**[Основы программирования – loftblog](https://www.youtube.com/watch?v=1oJcf6Q_VEc&list=PLY4rE9dstrJwoVF-svoIBhKzE-Ok39Zg_)**

Содержание:

[1. Основы программирования – #0 – Введение………………………………………………**……………………………………**… 1](#Урок0)

[2. Основы Программирования – #1 – Логика**.** Алгоритмы……………………………………………………………… 1](#Урок1)

[3. Основы Программирования – #2 – Типы данных……………………………………………………………………………**…** 2](#Урок2)

[4. Основы Программирования – #3 – Основные структуры данных………………………………………… 7](#Урок3)

[5. Основы Программирования – #4 – Функции……………………………………………………………………………………… 10](#Урок4)

[6. Основы Программирования – #5 – Арифметические и логические выражения……… 13](#Урок5)

[7. Основы Программирования – #6 – Условия……………………………………………………………………………………**…** 15](#Урок6)

[8. Основы Программирования – #7 – Циклы…………………………………………………………………………………………… 18](#Урок7)

[9. Основы Программирования – #8 – Инструменты для программиста……………………………… 24](#Урок8)

[10. Основы Программирования – #9 – Графика…………………………………………………………………………………… 25](#Урок9)

[11. Основы Программирования – #10 – События……………………………………………**……………………………………** 27](#Урок10)

[12. Основы Программирования – #11 – Первая программа **(**Калькулятор на JavaScript**)**……………………………………………………………………………………………………………………………………………………………………… 27](#Урок11)

[13. Основы Программирования – #12 – Баги……………………………………………………**……………………………………** 32](#Урок12)

[14. Основы Программирования – #13 – Заключение………………………………………………………………………… 33](#Урок13)

**[Основы программирования – #0 – Введение](#_top)**

**Программист** – человек, который пишет команды для устройств, которые выполняют определенные действия.

Нас везде окружают программируемые устройства.

Даже самый сложный процесс можно разбить на подзадачи и добиться его постепенно!!!

**[Основы Программирования – #1 – Логика. Алгоритмы](#_top)**

**Логика** – это правильное, рациональное мышление.

**Алгоритм** – это последовательность команд, то есть набор инструкций описывающий порядок действий для достижения конкретного результата.

**Программа** – это набор инструкций, которые идут друг за другом.

**Программа** – это алгоритм + структура данных!!!

**Команды (инструкции)** – называются statements – это наименьшая часть программы выполняющая конкретный программный код!!!

**Команда** – это задача, которую компьютер должен выполнить!!!

Из них состоит вся программа!!!

Чтобы компьютер понял команды для него нужно максимально подробно их писать!!!

Чем больше инструкций программы, тем программа лучше будет отвечать ожидаемым от нее параметрам!!!!

Программирование как и реальная жизнь не стоит на месте, а постоянно изменяется, расширяется функционал и т.д.

**Программирование** – это очень жизненная вещь, требующая постоянного дополнения – этот процесс бесконечный!!!

**Алгоритмы** бывают трех видов:

1. **Линейный** – это когда каждое действие идет друг за другом.

действие\_1 **->** действие\_2 **->** действие\_3 **->** действие\_4 **->** и т**.**д**.**

2. **Циклический** – когда действия повторяются по циклу

**|-->** **действие\_1** **->** **действие\_2** **->** **действие\_3** **->-----|**

**|--------<---<---<--действие\_4---<---<---<---<-----|**

3. **Ветвление** – выбор логики действий:

 **|->** на лево**;**

Идти **|->** на право**;**

 **|->** прямо**.**

4. **Комбинированный** – все три вида соединяются, переплетаются и т.д.

Каждая **программа** – это сложный набор инструкций, где есть все эти алгоритмы!!!

**Программирование** – это креативность прежде всего.

Если программа написана с малым набором команд – она легче, но она сложнее в доработке!!!

И наоборот – с большим количеством – тяжелее на оперативную память, но ее потом намного легче доработать и усовершенствовать.

**DRY (don't repeat yourself)** – никогда не повторяй себя!!!!

Все программы – это порядок действий!!!

**[Основы Программирования – #2 – Типы данных](#_top)**

Всю информацию, которую нам можно описать с помощью программирования нам можно осуществить с помощью данных.

**Данные** – это информация формализованная для передачи и обработки компьютером!!!

У данных бывают:

1. Типы – всего три:

1) Целое число **int** **(**integer**)**

2) Дробное число (с плавающей точкой после запятой) **float**

3) **number** – это просто число и целое и дробное в JavaScript

4) **double** – числа с плавающей точкой двойной точности!!!

5) **Строка** – фрагмент текста, последовательность символов в строке!!!

Она может состоять из букв, символов, чисел, знаков – из любых символов!!!

Обычно в одинарных или двойных кавычках 'строка' и "строка" – если строка началась с одинарных то и заканчивать нужно также.

Одинарные кавычки называются апострофы!!!!

Если апострофы нужны внутри строки!!!!

'it's cloudy day'

То применяется метод экранирования \'

'it\'s cloudy day'

Теперь эту строку компьютер интерпретирует правильно!!!

Строки можно склеивать

Команды в языках программирования

**char** – строка в phyton

**string** – строка в JavaScript

У каждого символа в строке есть свой номер, который соответствует его позиции. Его можно использовать для того, чтобы узнать отдельный символ или вырезать его с определенной строки.

Отсчет ведется с нуля!!!!

Например

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| з | е | л | е | н | ы | й |

Символ "н" соответствует номеру 4!!!!

Обычно индекс указывают в квадратных скобках!!!!

Если сделать два индекса можно получить срез строки!!!

Например

[4:6] = "ный"

В python

**print(**type**(**24**))**

**print(**type**(**24.3**))**

Консоль

**<**type 'int'**>**

**<**type 'float'**>**

**[**Finished **in** 1.4s**]**

Пример 2:

**print(**type**(**24**))**

**print(**type**(**24.3**))**

**print(**'It\'s cloudy day'**)**

Результат в окне консоли:

**<class** 'int'**>**

**<class** 'float'**>**

It's cloudy day

**[**Finished **in** 0.1s**]**

Вывод символа строки

**print(**'LOFTBLOG'**[**4**])**

Результат

B

Вывод отрезка строки

print**(**'LOFTBLOG'**[**4**:**8**])** #вывод с 4 по 8 символов!!!

Результат

BLOG

Пример 2

**print(**'LOFTBLOG'**[:**4**])** #вывод с 0 по 4 символов!!!

**print(**'LOFTBLOG'**[**4**:])** #вывод с 4 по последний символ!!!

Результат

LOFT

BLOG

В JavaScript

В конце строчки нужно обязательно ставить ";" в отличии от Python

console**.**log**();** //выведение в консоль

**.**charAt**(**n**);** //выведение из строки символа номер n

Например

console**.**log**(**'LoftBlog'**.**charAt**(**4**));**

Результат

"B"

Обрезать строку можно с помощью

Метод 1 – команда substring**(**n**)**

substring**(**n**);** //выделение всех символов после n

Например

console**.**log**(**'LoftBlog'**.**substring**(**4**));**

Результат

"Blog" //выделение всех символов после 4!!!

Пример 2

console**.**log**(**'LoftBlog'**.**substring**(**4**,**6**));** //выделение всех символов с 4 по 6!!!

Результат

"Bl"

Второй метод – slice**()**

console**.**log**(**'LoftBlog'**.**substring**(**4**));** //выделение всех символов после 4!!!

Консоль:

"Blog"

В этом методе можно указывать минусовые значения!!!

console**.**log**(**'LoftBlog'**.**substring**(-**6**));** //выделение 6-ти символов от конца строки!!!

Результат

"ftBlog"

Третий метод – substr**(**n**,**m**)**

substr**(**1**,**4**)** //здесь обязательно нужно указывать начальную позицию 1 и сколько символов нужно отрезать = 4!!!

Например

console**.**log**(**'LoftBlog'**.**substr**(**2**,**5**));**

Результат

"ftBlo"

3) Булевой тип данных (логический) – **ИСТИНА** или **ЛОЖЬ**

Вопрос, принимающий только один вариант ответа – ДА или НЕТ.

В языках программирования они имеют всего два значения

**true** – **ИСТИНА**

**false** – **ЛОЖЬ**

В Python тип данных определяется с помощью команды type**()**

**print(**type**(True))**

**print(**type**(False))**

Результат

**<class** 'bool'**>** #булевой(логический) тип данных

**<class** 'bool'**>** #булевой(логический) тип данных

В JavaScript тип данных определяется с помощью команды **typeof**

console**.**log**(typeof** **true);** //определение типа данных

console**.**log**(typeof** **false);** //определение типа данных

Результат

"boolean" //булевой(логический) тип данных

"boolean" //булевой(логический) тип данных

Что в JavaScript что в Python символы приравниваются с помощью двойного равно «==»

Например в JavaScript

console**.**log**(**3**==**3**);**

console**.**log**(**3**==**4**);**

Результат

**true**

**false**

Например в Python

**print(**3**==**3**)**

**print(**3**==**4**)**

Результат

**True**

**False**

Определение типов данных в Python

**print(**type**(**123.24**))**

Результат

**<class** 'float'**>**

Или строка

**print(**type**(**'Privet'**))**

Результат

**<class** 'str'**>**

Или булевое значение

**print(**type**(True))**

Результат

**<class** 'bool'**>**

Перевод из одного типа в другой

1. Команда **int()** - integer преобразует дробные числа в целые!!!

Например

**print(**int**(**45.456**))**

Результат

45

2. Команда **float()** преобразует целое число в дробное!!!

Например

**print(**float**(**45**))**

Результат

45.0

Строка

**print(**type**(**str**(**45**)))**

Консоль

**<class** 'str'**>**

3. Команда **bool()** преобразует в булевое значение!!!

**print(**bool**(**45**))** #число преобразуется в значение ИСТИНА

**print(**bool**(**0**))** #0 преобразуется в значение ЛОЖЬ

Консоль

**True**

**False**

Преобразование типов данных в JavaScript

console**.**log**(typeof(**15**));**

console**.**log**(typeof(**'Петр Николаевич'**));**

console**.**log**(**String**(**'152323'**));**

console**.**log**(**Boolean**(**45**));**

console**.**log**(**Number**(**'Петр Николаевич'**));**

console**.**log**(**Number**(true));**

console**.**log**(**Number**(false));**

Консоль

"number"

"string"

"152323"

**true**

NaN

1

0

В основном всего три типа данных

1. **Строка** – это текстовые данные

2. **Буль** – логическое значение **ИСТИНА** или **ЛОЖЬ**

3. **Число** – числовые данные

**Типизация данных** есть:

1. **Слабая** – невозможность быстро переводить одни типы данных в другие.

В JavaScript слабая типизация данных – нет специальной команды!!!

2. **Сильная** – возможность быстро переводить одни типы данных в другие.

В Python сильная типизация данных – с помощью команды type**()** можем ввести конкретный тип данных и перевести его в другой!!!

2. **Сильная** типизация данных – это определенная надежность разрабатываемой программы!!!

Она защищает программиста от простых, но трудно обнаруживаемых ошибок!!!!

С опечатками и невнимательностью!!!

Есть еще

1. **Динамическая** – когда переменная связывается с типом данных в момент присваивания значения, а не в момент объявления переменной!!!

В языке JavaScript

2. **Статическая** – когда переменная в любом участке кода, если она была привязана к конкретному типу данных, то она с этим топом данных и остается!!! Она не может его ПОМЕНЯТЬ!!!!

В языке Java.

Одна и та же переменная может принимать разные значение и разные типы данных!!!!

2. **Структуры**

В phyton

**print()** – перевести в консоль.

type**()** – тип числа.

Чтобы отобразить в консоли результат нужно нажать:

1. **Ctrl + S** (сохранить)

2. **Ctrl + B** (обновить)

В JavaScript

console**.**log**()**– отобразить в консоли результат

**typeof** – тип числа.

**[Основы Программирования – #3 – Основные структуры данных](#_top)**

**Структура данных** – это структура, позволяющая обрабатывать множество однотипных или логически связанных данных.

Самая простая структура данных – это **переменная**!!!

**Переменная** – это именованные и определенные ячейки памяти, в которые мы можем сохранять любые данные!!! и которые мы можем использовать в нашем коде в дальнейшем!!!!

Переменные нужны для того, чтобы запоминать фрагменты данных в программе!!!!

Для того, чтобы создать или объявить переменную в JavaScript используется ключевое слово

**var** после которого через пробел записывается имя переменной!!!

**var** name

Для того, чтобы записать значение переменной используется оператор присваивания, как знак равно в JavaScript!!!

**var** name **=** "milhouse"

Имя простой переменной характеризует номер ячейки – одной или нескольких, где хранится ее значение!!!!

Простая переменная может быть разного пита данных – число, строка, логические, буль!!!

Существует правило – как можно и как нельзя называть переменные:

- любые буквы и цифры;

- имя не может начинаться с цифры;

- знаки "-"**,** "/"**,** "#"**,** "@" – использовать нельзя!!!!

- пробелы использовать нельзя!!!!

- знак подчеркивания "\_" использовать вместо пробела можно!!!!

- заглавные и строчные буквы – различаются!!!! Например Dogs и dogs – это совершенно разные переменные!!!!

- нельзя имена переменных обозначать именами уже существующих команд!!! Например **print()** в python.

Переменные на языке Python

name **=** 'Milhouse'

**print(**name**)**

Консоль:

Milhouse

**[**Finished **in** 0.1s**]**

Переменные на языке JavaScript

**var** name **=** 'Milhouse'**;**

console**.**log**(**name**);**

Консоль:

"Milhouse"

undefined

Пример 2

**var** questionName **=** **prompt(**'Как вас зовут?'**);** //задание переменной questionName

console**.**log**(**questionName**);** //вывод в консоль значения введенного в переменную questionName

Вывод:

Как вас зовут**?**

"Петр"

undefined

Вторая структура данных – это массив!!!

**Массив** – это набор фиксированного количества элементов одного типа, имеющие возможность прочитать или записать элемент по индексу!!!!

Если нужно держать много данных в одном месте – их можно поместить в массив!!!

В язык JavaScript (со слабой и динамической типизацией данных) в массив можно добавлять любые переменные или типы данных – числа, строки, другие массивы и все это одновременно!!!!

В других языках – в массив можно добавлять только элементы одного типа данных!!!!

Каждый элемент массива имеет свой индекс!!!

Индекс обеспечивает прямой доступ к любому элементу массива!!!

Элементами массива могут быть как простые так и структурированные данные!!!!

Например, массив массивов – **многомерный массив**!!!

Число индексов массива обозначает его размерность – одномерный, двухмерный (один массив вложен в другой массив!!!), трехмерный и т.д.

В JavaScript массив записывается в виде

//элементы 0 1 2 3 4

**var** list **=** **[**'cheese'**,** 'milk'**,** 'limonade'**,** 'tea'**,** 'apple'**]**

console**.**log**(**list**);**

Консоль:

**[**"cheese"**,** "milk"**,** "limonade"**,** "tea"**,** "apple"**]**

undefined

Теперь из данного массива любой элемент можно вытащить с помощью индексов!!!!

Например

// 0 1 2 3 4

**var** list **=** **[**'cheese'**,** 'milk'**,** 'limonade'**,** 'tea'**,** 'apple'**]**

console**.**log**(**list **[**1**]);**

console**.**log**(**list **[**0**]);**

Консоль:

"milk"

"cheese"

undefined

В двухмерных массивах нужно указывать

list **[**2**][**3**]**

Например в Python

#эллементы массива 0 1

productMiniList **=** **[**'lemonade'**,**'Coca-Cola'**]** #

#эллементы массива 0 1 2 3 4

productList **=** **[**'apple'**,**'milk'**,**'chese'**,**productMiniList**,**'tea'**]** #

**print(**productList**[**3**][**1**])** #вывод в консоль элемента 3 массива productList[] элемента 1 подмасисва productMiniList []

Результат

Coca**-**Cola

В JavaScript

//эллементы массива 0 1

**var** productMiniList **=** **[**'lemonade'**,**'Coca-Cola'**];**

//эллементы массива 0 1 2 3 4

**var** productList **=** **[**'apple'**,**'milk'**,**'chese'**,**productMiniList**,**'tea'**];**

console**.**log**(**productList**[**0**]);** //вывод нулевого элемента массива productList[]

console**.**log**(**productList**[**3**][**1**]);** //вывод третьего элемента массива productList[] первого элемента подмассива productMiniList[]

Вывод:

"apple"

"Coca-Cola"

Структур данных на самом деле больше – **очереди**, **стейки**, **списки**, **хэши**, **таблицы** и др.

Но основные – это переменные и массивы!!!

Д/з – придумать и записать массив

**var** loft **=** **[**'лучшие уроки даёте'**,**'за урок!'**,** 'Спасибо,'**,**'очень хорошо у вас получается'**,**' продолжайте.'**];**

**var** blog **=** **[**'Агонь.'**,**'очень крутых программистов'**,** 'все хорошо'**,** loft**,**' но зачем так тыкать по'**,** 'мышке или тачпаду?'**];**

**var** author **=** **[**'Грачев Виталий'**,**'Николай Чернобаев'**,**'Серж Смайлов'**];**

console**.**log**(**loft**[**2**]);**

console**.**log**(**blog **[**3**][**1**]);**

console**.**log**(**author **[**1**]);**

Вывод:

"Спасибо,"

"за урок!"

"Николай Чернобаев"

undefined

**[Основы Программирования – #4 – Функции](#_top)**

Программа состоит из алгоритма!!!

Алгоритм состоит из команд, переменных и функций!!!

**Функция** – это именованная отдельно вызванная часть кода, которая решает конкретную задачу и создана для того, чтобы не повторять одну и ту же последовательность команд, инструкций несколько раз!!!!

И к которой можно обратиться из любой части программы в любое удобное время!!!

**Функция** – это алгоритм в алгоритме!!! Ее можно вызвать в любой удобный момент!!!

**Команда** – это вызов функции!!!!

Командой в одном алгоритме можно вызвать функцию, которая будет читать другой алгоритм!!!

Параметрам функции можно передавать переменные – числа, строки и т.д!!!!

Переменные внутри функции называются **аргументами**!!!!

У человека функция – ходить, тогда переменная – обувь!!!

В зависимости от того, когда функция запущена, зимой – ботинки, летом – босоножки и т.д.

При вызове функции язык программирования находит содержащиеся в ней команды и выполняет их!!!!

После этого программа возвращается в строке, откуда функция была вызвана и переходит к следующей команде.

Вызывать можно не только функции, которые идут в комплекте с языком программирования!!!

**print()** – команда или функция вывода в консоль.

Но мы можем создавать и свои собственные функции!!!

Чтобы создать функцию нужно записать наш алгоритм в определенной обертке и дать ей имя, а потом вызвать ее с помощью команды этого имени!!!

Чтобы всегда ее можно было вызвать как только она понадобится!!!

В Python определение новой функции начинается с ключевого слова

**def** после которого идет название функции, например

**def** greeting**()**

В скобочках функции **()** – можно передавать какие-то параметры, аргументы, которые будут использоваться внутри функции!!!!

В Python

**def** greeting**():**

**print(**'Hello!'**)**

Консоль

Error

В JavaScript

**function** greeting**()** **{**

console**.**log**(**'Hello!'**);**

**}**

greeting**();**

Консоль

"Hello!"

undefined

Например в Python

**def** greeting**():**

 **print(**'Hello it\'s Python function!'**)**

greeting**()** #вывод функции

Консоль

Hello it's Python function!

**[**Finished **in** 0.3s**]**

Например в JavaScript

**function** greeting**()** **{**

console**.**log**(**'Hello, it\'s JavaScript function!'**);**

**}**

greeting**();**

Консоль

"Hello, it's JavaScript function!"

greeting**(**a**,** b**,** c**);** – функция!!!

a, b, c – параметры функции!!!

height(1,45);

Например в JavaScript

**function** height**(**m**,**cm**)** **{**

**var** total**=(**100**\***m**)+**cm**;**

console**.**log**(**total **+** ' см'**);**

**}**

Консоль

145 cm

**Конкатенация** – операция склеивания объектов линейной структуры, обычно строк.

В Python

**def** height**(**m**,**cm**):**

 total **=** **(**m **\*** 100**)** **+** cm

 **print(**str**(**total**)** **+** ' cm tall'**)**

height**(**1**,**70**)**

Консоль

170 cm tall

**[**Finished **in** 0.1s**]**

В JavaScript

**function** height**(**m**,**cm**)** **{**

 **var** total **=** **(**100**\***m**)+**cm**;**

console**.**log**(**total **+** ' cm tall'**);**

**};**

height**(**1**,**70**);**

Консоль

"170 cm tall"

Пример 2

**function** height**(**m**,**cm**)** **{**

 **var** total **=** **(**100**\***m**)+**cm**;**

console**.**log**(**total**,** ' cm tall'**);**

**};**

height**(**1**,**70**);**

Консоль

170

" cm tall"

Пример 3

**var** m**=parseInt(prompt(**'Введите количество метров'**));**

**var** cm**=parseInt(prompt(**'Введите количество сантиметров'**));**

**function** height**(**m**,**cm**)** **{**

 **var** total **=** **(**100**\***m**)+**cm**;**

console**.**log**(**total**+** ' cm tall'**);**

**};**

height**(**m**,**cm**);**

Консоль

'Введите количество метров'

13

'Введите количество сантиметров'

10

"1310 cm tall"

Чтобы функция возвращала значение нужно добавить команду **return** и значение, которое нужно вернуть!!!!

Возвращение результатов функции нужно для того, чтобы возвращать определенный промежуточный результат вычисления.

Возвращение результатов функции в JavaScript

**function** calc**(**a**,**b**)** **{**

**var** sum **=** a **+** b**;**

**return** sum**;**

**};**

calc**(**4**,**5**);**

Консоль

9

Пример в Python

**def** calc**(**a**,**b**):**

 total **=** a**+**b

 **return** total

**print(**calc**(**4**,**5**))**

Консоль

9

Пример 2 в Python

**def** calc**(**a**,**b**):**

 total **=** a**+**b

 **return** total

**print(**calc**(**4**,**5**))**

Консоль

9

Пример 2 в JavaScript

**function** calc**(**a**,**b**)** **{**

**var** sum **=** a**+**b**;**

**return** sum**;**

**};**

console**.**log**(**calc**(**4**,**5**));**

Консоль

9

Д/з – Придумать свою произвольную функцию, которая реально помогает что-то делать!!!

**var** age **=** 15**;**

**if** **(**age **==** 15**)** **{**

 console**.**log **(**"Да, тебе уже 15!"**);**

**};**

Консоль

"Да, тебе уже 15!"

undefined

Пример 2

**var** age **=** 15**;**

**if** **(**age **>=** 18**)** **{**

 console**.**log **(**"Достигнут возраст совершеннолетия"**);**

**}** **else** **{**

 console**.**log **(**'Ещё нет восемнадцати'**);**

**};**

Консоль

Error

Пример 3

**var** age **=** 15**;**

**if** **(**age**<**16**)** **{**

 console**.**log **(**'Скорее всего вы учитесь в школе'**);**

**}** **else** **if** **(**age**>**18**)** **{**

 console**.**log **(**'Скорее всего вы – студент универа '**);**

**}** **else** **{**

 console**.**log **(**'Скорее всего вы – студент колледжа, ил учитесь в школе'**);**

**};**

Консоль

Error

**[Основы Программирования – #5 – Арифметические и логические выражения](#_top)**

Арифметические операции:

**\*** – умножение;

**/** – деление;

**+** – сложение;

**-** – вычитание.

В Node.js

12**+**3

**>**15

12**-**3

**>**9

Если несколько арифметических операций выполняются одна с другой то результат зависит от их порядка. Умножение и деление (\*, /) имеют более высокий приоритет.

В Node.js

**var** cookies **=** 8**;**

**var** apples **=** 4**;**

**var** food **=** cookies **+** apples**;**

console**.**log**(**food**)**

Консоль

12

**Логические выражения**

Результатом вычисления которых является **ИСТИНА** либо **ЛОЖЬ**

Для логических выражений используются операторы сравнения:

**>,** **<,** **=,** **>=,** **<=,** **<<,** **>>,** **!=**.

В Node.js

**>** 3**==**3

**true**

**>** 3**==**2

**false**

**>** **var** age **=** 7**;**

undefined

**>** age **==** 7**;**

**true**

**>** age **>** 11**;**

**false**

**>** age **<** 18**;**

**true**

**>**

Логические операции построенные при помощи логических операций называются **сложными**.

**var** age **=** 14**;**

**var** height **=** 150**;**

**(**age **>** 10**)** **&&** **(**height **>** 120**);**

**>** **var** age **=** 14**;**

undefined

**>** **var** height **=** 150**;**

undefined

**>** **(**age **>** 10**)** **&&** **(**height **>** 120**);**

**true**

**>** **(**age **>** 10**)** **&&** **(**age **<** 18**);**

**true**

**>** **(**age **>** 10**)** **||** **(**height **>** 160**);**

**true**

Д/з

**var** age **=** 18**;**

**var** height **=** 185**;**

**(**age **>** 16**)** **&&** **(**height **>** 170**);**

Консоль

Error

В Node.js

**>** **var** age **=** 18**;**

undefined

**>** **var** height **=** 185**;**

undefined

**>** **(**age **>** 16**)** **&&** **(**height **>** 170**);**

**true**

**>**

**[Основы Программирования – #6 – Условия](#_top)**

Алгоритм **ветвление** – выбор одного из нескольких условий!!!

Чтобы наша программа была умной – в определенной ее точке должно приниматься логическое решение.

Чтобы осуществить такое решение программисту в коде нужно задать одно или больше логических условий!!!

Которые будут выполнены и проверены программой!!!

И одного или несколько действий, которые будут выполнены если условие будет **ИСТИНА** – **true** или **ЛОЖЬ** – **false**.

На основе результата **true** или **false** можно принимать решение какую часть программы выполнять дальше – это называется ветвлением!!!!

Для осуществления ветвления в языках программирования есть условные операторы!!!

В большинстве языков есть три варианта логического выбора условных операторов:

1. **Единственный выбор** – **if**

2. **Двойной выбор** – **if/else**

3. **Множественный выбор** – **else if, switch, elif**

1. **Единственный выбор** – это утверждение с использованием оператора **if** содержит в себе условие с логическим выражением, в котором происходит логическое сравнение данных!!! В результате выбирается дальнейшее действие!!!

**if** **->** **|->** **true** – выполняется соответствующий код**!!!**

 **->** **|->** **false** – выполняется другой код**!!!**

2. **Двойной выбор** – когда вместе с **if** используется **else**

**if**

 **->** **|->** **true** – выполняется один блок команд**!!!**

**else**

 **->** **|->** **false** – выполняется другой блок команд**!!!**

**else** – это опциональный оператор и он может быль только один во всем выражении!!!!

3. **Множественный выбор** – позволяет выполнить множественную проверку выражения!!!!

И выполнить конкретный блок кода, как только результат будет считаться ИСТИНОЙ.

В языке Python

value **=** 15

**if** **(**value **==** 15**)** **:** **print(**"Yes!"**)**

**print(**"Nothing"**)**

Консоль

Yes**!**

Nothing

Пример 2

value **=** 10

**if** **(**value **==** 15**)** **:** **print(**"Yes!"**)**

**print(**"Nothing"**)**

Консоль

Nothing

В JavaScript

**var** year **=** 2017**;**

**if** **(**year **!=** 2017**)** **{**

console**.**log**(**'А вот и нет! Год другой!'**);**

**}**

console**.**log**(**'Скоро или уже 2017!'**);**

Консоль

"Скоро или уже 2017!"

Пример 2

**var** year **=** 2016**;**

**if** **(**year **!=** 2017**)** **{**

console**.**log**(**'А вот и нет! Год другой!'**);**

**}**

console**.**log**(**'Скоро или уже 2017!'**);**

Консоль

"А вот и нет! Год другой!"

"Скоро или уже 2017!"

2. Двойной выбор

var year **=** 2016**;**

**if** **(**year **!=** 2017**)** **{**

console**.**log**(**'А вот и нет! Год другой!'**);**

**}** **else** **{**

console**.**log**(**'Все верно! Уже 2017!'**);**

**}**

console**.**log**(**'Скоро или уже 2017!'**);**

Консоль

"А вот и нет! Год другой!"

"Скоро или уже 2017!"

Пример 2

**var** year **=** 2017**;**

**if** **(**year **!=** 2017**)** **{**

console**.**log**(**'А вот и нет! Год другой!'**);**

**}** **else** **{**

console**.**log**(**'Все верно! Уже 2017!'**);**

**}**

console**.**log**(**'Скоро или уже 2017!'**);**

Консоль

"Все верно! Уже 2017!"

"Скоро или уже 2017!"

В языке Python

value **=** 10

**if** **(**value **==** 15**)** **:**

 **print(**"Yes!"**)**

**else** **:**

 **print(**"No!"**)**

**print(**"Nothing"**)**

Консоль

No**!**

Nothing

**[**Finished **in** 0.1s**]**

3. Множественный выбор (используется оператор **elif** в Python)

value **=** 30

**if** **(**value **==** 15**)** **:**

 **print(**"Yes! It's 15"**)**

**elif** **(**value **==** 20**)** **:**

 **print(**"Yes! It's 20"**)**

**elif** **(**value **==** 25**)** **:**

 **print(**"Yes! It's 25"**)**

**elif** **(**value **==** 30**)** **:**

 **print(**"Yes! It's 30"**)**

**else** **:**

 **print(**"No! This is a strange value!"**)**

Консоль

Yes**!** It's 30

**[**Finished **in** 0.1s**]**

Пример 2

value **=** 23

**if** **(**value **==** 15**)** **:**

 **print(**"Yes! It's 15"**)**

**elif** **(**value **==** 20**)** **:**

 **print(**"Yes! It's 20"**)**

**elif** **(**value **==** 25**)** **:**

 **print(**"Yes! It's 25"**)**

**elif** **(**value **==** 30**)** **:**

 **print(**"Yes! It's 30"**)**

**else** **:**

 **print(**"No! This is a strange value!"**)**

Консоль

No**!** This **is** a strange value**!**

**[**Finished **in** 0.1s**]**

Множественный выбор в JavaScript

Функция **elif** в Pythone – это сокращенная **else if** в JavaScript

Пример

**var** year **=** 2017**;**

**if** **(**year **==** 2017**)** **{**

console**.**log**(**'Все верно! Уже 2017!'**);**

**}** **else** **if** **(**year **==** 2016**)** **{**

console**.**log**(**'Да! Но скоро уже 2017!'**);**

**}** **else** **if** **(**year **<=** 2015**)** **{**

console**.**log**(**'Ой! Вы откуда-то из прошлого!'**);**

**}** **else** **{**

console**.**log**(**'Скорее всего вы заглядываете в будущее!'**);**

**};**

Консоль

"Все верно! Уже 2017!"

Пример 2

**var** year **=** 2016**;**

**if** **(**year **==** 2017**)** **{**

console**.**log**(**'Все верно! Уже 2017!'**);**

**}** **else** **if** **(**year **==** 2016**)** **{**

console**.**log**(**'Да! Но скоро уже 2017!'**);**

**}** **else** **if** **(**year **<=** 2015**)** **{**

console**.**log**(**'Ой! Вы откуда-то из прошлого!'**);**

**}** **else** **{**

console**.**log**(**'Скорее всего вы заглядываете в будущее!'**);**

**};**

Консоль

"Да! Но скоро уже 2017!"

Синтаксис **switch**

**var** year **=** 2017**;**

**switch** **(**year**)** **{**

**case** 2017**:**

console**.**log**(**'Все верно! Уже 2017!'**);**

**break;**

**case** 2016**:**

console**.**log**(**'Да! Но скоро уже 2017!'**);**

**break;**

**case** 2015**:**

console**.**log**(**'Ой! Вы откуда-то из прошлого!'**);**

**break;**

**default:**

console**.**log**(**'Скорее всего вы заглядываете в будущее!'**);**

**};**

Консоль

'Все верно! Уже 2017!'

В функции **switch** **()** можно сравнивать только точные значения!!! Диапазон задать невозможно!!! В отличии от **else if** **(**year **<=** 2015**)**!!!!

Д.з – Придумать аналогичные условия **else if** в JavaScript и **elif** в Python

**[Основы Программирования – #7 – Циклы](#_top)**

**Циклы** используются для указания повторять действия пока условие является истинным!!!!

Пока условие **ИСТИНА** то цикл повторяется и повторяется бесконечно!!!

В случае условия **ЛОЖЬ** – цикл прекращается!!!

Цикл обозначает какие блоки кода нужно повторять и сколько раз!!!

Или до каких условий!!!!

Они используются для удобства и неиспользования для одних и тех же блоков кода многократно!!!!

Например

Цикл – мыть грузную посуду

Условие – пока грязная стопка не закончится!!!!

**Цикл** состоит из трех частей

1. **Оператор** – это название цикла!!!

2. **Условие** – это то при каких обстоятельствах цикл должен работать!!!

3. **Тело** – это код цикла, который должен сработать, если цикл прошел проверку!!!

**Итерация** – это повторение цикла!!!

Если нам нужно чтобы цикл повторился 10 раз – то это 10 итераций!!!

Типы циклов:

1. **Безусловные циклы** – цикл в котором пропущено условие!!!

То есть он бесконечно крутится/повторяется!!! На практике не используются – так как компьютер повиснет!!!!

2. **Условные циклы:**

1) С предусловием **while** – этот цикл выполняется пока истинное некоторое условие перед началом самого цикла!!!

Цикл **while** выполняется пока выполняется заданное условие!!! Если **ИСТИНА** цикл – повторяется, **ЛОЖЬ** – цикл заканчивается!!!

Если в условие цикла **while** задать значение ИСТИНА то оно никогда не станет ложным и цикл превратится в бесконечный цикл!!!

2) Цикл с пост условием – цикл **do... while** – это цикл, в котором условие выполняется после проверки тела цикла!!! То есть его тело всегда выполняется хотя бы один раз!!!

3) Цикл со счетчиком – цикл **for** – эти циклы повторяют блок команд заданное количество раз!!!

Циклы **for** хороши, когда количество повторений известны заранее!!!

В цикле **for** некоторая переменная называемая счетчиком, изменяет свое значение от некоторого начального до некоторого конечного значения с определенным шагом!!!!

И для каждого значения эта переменная и тело цикла выполняются один раз!!!

Условие проверяется перед каждой итерацией цикла!!!

После успешной проверки выполняется код в теле цикла!!!

Если проверка не прошла – код не выполняется!!!

И код выполняется со следующей строки после цикла!!!!

После выполнения счетчика переменная счетчика обновляется перед началом следующей итерации!!!

Если условие цикла не выполняется – цикл не запускается!!!

Если условие цикла выполняется всегда – образуется бесконечный цикл.

Цикл **for** – это самый универсальный используемый цикл!!!

Порядок выполнения цикла **for**

1. **Инициализация** – создание и инициализация используемых переменных!!!

2. **Условие** – проверяется выполнение условия!!!

Если оно не выполняется происходит выход из цикла!!!

3. **Блок кода (тело цикла)** – происходит выполнение цикла!!!

4. **Обновление** – происходит обновление управляющих переменных, которые мы задали в начале

5. Переход к пункту 2 – Условие!!!

4) Цикл просмотра – **for in** – представляет собой запись инструкции вида выполнить операцию x для всех элементов, входящих во множество y.

Этот цикл используется для перечисления полей, названий, массивов или объектов!!!

**Объект** – это структура данных!!!

Он описывается как список свойств и значений!!!

Объект – Машина

Список свойств – колеса, руль, цвет.

Методы – это функции объекта – ехать, стоять.

Цикл **for in** возвращает имеющиеся индексы, а не их значения!!!

По которым затем можно получить уже сами значения!!!

В различных языках программирование синтаксис данного цикла различен!!!

5) Цикл с выводом из середины – **break**

Многие языки программирования имеющие в своем составе циклические конструкции имеют специфические команды, позволяющие нарушать порядок работы этих конструкций!!!!

1. **Команда досрочного выхода из цикла** – **break** – применяется когда необходимо прервать выполнение цикла, когда условие выхода из цикла еще не достигнуто!!!

Эта команда завершает цикл даже бесконечный!!!

Вызов команды **break** внутри цикла приводит к его мгновенному завершению!!!! Даже если условие дает **ИСТИНА**!!!

Все команды после вызова цикла **break** – игнорируются!!!!

2. **Команда пропуска итерации** – **continue** – эта команда сразу переходит к следующему повтору цикла!!!

Применяется когда в теле цикла необходимо пропустить все команды до конца тела цикла!!! При этом сам цикл прерываться не должен!!!

Условие продолжения или выхода должно вычисляться обычным образом!!!

Команда **continue** прекращает выполнения текущей итерации цикла и переходит к следующей минуя все последующие команды в теле цикла!!!!

Эта команда сокращает вложенность сложных проверок и используется редко.

Она полезна, когда вначале итерации мы понимает что текущая итерация не может быть полезной и нужно сразу же переходить к следующей!!!

Командой **continue** можно пропустить вопрос не выходя из самого цикла!!!

Эта команда говорит, что нужно завершить текущий повтор цикла и сразу же перейти к следующему!!!

Эти команды и служат для управления ходом выполнения цикла!!!

Пример 1 в Python – Цикл с предусловием **while**

digit **=** 1

**while** digit **<=** 100**:**

 **print(**str**(**digit**)** **+** ' ok!'**)**

 digit **=** digit **+** 1

**print(**str**(**digit**)** **+** ' more than 100!'**)**

Консоль

1 ok**!**

2 ok**!**

**..**

100 ok**!**

101 more than 100**!**

**[**Finished **in** 0.1s**]**

Пример 1 в JavaScript – Цикл с предусловием **while**

**var** digit **=** 1**;**

**while** **(**digit **<=** 100**)** **{**

 console**.**log**(**digit **+** 'ok!'**);**

 digit **=** digit **+** 1**;**

**};**

console**.**log**(**digit **+** 'more than 100!'**);**

Консоль

"1ok!"

"2ok!"

**...**

"100ok!"

"101more than 100!"

Пример 2 в JavaScript – Цикл с постусловием **do while**

**var** digit **=** 2**;**

**do** **{**

console**.**log**(**digit **+** ' ok!'**);**

 digit **=** digit **+** 1**;**

**}** **while** **(**digit **<=** 10**);**

console**.**log**(**digit **+** ' more then 10!'**);**

Консоль

"2 ok!"

"3 ok!"

**...**

"10 ok!"

"11 more then 10!"

Пример 2 – Если больше 10

**var** digit **=** 11**;**

**do** **{**

console**.**log**(**digit **+** ' ok!'**);**

 digit **=** digit **+** 1**;**

**}** **while** **(**digit **<=** 10**);**

console**.**log**(**digit **+** ' more then 10!'**);**

Консоль

"11 ok!" //1 раз цыкл выполняется!!!

"12 more then 10!"

Пример 3 – Цикл **for** (со счетчиком)!!!

**var** arr **=** **[**1**,**4**,**6**,**3**,**10**];**

**for** **(**i**=**0**,**sum**=**0**;**i**<**arr**.**length**;**i**++)** **{**

sum**+=**arr**[**i**];**

**};**

console**.**log**(**sum**)**

Консоль

24 //1+4+6+3+10=24!!

**!**В Python нету такого же цикла со счетчиком но за то в нем есть цикл **for in** как и в JavaScript.

Пример 3 – Цикл **for in** в Python – Собирание суммы всех элементов массива вне зависимости от его длинны!!!

arr **=** **[**1**,**4**,**6**,**3**,**214**,**445**,**323463**,**63463**,**56**]**

sum **=** 0

**for** i **in** arr**:**

 sum**+=**i

**print(**sum**)**

Консоль

387655

**[**Finished **in** 0.4s**]**

Пример 4 – Цикл с прерыванием (выход из середины цикла) в Python

С использованием команд **brake** и **continue**

arr **=** **[**1**,**4**,**6**,**3**,**10**]** #массив

sum **=** 0 #начальное значение

**for** i **in** arr**:** #начало цикла for in

 **if** i **==** 3**:** **break** #прервать цикл при i == 3

 sum**+=**i**;** #суммирование

**print(**sum**)** #вывод суммы

Консоль

11 #1+4+6=11 (сумма всех чисел до тройки!!!)

Обязательно должна быть настроена табуляция иначе получиться неправильно значение!!!!

Например

arr **=** **[**1**,**4**,**6**,**3**,**10**]**

sum **=** 0

**for** i **in** arr**:**

 **if** i **==** 3**:** **break**

sum**+=**i**;**

**print(**sum**)**

Консоль

3 #а должно быть 11!!!

Элемент **continue** просто пропустит элемент i **==** 3!!! а остальные все числа суммирует!!!

Например

arr **=** **[**1**,**4**,**6**,**3**,**10**,**2**]**

sum **=** 0

**for** i **in** arr**:**

 **if** i **==** 10**:**

 # break

 **continue**

 sum**+=**i**;**

**print(**sum**)**

Консоль

16 #1+4+6+3+2=16 (десятка исключается!!!!)

Или несколько исключений

arr **=** **[**1**,**4**,**6**,**3**,**10**,**2**]**

sum **=** 0

**for** i **in** arr**:**

 **if** i **==** 10 **or** i **==** 6 **or** i **==** 2**:** #исключение 10, 6 и 2

 # break

 **continue**

 sum**+=**i**;**

**print(**sum**)**

Консоль

8 #1+4+3=8 (10, 6 и 2 исключается!!!!)

Пример 1 – В JavaScript

**var** arr **=** **[**1**,** 4**,** 6**,** 3**,** 10**];**

**for** **(**i**=**0**,**sum**=**0**;**i**<**arr**.**length**;**i**++)** **{**

 **if** **(**arr**[**i**]** **===** 6 **)** **{**

 **break;**

 //continue

 **}**

 sum**+=**arr**[**i**];**

**};**

console**.**log**(**sum**)**

Консоль

5 //1+4=5 (Все, что после 6 пропускается!!!)

Пример 2 – **continue** в JavaScript

**var** arr **=** **[**1**,** 4**,** 6**,** 3**,** 10**];**

**for** **(**i**=**0**,**sum**=**0**;**i**<**arr**.**length**;**i**++)** **{**

 **if** **(**arr**[**i**]** **===** 6 **)** **{**

 //break;

 **continue**

 **}**

 sum**+=**arr**[**i**];**

**};**

console**.**log**(**sum**)**

Консоль

18 //1+4+3+10=18 (Суммируется все, кроме 6!!!)

Пример 3 – Несколько исключений!!!

**var** arr **=** **[**1**,** 4**,** 6**,** 3**,** 10**];**

**for** **(**i**=**0**,**sum**=**0**;**i**<**arr**.**length**;**i**++)** **{**

 **if** **(**arr**[**i**]** **===** 6 **||** arr**[**i**]** **===** 10 **||** arr**[**i**]** **===** 3**)** **{**

 //break;

 **continue;**

 **}**

 sum**+=**arr**[**i**];**

**};**

console**.**log**(**sum**)**

Консоль

5 //1+4=5

В каждом языке синтаксис может выглядеть по-своему, но такие циклы есть в каждом языке или эмуляция каких-то видов этих циклов.

!Циклы могут быть **вложенными** – когда один цикл вложен в другой!!!

То есть внешний цикл повторяется лишь после того как внутренний цикл выполнит все свои повторы!!!

Д/з – Написать условие при котором сработает цикл на n-е число повторений!!!

**[Основы Программирования – #8 – Инструменты для программиста](#_top)**

**Редактор кода** – это минималистичный инструмент нацеленный в основном на то чтобы подсвечивать синтаксис языка, на котором пишет программист.

**IDE** (интегрированная среда разработки) – это комплексная программа в которую входит множество инструментов.

IDE имеет два рабочих окна:

1. **Текстовый редактор** – специфический редактор, который может обладать подсветкой имен, синтаксиса, созданием пользовательского интерфейса или отображением документации!!!

При помощи текстового редактора программист пишет и редактирует свои программы, которые называются исходным кодом!!

Язык программирования определяет синтаксис и семантику исходного кода.

В настоящее время активно используются среды разработки, которые включают в себя также:

1) **редактор кода** – для написания и редактирования программы

2) **дебаггеры** – отладчики для поиска и устранения ошибок;

3) **компоновщики** – помогают собрать программу из разных модулей.

4) **трансляторы** – могут транслировать программу из одного языка в другой.

**2. Консоль**:

Команды введенные в консоль выполняются сразу же как их напечатаешь. Это очень удобно для экспериментов, чтобы посмотреть, что значит та или иная функция в языке или как работает та или иная синтаксическая конструкция.

**Веб программирование**

Здесь код можно писать в абсолютно любой программе, которая умеет редактировать текст. Затем браузер считывает в пригодном для него формате и исполняет код, написанный внутри них.

Например html код

<h1>**Hello!**</h1>

<p>

**world!**

</p>

В обычном текстовом редакторе читать этот код очень трудно поэтому надо использовать специальные редакторы кода – Sublime text 3!!!

**Sublime text 3** – это текстовый редактор, но с дополнительным функционалом, необходимым программисту. Потому что данный редактор подсвечивает синтаксические конструкции. Благодаря чему мы можем оделить наполнение тегов от конструкции языка.

Практически любой редактор имеет самые необходимые возможности, которые нужны программисту. Если чего-то нет то его можно расшить посредством установки на него расширений или дополнительных плагинов.

Поэтому в своей работе программисты используют не только редактор но и IDE (интегрированную среду разработки), которая с коробки имеет уже гораздо больше, чтобы не требовалось дополнительно ничего устанавливать.

Например, **IDE phpstorm**

Для JavaScript Node.js

**>** **var** a **=** 2**;**

undefined

**>** **var** result **=** Math**.**pow**(**a**,** 3**);**

undefined

**>** console**.**log**(**result**);**

8

undefined

**>**

В IDE не нужно много программ, а все средства разработки уже есть в одном месте.

Д/з

OS **:** Windows 10

Browser**:** Google Chrome

Text Editor**:** Visual Studio Code

IDE**:** Visual Studio 2017**?**

**[Основы Программирования – #9 – Графика](#_top)**

**Компьютерная графика** – это область информатики изучающая способы формирования и обработки изображений.

Бывают несколько типов компьютерной графики, но самые известные из них

1. **Растровая графика** – теряет качество при увеличении. Единицей описания данного изображения являются пиксели.

В этих изображениях, как правило, хранятся данные о каждом пикселе.

Чтобы сохранить растровое изображение нужно хранить данные о каждом его пикселе плюс четвертый параметр – прозрачность пикселя.

Если увеличить данное изображение то по его краям будем иметь ребристость, потому что масштабирование таких изображений производится по различным алгоритмам. То есть у одного пикселя был скопирован и размножен каждый ближайший пиксель!!!!

Поэтому растровое изображение при увеличении его больше чем его исходный размер теряет в своем качестве, то есть у него появляются – ребристость, пикселизация, размытые края.

**Недостатки:**

1) при увеличении размера изображения увеличивается и количество памяти, которое он занимает!!! Потому что растет количество пикселей и данных об этих пикселях нужно хранить!!!

2) эти изображения плохо масштабируются!!! При увеличении – они заметно будут терять в своем качестве.

2. **Векторная графика** – не теряет качества при увеличении так как в данном изображении хранятся не данные пикселей, а характеристики фигуры!!!

В векторных изображениях вся графическая сцена представлена в виде набора фигур!!!

То есть такие изображения будут занимать памяти меньше.

При увеличении данных изображений объем памяти не будет увеличиваться так будут изменятся определенных характеристики фигур и все.

Векторные изображения выглядят не очень гладко и реалистично как растровые.

В векторных изображениях намного сложнее передать переход одного цвета к другому, градиенты и прочие полутона.

Именно поэтому для хранения фотографий и других сложных изображений предпочтительно использовать растровые форматы.

А для хранения боле простых изображений лучше использовать векторы.

В программировании компьютерная графика часто используется для построение графических интерфейсов либо для вывода данных на экран в виде визуального отображения либо инфографики.

Интерфейсы могут строиться при помощи различных инструментов либо при помощи сторонних программ, либо сред разработки.

В Веб программировании интерфейс строится при помощи двух языков:

1. **HTML** – для разметки

2. **CSS** – для придания стиля этой разметки.

Как правило, построение интерфейса начинается с проектирования и дизайна.

1. **Photoshop** – самый популярный редактор растровых изображений

2. **Adobe Illustrator** – самый популярный редактор векторных изображений

<button>**button**</button>– тег кнопки на языке html

**[Основы Программирования – #10 – События](#_top)**

**Событие** – это сообщение от программы, которое возникает в различных точках исполняемого кода при выполнении определенных условий.

События нужны для того, чтобы иметь возможность предусмотреть или обработать реакцию программы на определенное стечение обстоятельств.

Например:

Кнопка на системном блоке – включить компьютер.

Тоже самое и в программировании:

Для решения поставленной задачи создается обработчик события и как только программа попадает в заданное состояние – происходит это самое событие – посылается сообщение и обработчик перехватывая это сообщение обрабатывает наше событие.

В случае Веб программирования и языка программирования JavaScript самый простой пример события – это взаимодействие пользователя с графическим интерфейсом сайта.

Например, клик по кнопке.

В случае веб программирования из разметки веб браузер создает определенную модель документа, к которой дает возможность обращения при помощи JavaScript.

Различные методы и работы с разметкой лежат в объекте **document**.

Чтобы найти элемент по его **id** есть специальный метод .**GetElementById()**!!!

Куда нужно передать в виде строки идентификатор нужного элемента.

Вырабатывать события в JavaScript лучше всего при помощи метода **.addEventLister(**какое\_именно\_событие\_вырабатывать**,** сам\_обработчик**)** – добавить слушатель какого-то события.

События бывают разные:

1. Мышки;

2. Клавиатуры;

3. Другие.

Д/з – написать событие на сверстанной кнопке.

**[Основы Программирования – #11 – Первая программа (Калькулятор на JavaScript)](#_top)**

Последовательность работы программы:

1. Программа принимает входные данные – это **ввод**;

2. Читает программный код – **обрабатывает** эти данные;

3. Возвращает на выходе результат – **вывод**.

Последовательность действий:

1) **Ввод** – клавиатура и мышка, с помощью которых пользователь будет осуществлять действие с самой программой – вводить данные, нажимать на кнопки и т.д.

2) **Обработка** – это все функции, переменные, циклы то есть сам этап программирования!!!!

3) **Вывод** – экран и графика запрограммированные на работу!!!

Этап обработка – это сам процесс написание самого кода, то есть программирование!!!

Исходный код превращается в рабочий при помощи:

1. **Компиляции** – это когда исходный код программы преобразуется в код понятный компьютеру!!! То есть в машинный код!!!

Это преобразование происходит при помощи специальной программы, которая называется – компилятор!!!!

При этом исходный код остается у разработчика неизменным!!!

**Компилятор** – это переводчик с языка, который написал человек в язык, который понимает компьютер!!!

Это работает так: сначала компилятор читает весь код целиком и потом, если нет ошибок, выполняет ее!!!

Этот процесс называется Компиляцией!!!

2. **Интерпретация** – это когда специальная программа-интерпретатор получает исходный код, проверяет его и запускает построчно!!!

В этом и состоит разница с компилятором. Интерпретатор прочитал строку и если со строкой все в порядке – он ее выполняет!!! И переходит к следующей строке.

Этот процесс называется Интерпретацией!!!

При компиляции из программного кода единожды создается отдельный физический файл, содержащий в себе машинный код. То есть при компиляции программного кода всегда создается файл с машинным кодом!!! Который впоследствии уже и запускается!!!

Таким образом на конечной машине выполняется уже откомпилированный код, который содержится в выполняемом файле.

Выполнение исполняемого откомпилированного файла с машинным кодом происходит в разы быстрее чем при интерпретации!!!

То есть превращение программного кода в машинный на лету с последующем выполнением!!!

**Достоинства компиляции:**

1. Высочайшая скорость откомпилированного файла с машинным кодом

**Недостатки компиляции:**

1. Плохая переносимость программ в целом ибо при компиляции программный код преобразуется в машинный и для конкретной модели. То есть если начали программировать на Windows, а закончили на Mac – это может вызвать проблему!!!

При интерпретации программный код программы каждый раз при запуске программы последовательно читается интерпретатором!!! и превращается в машинный код. То есть построчно!!!

Таким образом программа читается на-лету, интерпретируется в машинный код и тут же выполняется!!!

**Достоинства интерпретации:**

Этот способ более удобен так как программный код может выполняться на любой платформе, не важно на какой – Mac, Windows и т.д.!!! При наличии соответствующего интерпретатора!!!

Например JavaScript код удобен тем что в любой современный браузер встроен интерпретатор JavaScript кода. Поэтому неважно на кой машине писать, редакторе и как писать!!! Потому что в итоге в браузере все это запуститься!!! и сразу интерпретируется на-лету!!!

То есть достоинство интерпретации – высочайшая переносимость программного кода, потому что преобразование машинного кода происходит на лету, на месте, на конкретной машине.

**Недостатки интерпретации:**

Низкая скорость работы по сравнении с компиляцией!!! Потому что код читается и интерпретируется на лету – это занимает определенное время!!!

Разработку программ можно сравнить со строительством дома!!!!

И прежде чем начать писать код нашей программы, как и в строительстве сначала необходимо составить план!!! – Можно это делать за компьютером, а можно и на бумаге!!! Если программа небольшая тогда проще это делать на бумаге!!!

Прежде всего нужно вспомнить какие алгоритмы и функции будут использоваться в нашей программе и попробовать их записать в виде алгоритмической схемы.

Например, в программе калькулятора, функции:

1. Функция нажатия кнопки с номерами – 0**,** 1**,** 2**,** 3**,** 4**,** 5**,** 6**,** 7**,** 8**,** 9;

Операции if else нажатия на кнопки!

2. Функция нажатия кнопки операций – **+,** **-,** **\*,** **/**;

Операции if else нажатия на кнопки операций!

3. Функция добавления кнопки с десятичной точкой – ".";

Операции if else нажатия на кнопки с десятичной точкой.

4. Функция очистки экрана – clear;

Операции if else нажатия на кнопки очистки экрана.

5. Функция what do – что делать калькулятору? – это будет цикл for!!!

В этой программе будет, как минимум, 5 функций – ветвление, цикл, переменные, функции.

Этот план – это очень хороший навык визуализации представления структурирования!!!

Готовый шаблон для функции – это снип заготовка!!!

Д/з – Выложить калькулятор

Код HTML:

<!DOCTYPE html>

<html lang=**"ru"**>

<head>

<meta charset=**"utf-8"**>

<meta http-equiv=**"X-UA-Compatible"** content=**"IE=edge"**>

<meta name=**"viewport"** content=**"width=device-width, initial-scale=1"**>

<!-- <meta name="apple-mobile-web-app-capable" content="yes" />-->

<title>**Simple Calc**</title>

<link href=**"** **https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/css/bootstrap.min.css"** rel=**"stylesheet"**>

<link href=**"css/style.css"** rel=**"stylesheet"**>

<link href=**"** **https://fonts.googleapis.com/css?family=VT323"** rel=**"stylesheet"**>

<script src=**"js/script.js"**></script>

</head>

<body>

<div class=**"wrapper"**>

<div class=**"maincontent"**>

<div class=**"container"**>

<main class=**'page'**>

<div class=**"container"**>

<div class=**"page\_left"**>

<div class=**"calc-wrapper"**>

<div class=**"calc"**>

<div class=**"page\_display"**>

<input disabled type=**"text"** value=**"0"** id=**"display"** class=**"calc\_display-input"** placeholder=**"00000000000000"**>

</div>

<div class=**"calc\_buttons"**>

<table class=**"calc\_table"**>

<tr class=**"calc\_buttons-row"**>

<td>

<button type=**"button"** id=**"ce"** class=**"clearBtn btn btn\_default btn\_hover\_red"**>**ce**</button>

</td>

<td>

<button type=**"button"** id=**"c"** class=**"clearBtn btn btn\_default btn\_hover\_red"**>**c**</button>

</td>

<td>

<button type=**"button"** value=**"Деление"** class=**"operation btn btn\_default btn\_hover\_blue"**>**/**</button>

</td>

<td>

<button type=**"button"** value=**"Умножение"** class=**"operation btn btn\_default btn\_hover\_blue"**>**\***</button>

</td>

</tr>

<tr class=**"calc\_buttons-row"**>

<td>

<button type=**"button"** class=**"number btn btn\_default "**>**7**</button>

</td>

<td>

<button type=**"button"** class=**"number btn btn\_default "**>**8**</button>

</td>

<td>

<button type=**"button"** class=**"number btn btn\_default "**>**9**</button>

</td>

<td>

<button type=**"button"** value=**"Вычитание"** class=**"operation btn btn\_default btn\_hover\_blue"**>**-**</button>

</td>

</tr>

<tr class=**"calc\_buttons-row"**>

<td>

<button type=**"button"** class=**"number btn btn\_default "**>**4**</button>

</td>

<td>

<button type=**"button"**class=**"number btn btn\_default "**>**5**</button>

</td>

<td>

<button type=**"button"** class=**"number btn btn\_default "**>**6**</button>

</td>

<td>

<button type=**"button"** value=**"Сложение"** class=**"operation btn btn\_default btn\_hover\_blue"**>**+**</button>

</td>

</tr>

<tr class=**"calc\_buttons-row"**>

<td>

<button type=**"button"** class=**"number btn btn\_default "**>**1**</button>

</td>

<td>

<button type=**"button"** class=**"number btn btn\_default "**>**2**</button>

</td>

<td>

<button type=**"button"** class=**"number btn btn\_default "**>**3**</button>

</td>

<td rowspan=**"2"**>

<button type=**"button"** value=**"Результат"** id=**"equal"** class=**"operation btn btn\_default btn\_hover\_blue"**>**=**</button>

</td>

</tr>

<tr class=**"calc\_buttons-row"**>

<td colspan=**"2"**>

<button type=**"button"** id=**"n0"** class=**"number btn btn\_default btn\_long"**>**0**</button>

</td>

<td>

<button type=**"button"** class=**"btn btn\_default "** id=**"decimal"**>**.**</button>

</td>

</tr>

</table>

</div>

</div>

</div>

</div>

<div class=**"page\_right"**>

<h1 class=**"title"**>

 **Простой калькулятор**

</h1>

<div class=**"about"**>

<div class=**"about\_btn"**>

<a href=**"#"** id=**"howWorkBtn"** class=**"about\_btn-link"**>

 **Как это работает?**

</a>

</div>

<div>

<ul id=**"operationsList"**>

</ul>

</div>

</div>

</div>

</div>

</main>

</div>

</div>

</div>

<script src=**"js/script.js"**></script>

</body>

</html>

Код CSS:

body **{**

 **background-image: url(**"../img/09pXgsGypas.jpg"**);**

**}**

**.**container **{**

 **display: flex;**

 **margin-top: 80px;**

**}**

**.**calc-wrapper **{**

 **background-color: #f1efef;**

 **width: 340px;**

 **height: 410px;**

 **border-radius: 10px;**

 **box-shadow: 1px 35px 91px -23px #050505;**

 **padding-left: 20px;**

**}**

**#display** **{**

 **background-color: #050505;**

 **padding: 0px 10px;**

 **width: 300px;**

 **font-size: 39px;**

 **border-radius: 8px;**

 **margin: 38px 0px;**

 **color: aliceblue;**

 **font-family:** 'Press Start 2P'**, cursive;**

 **text-align: right;**

**}**

**::**-webkit-input-placeholder **{**

 **color: rgba(193, 193, 193, 0.29);**

**}**

**.**title **{**

 **color: white;**

 **font-size: 50px;**

 **margin: 80px 100px;**

**}**

**.**about **{**

 **background-color: white;**

 **width: 220px;**

 **padding: 8px 20px;**

 **font-size: 20px;**

 **border-radius: 23px;**

 **margin-left: 100px;**

 **margin-top: -50px;**

**}**

**.**btn **{**

 **width: 68px;**

 **height: 42px;**

 **background-color: #f4f4f4;**

 **border: 2px solid #e3e3e3;**

 **border-radius: 6px;**

 **font-size: 20px;**

 **text-transform: uppercase;**

 **padding: 0px;**

 **color: #5b5c5d;**

 **cursor: pointer;**

 **margin: 3px;**

**}**

**.**btn**:hover** **{**

 **background-color: #b5b5b5;**

**}**

**.**btn\_hover\_red**:hover** **{**

 **background-color: rgba(235, 53, 53, 0.92);**

**}**

**.**btn\_hover\_blue**:hover** **{**

 **background-color: #2957e0;**

**}**

**#n0** **{**

 **width: 145px;**

**}**

**#equal** **{**

 **height: 92px;**

**}**

Код JavaScript:

**var** Numbers **=** **document.**querySelectorAll**(**'.number'**),**

 Operations **=** **document.**querySelectorAll**(**'.operation'**),**

 DecimalBtn **=** **document.**getElementById**(**'decimal'**),**

 ClearBtns **=** **document.**querySelectorAll**(**'.clearBtn'**),**

 HowWorkBtn **=** **document.**getElementById**(**'howWorkBtn'**),**

 Display **=** **document.**getElementById**(**'display'**),**

 MemoryCurrentNumber **=** 0**,**

 MemoryNewNumber **=** **false,**

 MemoryPendingOperation **=** ''**,**

 operationsList **=** **document.**getElementById**(**'operationsList'**);**

**for** **(var** i **=** 0**;** i **<** Numbers**.**length**;** i**++)** **{**

 **var** number **=** Numbers**[**i**];**

 number**.**addEventListener**(**'click'**,** **function** **(**e**)** **{**

 numberPress**(**e**.**target**.**textContent**);**

 **});**

**}**

**for** **(var** i **=** 0**;** i **<** Operations**.**length**;** i**++)** **{**

 **var** operationBtn **=** Operations**[**i**];**

 operationBtn**.**addEventListener**(**'click'**,** **function** **(**e**)** **{**

 operation**(**e**.**target**.**textContent**);**

 **});**

**}**

**for** **(var** i **=** 0**;** i **<** ClearBtns**.**length**;** i**++)** **{**

 **var** clearBtn **=** ClearBtns**[**i**];**

 clearBtn**.**addEventListener**(**'click'**,** **function** **(**e**)** **{**

 clear**(**e**.**srcElement**.**id**);**

 **});**

**};**

DecimalBtn**.**addEventListener**(**'click'**,** decimal**);**

HowWorkBtn**.**addEventListener**(**'click'**,** howWork**);**

**function** numberPress**(**number**)** **{**

 **if** **(**MemoryNewNumber**)** **{**

 display**.**value **=** number**;**

 MemoryNewNumber **=** **false;**

 **}** **else** **{**

 **if** **(**display**.**value **===** '0'**)** **{**

 display**.**value **=** number**;**

 **}** **else** **{**

 display**.**value **+=** number**;**

 **};**

 **};**

**};**

**function** operation**(**op**)** **{**

 **var** localOperationMemory **=** display**.**value**;**

 **if** **(**MemoryNewNumber **&&** MemoryPendingOperation **!==** '='**)** **{**

 display**.**value **=** MemoryCurrentNumber**;**

 **}** **else** **{**

 MemoryNewNumber **=** **true;**

 **if** **(**MemoryPendingOperation **===** '+'**)** **{**

 MemoryCurrentNumber **+=** **parseFloat(**localOperationMemory**);**

 **}** **else** **if** **(**MemoryPendingOperation **===** '-'**)** **{**

 MemoryCurrentNumber **-=** **parseFloat(**localOperationMemory**);**

 **}** **else** **if** **(**MemoryPendingOperation **===** '\*'**)** **{**

 MemoryCurrentNumber **\*=** **parseFloat(**localOperationMemory**);**

 **}** **else** **if** **(**MemoryPendingOperation **===** '/'**)** **{**

 MemoryCurrentNumber **/=** **parseFloat(**localOperationMemory**);**

 **}** **else** **{**

 MemoryCurrentNumber **=** **parseFloat(**localOperationMemory**);**

 **};**

 display**.**value **=** MemoryCurrentNumber**;**

 MemoryPendingOperation **=** op**;**

 **};**

**};**

**function** clear**(**id**)** **{**

 **if** **(**id **===** 'ce'**)** **{**

 display**.**value **=** '0'**;**

 MemoryNewNumber **=** **true;**

 **}** **else** **if** **(**id **===** 'c'**)** **{**

 display**.**value **=** '0'**;**

 MemoryNewNumber **=** **true;**

 MemoryCurrentNumber **=** 0**;**

 MemoryPendingOperation **=** ''**;**

 **};**

**};**

**function** decimal**()** **{**

 **var** localDecimalMemory **=** display**.**value**;**

 **if** **(**MemoryNewNumber**)** **{**

 localDecimalMemory **=** '0.'**;**

 MemoryNewNumber **=** **false;**

 **}** **else** **{**

 **if** **(**localDecimalMemory**.**indexOf**(**'.'**)** **===** **-**1**)** **{**

 localDecimalMemory **+=** '.'**;**

 **}**

 **};**

 display**.**value **=** localDecimalMemory**;**

**};**

**function** howWork**(**argument**)** **{**

 **for** **(var** i **=** 0**;** i **<** Operations**.**length**;** i**++)** **{**

 **var** newLi **=** **document.**createElement**(**'li'**);**

 **var** operationText **=** Operations**[**i**].**value**;**

 newLi**.**innerText **=** operationText**;**

 operationsList**.**appendChild**(**newLi**);**

 **};**

**};**

Вывод в браузере:



**[Основы Программирования – #12 – Баги](#_top)**

**Ошибки** **в программе** – это **баги**.

А выявление этих ошибок называется **отладкой** или **дебаггингом**.

**Баг** – это жук (в переводе с английского).

В программах, в основном, бывает три основных типа ошибок

1. **Ошибки синтаксиса** – это опечатки в ключевых словах!!

2. **Ошибки выполнения** – возникают тогда когда в коде идет сложение числа и строки или деление на ноль!!!

3. **Логические ошибки** – это ошибки в том как программист придумал код. В отличии от синтаксических ошибок программы с логическими изъянами работают, НО ведут они себя не так как разработчик задумал в начале!!!

Этот вид ошибок связан на этапе принятия решений программистом!!!

Пример 1

Код в Python

i **=** 5

**while** i **<** 15**:**

 **print(**i**)**

 i **=** i **+** 2

Консоль

5

7

9

11

13

Пример 2 – Сложение числа и строки

**>print(**'dima' **+** 1**)**

или деление на ноль

**>print(**10**/**0**)**

Консоль

Error #ошибка выполнения – сложение числа и строки

Пример 3 – Логическая ошибка

age **=** 7

**if** **(**age **<** 5 **and** age **>** 8**)**

**print(**age**)**

Консоль

Error #age не может быть одновременно <5 и >8.

**Как писать код без ошибок?**

1. Проверить код – все ли ключевые слова в тексте программы написаны правильно?

2. Если программа должна вывести текст – заключены ли сообщения в одинарные кавычки?

3. Нет ли в строке лишних пробелов?

4. Все ли в порядке в строках – перед и после выделенной строки с ошибкой.

5. Проверял ли мой код сторонний человек – он свежим взглядом может заметить упущенную ошибку!!!

Д/з – Какие баги были и как мы их смогли победить?

**[Основы Программирования – #13 – Заключение](#_top)**

Самые интересные направления:

1. **Веб разработка**

2. **Фронтэнд разработка**