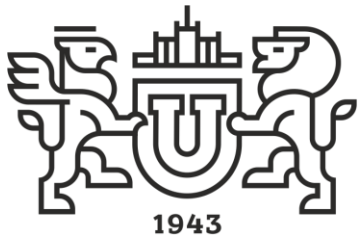


Южно-Уральский государственный университет

Национальный исследовательский университет

КРОК

Клиентская часть клиент-серверного взаимодействия. Основы JS

КРОК

Челябинск, ул. Карла Маркса, д. 38

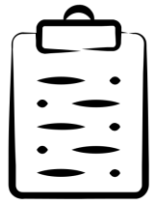
Смирнов Анатолий
Технический менеджер

Антонов Сергей
Тимлид группы разработчиков, старший инженер-разработчик

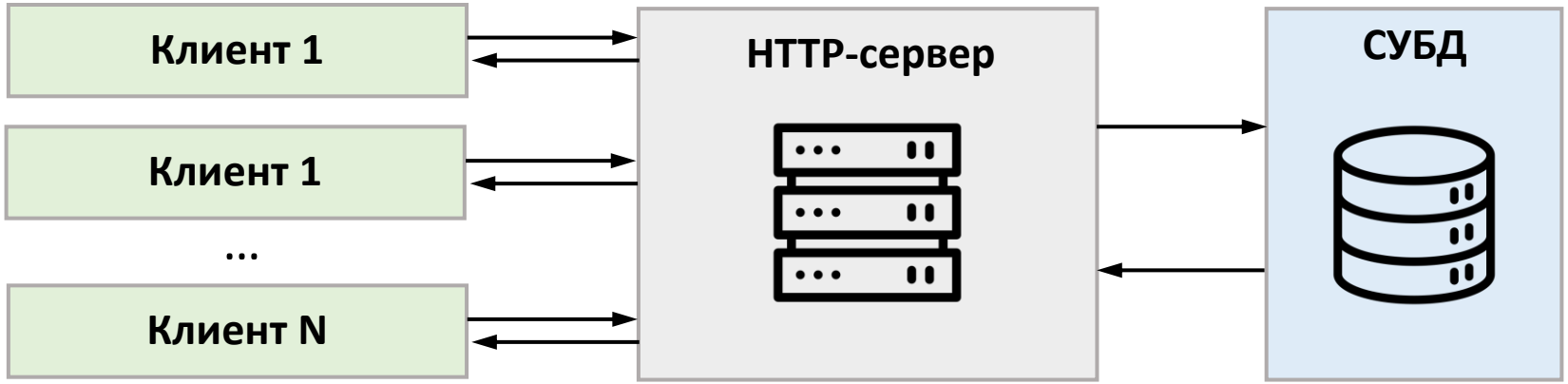
Кузнецов Сергей
Инженер-разработчик

Задачи клиентской части

КРОК



- Взаимодействие с пользователем.
- Передача пользовательского запроса серверу.
- Получение запроса от серверной части (программы-сервера) представление его в удобном для пользователя виде.

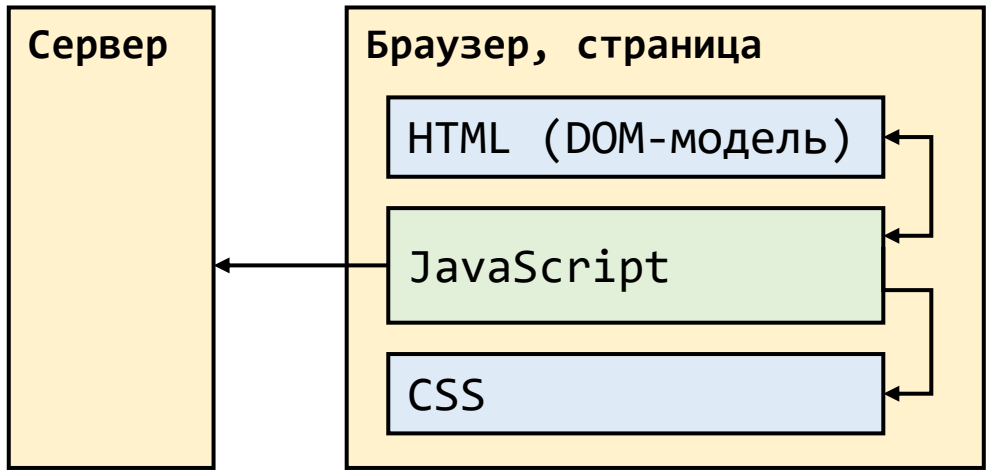


Теперь мы тут

JavaScript (JS)

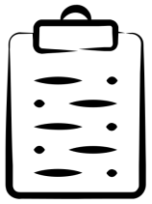
КРОК

- JavaScript (часто просто JS) — это легковесный, интерпретируемый, объектно-ориентированный язык с функциями первого класса. Наиболее широкое применение находит как язык сценариев веб-страниц, но также используется и в других программных продуктах, например, node.js
- Мультипарадигменный язык с динамической типизацией, который поддерживает объектно-ориентированный, императивный и декларативный (например, функциональное программирование) стили программирования

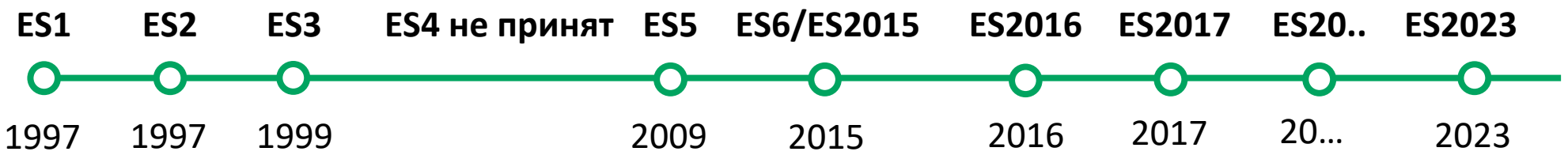


Стандарты JS

КРОК



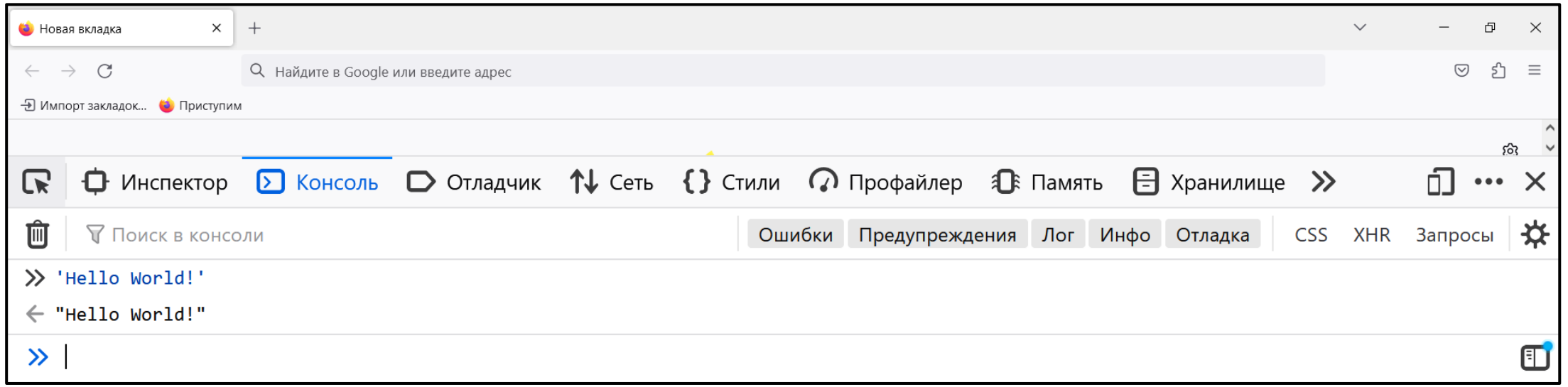
- JavaScript создавался как скриптовый язык для Netscape. После чего он был отправлен в ECMA International для стандартизации (ECMA — это ассоциация, деятельность которой посвящена стандартизации информационных и коммуникационных технологий). Это привело к появлению нового языкового стандарта, известного как ECMAScript.
- Последующие версии JavaScript уже были основаны на стандарте ECMAScript. Проще говоря, ECMAScript — стандарт, а JavaScript — самая популярная реализация этого стандарта.
- ES — это просто сокращение для ECMAScript. Каждое издание ECMAScript получает аббревиатуру ES с последующим его номером.
- После 2015 стандарты выходят каждый год.



Использование JS

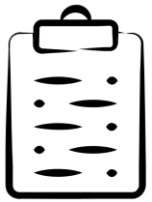
КРОК

- Для выполнения кода JS достаточно любого современного браузера.
- F12 и вкладка «Консоль» («Console») – вызов консоли браузера для просмотра вывода сообщений и выполнения кода на JS.



Использование JS в HTML

КРОК

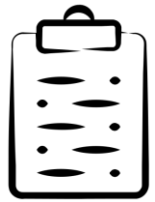


- Код JS содержится между тэгами <script> </script> либо в теге <script> </script> указан путь до файла в атрибуте src
- Рекомендуется подключать файлы JS в секции <head>

```
<!DOCTYPE html>
<html lang="">
    <head>
        <meta charset="utf-8">
        <script src="/path/script.js"></script>
        <script>
            // Код
        </script>
    </head>
    <body>
        <div></div>
        <script src="/path/script2.js"></script>
        <script>
            // Код
        </script>
    </body>
</html>
```

JS. Особенности

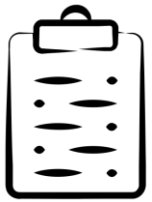
КРОК



- Синтаксис СИ-подобный
- Точка с запятой в конце не обязательная
- Оператор присваивания =
- Нет отдельного типа для целых чисел
- Принятый стиль именования переменных: camelCase

JS. Объявление переменных

КРОК



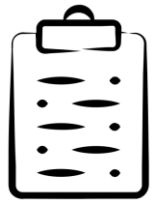
Объявить переменные можно тремя способами: var, let, const.

- var считается устаревшим, до 2015 года (ES5) был единственным способом объявления переменных
- let для любых переменных, когда требуется их изменять
- const не может быть изменено новым присваиванием, а также не может быть переопределено (но можно менять свойства объектов и элементы массива)

```
> var x = 1; let y = 5; const t = 1
< undefined
> x
< 1
> x = 2
< 2
> t = 10
✕ ▶Uncaught TypeError: Assignment to constant variable.
    at <anonymous>:1:3
```

JS. Комментарии и вывод данных

КРОК



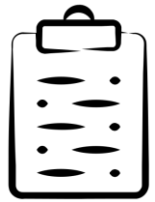
- // для однострочных комментариев
- /* */ для многострочных комментариев
- Вывод данных в консоль из любого участка кода осуществляется при помощи объекта console и методов log() error() и warn().
- Самое распространенное применение console.log().

```
> let x = 5; /* Комментарий */ console.log(x) // Комментарий
  5
```

```
> console.warn('Предупреждение!')
▶ Предупреждение!
undefined
> console.error('Ошибка!')
▶ Ошибка!
undefined
```

JS. Операторы сравнения

КРОК

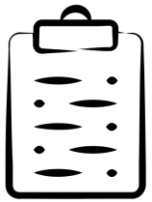


- Больше/меньше: a > b, a < b
- Больше/меньше или равно: a >= b, a <= b
- Равно: a == b
- Строго равно a === b (без преобразования типов)

```
> 2 > 3
< false
> 1 == '1'
< true
> 1 === '1'
< false
```

JS. Типы данных

КРОК

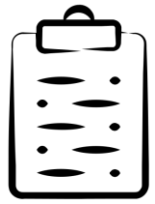


- Js – язык с динамической типизацией
- Есть восемь основных типов данных в JavaScript (7 примитивных и один сложный)

Тип	Назначение
number	Для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном ±(253-1).
bigint	Для целых чисел произвольной длины.
boolean	Для true/false.
null	Для неизвестных значений – отдельный тип, имеющий одно значение null
undefined	Для неприсвоенных значений – отдельный тип, имеющий одно значение
string	Для строк. Строка может содержать ноль или больше символов, нет отдельного символьного типа.
symbol	Для уникальных идентификаторов
object	Для более сложных структур данных

JS. Типы данных. number и bigint

КРОК



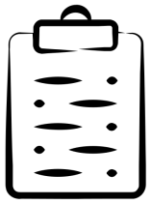
- Числовой тип данных (number) представляет как целочисленные значения, так и числа с плавающей точкой.
- В JavaScript тип number не может безопасно работать с числами, большими, чем (253-1) (т. е. 9007199254740991) или меньшими, чем -(253-1) для отрицательных чисел.
- Деление на 0 безопасно

```
> n = 5
< 5
> m = 5.123
< 5.123
> x = 1 / 0
< Infinity
> Infinity * Infinity
< Infinity
> Infinity / Infinity
< NaN
```

```
> 2 ** 53
< 9007199254740992
> 2 ** 53 + 1
< 9007199254740992
> 2 ** 53 === 2 ** 53 + 1
< true
```

JS. Типы данных. boolean, null, undefined

КРОК



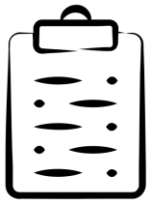
- Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь)
- Специальное значение null не относится ни к одному из типов, описанных выше. Оно формирует отдельный тип, который содержит только значение null
- Специальное значение undefined также стоит особняком.. Оно означает, что «значение не было присвоено». Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined

```
> x = true
< true
> x = false
< false
> x = 1 === 3
< false
> x = 2 + 1 === 3
< true
```

```
> let x
< undefined
> x
< undefined
> x = null
< null
> x
< null
```

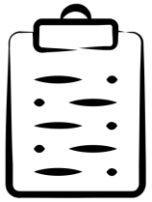
JS. Типы данных. Строки

КРОК



Строка (string) в JavaScript должна быть заключена в кавычки:

- Двойные кавычки: "Привет".
- Одинарные кавычки: 'Привет'.
- Обратные кавычки: `Привет`.



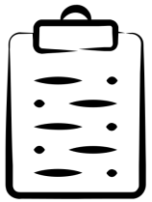
- Двойные или одинарные кавычки являются «простыми», между ними нет разницы в JavaScript.
- Обратные же кавычки имеют расширенную функциональность. Они позволяют нам встраивать выражения в строку, заключая их в ${…}

```
> let name = "Иван"
<· undefined
>
  let x = `Привет, ${name}!`
<· undefined
> x
<· 'Привет, Иван!'
```

```
> 'Hello, ' + "World" + `!`
<· 'Hello, World!'
```

JS. Типы данных. Строки. Методы и свойства

КРОК



Примеры работы со строками:

- `length` – длина строки.
- `substr(start [, length])` - возвращает часть строки от start длины length.
- `substring(start [, end])` возвращает часть строки между start и end (не включая) end.
- метод `includes(substr, pos)` возвращает true, если в строке str есть подстрока substr, либо false, если нет.
- Методы `toLowerCase()` и `toUpperCase()` меняют регистр символов.

```
> x = 'Hello'
< 'Hello'
> x.length
< 5
> 'World'.length
```

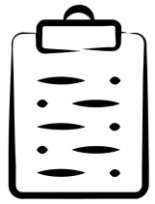
```
> x = 'Hello world'
< 'Hello world'
> x.substr(3, 8)
< 'lo world'
> x.substring(3, 8)
< 'lo wo'
```

```
> 'Hello world'.includes('wor')
< true
> 'Hello world'.includes('wor1')
< false
```

```
> x.toLowerCase()
< 'hello world'
> x.toUpperCase()
< 'HELLO WORLD'
```

JS. Типы данных. Строки. Доступ к символам

КРОК



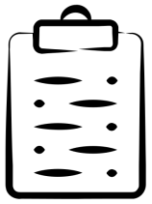
- Получить символ, который занимает позицию pos, можно с помощью квадратных скобок: [pos]
- Содержимое строки в JavaScript нельзя изменить. Нельзя взять символ посередине и заменить его. Как только строка создана — она такая навсегда.

```
> x = 'Hello'
< 'Hello'
> x[0]
< 'H'
> x[1]
< 'e'
> x[x.length - 1]
< 'o'
```

```
> let str = 'Hi'
< undefined
> str[0] = 'h'
< 'h'
> str
< 'Hi'
```

JS. Типы данных. Массивы

КРОК



Существует два варианта синтаксиса для создания пустого массива

```
let arr = new Array();
let arr = [];
```

Практически всегда используется второй вариант синтаксиса. В скобках мы можем указать начальные значения элементов:

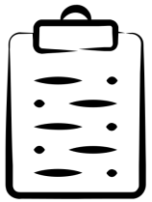
```
let fruits = ["Яблоко", "Апельсин", "Слива"];
```

- Элементы массива нумеруются, начиная с нуля.
- Для получения элемента нужно указать его номер в квадратных скобках.

```
> let fruits = ["Яблоко", "Апельсин", "Слива"];
  fruits[0]
'Яблоко'
> fruits[1] = 5
5
> fruits
▶ (3) ['Яблоко', 5, 'Слива']
```

JS. Типы данных. Массивы. Методы и свойства

КРОК



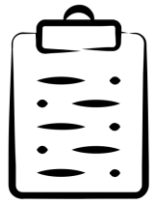
Пример работы с массивами:

- length – длина массива
- push() добавляет элемент в конец.
- pop() удаляет последний элемент.
- join(<разделитель>) – возвращает строку, которая соединяет все элементы массива через разделитель

```
> let fruits = ["Яблоко", "Апельсин", "Слива"];
< undefined
> fruits.length
< 3
> fruits.push("Дыня");
< 4
> fruits.join(" ")
< 'Яблоко Апельсин Слива Дыня'
```

JS. Типы данных. Объекты

КРОК



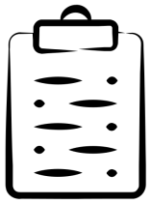
- Объекты же используются для хранения коллекций различных значений и более сложных сущностей. В JavaScript объекты используются очень часто, это одна из основ языка.
- Объект может быть создан с помощью фигурных скобок {…} с необязательным списком свойств. Свойство – это пара «ключ: значение».

```
let user = {      // объект
    name: "Иван",  // под ключом "name" хранится значение "John"
    age: 30        // под ключом "age" хранится значение 30
};
```

```
> user.name
< 'Иван'
> user.flights = 20
< 20
> user
< ▶ {name: 'Иван', age: 30, flights: 20}
```

JS. Типы данных. Объекты. Доступ к свойству

КРОК



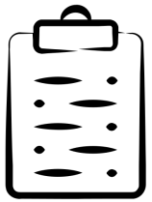
Получить доступ к свойству можно либо через точку либо через квадратные скобки и обращению к свойству в виде строки

```
obj.key
obj['key']
```

```
> let user = {
      name: "Иван",
      age: 30
  }
<· undefined
> user.name
<· 'Иван'
> user['name']
<· 'Иван'
> let x = 'name'
<· undefined
> user[x]
<· 'Иван'
```

JS. Типы данных. Объекты. Методы

КРОК



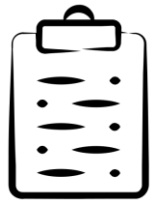
- `Object.keys(<объект>)` – получить массив ключей.
- `Object.values(<объект>)` – получить массив значений.
- `Object.entries(<объект>)` – получить массив в формате [key, value].

```
> let x = {
      key1: 'value 1',
      '5': 4,
      key3: true
  }
< undefined
> Object.keys(x)
< ▶ (3) ['5', 'key1', 'key3']
> Object.values(x)
< ▶ (3) [4, 'value 1', true]
```

```
> Object.entries(x)
< ▼ (3) [Array(2), Array(2), Array(2)]
    ▶ 0: (2) ['5', 4]
    ▶ 1: (2) ['key1', 'value 1']
    ▶ 2: (2) ['key3', true]
      length: 3
    ▶ [[Prototype]]: Array(0)
> Object.entries(x)[1]
< ▶ (2) ['key1', 'value 1']
```

JS. Типы данных. Объекты и JSON

КРОК

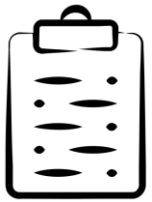


- `JSON.stringify(<объект>)` возвращает JSON-строку
- `JSON.parse(<строка формата JSON>)` возвращает объект

```
> let x = {
      key1: 'value 1',
      '5': 4,
      key3: true
  }
< undefined
> let str = JSON.stringify(x)
< undefined
> str
< '{"5":4,"key1":"value 1","key3":true}'
> let y = JSON.parse(str)
< undefined
> y
< ▼ {5: 4, key1: 'value 1', key3: true}
      5: 4
      key1: "value 1"
      key3: true
```

JS. Ссылки

КРОК



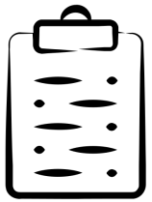
- Оператор присваивания для объектов (в том числе и массивов), присваивает ссылку на исходный объект
- Для копирования необходимы дополнительные действия

```
> a = [1 , 2]
◂ ▸ (2) [1, 2]
> b = a
◂ ▸ (2) [1, 2]
> b.push(1111)
◂ 3
> a
◂ ▸ (3) [1, 2, 1111]
```

```
> a = [1 , 2]
◂ ▸ (2) [1, 2]
> b = [...a] // Копия
◂ ▸ (2) [1, 2]
> b.push(1111)
◂ 3
> a
◂ ▸ (2) [1, 2]
```

JS. Условия

КРОК

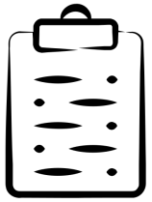


Инструкция if(...) вычисляет условие в скобках и, если результат true, то выполняет блок кода

```
> let x = 5

  if (x < 10) {
      console.log('Меньше');
  } else if (year > 2015) {
      console.log('Больше');
  } else {
      console.log('Равно');
  }

  Меньше
```



Тернарный оператор «?»

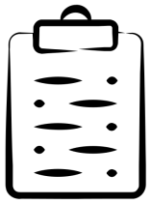
```
let result = условие ? значение1 : значение2;
```

```
> y = (x < 10) ? 'Меньше' : 'Больше'
  console.log(y)

  Меньше
```

JS. Базовые циклы

КРОК



- while – Проверяет условие перед каждой итерацией.
- do..while – Проверяет условие после каждой итерации.
- for (;;) – Проверяет условие перед каждой итерацией, есть возможность задать дополнительные настройки.

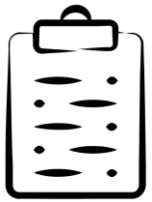
```
> let i = 0;
  while (i < 3) {
    console.log(i);
    i++;
  }
  0
  1
  2
```

```
> let i = 0;
  do {
    console.log(i);
    i++;
  } while (i < 3);
  0
  1
  2
```

```
> for (let i = 0; i < 3; i++) {
    console.log(i);
  }
  0
  1
  2
```

JS. Циклы for ... of и for ... in

КРОК



- Основная разница между этими операторами заключается в том, что for...in проходит по всем перечисляемым свойствам объекта тогда как for...of проходит только по значениям элементов масива.

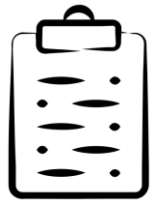
```
> const x = [222, 333, 55]
< undefined
> for (let i = 0; i < x.length; ++i) {
      console.log(x[i])
  }
  222
  333
  55
```

```
> for (let index in x) {
      console.log(index, x[index])
  }
  0 222
  1 333
  2 55
```

```
> for (let value of x) {
      console.log(value)
  }
  222
  333
  55
```

JS. Циклы for ... of и for ... in

КРОК



Цикл for .. of для массивов является предпочтительней, т.к. он быстрее и перебирает только элементы массива.

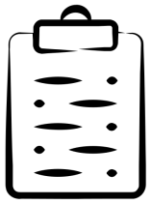
```
> const x = [222, 333, 55];
  x.some = 'свойство!';
'свойство!'
> for (let i = 0; i < x.length; ++i) {
      console.log(x[i])
  }
222
333
55
```

```
> for (let index in x) {
      console.log(index, x[index])
  }
0 222
1 333
2 55
some свойство!
```

```
> for (let value of x) {
      console.log(value)
  }
222
333
55
```

JS. Циклы по объекту

КРОК



for .. of не работает для обычных объектов, необходимо использовать либо for .. in, либо методы values(), keys(), entries().

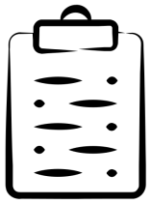
```
> x = {
      key1: 111,
      key2: 222
  }
◂ ▶ {key1: 111, key2: 222}
> for (let value of x) {
      console.log(value)
  }
✕ ▶ Uncaught TypeError: x is not iterable
      at <anonymous>:1:19
```

```
> for (let value of Object.values(x)) {
      console.log(value)
  }
  111
  222
```

```
> for (let value in x) {
      console.log(value, x[value])
  }
  key1 111
  key2 222
```

JS. Функции

КРОК



Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

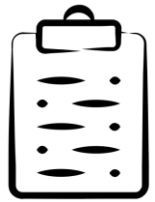
```
function имя(параметры) {
    ...тело...
}
```

```
> function logHello() {
      console.log('Hello');
  }
< undefined
> logHello()
  Hello
```

```
> function logHello(str) {
      console.log('Hello, ' + str);
  }
< undefined
> logHello('Иван')
  Hello, Иван
```

JS. Функции. Возврат значения

КРОК



Функция может вернуть результат, который будет передан в вызвавший её код

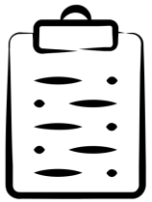
```
function имя(параметры) {
    ...тело...
    return ... результат ...
}
```

```
> function logHello() {
      console.log('Hello');
  }
< undefined
> logHello()
  Hello
```

```
> function logHello(str) {
      console.log('Hello, ' + str);
  }
< undefined
> logHello('Иван')
  Hello, Иван
```

JS. Функции обратного вызова

КРОК



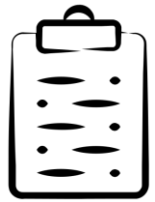
- Функция обратного вызова (функция высшего порядка, функция второго порядка, коллбэк-функция) - это функция, переданная в другую функцию в качестве аргумента

```
> function a() {
      return 123
  }
< undefined
> function b(arg) {
      return arg
  }
< undefined
> b(a)
< ƒ a() {
      return 123
  }
```

```
> function a() {
      return 123
  }
< undefined
> function b(arg) {
      return arg()
  }
< undefined
> b(a)
< 123
```

JS. Функции обратного вызова

КРОК



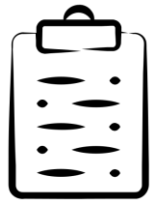
Вложенность может быть достаточно большая

```
> function b(arg) {
      return arg()
  }
< undefined
> b(function () {return 5})
< 5
```

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

JS. Функциональный стиль для массивов

КРОК

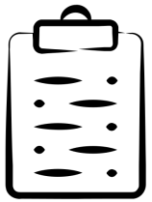


Популярные методы для работы с массивами:

- .forEach()
- .map()
- .filter()
- .sort()
- .find()

JS. Функциональный стиль для массивов. forEach

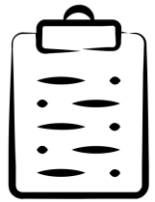
КРОК



- Метод forEach() выполняет указанную функцию один раз для каждого элемента в массиве
- Нет возможности сделать break

```
arr.forEach(function(currentValue, index, array) {
    // currentValue - текущее значение
    // index - индекс
    // array ссылка на сам массив
});
```

```
> arr = [55, 66, 'Hello'];
◂ ▶ (3) [55, 66, 'Hello']
> arr.forEach(function(currentValue, index, array) {
    console.log(`Индекс: ${index}, значение: ${currentValue}`)
  });
  Индекс: 0, значение: 55
  Индекс: 1, значение: 66
  Индекс: 2, значение: Hello
```



- Если необходимо прервать цикл (break), то выбирать for .. of
- forEach быстрее, чем for .. of, но медленнее, чем `for(let i = 0; i < …)`

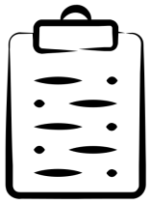
Аналогичный результат:

```
> arr = [55, 66, 'Hello'];
< ▶ (3) [55, 66, 'Hello']
> arr.forEach(function(value) {
      console.log(value)
  });
  55
  66
  Hello
```

```
> for (let value of arr) {
      console.log(value)
  }
  55
  66
  Hello
```

JS. Функциональный стиль для массивов. map

КРОК



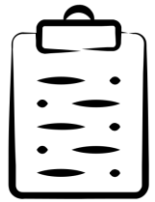
- Метод map() создаёт новый массив с результатом вызова указанной функции для каждого элемента массива

```
const newArray = arr.map(function (currentValue, index, array) {
    // Возвращает элемент для new_array
    // например, return currentValue * 3
})
```

```
> arr = [5, 6, 'Hello'];
◂ ▶ (3) [5, 6, 'Hello']
> newArr = arr.map(function(value) {
      return value * 2
  });
◂ ▶ (3) [10, 12, NaN]
> arr
◂ ▶ (3) [5, 6, 'Hello']
```

JS. Функциональный стиль для массивов. filter

КРОК



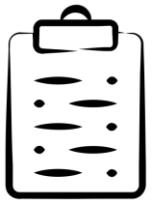
- Метод filter() создаёт новый массив со всеми элементами, прошедшими проверку, задаваемую в передаваемой функции.

```
const newArray = arr.filter(function (currentValue, index, array) {
    // если результат return === true, то элемент попадат в
    // результирующий массив
})
```

```
▶ (5) [2, 3, 10, 5, 7]
> arr.filter(function(value) {
    return value > 3
  })
▶ (3) [10, 5, 7]
```

JS. Функциональный стиль для массивов. find

КРОК



- Метод find() возвращает значение первого найденного в массиве элемента
- filter() возвращает массив, а find возвращает значение массива

```
const newArray = arr.find(function (currentValue, index, array) {
    // если результат return === true, то возвращается
    // текущее значение, выполнение прекращается
})
```

```
> arr = [2, 3, 10, 5, 7]
< ▶ (5) [2, 3, 10, 5, 7]
> arr.find(function(value) {
    return value > 3
  })
< 10
```

Пример 1

КРОК

Напишите функцию firstUp(str), возвращающую строку str с заглавным первым символом.

```
function firstUp(str) {
    return str[0].toUpperCase() + str.substr(1, str.length -1)
}
```

```
> function firstUp(str) {
      return str[0].toUpperCase() + str.substr(1, str.length -1)
  }
< undefined
> firstUp('hello')
< 'Hello'
```

Пример 2

КРОК

Напишите функцию userJoin(array, smb), возвращающую строку, которая соединяет элементы массива array строкой smb (аналог встроенной join())

```
function userJoin(array, smb) {
    let str = '';
    const len = array.length;

    for (let i = 0; i < len; i++) {
        str += array[i];
        if (i !== len - 1) {
            str += smb;
        }
    }

    return str;
}
```

```
> userJoin([1, 2], '___')
< '1___2'
> userJoin([1, 2, 3, '123'], '___')
< '1___2___3___123'
> userJoin([1, 2, 3, '123'], ' ')
< '1 2 3 123'
```

Пример 3

КРОК

Получить массив из исходного, отфильтровать только те значения, которые строго больше 3. Все элементы полученного массива умножить на 5.

```
const result = a.filter(function(value) {
    return value > 3
}).map(function(value) {
    return value * 5
});
```

```
> const a = [1, 5, 7, 8, 4, 2, 3, 9];
< undefined
> const result = a.filter(function(value) {
      return value > 3
  }).map(function(value) {
      return value * 5
  });
< undefined
> result
< ▶ (5) [25, 35, 40, 20, 45]
```

Южно-Уральский государственный университет
Национальный исследовательский университет

1943

КРОК

Спасибо за внимание!

КРОК
Челябинск, ул. Карла Маркса, д. 38