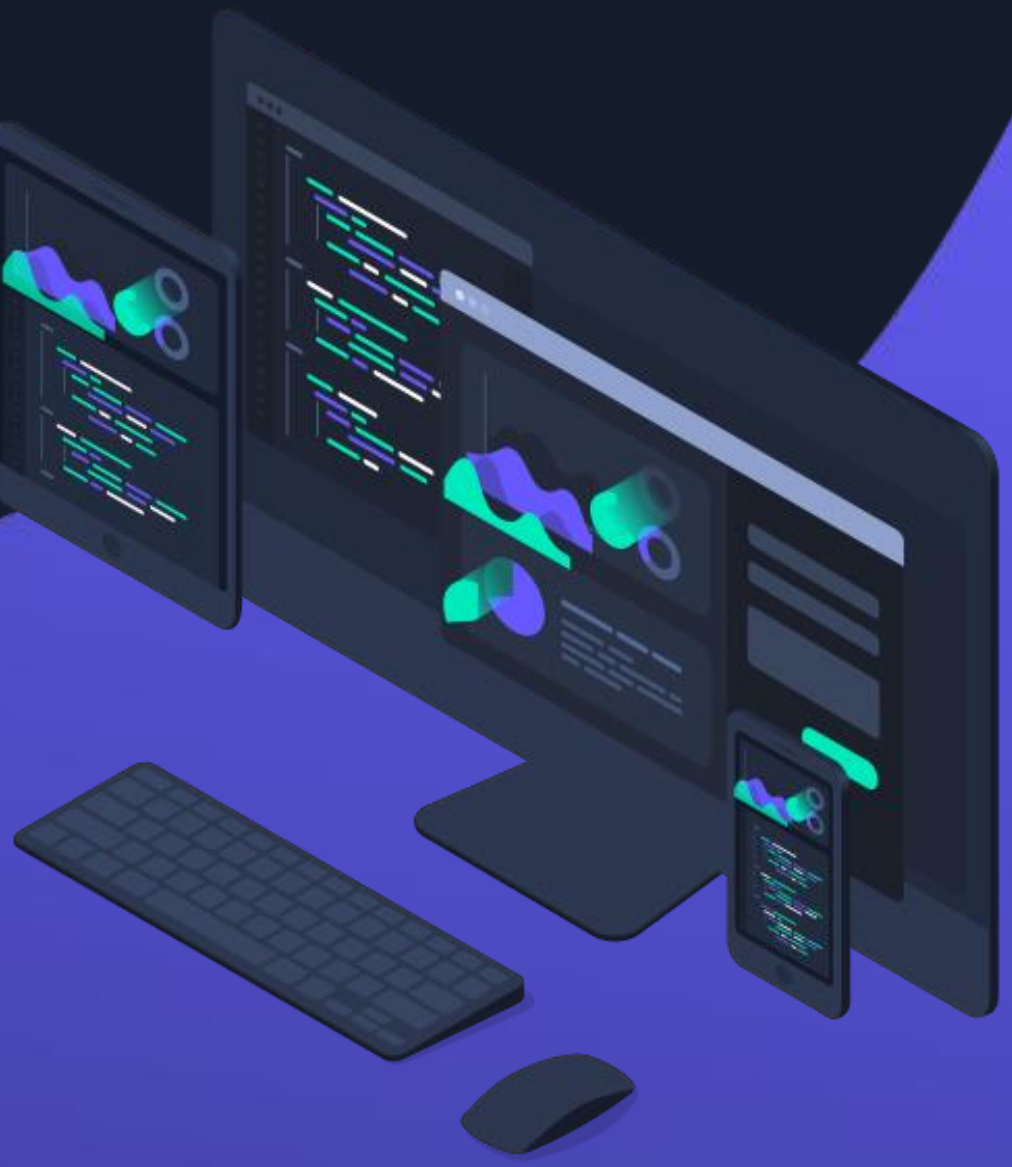
ITEA

ONLINE



ITEA ONLINE

На сегодняшнем занятии:

1. indexOf
2. lastIndexOf
3. find
4. findIndex
5. forEach
6. filter
7. every
8. some
9. map
10. reduce
11. reduceRight
12. isArray
13. isNaN, Number.isNaN

I T E A
ONLINE

indexOf

`arr.indexOf(item, from)` ищет `item`, начиная с индекса `from`, и возвращает индекс, на котором был найден искомый элемент, в противном случае -1. Если `from` не указан – по умолчанию стоит 0.

```
const arr = ["test", -1, 0, "asd", 1, 2, "4"];
let i = arr.indexOf(0); // 2
```

```
const arr = ["test", false, -1, 0, "asd", false, 1, 2, "4"];
let i = arr.indexOf(false, 2); // 5
```

Обратите внимание, что метод использует строгое сравнение ===. Таким образом, если мы ищем false, он находит именно false, а не ноль.

ITEA ONLINE

lastIndexOf

`arr.lastIndexOf(item, from)` ищет `item`, начиная с индекса `from`, и возвращает индекс, на котором был найден искомый элемент, в противном случае -1. Разница с `arr.indexOf()` в том, что в данном случае метод начинает поиск с конца массива.

```
const arr = ["test", -1, 1, 0, "asd", 1, 2, "4"];
let i = arr.lastIndexOf(1); // 5
```

Этот метод так же использует строгое сравнение ===.

I T E A
ONLINE

find

Представьте, что у нас есть массив объектов. Как нам найти объект с определённым условием? Здесь пригодится метод `find()`. Функция вызывается по очереди для каждого элемента массива

```
let result = arr.find(function(item, index, array) {
  // если true - возвращается текущий элемент и перебор прерывается
  // если все итерации оказались ложными, возвращается undefined
});
```

```
const a = [1, 2, "4", 11, 3, 4, "3"];
let i = a.find((item) => {
    return item > 5;   // i: 11
})
```

```
const a = ["test", -1, 0, "asd", 1, 2, "4"];
let i = a.find((item) => {
    return !isNaN(item); // i: -1
})
```

ITEA ONLINE

findIndex

Метод `arr.findIndex` – по сути, то же самое, но возвращает индекс, на котором был найден элемент, а не сам элемент, и -1, если ничего не найдено.

```
const a = [1, 2, "4", 3, 4, "3", 10];
let i = a.findIndex((item) => {
    return item === "3";   // i: 5
})
```

```
const a = ["test", "-1a", "10", "asd", 1, 2, "4"];
let i = a.findIndex((item) => {
    return !isNaN(item); // i: 2
})
```

ITEA ONLINE

forEach

Метод `forEach()` выполняет переданную функцию один раз для каждого элемента, находящегося в массиве в порядке возрастания. Она не будет вызвана для удалённых или пропущенных элементов массива. Однако, она будет вызвана для элементов, которые присутствуют в массиве и имеют значение `undefined`.

```
arr.forEach(function(currentValue, index, array) {
    //your iterator
}[, thisArg]);
```

```
const arr = [1, 3, 7]

arr.forEach((item) => {
  console.log("item: " + item); // "item: 1", "item: 3", "item: 7"
});
console.log(arr); // [1, 3, 7]
```

ITEA ONLINE

filter

Метод `find` ищет один (первый попавшийся) элемент, на котором функция-колбэк вернёт true. На тот случай, если найденных элементов может быть много, предусмотрен метод `arr.filter(fn)`. Синтаксис этого метода схож с `find`, но `filter` возвращает массив из всех подходящих элементов:

```
let results = arr.filter(function(item, index, array) {
  // если true - элемент добавляется к результату, и перебор продолжается
  // возвращается пустой массив в случае, если ничего не найдено
});
```

```
const arr = ["test", "-1a", "10", "asd", 1, 2, "4", 6];
let i = arr.filter((item) => {
    return item > 3; // ["10", "4", 6]
})
```

ITEA ONLINE

every

Метод `every()` проверяет, удовлетворяют ли все элементы массива условию, заданному в передаваемой функции. Метод возвращает `true` при любом условии для пустого массива.

```
const arr = [1, 2, 4, 6, 5];
const even = arr.every((item) => { // false
   return item % 2 === 0;
});
```

ITEA ONLINE

some

Метод `some()` проверяет, удовлетворяет ли какой-либо элемент массива условию, заданному в передаваемой функции. Метод возвращает `false` при любом условии для пустого массива.

```
const arr = [1, 3, 3, 2, 5];
const even = arr.some((item) => { // true
        return item % 2 === 0;
});
```

ITEA ONLINE

map

Метод `arr.map()` является одним из наиболее полезных и часто используемых.

Он вызывает функцию для каждого элемента массива и возвращает массив результатов выполнения этой функции.

```
const arr = [1, 2, 4, 6, 5];
let result = arr.map((item) => {
    return item += " text";
});

console.log(arr);    // [1, 2, 4, 6, 5]
console.log(result); // ["1 text", "2 text", "4 text", "6 text", "5 text"]
```

```
let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item => item.length);
console.log(lengths); // [5, 7, 6]
```

I T E A
ONLINE

reduce

Методы `arr.reduce` и `arr.reduceRight` похожи на предыдущие методы, но они немного сложнее. Они используются для вычисления какого-нибудь единого значения на основе всего массива. Функция применяется по очереди ко всем элементам массива и «переносит» свой результат на следующий вызов.

```
let value = arr.reduce(function(previousValue, item, index, array) {
  // ...
}, [initial]);
```

previousValue – результат предыдущего вызова этой функции, равен initial при первом вызове (если передан initial),

item – очередной элемент массива,

index – его индекс,

array – сам массив.

I T E A
ONLINE

reduce

```
let arr = [1, 2, 3, 4, 5];
let result = arr.reduce((sum, current) => sum + current, 0);
alert(result); // 15
```

Здесь мы использовали наиболее распространённый вариант `reduce`, который использует только 2 аргумента.

При первом запуске `sum` равен `initial` (последний аргумент `reduce`), то есть 0, а current – первый элемент массива, равный 1. Таким образом, результат функции равен 1.

При втором запуске `sum = 1`, и к нему мы добавляем второй элемент массива (2).

При третьем запуске `sum = 3`, к которому мы добавляем следующий элемент, и так далее…

ITEA ONLINE

reduceRight

Метод `arr.reduceRight` работает аналогично, но проходит по массиву справа налево.

```
let flattened = [[0, 1], [2, 3], [4, 5]].reduceRight(function(a, b) {
  return a.concat(b);
}, []);
// flattened равен [4, 5, 2, 3, 0, 1]
```

При отсутствии `initial` в качестве первого значения берётся последний элемент массива, а перебор стартует со второго с конца. В `reduce` наоборот – берётся первый элемент, а перебор стартует со второго элемента.

I T E A
ONLINE

isArray()

Массивы не образуют отдельный тип языка. Они основаны на объектах. Поэтому `typeof` не может отличить простой объект от массива:

```
alert(typeof {}); // object
alert(typeof []); // тоже object
```

Массивы используются настолько часто, что для этого придумали специальный метод: `Array.isArray(value)`. Он возвращает `true`, если `value` массив, и `false`, если нет.

```
alert(Array.isArray({})); // false
alert(Array.isArray([])); // true
```

I T E A
ONLINE

isNaN & Number.isNaN

Метод `Number.isNaN()` определяет, является ли переданное значение `NaN`. В отличие от глобальной функции `isNaN()`, `Number.isNaN()` не имеет проблемы принудительного преобразования параметра в число. Это значит, что в него безопасно передавать значения, которые обычно превращаются в `NaN`, но на самом деле `NaN` не являются. Также это значит, что метод возвращает true только для числовых значений, имеющих значение `NaN`.

`NaN == NaN` и `NaN === NaN` в качестве значения вернут false

```
Number.isNaN(NaN);  // true
Number.isNaN(0 / 0) // true

Number.isNaN('NaN');      // false
Number.isNaN(undefined);  // false

isNaN(undefined);  // true
```

```
isNaN("a");         // true
(попытается преобразовать строку «a» в число,
получится NaN и потом вернет true)

Number.isNaN("a"); // false
```

I T E A
ONLINE

Формат JSON

Обозначение объектов JavaScript (JSON - JavaScript Object Notation) - стандартный текстовый формат для представления структурированных данных на основе синтаксиса объекта JavaScript. Он обычно используется для передачи данных в веб-приложениях (например, отправка некоторых данных с сервера клиенту, таким образом чтобы это могло отображаться на веб-странице или наоборот).

JSON существует как строка, что необходимо при передаче данных по сети. Он должен быть преобразован в собственный объект JavaScript, если вы хотите получить доступ к данным. JavaScript предоставляет глобальный объект `JSON`, который имеет методы для преобразования объекта в строку.

I T E A
ONLINE

Формат JSON

Преобразование строки в объект называется десериализацией, в то время как преобразовании объекта в строку, таким образом, чтобы он мог быть передан через сеть, называется сериализацией.

Объект JSON может быть сохранён в собственном файле, который в основном представляет собой текстовый файл с расширением .json

Если бы мы загрузили этот объект в программу JavaScript, создали переменную с названием person, мы могли бы затем получить доступ к данным внутри неё, используя те же самые точечную и/или скобочную нотации

```
{
  "name": "Molecule Man",
  "age": 29,
  "secretIdentity": "Dan Jukes",
  "powers": [
     "Radiation resistance",
     "Turning tiny"
  ]
}
```

ITEA ONLINE

Формат JSON

- JSON - это чисто формат данных - он содержит только свойства, без методов
- JSON требует двойных кавычек, которые будут использоваться вокруг строк и имён свойств. Одиночные кавычки недействительны.
- Даже одна неуместная запятая или двоеточие могут привести к сбою JSON-файла и не работать. Вы должны быть осторожны, чтобы проверить любые данные, которые вы пытаетесь использовать (хотя сгенерированный компьютером JSON с меньшей вероятностью включает ошибки, если программа генератора работает правильно). Вы можете проверить JSON с помощью приложения вроде JSONLint
- В отличие от кода JavaScript, в котором свойства объекта могут не заключаться в двойные кавычки, в JSON в качестве свойств могут использоваться только строки заключённые в двойные кавычки.

ITEA ONLINE

Q&A

ITEA ONLINE

Совет из жизни: Тайм-менеджмент

1. Что это такое
2. Вам постоянно нехватает времени?
3. Планируйте свой день/неделю/месяц

ITEA

ONLINE