**Типы данных**

|  |  |
| --- | --- |
| Простые типы | Объекты |
| **Числа** 1, 2.5 | **Массивы** [] |
| **Строки** ‘name’ | **Функции** () |
| **Логические** true/false | **Ошибки** |
| **Symbo**l | **Регулярные выражения** |
| **Undefined** (существует, но не имеет значения) | **Объекты** {} |
| **null** не существует чего-то | **Объекты даты** |

**typeof()** – определение типа данных

**Операторы**

i = i + 1 == i++ инкремент:

i = i + 1 == i-- декремент

Постфиксные i=0; alert(i++); // 0

Префиксные i=0; alert(++i); // 1

% - остаток от деления 5%2=1

\*\* - возведение в степень

|  |  |
| --- | --- |
|  | **Группировка** *()* |
|  | **Вызов функции** |
|  | **Постфиксный инкр/декр** *i++* |
|  | **Логическое отрицание** *!*  **Унарные** *+/-*  **Префиксный инкр/декр** *++i*  **typeof()**  **void**  **delete** |
|  | **Возведение в степень**  *\*\** |
|  | **Умножение** *\**  **Деление** */*  **Остаток от деления** *%* |
|  | **Сложение** *+*  **Вычитание** *-* |
|  | **Меньше/больше** *<, >, <=, >=* |
|  | **Равенство** *==*  **Строгое равенство** *===* |
|  | **Логическое И** *&&* |
|  | **Логическое ИЛИ** *||* |
|  | **Присваивание** *=* |

**Логика**

**НЕ – !**

**И – && (строгое)**

**ИЛИ - || (мягкое)**

**false: 0, “”, null, undefined**

**?? –** оператор нулевого слияния

|  |  |
| --- | --- |
| let user;  alert(user **??** ”Anonim”); // “Anonim” | user = ‘Иван’;  alert(user **??** “Anonim”); // ‘Иван’ |

Если первый оператор (user) не определен (null, undefined), то возвращается оператор2(“Anonim”)

**2** && **1** && **null** && **0** && **undefined; // null**

Если слева **false** – возвращается сам операнд, если **true** – возвращается правый

**0** || **“”** || **2** || **undefined** || **true** || **false; // 2**

Если слева **true** – возвращается сам операнд, если **false** – возвращается правый

**Общение с пользователем**

**alert(‘Hello’);** – всплывающее окно-предупреждение (останавливает выполнение всех скриптов на странице

**confirm(‘Вы здесь’);** - всплывающее окно - вопрос пользователю, подразумевает ответ: Да/Нет

*let a = confirm(‘?’)* // true/false

**promt(‘Сколько вам лет?’, ‘0’)** – пользовательское окно ввода (вопрос пользователю, значение по умолчанию, в поле ввода уже будет введено)

Если пользователь нажмет “ОТМЕНА”, **promt** вернет **null**

**Числа**

**1e3** = 1000

**0x**ff = 255 (16-ричная система)

**0b**11 = 3 (двоичная система)

**0o**11 = 9 (восьмеричная система)

Ошибки нецелых чисел

0.1 +0.2 !== 0.3 0.1 +0.2 == 0.300000004

Решение – sum = 0.1 +0.2*; +sum.toFixed(2)* == 0.30

Округления

**Math.floor**(2.2) = 2 - в меньшую сторону

**Math.ceil**(2.2) = 3 - в большую сторону

**Math.round**(2.8) = 3 - до ближайшего целого

**Math.trunc**(5.2) = 5 - удаление дробной части

**Math.random()**; // псевдослучайное число от 0..1

num = 1.23456;

**num.toFixed**(2) = ‘1,23’ - округление до n-знаков после запятой в ближайшую сторону, представляет число в строковом формате

Считывание целых чисел из строки

parseInt(‘100px’); //100

parseInt(‘2.5’); //2

parseInt(‘a123); //NaN

parseInt(‘ff, 16’); //255 – шестнадцатиричная система

Считывание чисел с точкой из строки

parseFloat (‘12.5em’); //12.5

parseFloat (‘122.5.9’); //122.5

Math.max(1, 2, 5, 19, -9); // 11 – возвращает max число из перечисленных инструментов

Math.min(1, 2, 5, 19, -9); // -9 – возвращает min число из перечисленных инструментов

Math.pow(2, 5); //32 – возведение числа в степень

**Number() –** преобразование в число, аналогично **+’55’ –** унарный +

**isNaN(NaN); isNaN(‘str’);** // true – проверяет, является ли значение **NaN**

**(1234).toString(2) -**  преобразование числа строку в двоичной системе счисления

**(1234).toString() -** преобразование числа строку в десятичной системе счисления

**1234..toString(16) -**  преобразование числа строку в шестнадцатиричной системе счисления

**Строки**

Обратные кавычки?

Спецсимволы

**\n** – перенос строки

**\\** – обратный слеш

**\t** – табуляция

**\’, \”** – кавычки

**String()** – преобразование в строку, аналогично **“str” + 1**

Строки неизменяемы, можно создать новую строку и перезаписать существующую переменную.

Свойства и методы:

|  |  |  |
| --- | --- | --- |
| Длина строки | let b = “*Sdfgh*”.**length**; | // b = **5** |
| Доступ к символам | let str = “*Sdfgh”*;  let b = str**[0]**;  let b = str.**at(0)**;  let b = str[str.lenght **-1**];  let b = str.at**(-1**);  let b = str**[-1];** | // b= “**S**”  // b= “**S**”  // b = “**h**” (последний символ)  // b = “**h**” (последний символ)  // **undefined** ( [] не допускает отрицательные значения) |
| Перебор строки циклом **for..of** | **for** (let char **of** "Hello") {  alert(char); } | // H,e,l,l,o |
| Изменение регистра | let a = str.**toUpperCase()**;  let a = str.**toLowerCase();**  let a = str[1].**toUpperCase();** | // a= “SDFGH”  // a= “s*dfgh*”  // a= “D” отдельного символа |
| Код символа в юникоде | let a = "z".**codePointAt**(0) | // 122 (0 –позиция в строке) |
| Создает символ по коду | let a = **String.fromCodePoint**(90) | // Z |
| Поиск совпадения в строке по подстроке, чувствителен к регистру | 'Widget with id'.**indexOf**('wit')  'Widget with id'.**indexOf**('Wi')  'Widget with id'.**indexOf**('Wit')  'Widget with id'.**indexOf**('i', 6) | // 7 (позиция совпадения)  // 0 (позиция совпадения)  // -1 (нет совпадения)  // 8 (начинать поиск с 6-й поз. ) |
| Поиск с конца строки | str.**lastIndexOf**(substr, position) |  |
| Получение подстроки  str.**slice(start [, end])** | let str = "stringify";  let a = str.**slice(0, 5)**;  let a = str.**slice(2);**  let a = str.**slice(-4, -1);** | // a = ‘strin’ (не включая 5)  // a = ‘ringify’ (с 2 поз. до конца)  // a= ‘gif’ |
| Возвращает подстроку  str.**substring(start [, end])**  Отрицательные значения не поддерживает -3 = 0 | let str = "stringify";  let a = str.**substring(2, 6);**  let a = str.**substring(6, 2);**  let a = str.**slice(6, 2);** | // a = ‘ring’  // a = ‘ring’  // a = “”(пустая сторока) |
|  |  |  |

**Объект**

|  |  |
| --- | --- |
| let obj = {  name: “John”,  age: 25,  isMarried: true  }; | obj.name // John  obj[“name”] // John |
| Создание свойства и присваивание ему значения | **obj.bool = false;** |
| Удаление свойства | **delete** *obj.bool***;** |
| Перебор свойств объекта циклом **for … in** | **for (**let *key* **in** obj**){**  console.log(‘Свойство ’ + key +  ‘имеет значение’ + obj[key]);  **};** |
| Количество свойств в объекте | **Object.***keys(*obj*).***length** // 3 |

**Массив**

let arr = [‘str’, 25, true, 11];

arr[1] ; // 25

arr.lenght; // 4 (длина массива)

**Условные инструкции**

let a = 10;

**if** (a = 10) {

console.log(‘10’);

} **else if** (a>10) {

console.log(‘Больше 10’);

} **else** {

console.log(‘Меньше 10’);

}

Тернарный оператор

let a = 5;

(a == 5) **?** console.log(“true”) **:** console.log(“false”); // условие (a == 5), знак ?, true : false

Для проверки конкретных значений переменной.

let a = 2;

**swith (**a**) {**

**case 2:**

console.log(‘Верно’);

**break;**

**case 3:**

**case 1:**

console.log(‘Чуть-чуть не угадал’);

**break;**

**default:**

console.log(‘Совсем не то’);

**}**

**Циклы**

|  |  |
| --- | --- |
| **WHILE** | **DO WHILE** |
| let num = 50;  **while (**num < 55**) {**  console.log(num);  num++;  **}**  let i = 3;  **while(**i**)** console.log(i--); | Сначала что-то сделать, а потом проверить условие (если необходимо, чтобы тело цикла исполнилось хотя бы 1 раз)  let num = 30;  **do {**  console.log(num);  num++;  **} while (**num < 36**)**; |

|  |  |
| --- | --- |
| **FOR** | |
| **for(**let i = 0; i < 10; i++**) {**  console.log(‘Число’, i);  **}**  let i = 0 – переменная существует только внутри цикла  **for (**начало; условие; шаг**) {**  тело цикла;  **}** | let i = 0;  **for(** ;i < 5; i++**) {**  alert(i);  **}**  let i = 0;  **for(** ;i < 5; **) {**  alert(i++);  **}** |

|  |  |
| --- | --- |
| Прерывание цикла  **break** | Пропуск итерации  **continue** |
| **for(**let i = 0; i < 10; i++**) {**  **if (** i == 6 **) break;**  console.log(i);  **}** | **for(**let i = 0; i < 10; i++**) {**  **if (** i == 6 || i == 8 **) continue;**  console.log(i);  **}** |

**Метки???????????????????**

**Функции**

* Создание и вызов функции

**function** *showTime***(**time**){**

alert**(**time**);**

**};**

*showTime***(**10**);**

* Виды функций

|  |  |
| --- | --- |
| **function declaration** | **function expression** (функциональное выражение) |
| **f1(5);** *//функция будет работать до ее объявления*  **function f1() {**  **…………….**  **}** | **a();** *//функция не будет работать, ошибка*  **let a = function(){**  **………**  **}** |

* Области видимости

|  |  |  |
| --- | --- | --- |
| **Локальная** | **Глобальная** | **Локальная и глобальная** |
| Локальная переменная видна только внутри функции, за пределами функции ее не существует  ***function func(){***  *let a =* ***10****;*  *alert(a);*  *};*  *func(); //****10*** *alert*  ***console.log(a); // error***  ***(****is not defined****)*** | Глобальная переменная видна внутри функции  let a =**10**;  **function** *func(){*  *console.log(a);*  *};*  *func(); //* ***10***  Глобальная переменная изменяется внутри функции и вне функции существует уже измененная  **function** *func(){*  *a =* ***20****;*  *};*  *func();*  *console.log(a); //* ***20*** | Локальная и глобальная переменные с одинаковым именем – разные переменные внутри и вне функции  let a =**10**;  **function** *func(){*  *let a =* ***20****;*  *console.log(a);*  *};*  *func(); //* ***20***  *console.log(a); //* ***10*** |

Функция заканчивает выполнение, когда встречает **return** или **}**.

* Стрелочные функции

**let a = (x,y) => x+y;**

console.log(a(1,3)); //4

Не имеет своего контекста вызова

Чаще всего используется в обработчиках событий

* Замыкания (вложенные функции)

**Linux**

**pwd** – путь к файлу

**ls (-al)** – содержимое текущей директории (+скрытые, расширенные свойства файлов)

**mkdir –** создание папки

**touch –** создание файла **touch** {text, asd}.txt – создание нескольких файлов

**mv -**  переименование файла/папки

**rm (-R) –** удаление файла (папки)

**cp –** копировать

**&&** - соединение 2 команд mkdir javascript **&&** cd javascript

**| -** конвеер – перенаправляет вывод предыдущей команды в последующую

pip3 freeze **|** grep requests – сначала выводится список приложений pip3, а потом фильтруется grepом (находит приложения включающие слово requests

**> -** записать в файл (перезаписывает содержимое файла)

**>> -** записать в файл (добавляет запись в файл в конец файла)

**stat** text.txt – просмотреть инфу о файле (дата создания например)

**grep** – утилита сортировки и фильтрации

**grep** “abs” *text.txt* – поиск “вхождения”/сочетания “ abs” в файле *text.txt*

**-i** – нечувствительно к регистру

**-w** – полное вхождение (“absd” не войдет, только точно “abs” )

**-n** – показывает номер строк

**-l** – покажет только название файлов, где найдены вхождения

**-c** – покажет только название файлов, где найдены вхождения и количество вхождений в каждом файле

**-v** – инвертировать поиск – показать все, кроме найденых

**-A2** – покажет контекст - строку в которой вхождение и 2 строки после

**-B4** – покажет контекст - строку в которой вхождение и 4 строки до

**-C3** – покажет контекст - строку в которой вхождение и 3 строки до и после

**\*.txt** – поиск вхождения только в текстовых файлах

Регулярные выражения

**grep** –P “\d{3}-\d{2}” *1.txt*

где –P – тип регулярных выражений (перл), d{3} – 3 любые цифры, *1.txt –* файл, где ищем

**grep** –P “\d{3}-\d{2}” *1.txt > 2.txt –* запись результатов в файл *2.txt* (запишет строки с вхождениями)

-o – запишет не строки, а результаты вхождения (123, 456, 195 и т.д)

-f – использование паттерна (создаем файл с регулярным выражением в текстовом виде)

grep –owiPf patterns.txt 1.txt

Алиасы

Во время работы в терминале, временные

**$alias** – показывает все заданные алиасы

**$alias** sv = ‘source vevc/bin/activate’ – (значения, аргументы, путь в исполняемому файлу)

**$unalias** sv

Постоянные

Редактируем файл ~/.bachrc – дописываем в конец нужные нам алиасы

**GIT**

**git init –** инициализация папки проекта

echo >> README.md - создание файла документации/описание проекта

Глобальная конфигурация

**git config - - global user.name “Name”**

**git config - - global user.email**

**git config - - list** – информация о конфигурации

Локальная конфигурация – для конкретного проекта

**git config - - global user.name “Name”**

**git config - - global user.email** [**ex@gmail.com**](mailto:ex@gmail.com)

Добавление в отслеживание

**git add –A** – все файлы в папке

**git add \*.txt** – все текстовые файлы

**git add text.txt** – конкретный файл

**git rm --cached text.txt** – убрать из индекса (отслеживания)

**git commit –am “Первый коммит”** – фиксация изменений

**git commit –-amend** – редактировать/заменить содержимое предыдущего коммита, если необходимо внести мелкие правки, заменить недочеты, чтобы не засорять мелочью историю

**git status** – состояние файлов в проекте

Отслеживание изменений

**git log** – логи изменений

**git log** **--pretty=format:“%h\_%ad | %s%d\_[%an]“ --graph --date = short** - вывод лога в удобном виде ( **h —** сокращенный формат хеша, **s —** комментарий к коммиту, **ad** — дата коммита, **an —** автор коммита)

**--pretty=online** - в одну строчку

**--max-count=3 —** вывести последние 3 строчки

**--date = short —** сокращенный формат даты

**--author=“Yakovkireev“ —** по автору коммита

**git log –-help, git add –-help -**  документация по команде

**git checkout** ХЕШ КОММИТА - откат к нужному коммиту по хешу

**git checkout text.txt -** откат к последнему состоянию конкретного файла

**git checkout main -** откат к последней версии ветки MAIN

Файл **.gitignore** - записать в него папки, файлы, которые требуется игнорировать (log.txt, logs)

Подключение репозитория GITHUB

- Создать репозиторий на github

- **git remote add origin https://github.com/…** - добавление репозитория

- **git puch -u origin main –** первый залив проекта на гитхаб

- **git puch –** отправка изменений на гитхаб из локального репозитория

- **git clone https://github.com/… –** клонирование репозитория

Токен на гитхабе: создание Настройки/ настройки разработчика

**git pull = fetch + merge –** вытянуть с репозитория изменения на локальный репозиторий

**git fetch** – обновление локальной копии удаленного репозитория(комп узнает изменения и показывает разницу)