, 100, 100);

ctx.fillStyle = 'red';
```

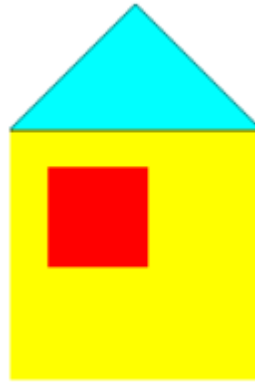
```
ctx.fillRect(65,165,40,40);  
var ctx2 = canvas.getContext('2d');  
ctx2.beginPath();  
ctx2.fillStyle = 'cyan';  
ctx2.arc(100,130,10,0,2*Math.PI);  
ctx2.stroke();  
ctx2.fill();  
</script>  
</body>  
</html>
```

Результат:



Обратите внимание, что в этом примере использовались два контекста изображения – ctx и ctx2. Если их объединить, то окружность не будет нарисована:

```
<script>
    var canvas = document.getEle-
mentById('canvas');
    var ctx = canvas.getCon-
text('2d');
    ctx.beginPath();
    ctx.moveTo(50, 150);
    ctx.lineTo(100, 100);
    ctx.lineTo(150, 150);
    ctx.lineTo(50,150);
    ctx.stroke();
    ctx.fillStyle = 'yellow';
    ctx.fillRect(50, 150, 100, 100);
    ctx.fillStyle = 'red';
    ctx.fillRect(65,165,40,40);
    ctx.fillStyle = 'cyan';
    ctx.arc(100,130,10,0,2*Math.PI);
    ctx.stroke();
    ctx.fill();
</script>
```

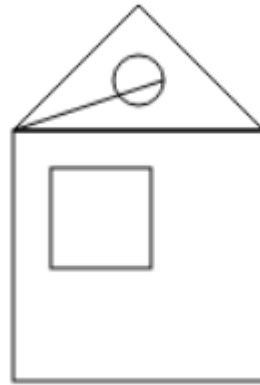


Вариант без заливки, показывающий автоматическую обводку:

```

<script>
    var canvas = document.getEle-
mentById('canvas');
    var ctx = canvas.getCon-
text('2d');
    ctx.beginPath();
    ctx.moveTo(50, 150);
    ctx.lineTo(100, 100);
    ctx.lineTo(150, 150);
    ctx.lineTo(50,150);
    ctx.stroke();
    ctx.strokeRect(50, 150, 100,
100);
    ctx.strokeRect(65,165,40,40);
    ctx.arc(100,130,10,0,2*Math.PI);
    ctx.stroke();
</script>

```



Отрисовку изображения можно упростить и перенести путь:

```

<script>
var canvas = document.getEle-
mentById('canvas');
var ctx = canvas.getContext('2d');
ctx.moveTo(50, 150);
ctx.lineTo(100, 100);
ctx.lineTo(150, 150);
ctx.stroke();

```




<pre> ctx.strokeRect(50, 150, 100, 100); ctx.strokeRect(65,165,40,40); ctx.beginPath(); ctx.arc(100,130,10,0,2*Math.PI); ctx.stroke(); ctx.closePath(); &lt;/script&gt; </pre>	
--	--

Создание полуокружности имеет следующие параметры:

- ctx.arc(x,y,r,0,Math.PI, true); - выпукла вверх
- ctx.arc(x,y,r,0,Math.PI, false); - выпукла вниз

Пример скрипта:

<pre> &lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt; &lt;title&gt;Web Page Design&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;canvas id="canvas" width="500" height="500"&gt;&lt;/canvas&gt; &lt;script&gt; var canvas = document.getEle- mentById('canvas'); var ctx = canvas.getContext('2d'); ctx.beginPath(); ctx.arc(100,130,50,0,Math.PI, true); ctx.closePath(); ctx.stroke(); ctx.beginPath(); </pre>	
--	--



<pre>ctx.arc(100,230,50,0,Math.PI, false); ctx.closePath(); ctx.stroke(); &lt;/script&gt; &lt;/body&gt; &lt;/html&gt;</pre>	
---	--

Для создания многоугольников необходимо объявить массив `points`, содержащий значения координат точек, образующий вершины многоугольника. Массив является двумерным, т.к. для каждой точки используются 2 координаты – `x,y`. Чтобы соединить последнюю точку с первой, можно использовать команду замыкания контура (закрытия пути) – `ctx.closePath()`.

Скрипт:

```
<!DOCTYPE html>
<html><head>
<title>Web Page Design</title>
</head>
<body>
<canvas id="canvas" width="500" height="500"></canvas>
<script>
const points=[[100,200], [150,100], [200,200], [150,300] ];
var canvas=document.getElementById("canvas")
var ctx = canvas.getContext('2d');
ctx.beginPath();
ctx.moveTo(points[0][0], points[0][1])
ctx.lineTo( points[1][0], points[1][1])
ctx.lineTo( points[2][0], points[2][1])
ctx.lineTo( points[3][0], points[3][1])
ctx.closePath();
```

```
ctx.stroke();
```

```
</script> </body> </html>
```

Результат:

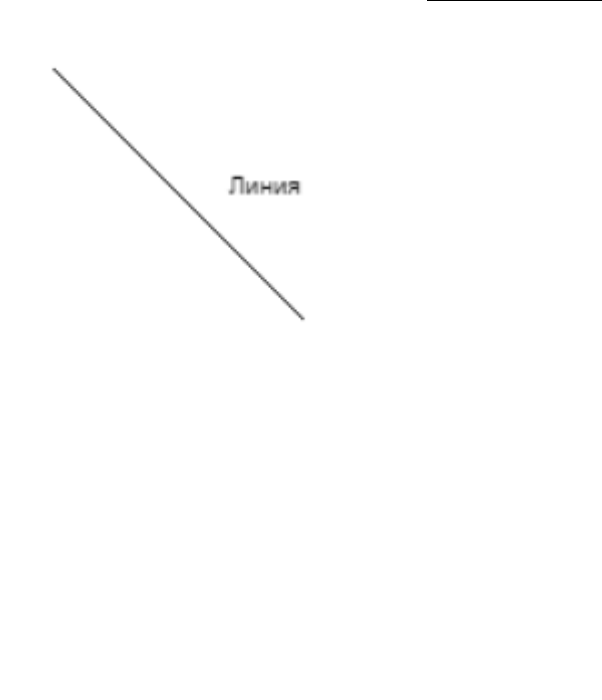


С помощью команды `fillText(s,x,y)` можно вставлять текст в графическую область.

`s` – строка, которую необходимо вывести;

`x,y` - координаты вывода текста.

Пример:

<pre>&lt;script&gt; var canvas=document.getEle- mentById("canvas") var ctx = canvas.getContext('2d'); ctx.beginPath(); ctx.moveTo(50,100) ctx.lineTo(150,200) ctx.fillText("Линия",120,150) ctx.closePath(); ctx.stroke(); &lt;/script&gt;</pre>	
--	--

В команды для изображений часто подставляются числовые данные. Их можно обрабатывать с помощью оператора цикла и получать интересные рисунки от мишени до градиента.

С помощью цикла можно изменить:

- начальные и конечные координаты точки (смещение линий);
- толщину линии;
- цвет (задается выражением RGB).

Рассмотрим эти методы на примере изображения наклонных линий:

```
<!DOCTYPE html>
```

```
<html> <head>
```

```
<title>Web Page Design</title>
```

```
<script type=text/javascript>
```

```
// анонимная функция – загрузчик окна (событие)
```

```
window.onload=function()
```

```
{
```

```
// доступ к графической области №1 (с индексом 0) и вызов через ссылку  
метода возврата ссылки графического контекста
```

```
let ctx=document.getElementsByTagName("canvas")[0].getContext("2d")
```

```
// длина и ширина окна
```

```
ctx.canvas.width=600
```

```
ctx.canvas.height=400
```

```
let x=10, y=10, l=200
```

```
// цикл, задающий количество линий (8 штук), их толщину и цвет
```

```
for (let k=0;k<8;k++)
```

```
{
```

```

    ctx.beginPath()

    ctx.lineWidth=1+k;

ctx.strokeStyle="rgb("+((255-k*20))+","+((20*k))+",255)"

    x+=10

ctx.moveTo(x,y)

    ctx.lineTo(x+1/5,y+1)

    ctx.stroke()

    ctx.closePath()

} }

</script> </head>

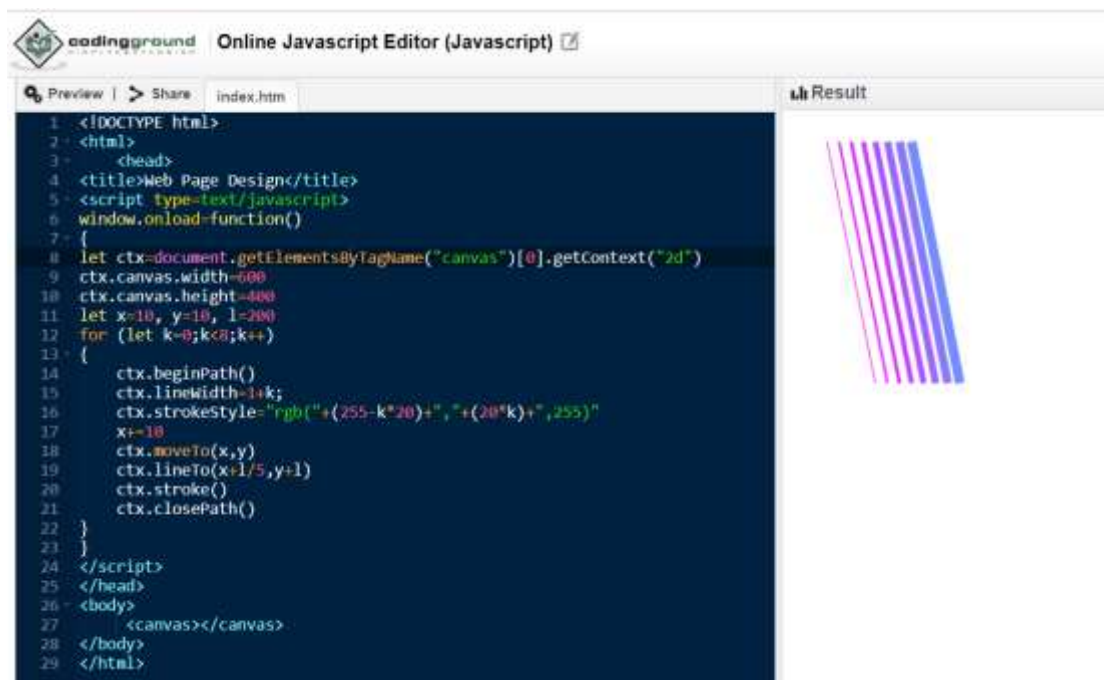
<body>

    <canvas></canvas>

</body> </html>

```

Результат:



Пояснения к коду:

В предыдущем примере настройки графической области производились внутри веб-страницы, а именно, тега `<canvas>`. В самом скрипте происходило связывание графической области с контекстом отображения. В данном примере и настройки, и связывание графической области с контекстом содержатся в коде функции – обработчика события загрузки документа.

С помощью команды `ctx=document.getElementsByTagName("canvas")[0].getContext("2d")` мы получим ссылку на объект графического контекста, связанного с графической областью документа. Значением выражения `document.getElementsByTagName("canvas")` является ссылка на коллекцию, содержащую ссылки на графические области. Т.к. в этом примере всего одна графическая область, то ее индекс будет равен нулю. А через эту ссылку вызывается метод `getContext("2d")`, который возвращает ссылку на объект графического контекста.

Фактически две строки вида:

```
var canvas = document.getElementById('canvas');
```

```
var ctx = canvas.getContext('2d');
```

мы заменили одной строкой с указанием индекса графического окна.

Если высоту и ширину графической области задать в скрипте, то соответствующий метод должен относиться к самой графической области, поэтому в команде сначала указывается графический контекст, затем графическая область, а в конце – сам метод:

```
ctx.canvas.width=600
```

Для рисования наклонных линий разного цвета и толщины объявляются вспомогательные переменные `x,y` – координаты точки, `l` – длина каждой линии.

Непосредственное создание линий реализуется через оператор цикла, в котором переменная `k` пробегает 8 значений (от 0 до 7). Каждый виток цикла начинается с создания нового пути для контура – `ctx.beginPath()`.

Затем командой `ctx.lineWidth=1+k` определяется толщина линии от 1 до 8 пикселей (с увеличением толщины каждой линии на единицу).

Цвет линии задается выражением `ctx.strokeStyle="rgb("+ (255-k*20)+", "+ (20*k)+", 255)"`

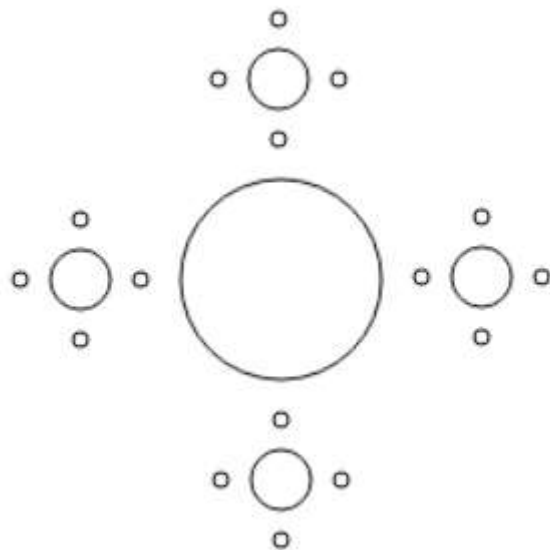
В правой части содержится текстовая строка, которая выражает количественную меру для красной (r), зеленой (g) и синей (b) составляющей цвета в диапазоне от 0 до 255.

В данном примере для каждого нового витка цикла количество красного цвета уменьшается, зеленого увеличивается, а синего остается неизменным. Таким образом, цвет линий меняется от розового к фиолетовому.

Командой `x+=10` увеличивается горизонтальная координата для получения параллельных линий, расположенных на равном расстоянии (10 пикселей) друг от друга. После команды рисования линии и отображения на экране контур замыкается.

## 8.2. Рекурсия и фракталы

Написать скрипт построения изображения, представленного на рисунке:



Анализ изображения: большая окружность окружена четырьмя окружностями поменьше, каждая из которых в свою очередь окружена четырьмя окружностями с еще меньшим радиусом.

Алгоритм решения:

- описать функцию изображения одной окружности;
- вычислить радиус для маленьких окружностей (орбиты);
- в цикле рассчитать координаты центра окружности следующего уровня;
- заново вызвать функцию с новыми параметрами.

Математические формулы, использующие вызов функции через предыдущие значения, называются рекуррентными. А рекурсии, предназначенные для построения однотипных изображений с уменьшением значений параметров, называются фракталами.

Пусть центр первоначальной окружности имеет координаты  $(x, y)$ , а радиус –  $r$ . Чтобы изобразить четыре окружности вокруг начальной, необходимо знать расстояние  $r_1$  от центра большой окружности до центров окружностей окружения (радиусы орбиты –  $r_0$ ). Пусть  $\frac{r_0}{r} = k_1, \frac{r_1}{r} = k_2$ .

Значения  $x_1, y_1$  зависят от  $x, y$  следующими соотношениями:

$$x_1 = x + dx, y_1 = y + dy, \text{ где } dx = r_1 \cdot \cos \alpha, dy = r_1 \cdot \sin \alpha, \alpha = \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi$$

Если осуществить вызов функции из основной программы с начальными параметрами, то рекурсивные вызовы никогда не закончатся. Т.е. функция будет работать бесконечно. Чтобы этого не случилось, надо ввести ограничение  $n$  – количество уровней. С каждым вызовом это количество уменьшается на единицу, т.е. функция содержит условие  $\text{if } (n > 0)$ .

В основной программе запрашиваются радиус  $r$  большой окружности, число уровней  $n$  и значения коэффициентов  $k_1, k_2$ . Изменяя эти значения, можно получить разные изображения.

Сам код:

```
<!DOCTYPE html>
<html><head>
<title>Web Page Design</title>
```

```

</head>
<body>
<canvas id="canvas" width="500" height="500"></canvas>
<script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
function picture(x,y,r,r1,n)
{
  let x1,y1
  if (n>0)
    {
  ctx.beginPath();
  ctx.arc(x,y,r,0,2*Math.PI);
  ctx.stroke()
  ctx.closePath();
  r1=Math.floor(r*k2)
  for (let i=1;i<5;i++)
    {
  x1=Math.floor(x+r1*Math.cos(Math.PI/2*i))
  y1=Math.floor(y+r1*Math.sin(Math.PI/2*i))
  picture(x1,y1,Math.floor(r*k1), r1,n-1)
    } } }
let n=parseInt(prompt())
let x=250
let y=250

```



```
let r=parseInt(prompt())  
k1=0.3  
k2=2  
let r1=0  
picture(x,y,r,r1,n)  
</script> </body> </html>
```

### 8.3. Работа с модулем «Черепашка»

В чистом виде в JavaScript функций turtle не существует, но можно подключить сторонний модуль turtle.js для создания изображений с помощью движения объекта.

Рассмотрим код для рисования правильного шестиугольника, используя в цикле команды «вперед» и «повернуть». Параметром первой команды является количество шагов (пикселей), а второй – угол поворота.

Подключить модуль можно с сервера <https://cdn.jsdelivr.net/gh/IntEgr8/turtlejs/turtle.js>.

После этого создается новый объект с параметрами – координатами окна:

```
Const имя = new Turtle(x_нач, y_нач, x_кон, y_кон)
```

В параметрах метода setStyles указывается цвет и толщина линии. После окончания работы с «черепашкой» необходимо завершить работу с помощью метода .done().

Основные команды модуля turtle.js:

forward(x) – движение вперед на x пикселей;

backward(x) – движение назад на x пикселей, равнозначно forward(-x);

rotate(alpha) – поворот на угол  $\alpha$  (в градусах);

home() – возврат в начальную позицию.

Скрипт:

```
<!DOCTYPE html>

<html>

<head>

  <title>Turtle</title>

</head>

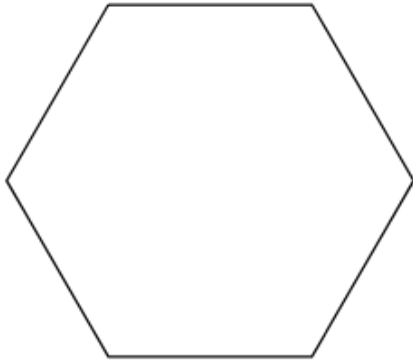
<body>

<script src="https://cdn.jsdelivr.net/gh/1ntEgr8/turtlejs/turtle.js"></script>

  <script >

const turtle = new Turtle(100, 100, 1000, 1000)
for (var count = 0; count < 6; count++) {
turtle.setStyles({"stroke": "black",
  "stroke-width": "1px"});
turtle.forward(90);
turtle.rotate(60);
}
turtle.done();
</script> </body> </html>
```

Результат:



Пояснение к коду:

Полный цикл многоугольника – 360 градусов, поэтому переменная цикла пробегает 6 значений (от 0 до 5), а угол поворота равен 60 градусов. В команде `turtle.forward(90)` число 90 указывает расстояние, т.е. длину стороны шестиугольника.

#### 8.4. Анимация

При работе с графикой могут применяться понятия «динамическая графика» или «анимация». Обычно это смена статических картинок по какому-либо условию.

Имитация движения достигается следующим образом: по истечении определенного времени (действия таймера, витков цикла, функции-задержки) цвет фигуры меняется на фоновый, и строится аналогичная фигура со смещенными координатами.

В JavaScript для этих целей применяются методы объекта `window` `setTimeout()` и `setInterval()`. Первый метод позволяет вызвать некоторую функцию через определенный промежуток времени. Второй метод позволяет периодически выполнять определенные действия из заданной функции. Ее вид:

```
setInterval(имя функции, время в миллисекундах, необязательные аргументы);
```

Интервал между вызовами функции с заданным именем (1 параметр) задается во втором параметре. Для прекращения процесса периодического вызова функции используется метод `clearInterval()`. Ее параметром является значение результата вызова первого метода.

Рассмотрим пример, в котором, цвет окружности периодически меняется с красного на зеленый:

```
<!DOCTYPE html>

<html><head>

<title>Web Page Design</title>

</head>

<body>

<canvas id="canvas" width="500" height="500"></canvas>

<script>

var canvas = document.getElementById('canvas');

var ctx = canvas.getContext('2d');

function draw1()

{

ctx.fillStyle="red"

ctx.beginPath()

ctx.arc(320,240,40,0,2*Math.PI);

ctx.closePath()

ctx.fill()

}

setInterval(draw1,1000);

function draw2()

{
```

```
ctx.fillStyle="green"
ctx.beginPath()
ctx.arc(320,240,40,0,2*Math.PI);
ctx.closePath()
ctx.fill()
}
setInterval(draw2,1500);
</script>
</body>
</html>
```

Результат:



Эта функция-таймер может использоваться и по своему прямому назначению – отсчитывать время в секунду. Продемонстрируем это на примере цифровых часов:

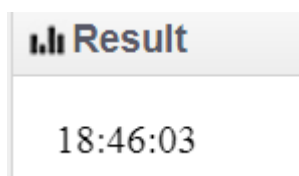
```
<!DOCTYPE html>
<html><head>
<title>Web Page Design</title>
</head>
<body onload="setInterval(myTime,1000)">
  <div id="myblock"></div>
```

```

<script>
function myTime()
{
var time=new Date()
document.getElementById("myblock").innerHTML=time.toLocaleTime-
String()
}
</script>
</body>
</html>

```

Результат:



## 8.5. Загрузка графических файлов

Для загрузки графического файла на веб-страницу достаточно знать тег языка HTML - ``. Тег `<img>` должен располагаться внутри любого блочного тега, например, нового абзаца `<p></p>`. Параметр тега `<img>` - `src` – содержит URL-адрес графического изображения. Дополнительно можно указать ширину и высоту изображения. Но в таком случае ничего поверх этой картинки изобразить уже нельзя. А можно ли совместить канву для рисования `<canvas>` с готовыми изображениями?

Конечно. Для этих целей служит объект изображения `Image()`. Для работы с ним сначала надо определить переменную для записи ссылки на этот объект. После этого необходимо вызвать метод `drawImage()`, который рисует изображение, содержимое другого элемента `<canvas>` или видео. Также, метод

`drawImage()` может нарисовать часть изображения и/или увеличить/уменьшить размер изображения.

Его синтаксис:

`drawImage(переменная типа Image, x,y)`, где `x` и `y` – координаты начальной позиции изображения, начиная с верхнего левого угла.

С помощью метода `имя.src` можно получить ссылку на файл с изображением, например:

```
pict.src="графический файл.jpg".
```

Рассмотрим пример скрипта, в котором на графической области изображается прямоугольник с заливкой, внутрь прямоугольника помещается изображение из файла, а под изображением (на холсте) с помощью метода `fillText()` помещается текстовая запись.

```
<html> <head>
<title>Рисунок</title>
<script type=text/javascript>
let pict,cnv,ctx
window.onload=function()
{
pict=new Image()
cnv=document.getElementById("mycanvas")
cnv.title="Картинка"
ctx=cnv.getContext("2d")
ctx.font="30px Arial"
set()
pict.onload=function()
{
```

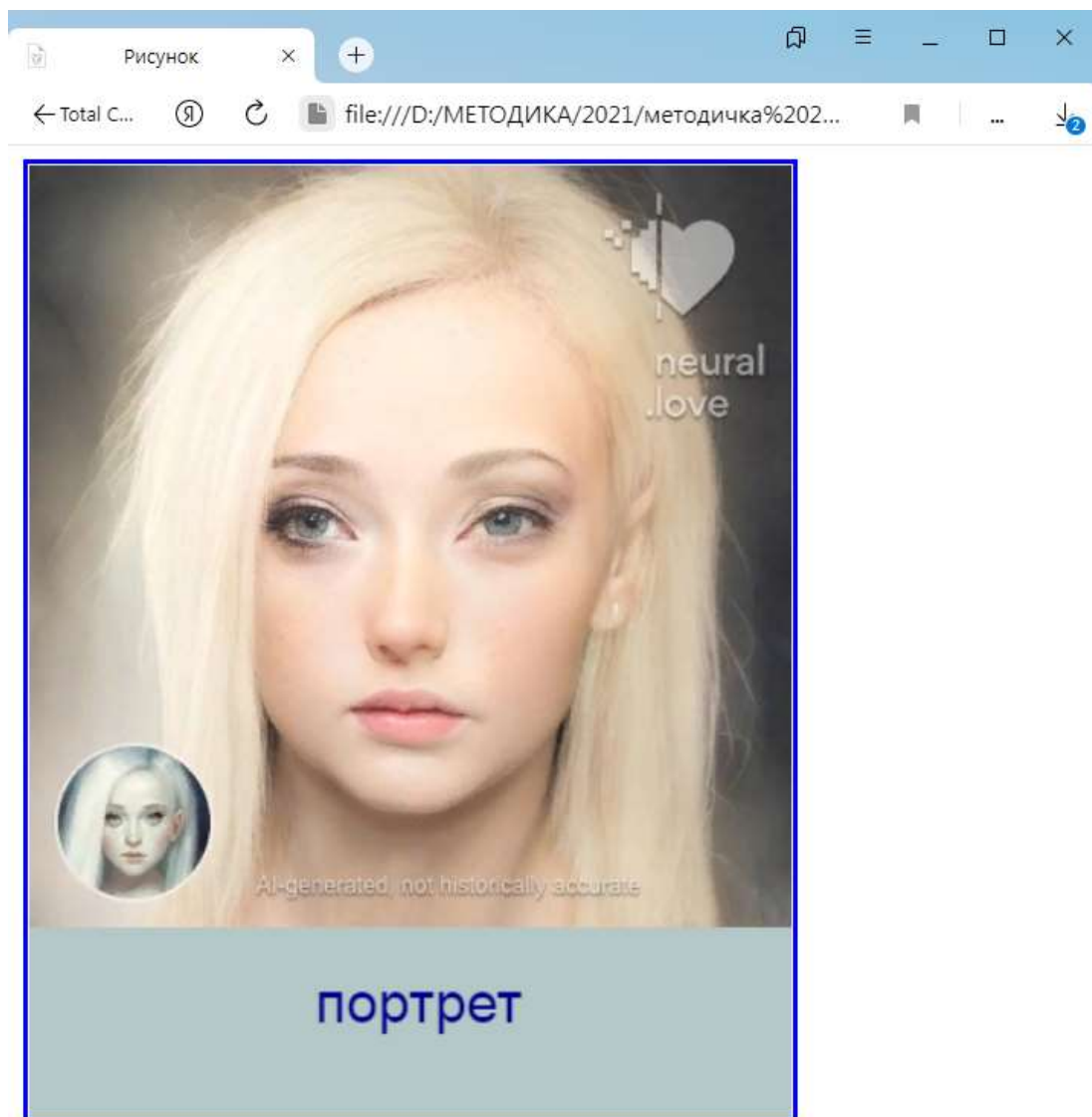
```

ctx.drawImage(pict,0,0)
}
cnv.onclick=function()
{
set()
} }
function set()
{
ctx.fillStyle="rgb(180,200,200)"
ctx.fillRect(0,0,cnv.width, cnv.height)
pict.src="portrait-preview.jpg"
ctx.fillStyle="darkblue"
ctx.fillText("портрет",150,450)
}
</script> </head> <body>
<canvas height="500", width="400" id="mycanvas" style="border:3px solid
#0000ff;"></canvas>
</body>
</html>

```

Результат:





Пояснения к коду:

Создание объекта изображения `pic=new Image()` производится в обработчике события, связанного с загрузкой документа `window.onload`. Внутри этого же обработчика добавляется получение ссылки на объект графической области `cnv=document.getElementById("mycanvas")`; всплывающая подсказка для графической области (при наведении курсора мыши) – метод `cnv.title`; размер и тип шрифта для надписи `ctx.font`. После чего происходит вызов функции `set()`, выполняющей заливку графической области и рисование текста.

В том же обработчике загрузки документа вызывается обработчик события, связанного с загрузкой изображения из файла: `picl.onload`, в котором затем вызывается метод отрисовки изображения `ctx.drawImage(picl,0,0)`.

В первоначальном загрузчике также происходит обработка события, связанного со щелчком мышью в графической области `cnv.onclick`, где заново производится вызов функции `set()`, выполняющей заливку графической области и рисование текста.

Наконец, содержимое функции `set()`:

```
ctx.fillStyle="rgb(180,200,200)" //установка светло-голубого цвета для заливки прямоугольника
```

```
ctx.fillRect(0,0,cnv.width, cnv.height) // рисование закрашенного прямоугольника от начальной точки с координатами (0,0) до нижнего угла графической области
```


```
picl.src="portrait-preview.jpg" //вставка файла изображения в графическую область
```

```
ctx.fillStyle="darkblue" //установка темно-синего цвета для надписи
```

```
ctx.fillText("портрет",150,450) //надпись под рисунком
```

## 8.6. Скрипты, содержащие графические примитивы

### Задача №1

Построение ромба из одномерного массива точек	
<pre>&lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt; &lt;title&gt;Web Page Design&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;canvas id="canvas" width="500" height="500"&gt;&lt;/canvas&gt; &lt;script&gt; var poly=[100,200, 150,100, 200,200, 150,300 ]; var canvas=document.getElementById("canvas") var ctx = canvas.getContext('2d'); ctx.beginPath(); ctx.moveTo(poly[0], poly[1]); for( item=2 ; item &lt; poly.length-1 ; item+=2 ) {ctx.lineTo( poly[item] , poly[item+1] )} ctx.closePath(); ctx.stroke(); &lt;/script&gt; &lt;/body&gt; &lt;/html&gt;</pre>	

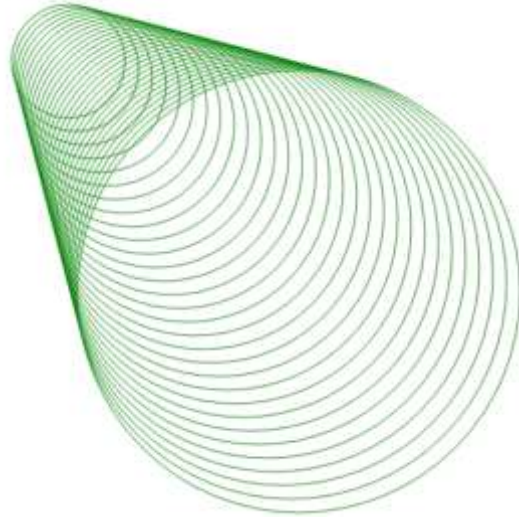
Пояснения к коду:

- 1) массив точек можно называть любым именем, не только points.
- 2) в скрипте используется одномерный массив, но в вызове метода рисования линии используются две координаты.
- 3) для однотипных параметров координат применяется цикл, в котором берутся точки [2, 3], [4,5] и т.д.

## Задача №2

Изобразить смещенные окружности (подзорная труба)

```
<!DOCTYPE html>
<html><head>
<title>Web Page Design</title>
</head>
<body>
<canvas id="canvas" width="800"
height="800"></canvas>
<script>
var x=200;
var y=200;
var xr=50;
var canvas=document.getEle-
mentById("canvas")
var ctx = canvas.getContext('2d');
ctx.strokeStyle="Green"
for( var i=0 ; i < 30 ; i++ )
{ctx.beginPath();
ctx.arc(x,y,xr,0,360)
x=x+7
y=y+7
xr=xr+5
ctx.closePath();
ctx.stroke();
}
</script> </body> </html>
```



Пояснения к коду:

- 1) Параметр цикла показывает количество окружностей ( $i < 30$ ).
- 2) В теле цикла меняются координаты центра и радиус окружности.

Задача №3.

Начертить график функции  $y=(x+1)(x-2)(x-3)$  на интервале  $(-2; 6)$ .

1 способ – с помощью построения коротких отрезков (в цикле).

```
<!DOCTYPE html>
<html><head>
<title>Web Page Design</title>
</head>
<body>
<canvas id="canvas" width="520" height="520"></canvas>
<script>
var can = document.getElementById('canvas');
var ctx = can.getContext('2d');
var step = 20;//шаг решетки
//функция
function func(x) {
    return((x+1)*(x-2)*(x-3));
}
function drawgrid(){
    for (var i = step; i<can.width; i+=step)
    {//вертикальные
        ctx.beginPath();
        ctx.strokeStyle = 'black';
        ctx.lineWidth = 1;
        ctx.moveTo(i, 257);
```

```

    ctx.lineTo(i, 263);
    ctx.closePath();
    ctx.stroke();
}
for (var i = step; i<can.height; i+=step)
{ //Горизонтальные
    ctx.beginPath();
    ctx.moveTo(257, i);
    ctx.lineTo(263, i);
    ctx.closePath();
    ctx.stroke();
}
}
function drawAxle(){
    // ось X
    ctx.beginPath();
    ctx.moveTo(0, can.height/2);
    ctx.lineTo(can.width, can.height/2);
    ctx.strokeStyle = 'black';
    ctx.lineWidth = 1;
    ctx.closePath();
    ctx.stroke();
    // ось Y
    ctx.beginPath();
    ctx.moveTo(can.width/2, 0);
    ctx.lineTo(can.width/2, can.height);
    ctx.strokeStyle = 'black';
    ctx.closePath();
    ctx.stroke();
}

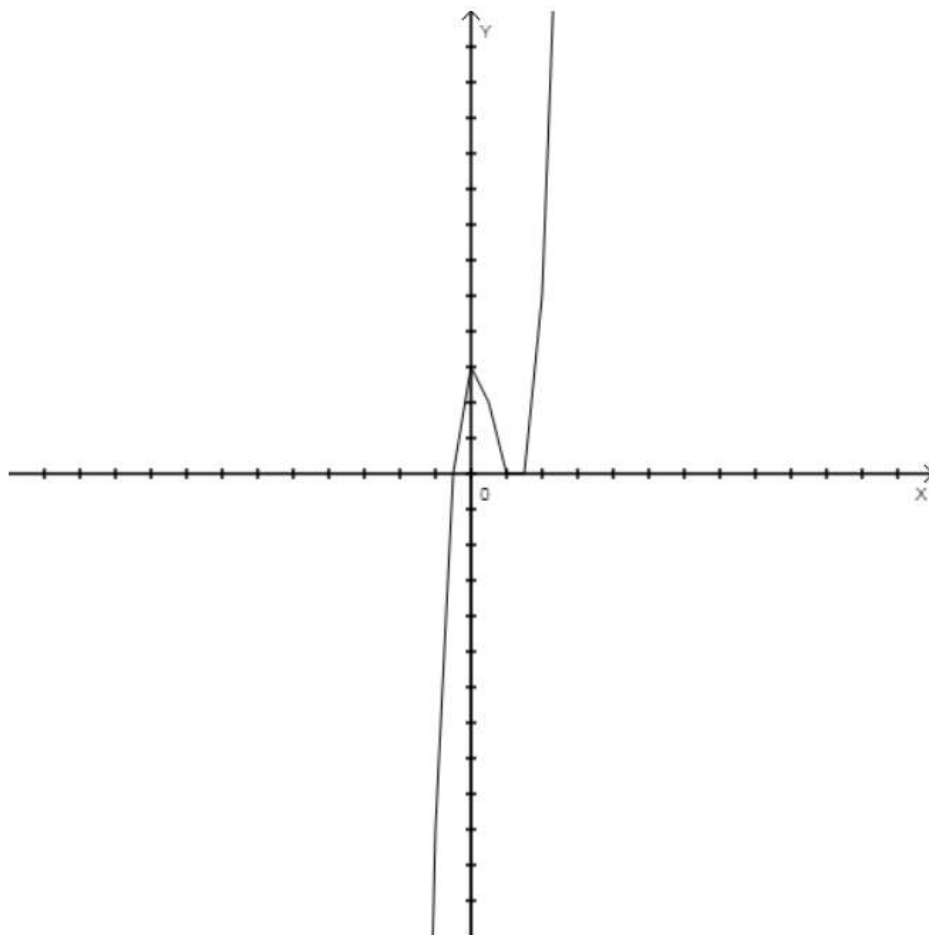
```

```

}
function drawGraph(){
//график из отрезков
ctx.strokeStyle = "black";
ctx.beginPath();
cx = can.width / 2;
cy = can.height / 2;
ctx.moveTo(cx, cy);
for (i = -500; i<500; i++) {
  x = i;
  y = -func(i);
  ctx.lineTo(cx+10*x, cy+10*y);
}
ctx.lineWidth = 1;
ctx.stroke()
}
function ar_text()
{
//подписи и стрелки
ctx.beginPath();
ctx.moveTo(515,255);
ctx.lineTo(520,260);
ctx.lineTo(515,265);
ctx.moveTo(255,5);
ctx.lineTo(260,0);
ctx.lineTo(265,5);
  ctx.strokeStyle = 'black';
  ctx.stroke();
ctx.fillText("X", 510,275);

```

```
ctx.fillText("Y", 265,15);
ctx.fillText("0", 265,275);
}
drawgrid();
drawAxle()
drawGraph();
ar_text();
</script> </body> </html>
```



2 способ – с помощью модуля (библиотеки) chart.js.

```
<!-- Подключаем JQuery и Chart.js -->
<script type="text/javascript"
```

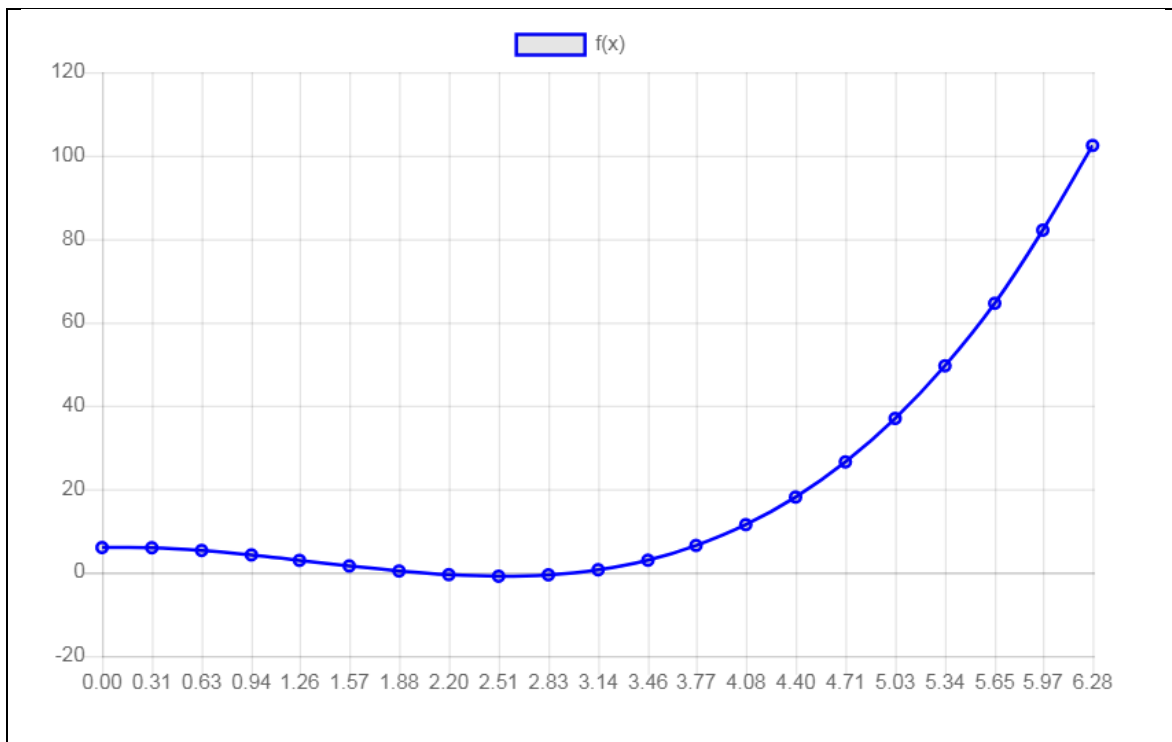


```
src="https://ajax.goog-
leapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script type="text/javascript" src="https://cdnjs.cloud-
flare.com/ajax/libs/Chart.js/2.7.2/Chart.bundle.min.js"></script>
<!-- Готовим контейнер для диаграммы -->
<div id="content" align="center">
  <canvas id="myChart" width="628" height="400"></canvas>
</div>
<script type="text/javascript">
  //Готовим диаграмму
  function Diagram () {
    var ctx = document.getElementById("myChart");
    var myChart = new Chart (ctx, {
      type: 'line',
      data: {
        labels: [], //Подписи оси x
        datasets: [
          {
            label: 'f(x)', //Метка
            data: [], //Данные
            borderColor: 'blue', //Цвет
            borderWidth: 2, //Толщина линии
            fill: false //Не заполнять под графиком
          }
        ]
      },
      //Можно добавить другие графики
    ],
    options: {
      responsive: false, //Вписывать в размер canvas
```

```

scales: {
  xAxes: [{
    display: true
  }],
  yAxes: [{
    display: true
  }]
} }
});
//Заполняем данными
for (var x = 0.0; x<=2*Math.PI; x+=Math.PI/10) {
  myChart.data.labels.push("+x.toFixed(2));
  myChart.data.datasets[0].data.push(f(x).toFixed(2));
}
//Обновляем
myChart.update();
function f(x) { //Вычисление нужной функции
  return((x+1)*(x-2)*(x-3));
}
}
//Ставим загрузку диаграммы на событие загрузки страницы
window.addEventListener("load", Diagram);
</script>

```



#### Задача №4

Нарисовать зигзаг из линий. В программе задаются координаты начальной точки (x,y), а также расстояние между зубцами по горизонтали и высота зубцов зигзага. Вывести это значение на экран.

```

<!DOCTYPE html>
<html><head>
<title>Web Page Design</title>
</head>
<body>
<canvas id="canvas" width="600" height="600"></canvas>
<script>
var x=parseInt(prompt());
var y=parseInt(prompt());
var a=parseInt(prompt());
var canvas=document.getElementById("canvas")
var ctx = canvas.getContext('2d');

```

```
ctx.beginPath();
ctx.strokeStyle="black"
for( var i=0 ; i < 11 ; i++ )
{
ctx.moveTo(x,y);
ctx.lineTo(x+a/2,y-a);
ctx.moveTo(x+a/2,y-a);
ctx.lineTo(x+a,y);
x=x+a;
}
ctx.stroke();
ctx.fillText(a.toString(), 50,25);
</script> </body> </html>
```

Результат:



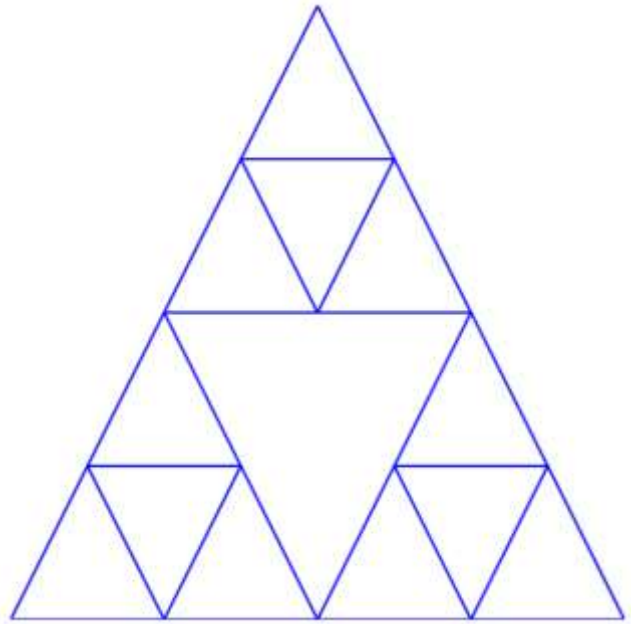
Задача №5

Нарисовать треугольник Серпинского

```

<!DOCTYPE html>
<html>
<head>
  <title>Треугольник
Серпинского</title>
</head>
<body>
  <canvas id="canvas" width="500"
height="500"></canvas>
  <script>
    var depth = Num-
ber(prompt('Введите глубину
рекурсии'));
    var canvas = document.getEle-
mentById("canvas");
    var ctx = canvas.getCon-
text("2d");
    var p0 = { //координаты вер-
шин начального треугольника
      x: 0,
      y: 500 },
    p1 = {
      x: 250,
      y: 0 },
    p2 = {
      x: 500,
      y: 500 };

```



```

function drawTriangle (p0, p1,
p2) //функция для отрисовки тре-
угольника

    ctx.beginPath();
    ctx.moveTo(p0.x, p0.y);
    ctx.lineTo(p1.x, p1.y);
    ctx.lineTo(p2.x, p2.y);
    ctx.lineTo(p0.x, p0.y);
    ctx.strokeStyle = "blue";
    ctx.lineWidth = 2;
    ctx.stroke();
}

function drawFract (p0, p1, p2,
limit){ //лимит - до какого момента
будет выполняться рекурсия
    if( limit > 0 ){
        var pA = { // середины
каждой пары вершин
            x: (p0.x + p1.x) / 2,
            y: (p0.y + p1.y) / 2
        },
        pB = {
            x: (p1.x + p2.x) / 2,
            y: (p1.y + p2.y) / 2
        },
        pC = {
            x: (p2.x + p0.x) / 2,
            y: (p2.y + p0.y) / 2
        };

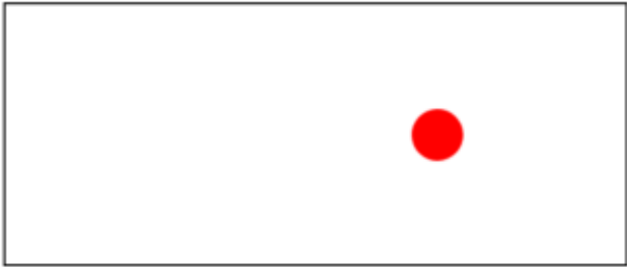
```

<pre> drawFract(p0, pA, pC, limit-1); //рекурсивный вызов функции     drawFract(pA, p1, pB, limit-1);     drawFract(pC, pB, p2, limit- 1);     } else {         drawTriangle(p0,p1,p2);     } } drawFract(p0, p1, p2, depth-1) &lt;/script&gt; &lt;/body&gt; &lt;/html&gt; </pre>	
---	--

Пояснения к коду:

В треугольнике проводятся три средние линии. В результате он разбивается на четыре новых треугольника. К трем из них, примыкающим к вершинам первоначального треугольника, применяется та же рекурсивная функция.

Задача №6.

<p>Организовать горизонтальное движение шарика внутри прямоугольника. Шарик движется до правой границы прямоугольника, отталкивается от нее, движется влево, отталкивается от левой границы прямоугольника и т. д., пока не будет нажата клавиша.</p>	
<pre> &lt;!DOCTYPE html&gt; &lt;html&gt;&lt;head&gt; &lt;title&gt;Web Page Design&lt;/title&gt; &lt;/head&gt; &lt;body&gt; </pre>	

```
<canvas id="canvas" width="480"
height="200" style="border:2px
solid #000000;">
</canvas>
<script>
var canvas = document.getEle-
mentById("canvas");
var ctx = canvas.getContext("2d");
var ballRadius = 20;
var ball = {x:0, y: 100} // коорди-
наты центра мяча
  ,speed = {x: -10, y: 28} // ско-
рость мяча
  ,box = {x: 480, y: 200} // ши-
рина/высота коробки
;
function update(p) { // обновляет
координаты
  ball[p] += speed[p]/10; // с учётом
скорости
  if( ball[p] > box[p]) { // вылетели
за дальний конец коробки?
    ball[p] = 2 * box[p] - ball[p];
    speed[p] *= -1;
  } else if( ball[p] < 0) { // вылетели
за другой конец?
    ball[p] = -ball[p];
    speed[p] *= -1;
  }
}
```



```
}  
function draw() {  
  update('x');  
  ctx.clearRect(0, 0, box.x, box.y); //  
очистить экран  
  ctx.beginPath();  
  ctx.arc(ball.x, ball.y, ballRadius,  
20, Math.PI * 2, true);  
  ctx.fillStyle = "red";  
  ctx.fill();  
ctx.closePath();  
}  
setInterval(draw, 10);  
</script>  
</body>  
</html>
```

## 8.7. Контрольные вопросы и тесты

### Дополнительные вопросы

1. Можно ли на холсте изобразить окружность, если в JS имеется только метод рисования дуги?
2. Когда используется метод fill(), а когда stroke()?
3. Обязательно ли закрывать траекторию изображения методом closePath() после создания изображения?
4. Какое понятие лежит в основе построения фракталов?
5. Можно ли на холст вставить готовое изображение, а поверх него нарисовать что-нибудь?
6. В какую часть графического экрана можно вывести текст?

7. Какие методы применяются для анимации изображения?
8. Можно сначала изобразить незакрашенный прямоугольник, а потом закрасить его отдельной командой?
9. Какой массив, одномерный или двумерный, используется при рисовании многоугольника по его вершинам?
10. Можно ли изменить размер шрифта для надписи в графической области?

### Тесты с выбором варианта ответа

1. Холст для рисования на веб-странице:
  - 1) picturebox;
  - 2) image;
  - 3) canvas;
  - 4) draw.
2. Связать графическую область с инструментами изображения можно при помощи:
  - 1) операционной системы;
  - 2) контекста отображения;
  - 3) формы веб-страницы;
  - 4) диалогового окна.
3. Для создания незакрашенного прямоугольника в JS используется метод:
  - 1) fillRect();
  - 2) strokeRect();
  - 3) rectangle();
  - 4) lineTo().

4. Какое слово не описывает команду `beginPath()`?

- 1) контур;
- 2) путь;
- 3) отрезок;
- 4) траектория.

5. С помощью массива с координатами точек можно изобразить:

- 1) любую кривую;
- 2) только ромб;
- 3) окружность;
- 4) многоугольник.

6. При изображении дуги, выпуклой вверх, последний параметр метода:

- 1) `false`;
- 2) `true`;
- 3) `Math.PI`;
- 4) не используется.

7. Текст на изображение можно поместить методом:

- 1) `fillText(s,x,y)`;
- 2) `TextOut(x,y,s)`;
- 3) `Stroke.ToString(s)`;
- 4) `fillStyle="text"`.

8. Создание объекта изображения имеет вид:

- 1) `pict=new Image()`;
- 2) `image=new Object()`;
- 3) `cnv = canvas()`;
- 4) `ctx=context()`.

9. Метод отрисовки изображения:

- 1) `cnv.drawImage(ctx,x,y);`
- 2) `ctx.fillText(pict,x,y);`
- 3) `ctx.drawImage(pict,x,y);`
- 4) `ctx.fillRect(pict,x,y,width,height);`

10. Команда модуля `turtle.js` движение вперед на `x` пикселей:

- 1) `backward(x);`
- 2) `forward(x);`
- 3) `rotate(x);`
- 4) `home(x);`

11. Графические координаты определяются в...

- 1) пунктах;
- 2) дюймах;
- 3) пикселях;
- 4) миллиметрах.

12. Выполнение команды вида `pict.src` приводит:

- 1) к привязке изображения к определенной графической области;
- 2) к непосредственному отображению изображения в документе;
- 3) к передаче ссылки на объект изображения;
- 4) к началу загрузки изображения в документ.

### Тесты без выбора варианта ответа

1. С помощью метода `.strokeStyle="rgb(r,g,b)"` построить линию:

- А) оранжевого цвета;
- Б) серого цвета.

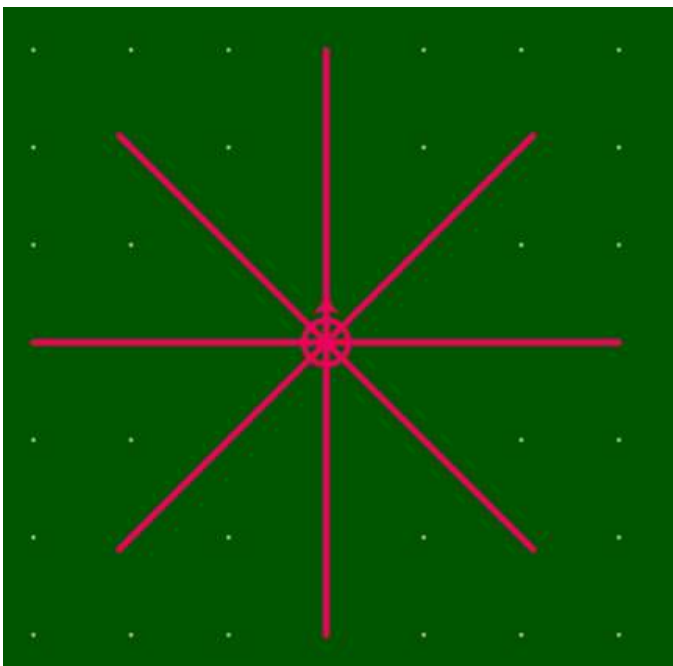
2. Добавить в каждый из рассмотренных скриптов метод `setInterval()` по смыслу:

- А) дым из трубы дома;
- Б) вращение ромба;
- В) смена цвета смещенных окружностей;

3. Дан скрипт модуля «Черепашка» на сайте <http://blockly.ru/apps/js-turtle/index.html>:

```
for (var count = 0; count < 8; count++) {  
  Turtle.penColour('#e8065c');  
  Turtle.moveForward(150);  
  Turtle.moveBackward(150);  
  Turtle.turnRight(45);  
}
```

для изображения следующей фигуры:



Переделать этот скрипт с учетом функций модуля turtle.js.

### **8.8. Задачи для самостоятельной работы**

- 1) Нарисовать 5 олимпийских колец.
- 2) Нарисовать из прямых линий кораблик. Иллюминатор изобразить с помощью окружности.
- 3) Нарисовать из прямых линий ракету. Иллюминатор изобразить с помощью окружности.
- 4) Нарисовать из прямоугольников трехцветный флаг России.
- 5) Из окружностей и дуг изобразить смайлик.
- 6) Из прямоугольников изобразить паровоз. Колеса изобразить с помощью окружностей.
- 7) Из прямоугольников изобразить грузовик. Колеса изобразить с помощью окружностей.
- 8) Нарисовать елку из треугольников.
- 9) Изобразить пятиконечную звезду.
- 10) Изобразить равнобедренную трапецию.
- 11) Изобразить вишню: плоды – восьмиугольники, листья – параллелограммы, ветка – линии.
- 12) Нарисовать заштрихованный треугольник.
- 13) Изобразить по точкам пятиугольник.
- 14) Изобразить рыбок и воздушного змея с помощью выпуклых дельтоидов.
- 15) Изобразить по точкам семиугольник.
- 16) Изобразить по точкам четырехугольную наклонную призму.
- 17) Изобразить ромбовидную спираль от края к центру.

- 18) Изобразить нотный стан – 12 групп по 5 линий на странице.
- 19) Изобразить цветные линии, выбрав цвет случайным образом.
- 20) Изобразить звездное небо из точек разного цвета.
- 21) Нарисовать пирамиду из треугольников, направленных вершиной вниз.
- 22) Изобразить динамические прямоугольники с изменяющимися сторонами от большего к меньшему.
- 23) Нарисовать диагональную спираль, длина каждого следующего звена которой увеличивается на единицу.
- 24) Подготовить лист в косую линейку, как в тетради-прописи для младших школьников (с полями).
- 25) Ввести радиусы  $R_1$ ,  $R_2$ . Нарисовать кольцо и указать около него, какой из радиусов больше, а какой меньше.
- 26) Запросить с клавиатуры число  $x$ . При  $x=3$  нарисовать треугольник, при  $x=4$  – квадрат, при  $x=5$  – пятиугольник из линий.
- 27) Нарисовать цветок, лепестками которого будут служить прямые линии. Размер лепестков и их количество ввести с клавиатуры.
- 28) Нарисовать цепочку из квадратов, расположенных на диагонали экрана. С клавиатуры вводятся значение ребра квадрата и значение ребра малого квадрата, получающегося при наложении квадратов друг на друга.
- 29) Нарисовать шахматную доску  $8 \times 8$ , начиная со светлого квадрата. Сверху клетки подписать восьмью латинскими буквами. Снизу – цифрами.
- 30) Нарисовать гроздь винограда. С клавиатуры задается число рядов ягод и их радиус.
- 31) Нарисовать мишень из концентрических окружностей. В центре поместить число «100», а по краям – 50, 20, 10 и 5 соответственно.
- 32) Нарисовать треугольник и его высоту. Расставить точки вершин – А, В, С и отметить прямой угол.

33) Изобразить окружность заданного с клавиатуры радиуса  $R$ . В центре окружности вывести  $O$ , а в левом верхнем углу – площадь этой окружности.

34) Запросить с клавиатуры число  $x$ . При  $x=1$  на экран выводится прямоугольник, а при  $x=2$  изображается эллипс. Подписать выведенную фигуру.

35) Запросить с клавиатуры число  $x$ . При  $x=1$  на экран выводится закрашенный прямоугольник красного цвета, при  $x=2$  – зеленого цвета, при  $x=3$  – прямоугольник синего цвета.

36) Нарисовать квадратную спираль. Начало координат, длина звена и количество витков вводятся с клавиатуры.

37) Изобразить столбиковую диаграмму из четырех прямоугольников разных цветов. Справа поместить легенду с названием диаграммы и описанием цвета. Например, «Курсы». Красный – первый, синий – второй, зеленый – третий, черный – четвертый.

38) Изобразить круговую диаграмму из четырех секторов окружности. Справа поместить легенду с названием диаграммы и описанием цвета. Например, «Предметы». Красный – математика, синий – физика, зеленый – информатика, черный – химия.

39) Изобразить блок-схему алгоритма «ветвление». Внутри ромба написать слово «условие», а внутри прямоугольника – «команда».

40) Нарисовать штриховку, используя функции рисования прямоугольника и линии. Количество линий задается с клавиатуры.

41) Нарисовать снежинку Коха.

42) Нарисовать кривую Коха.

43) Нарисовать ковёр Серпинского.

44) Нарисовать пятиугольник Дюрера.

45) Нарисовать кривую Минковского.

46) Нарисовать кривую дракона.



- 47) Нарисовать фрактал Мандельброта.
- 48) Нарисовать фрактал «коробка».
- 49) Нарисовать снежинку из 6 лучей.
- 50) Нарисовать вложенные восьмиугольники.
- 51) Нарисовать ступенчатую пирамиду.
- 52) Случайным образом заполнить экран квадратами различных цветов и размеров.
- 53) Изобразить окружность в виде правильного 60-угольника.
- 54) Изобразить скругленную спираль, идущую от центра.
- 55) Нарисовать ряд правильных  $n$ -угольников, центры которых лежат на горизонтальной прямой на равном расстоянии друг от друга. Сторона каждого следующего  $n$ -угольника больше стороны предыдущего на одно и то же число.
- 56) Нарисовать спираль из правильных  $n$ -угольников, поворачивающихся относительно своей вершины на один и тот же угол. Сторона каждого следующего  $n$ -угольника больше стороны предыдущего на одно и то же число.
- 57) Нарисовать фигуру, которая получается вращением прямоугольника вокруг центра симметрии.
- 58) Нарисовать кубик, используя изображение квадрата для грани.
- 59) Проиллюстрировать биологический цикл жизни дерева: дерево зеленеет и растет, потом желтеет и увядает (уменьшается в размере).
- 60) Нарисовать цветик-семицветик, у которого поочередно после нажатия клавиши улетает лепесток (исчезает с экрана).
- 61) Организовать зигзагообразное движение мячика внутри трубы слева направо.
- 62) Смоделировать скачки мяча, брошенного на поверхность: мяч совершает несколько движений с максимальной амплитудой, затем амплитуда уменьшается, и мяч постепенно останавливается.

63) Написать программу, в которой Колобок (шарик) поднимается по ступенькам вверх, движется по площадке и падает вниз.

64) Нарисовать циркового клоуна, у которого из стороны в сторону движутся глаза.

## Список используемых источников

1. Мишенин А.И. Сборник задач по программированию: учеб. пособие / А.И. Мишенин. – М.: Финансы и статистика; ИНФРА-М, 2009. – 224 с.
2. Златопольский Д.М. Сборник задач по программированию. – 3-е изд., перераб. и доп. – СПб: БХВ-Петербург, 2011. – 304 с.: ил.
3. Могилев А.В. и др. Практикум по информатике: Учеб. пособие для студ. высш. учеб. заведений / А.В. Могилев, Н.И. Пак, Е.К. Хеннер; Под ред. Е.К. Хеннера. – М.: Издательский центр «Академия», 2002. – 608 с.
4. Расторгуев И.С., Никольский А.П. Привет, JavaScript! Моя первая книга по программированию. – СПб.: «Наука и техника», 2020. – 256 с.: ил.
5. Прохоренок Н.А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. – 5-е изд., перераб. и доп. / Н.А. Прохоренок, В.А. Дронов. – СПб.: БХВ-Петербург, 2021. – 912 с.: ил.
6. Васильев А.Н. JavaScript в примерах и задачах / Алексей Васильев. – М.: Эксмо, 2019. – 720 с.