**Рабочее пространство**

VSCode <https://code.visualstudio.com/>

Список необходимых плагинов для VSCode:

1. All Autocomplete
2. Auto Close Tag
3. Auto Complete Tag
4. Auto Rename Tag
5. Beautify
6. Code Runner
7. Import Cost
8. JavaScript (ES6) code snippets
9. jshint
10. Live Server
11. Multiple clipboards for VSCode
12. Reactjs code snippets
13. Sass
14. Theme - Oceanic Next
15. vscode-icons
16. ESLint

VSCode -> Emmet :

|  |  |
| --- | --- |
| ! <Tab> | <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta http-equiv="X-UA-Compatible" content="IE=edge">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body> |
| .class < Tab > | <div class="class"></div> |
| .class>button < Tab > | <div class="class"><button> </button></div> |
| .class>button>div < Tab > | <div class="class">      <button>          <div></div>      </button>  </div> |

Плагин jshint

Конфиг плагина jshint – файл .jshintrc, помещается в папку проекта или в папку выше, которая включает проекты, где должен действовать конфиг.

Окружение Node.js

<https://nodejs.org/en/>

**4. Подключение jscript**

На уровне файла index.html создаём папку js/ и в ней файл script.js.

В html перед закрывающим тегом body:

<body>

    <p> Hello!</p>

…

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

</body>

Атрибут тега script

type="text\javascript"

устаревший. Его надо удалять.

Проверка:

а) На веб-странице F12 или ПКМ-«Исследовать элемент». Вкладка Console. Там вывод js.

б) Или консоль в VSCode – вкладка Output. Выделить нужную часть кода, ПКМ-Run code. В Output’е видим вывод.

**5.1 Объявление переменных**

По стандарту ES6

let number = 5;

const width = 1;

Имена регистрозависимые.

Вариант

var name = Ivan;

устаревший. Не использовать.

let и const не подвержены всплытию переменной, как var, видны только после объявления и в блоке кода, ограниченном фигурными скобками { и }.

Новый стандарт не поддерживается старыми браузерами. Проверить применимость (поддержку возможностей) конструкций языка на сайте <https://caniuse.com/>.

**5.2 Режим строгого современного кода**

Директива

"use strict";

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode>

**5.3 Стилистические правила**



**6. Классификация типов данных в JavaScript**

****

<https://medium.com/@hydrock/bigint-новый-тип-данных-в-js-dd5c29446570>

<https://learn.javascript.ru/symbol>

//number

let number = 4.6;

//string

let name = 'Alex';

let name = "Alex";

let name = `Alex`;

//null

console.log(result);

//undefined

let und;

console.log(und);

//object

const obj = {

    name: "John",

    age: 25,

    isMarried: false

};

console.log(obj.name);

console.log(obj["name"]);

const emptyObj = {}; //пустой объект

//array

let arr = ['plum.png', 'orange.jpg', 6, 'apple.bmp'];

console.log(arr[1]);  // orange.jpg

const emptyArr = []; //пустой массив

**7. Простое общение с пользователем**

Alert

alert('Привет!');



Confirm

const result = confirm('Are You here?');



const answer = prompt("Как вас зовут?", "");



Если пользователь нажмет Отмена, то answer=null.

Все ответы пользователя имеют тип данных string. Если пользователь ничего не введёт, то answer=’’.

Проверка типа данных:

console.log(typeof(answer));

string

(https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/typeof)

Для преобразования в число можно использовать оператор +

const answer = +prompt("Сколько вам лет?", "");

console.log(typeof(answer));

number

Заполнение массива ответами пользователя

const answers = [];

answers[0] = prompt("Ваше имя?", "");

answers[1] = prompt("Ваша фамилия?", "");

answers[2] = prompt("Ваш возраст?", "");

document.write(answers);



console.log(typeof(answer));

object

(<https://developer.mozilla.org/ru/docs/Web/API/Document/write> )

Команда

document.write(answers);

устаревшая, полностью заменяет содержимое страницы.

Однако alert, prompt и т.п. блокируют построение страницы, поэтому Hello! остаётся.

**8. Интерполяция (ES6)**

- вставка значения переменной в строку.

Используется вместо конкатенации строк.

Например, два алерта идентичны

const user = "Ivan";

alert('Привет, ' + user + '!');

alert(`Привет, ${user}!`);

В строке, использующей интерполяцию обязательно использование обратных кавычек (на клавише Ё) (см. второй алерт).

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Template_literals>

**9. Операторы в JS**

+, -, \*, / - арифметические

Конкатенация

console.log('4' + "5");

// expected 45

Унарный плюс (преобразует тип string к number)

console.log(4 + +'5');

// expected 9

Инкремент, декремент (постфиксная форма)

let a = 10,

    b = 10;

a++;

b--;

console.log(a);

// expected 11

console.log(b);

// expected 9

Инкремент, декремент (префиксная форма)

let a = 10,

    b = 10;

++a;

--b;

console.log(a);

// expected 11

console.log(b);

// expected 9

Различие этих форм в выполнении «на месте»:

Постфиксная форма сначала возвращает исходное значение

let a = 10,

    b = 10;

console.log(a++);

// expected 10

console.log(b--);

// expected 10

Префиксная форма

let a = 10,

    b = 10;

console.log(++a);

// expected 11

console.log(--b);

// expected 9

Остаток от деления

console.log(8%3);

// expected 2

=

Присваивание

let a = 10;

Сравнение

console.log(2\*4 == 8);

// expected true

Строгое сравнение (по значению и по типу данных)

console.log(2\*4 == "8"); //число сравнивается со строкой

// expected true

console.log(2\*4 === "8"); //число сравнивается со строкой

// expected false

Логическое И и ИЛИ

let isChecked = true,

    isClosed = false;

console.log(isChecked || isClosed); //ИЛИ

// expected true

console.log(isChecked && isClosed); //И

// expected false

Отрицание !

console.log(isChecked && !isClosed);

// expected true

Не равно !=

console.log(6 != 8);

// expected true

console.log(6 != '6');

// expected false

console.log(6 !== '6');

// expected true

!= - нестрогое сравнение,

!== - сравнение строгое по типу.

[Таблица](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Operator_Precedence#table) приоритетов операторов

Данная таблица упорядочена с самого высокого приоритета (20) до самого низкого (1).

|  |  |  |  |
| --- | --- | --- | --- |
| **Приоритет** | **Тип оператора** | **Ассоциативность** | **Конкретные операторы** |
| 20 | [Группировка](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Grouping) | не определено | ( … ) |
| 19 | [Доступ к свойствам](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Property_Accessors#dot_notation) | слева направо | … . … |
| [Доступ к свойствам с возможностью вычисления](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Property_Accessors#bracket_notation) | слева направо | … [ … ] |
| [new](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/new) (со списком аргументов) | не определено | new … ( … ) |
| [Вызов функции](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Functions) | слева направо | … ( … ) |
| 18 | [new](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/new) (без списка аргументов) | справа налево | new … |
| 17 | [Постфиксный инкремент (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators) | не определено | … ++ |
| [Постфиксный декремент (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators) | … -- |
| 16 | [Логическое отрицание (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#logical_not) | справа налево | ! … |
| [Побитовое отрицание (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#bitwise_not) | ~ … |
| [Унарный плюс (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#unary_plus) | + … |
| [Унарный минус (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#unary_negation) | - … |
| [Префиксный инкремент (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#increment) | ++ … |
| [Префиксный декремент (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#decrement) | -- … |
| [typeof](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/typeof) | typeof … |
| [void](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/void) | void … |
| [delete](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/delete) | delete … |
| [await](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/await) | await … |
| 15 | [Возведение в степень (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#exponentiation) | справа налево | … \*\* … |
| 14 | [Умножение (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#multiplication) | слева направо | … \* … |
| [Деление (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#division) | … / … |
| [Остаток (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#remainder) | … % … |
| 13 | [Сложение (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#addition) | слева направо | … + … |
| [Вычитание (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#subtraction) | … - … |
| 12 | [Побитовый сдвиг влево](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators) | слева направо | … << … |
| [Побитовый сдвиг вправо (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators) | … >> … |
| [Сдвиг вправо с заполнением нулей (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators) | … >>> … |
| 11 | [Меньше (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#less_than_operator) | слева направо | … < … |
| [Меньше или равно (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#less_than__or_equal_operator) | … <= … |
| [Больше (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#greater_than_operator) | … > … |
| [Больше или равно (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#greater_than_or_equal_operator) | … >= … |
| [in](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/in) | … in … |
| [instanceof](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/instanceof) | … instanceof … |
| 10 | [Равно (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#equality) | слева направо | … == … |
| [Не равно (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#inequality) | … != … |
| [Строго равно (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#identity) | … === … |
| [Строго не равно (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#nonidentity) | … !== … |
| 9 | [Побитовое «И» (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#bitwise_and) | слева направо | … & … |
| 8 | [Побитовое исключающее «ИЛИ» (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#bitwise_xor) | слева направо | … ^ … |
| 7 | [Побитовое «ИЛИ» (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#bitwise_or) | слева направо | … | … |
| 6 | [Логическое «И» (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#logical_and) | слева направо | … && … |
| 5 | [Логическое «ИЛИ» (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#logical_or) | слева направо | … || … |
| 4 | [Условный](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Conditional_Operator) | справа налево | … ? … : … |
| 3 | [Присваивание (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#Assignment_operators) | справа налево | … = … |
| … += … |
| … -= … |
| … \*\*= … |
| … \*= … |
| … /= … |
| … %= … |
| … <<= … |
| … >>= … |
| … >>>= … |
| … &= … |
| … ^= … |
| … |= … |
| 2 | [yield](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/yield) | справа налево | yield … |
| [yield\*](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/yield*) | yield\* … |
| 1 | [Запятая / Последовательность](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Comma_Operator) | слева направо | … , … |

Побитовые операторы <https://learn.javascript.ru/bitwise-operators>

**10. Система контроля версий Git и сервис GitHub**

<https://git-scm.com/download/win>

Документация по Git <https://git-scm.com/book/ru/v2/Введение-О-системе-контроля-версий>

Обучение Git <https://githowto.com/ru>

Интерактивное обучение Git <https://learngitbranching.js.org/?locale=ru_RU>

После установки Git идем в терминал, в папку проекта (можно прямо в терминале VSCode, но сначала перезапустить его). Инициализируем пустой локальный репозиторий Git для данного проекта:

… \Project\_1\_Udemy> git init

В проекте появилась папка .git. Конфигурируем пользователя

…\Project\_1\_Udemy> git config --local user.name "RustemV"

…\Project\_1\_Udemy> git config --local user.email rustemv@gmail.com

(для всех проектов –global. . При указании параметра --global используется файл ~/.gitconfig , он хранит настройки конкретного пользователя. В системах семейства Windows Git ищет файл .gitconfig в каталоге $HOME (C:\Documents and Settings\$USER или C:\Users\$USER для большинства пользователей).

Три состояния файлов локального репозитория:

…\Project\_1\_Udemy> git status

- файлы созданы

- файлы индексируются (tracked)

- созданы контрольные точки (commits)

Добавить файлы проекта в локальный репозиторий git

…\Project\_1\_Udemy> git add -A

(-A или --all)

Эту команду выполнять каждый раз при создании нового файла. Или персонально файл:

…\Project\_1\_Udemy> git add main.css

Создание контрольной точки

…\Project\_1\_Udemy> git commit –a –m”текст\_комментария”

(Петриченко рекомендует перед созданием контрольной точки каждый раз выполнять git add -A «Для правильной работы системы контроля версий каждый раз нужно добавлять файлы в индекс»)

Просмотр логов с контрольными точками

…\Project\_1\_Udemy> git log

Выход из лога q

Сервис GitHub

<https://github.com/>

Заводим аккаунт [Rv-csi@yandex.ru](mailto:Rv-csi@yandex.ru) / @Github1812#

Создаем новый удалённый репозиторий (кнопка New). По инструкции на странице привязываем локальный репозиторий к удаленному: в локальном терминале

…\Project\_1\_Udemy> git remote add origin https://github.com/RustemV/Node.js\_Project1.git

…\Project\_1\_Udemy> git branch -M main

…\Project\_1\_Udemy> git push -u origin main

Теперь каждый раз после создания очередной точки

…\ Project\_1\_Udemy > git commit –a –m”текст\_комментария”

отправляем её в удаленный репозиторий на Github

…\ Project\_1\_Udemy > git push

**11. Работа с GitHub с разных компьютеров**

а) Клонирование проекта

Допустим, проект был создан на компьютере № 1.

На компьютере № 2, где еще нет проекта, необходимо клонировать репозиторий. Для этого в Github нужно скопировать путь к нему типа «https://github.com/RustemV/Project\_1\_Udemy.git»



Затем в терминале VSCode:

- перейти в рабочую папку

- выполнить

…\Work> git clone скопированный\_путь имя\_папки\_проекта

Например,

…\Work> git clone https://github.com/RustemV/Project\_1\_Udemy.git Project\_1

На компьютере № 2 создается папка Project\_1 со всеми файлами проекта в ней.

При этом локальный репозиторий уже связан с удаленным. Привязывать не нужно.

После изменения проекта на этом компе № 2 создаем commit и отправляем на Github

…\ Project\_1\_Udemy > git add -A

…\ Project\_1\_Udemy > git commit –a –m”modified at PC#2”

…\ Project\_1\_Udemy > git push

Теперь, когда нужно работать с проектом на компьютере № 1 выполняем в терминале в папке проекта

…\ Project\_1\_Udemy > git pull

А затем, после модификации проекта, обычным образом создается контрольная точка

…\Work > git add -A

…\Work > git commit –a –m”modified at PC#1”

…\Work > git push

Если к моменту выполнения git push , в репозитории на Github’е произошли изменения, то будет выведено предупреждение

To https://github.com/RustemV/Project\_1\_Udemy.git

! [rejected] main -> main (fetch first)

error: failed to push some refs to 'https://github.com/RustemV/Project\_1\_Udemy.git'

hint: Updates were rejected because the remote contains work that you do

hint: not have locally. This is usually caused by another repository pushing

hint: to the same ref. You may want to first integrate the remote changes

hint: (e.g., 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

с предложением предварительно выполнить интеграцию удаленных изменений с помощью git pull . При выполнении git pull произойдет merge – слияние изменений внесенных в локальном репозитории с изменениями в удаленном репозитории. Ввести комметарий, Ctrl+C 2 раза, :wq!

В списке коммитов на Github появится Merge.

Игнорирование

Чтобы Git не помещал некоторые файлы, лежащие в проекте, в репозиторий, в папку проекта нужно положить файл .gitignore, в котором указываются игнорируемые файлы.

Пример .gitignore

# See https://help.github.com/articles/ignoring-files/ for more about ignoring files.

# dependencies

/node\_modules

/.pnp

.pnp.js

# testing

/coverage

# production

/build

# misc

.DS\_Store

.env.local

.env.development.local

.env.test.local

.env.production.local

npm-debug.log\*

yarn-debug.log\*

yarn-error.log\*

GitKraken

Программа для управления коммитами <https://www.gitkraken.com/>

**12. Практика, ч.1. Начинаем создавать приложение**

Задание

/\* Задание на урок: \*/

'use strict';

/\*1) Создать переменную numberOfFilms и в неё поместить ответ от пользователя на вопрос:

'Сколько фильмов вы уже посмотрели?'\*/

let numberOfFilms;

numberOfFilms = +prompt('Сколько фильмов Вы уже посмотрели?', '');

console.log(numberOfFilms);

/\*2) Создать объект personalMovieDB и в него поместить такие свойства:

    - count - сюда передается ответ на первый вопрос

    - movies - в это свойство поместить пустой объект

    - actors - тоже поместить пустой объект

    - genres - сюда поместить пустой массив

    - privat - в это свойство поместить boolean(логическое) значение false \*/

let personalMovieDB = {

    count: numberOfFilms,

    movies: {},

    actors: {},

    genres: [],

    privat: false

};

/\*3) Задайте пользователю по два раза вопросы:

    - 'Один из последних просмотренных фильмов?'

    - 'На сколько оцените его?'

Ответы стоит поместить в отдельные переменные

Записать ответы в объект movies в формате:

    movies: {

        'logan': '8.1'

    }

Проверить, чтобы все работало без ошибок в консоли \*/

let a = prompt('Один из последних просмотренных фильмов?', '');

let b = prompt('На сколько оцените его?', '');

personalMovieDB.movies[a] = b;

a = prompt('Один из последних просмотренных фильмов?', '');

b = prompt('На сколько оцените его?', '');

personalMovieDB.movies[a] = b;

console.log(personalMovieDB);

При обращении к объекту moves использован синтаксис movies[a], а не movies.a. Такое обращение работает более надежно в случае ввода в качестве значения переменной a произвольного текста с пробелами.

**13. Условия**

<https://developer.mozilla.org/ru/docs/Learn/JavaScript/Building_blocks/conditionals>



Оператор if

if (a == b) {

    console.log('OK!');

} else {

    console.log('Error')

}

Запись if с помощью тернарного оператора

(a == b) ? console.log('OK!') : console.log('Error');

Вложенный if

if (a == b) {

    console.log('OK!');

} else if (a > c) {

    console.log('Error 1')

} else {

    console.log('Error 2')

}

Проверка на строгое соответствие с помощью switch

switch(a) {

    case 49:

        console.log('Меньше');

        break;

    case 50:

        console.log('Равно');

        break;

    case 51:

        console.log('Больше');

        break;

    default:

        console.log('Вне границ');

        break;

}

**14. Циклы**

<https://learn.javascript.ru/while-for>



While

let n = 50;

while (n <= 55) {

    console.log(n);

    n++;

}

Do … while

n = 50;

do {

    console.log(n);

    n++;

}

while ( n <= 55);

For

let num = 51;

for (let i = 1; i<=8; i++) {

    console.log(num);

    num++;

}

Прерывание цикла break

let num = 51;

for (let i = 1; i<0; i++) {

    if (i === 10 ){

        break;

    }

    console.log(num);

    num++;

}

Прерывание цикла continue. Contrinue прерывает только одну итерацию и переходит к следующей.

let num = 51;

for (let i = 1; i<0; i++) {

    if (i === 10 ){

        continue;

    }

    console.log(num);

    num++;

}

**15. Практика, ч2. Применяем условия и циклы**

Задание

/\* Задание на урок: \*/

"use strict";

let personalMovieDB = {

    count: 5,

    movies: {},

    actors: {},

    genres: [],

    privat: false

};

/\* 1) Автоматизировать вопросы пользователю про фильмы при помощи цикла  \*/

let a, b;

for (let i = 1; i <= 2; i++) {

    a = prompt('Один из последних просмотренных фильмов?', '');

    b = prompt('На сколько оцените его?', '');

    personalMovieDB.movies[a] = b;

}

console.log(personalMovieDB);

/\* 2) Сделать так, чтобы пользователь не мог оставить ответ в виде пустой строки,

отменить ответ или ввести название фильма длинее, чем 50 символов. Если это происходит -

возвращаем пользователя к вопросам опять \*/

let aa, bb;

console.log(aa);

for (let i = 1; i <= 2; i++) {

    do {

        //aa = '';

        aa = prompt('Один из последних просмотренных фильмов?', '');

    } while ((aa == '') || aa == null || (aa.length > 50));

    do {

        //bb = '';

        bb = prompt('На сколько оцените его?', '');

    } while (bb == '' || bb == null)

    personalMovieDB.movies[aa] = bb;

}

console.log(personalMovieDB);

/\* 3) При помощи условий проверить  personalMovieDB.count, и если он меньше 10 - вывести сообщение

"Просмотрено довольно мало фильмов", если от 10 до 30 - "Вы классический зритель", а если больше -

"Вы киноман". А если не подошло ни к одному варианту - "Произошла ошибка" \*/

let personalMovieDB.count = +prompt('Сколько фильмов Вы уже посмотрели?', '');

let cnt = personalMovieDB.count;

if (cnt < 10) {

    alert('мало');

} else if (cnt > 30) {

    alert('много');

} else if (cnt >=10 && cnt <=30) {

    alert('классика');

} else {

    alert('ошибка');

}

/\* 4) Потренироваться и переписать цикл еще двумя способами \*/

// 4.1)

let aa, bb;

let i = 0;

do {

    do {

        //aa = '';

        aa = prompt('Один из последних просмотренных фильмов?', '');

    } while ((aa == '') || aa == null || (aa.length > 50));

    do {

        //bb = '';

        bb = prompt('На сколько оцените его?', '');

    } while (bb == '' || bb == null)

    personalMovieDB.movies[aa] = bb;

    i++;

} while (i < 2);

console.log(personalMovieDB);

// 4.2)

let aa, bb;

let i = 0;

while (i < 2) {

    do {

        //aa = '';

        aa = prompt('Один из последних просмотренных фильмов?', '');

    } while ((aa == '') || aa == null || (aa.length > 50));

    do {

        //bb = '';

        bb = prompt('На сколько оцените его?', '');

    } while (bb == '' || bb == null)

    personalMovieDB.movies[aa] = bb;

    i++;

}

console.log(personalMovieDB);

// Код возьмите из предыдущего домашнего задания

Вариант преподавателя

Начало формы

Конец формы

Начало формы

Конец формы

|  |  |
| --- | --- |
|  | /\* Задание на урок: |
|  |  |
|  | 1) Автоматизировать вопросы пользователю про фильмы при помощи цикла |
|  |  |
|  | 2) Сделать так, чтобы пользователь не мог оставить ответ в виде пустой строки, |
|  | отменить ответ или ввести название фильма длинее, чем 50 символов. Если это происходит - |
|  | возвращаем пользователя к вопросам опять. (К любой строке можно обратиться как |
|  | str.length - и получить её длину) |
|  |  |
|  | 3) При помощи условий проверить personalMovieDB.count, и если он меньше 10 - вывести сообщение |
|  | "Просмотрено довольно мало фильмов", если от 10 до 30 - "Вы классический зритель", а если больше - |
|  | "Вы киноман". А если не подошло ни к одному варианту - "Произошла ошибка" |
|  |  |
|  | 4) Потренироваться и переписать цикл еще двумя способами\*/ |
|  |  |
|  | 'use strict'; |
|  |  |
|  | const numberOfFilms = +prompt('Сколько фильмов вы уже посмотрели?', ''); |
|  |  |
|  | const personalMovieDB = { |
|  | count: numberOfFilms, |
|  | movies: {}, |
|  | actors: {}, |
|  | genres: [], |
|  | privat: false |
|  | }; |
|  |  |
|  | for (let i = 0; i < 2; i++) { |
|  | const a = prompt('Один из последних просмотренных фильмов?', ''), |
|  | b = prompt('На сколько оцените его?', ''); |
|  |  |
|  | if (a != null && b != null && a != '' && b != '' && a.length < 50) { |
|  | personalMovieDB.movies[a] = b; |
|  | console.log('done'); |
|  | } else { |
|  | console.log('error'); |
|  | i--; |
|  | } |
|  | } |
|  |  |
|  | if (personalMovieDB.count < 10) { |
|  | console.log("Просмотрено довольно мало фильмов"); |
|  | } else if (personalMovieDB.count >= 10 && personalMovieDB.count < 30) { |
|  | console.log("Вы классический зритель"); |
|  | } else if (personalMovieDB.count >= 30) { |
|  | console.log("Вы киноман"); |
|  | } else { |
|  | console.log("Произошла ошибка"); |
|  | } |
|  |  |
|  | console.log(personalMovieDB); |

**16. Функции**

Несколько подходов:



16.1 Functioon Declaration

function *имя\_функции*(*аргумент1*, *аргумент2*) {

тело функции

return *возвращаемое\_значение*

}

*имя\_функции*(*a, b);*

return останавливает выполнение функции и возвращает результат на уровень выше. Код после return не выполняется.

Переменные, объявленные внутри функции – локальные, существуют только внутри функции, не видны вне функции. Переменные, не объявленные внутри функции, глобальные, также могут использоваться внутри функции. Поиск переменных на уровне выше и далее на следующем уровне и т.д. называется замыканием функции. <https://medium.com/nuances-of-programming/я-никогда-не-понимал-замыкания-в-javascript-часть-первая-3c3f02041970>

<https://learn.javascript.ru/closures>

*имя\_функции* принято начинать с глагола: showForm, getName.

Functioon Declaration создается интерпретатором до выполнения кода. Поэтому её можно вызвать выше, чем она объявлена. Аналогично переменным var.

function calc(a, b) {

    return (a + b);

console.log(‘end’); //unreachable code

}

console.log(calc(4, 6));

console.log(calc(7, 56));

16.2 Function Expression – функциональное выражение

Объявление функции в контексте какого-либо выражения, например присваивания

const *имя\_функции* = function (*аргумент\_1, аргумент\_2*) {

тело функции

return *возвращаемое\_значение*;

};

Сравнение с Functioon Declaration

//Function Declaration

function sum(a, b) {

    return a + b;

}

let s = sum(2, 4);

//Function Expression

const sum = function (a, b) {

    return a + b;

};

s = sum(2, 4);

16.3 Стрелочные функции

Введены стандартом ES6.

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Functions/Arrow_functions>

const *имя\_функции* = (*аргумент\_1, аргумент\_2*) => {

тело функции

return *возвращаемое\_значение*;

}

Если в стрелочной функции одна строка, то можно опустить фигурные скобки – см. код выше, что сильно экономит объем.

//Стрелочная функция

const sum = (a, b) => a + b;

s = sum(2, 4);

console.log(s);

Такая функция не имеет контекста вызова, поэтому их используют для обработчиков событий.

**17. Методы и свойства строк и чисел**

**17.1 Строки**

Посмотреть весь список можно в контекстном меню редактора



Или в документации <https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String>

<https://learn.javascript.ru/string>

Также в консоли браузера:

Console.dir(String)

Свойство length – длина строки

let str = "Hello world!";

console.log(str.length);  // 12

\* также пименимо к массивам

Получение символа (readonly)

let str = "Hello world!";

console.log(str[6]);  // w

Метод приведения к верхнему и нижнему регистру

let str = "Hello world!";

console.log(str.toUpperCase());  // HELLO WORLD!

console.log(str.toLowerCase());  // hello world!

console.log(str);               // Hello world!

Сама строка не меняется

Метод поиска подстроки

let str = "Hello world!";

console.log(str.indexOf('ello'));  // 1

Может использоваться как проверка вхождения подстроки в строку. При отсутствии подстроки возвращается -1.

Метод извлечения подстроки

let str = "Hello world!";

console.log(str.slice(6, 11));  // world

возвращает с 6-го до 11-го (не включая!) символа.

При отрицательных аргументах отсчет справа.

При отсутствии 2-го аргумента подстрока извлекается до конца строки.

Метод substring

То же что и slice, но не поддерживает отрицательные аргументы (воспринимаются как 0).

Метод substr

Вырезает подстроку с *аргумент\_1* символа длиной *аргумент\_2* символов.

let str = "Hello world!";

console.log(str.substring(6, 11));  // world

let str = "Hello world!";

console.log(str.substr(6, 5));  // world

**17.2 Числа**

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number>

<https://learn.javascript.ru/number>

Есть минибиблиотека в браузере Math.

Можно увидеть её состав, если в консоли браузера набрать

>Math.



Округление чисел

const num = 12.2;

console.log(Math.round(num)); // 12

Извлечение чисел

const text = "12.2px";

console.log(parseInt(text)); // 12

console.log(parseFloat(text)); // 12.2

**18. Практика , ч3. Используем функции**

**19. Callback- функции**

Асинхронное выполнение

'use strict';

function first() {

    setTimeout(function() {

        console.log('1st');

    }, 500);

}

function second() {

    console.log('2nd');

}

first();

second();

Вывод будет:

2nd

1st

т.к. в первой функции введена задержка, несмотря на то, что вызов функции first был первым.

Для того, чтобы функция была выполнена после того, как будет выполнена первая, используют callback-функции. <https://learn.javascript.ru/callbacks>

Сallback - это функция, переданная в другую функцию в качестве аргумента, которая затем вызывается по завершению какого-либо действия. Как правило они используются для асинхронного кода, чтобы быть уверенным, что функция не сработает, пока не закончит свою работу другая функция.

// объявление функции

function *имя\_функции* (*аргумент\_1*, *имя\_callback-функции*) { //см. примечание[[1]](#footnote-1)

тело функции;

*имя\_callback-функции* (); // вызов callback-функции в объявлении функции

}

// объявление callback-функции

function *func\_сallbck() {*

тело callback-функции;

*}*

//вызов функции

*имя\_функции* (*arg\_example*, *func\_сallbck*);

Передача callback-функции в качестве аргумента при вызове функции может быть в виде анонимной функции (тогда предварительное объявление callback-функции отсутствует):

*имя\_функции* (*arg\_example*, function () {

тело callback-функции;

} );

**20. Объекты. Деструктуризация объектов (ES6)**

<https://javascript.ru/tutorial/object/intro>

Пример объекта

const options = {

    name: 'test',    //свойства

    width: 1024,

    height: 1024,

    colors: {

        border: 'black', // вложенные свойства

        bg: 'red',

    },

}

**20.1 Свойства**

Обращение к свойству - через точку

console.log(options.name);

Обращение к свойству - через [ ]. Внимание – кавычки!

console.log(options['name']);

и к вложенному свойству

console.log(options['colors']['border']);

(см. ниже Деструктуризация объекта).

Удаление свойства

delete options.name;

Перебор всех свойств объекта for in

<https://learn.javascript.ru/object-for-in>

for (let key in options) {

    console.log(`Свойство ${key} имеет значение: ${options[key]}`);

}

И то же для вложенных свойств

for (let key in options) {

    if (typeof(options[key]) === 'object') {

        for (let i in options[key]) {

            console.log(`Свойство ${i} имеет значение: ${options[key][i]}`);

        }

    } else {

        console.log(`Свойство ${key} имеет значение: ${options[key]}`);

    }

}

Вывод:

Свойство name имеет значение: test

Свойство width имеет значение: 1024

Свойство height имеет значение: 1024

Свойство border имеет значение: black

Свойство bg имеет значение: red

Определить размер (количество свойств) объекта:

let counter = 0;

for (let key in options) {

        counter++;

}

**20.2 Методы**

Определить размер (количество свойств) объекта.

Метод keys() возвращает массив ключей объекта. Для определения которого используем length:

console.log(Object.keys(options).length); // 4

Создание метода в объекте:

const options = {

    name: 'test',   //свойства

    width: 1024,

    height: 1024,

    colors: {

        border: 'black',

        bg: 'red',

    },

    makeTest: function() {      // метод

        console.log('test message');

    }

}

Обращение к методу

options.makeTest();

Дескрипторы, геттеры и сеттеры свойств

<https://learn.javascript.ru/descriptors-getters-setters>



**20.3 Деструктуризация объекта**

<https://learn.javascript.ru/destructuring#destrukturizatsiya-obekta>

Для того, чтобы при обращении к свойству не использовать конструкцию

console.log(options['colors']['border']);

используется деструктуризация

const {border, bg} = options.colors;

^ ^ ^

нужные свойства свойство верхнего уровня

Теперь они доступны в виде:

console.log(border); //black

console.log(bg); //red

**21. Массивы и псевдомассивы**

const arr = [2, 4, 6, 8, 10];

Длина массива

arr.length

length всегда равна «индекс последнего элемента» + 1.

Удаление последнего элемента

arr.pop();

Добавление элемента в конец

arr.push(12);

Перебор массива

В цикле

for (let i = 0; i < arr.length; i++) {

    console.log(arr[i]);

}

Или **for… of**

for (let value of arr) {

    console.log(value);

}

(в отличие от for… in вызывает на каждом шаге значение, а не ключ. For… in вызывает на каждом шаге ключ и проходит по свойствам в произвольном порядке, поэтому его не следует использовать для Array).

Или с помощью

**Метода forEach(),**

в который передается callback-функция с тремя параметрами: значение элемента, индекс, и сам массив callbackfn(**value: number**, index: number, array: number[]): void

const arr = [2, 4, 6, 8, 10];

arr.forEach(function(val, i, arr) {

    console.log(`${i}: ${val} входит в массив ${arr}`);

})

Вывод:

0: 2 входит в массив 2,4,6,8,10

1: 4 входит в массив 2,4,6,8,10

2: 6 входит в массив 2,4,6,8,10

3: 8 входит в массив 2,4,6,8,10

4: 10 входит в массив 2,4,6,8,10



**Сортировка arr.sort()**

Без ф-ции сравнения

const arr = [2, 14, 26, 78, 10];

arr.sort();

console.log(arr);

Вывод:

[ 10, 14, 2, 26, 78 ]

- сортировка элементов как строки

С ф-цией сравнения (callback)

const arr = [2, 14, 26, 78, 10];

function compareNum(a, b) {

    return a - b;

}

arr.sort(compareNum);

console.log(arr);

Вывод:

[ 2, 10, 14, 26, 78 ]

- сортировка как числа. См. <http://algolist.ru/sort/quick_sort.php>

**Псевдомассивы** –

массивоподобные структуры, но без стандартных свойств и методов.

**22. Передача по ссылке или по значению, Spread оператор (ES6-ES9)**

В случае создания копии

a = b

для примитивных типов данных происходит передача по значению, а для сложных типов данных (массивы, объекты) – по ссылке.

<https://medium.com/@stasonmars/копирование-объектов-в-javascript-d25c261a7aff>

Для создания полноценной копии объекта/массива (по значению) – 4 способа.

**22.1 Нужно произвести поэлементное присваивание:**

function copyObj(srcObj) {

    let dstObj = {}

    let key;

    for (key in srcObj) {

        dstObj[key] = srcObj[key];

    }

    return dstObj;

}

newMyNumbers = copyObj(myNumbers);

Если элементом будет сложный тип, то этот элемент присвоится по ссылке. Это называется поверхностная копия.

Присваивание по значению на всех уровнях вложенности – глубокое копирование (позже).

**22.2 С помощью метода Object.assign:**

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/assign>

newMyObj = Object.Assign({}, myObj);

- объединение объекта с пустым объектом.

**22.3 Для массивов – с помощью метода slice:**

newMyArr = myArr.slice();

**22.4 C помощью Spread оператора … (для массивов начиная с ES6, для объектов – с ES9)**

Оператор развертывания …

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Spread_syntax>

Копирование массива

const newMyArr = [...myArr];

Копирование объекта

const newMyObj = [...myObj];

**23. Основы ООП, прототипно-ориентированное наследование**

Имеем объект

const soldier = {

    health: 400,

    armor: 100

}

И объект

const john = {

    health: 100

}

Прототипное наследование

Устаревший синтаксис

john.\_\_proto\_\_ = soldier;

Теперь можно обратиься к свойству john.armor

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Object_initializer>

Вместо св-ва \_\_proto\_\_ сейчас нужно использовать методы

Object.create()

Object.getPrototypeOf()

Object.setPrototypeOf()

**24. Практика , ч4. Используем объекты**

1. при передаче callback-функции в качестве аргумента скобки не пишутся, т.к. это не вызов функции [↑](#footnote-ref-1)