PHASE 1

WEEK 2

DAY 1

ELBRUS/BOOTCAMP

Objects Proto Prototype

ELBRUS/BOOTCAMP

OBJECT

OBJECT

Объект в JavaScript - неупорядоченный набор пар "ключ: значение".

```
let object = {} // пустой объект
```

```
let object = {
 key: value,
 key: value,
 key: value
}
```

ПОЛУЧЕНИЕ ЗНАЧЕНИЯ ПО КЛЮЧУ

```
let colors = {
 red: "rgb(255, 0, 0)",
 lime: "#00FF00",
 blue: "0f0"
}

Доступ к ключам
colors.red;            // "rgb(255, 0, 0)"

let key = "red";         // ключ как переменная
colors[key];             // "rgb(255, 0, 0)"

Object.keys(colors); //[ 'red', 'lime', 'blue' ]
```

ПРОВЕРКА СВОЙСТВ ОБЪЕКТА

```
let obj = {
 x: "first",
 y: "second"
};

"x" in obj;          // true
"z" in obj;          // false

obj.z = "new";
delete obj.x;        // Удаление свойства
"x" in obj;          // false
"z" in obj;          // true
```

ПЕРЕБОР КЛЮЧЕЙ

```
let obj = {
 x: "first",
 y: "second",
 z: "third"
};
for (let key in obj) {
 // key - текущий ключ
 // obj[key] - текущее значение
 if (obj.hasOwnProperty(key)) {
 }
}
```

МЕТОДЫ ОБЪЕКТА

```
let car = {
 model: "toyota",
 year: 2017,
 show: function() {
   console.log(car.model)
 }
};

car.show();
```

THIS

this - это специальная переменная, которая ссылается, на тот объект, в котором она на данный момент находится.

```
let obj = {
 model: "toyota",
 year: 2017,
 show: function() {
   console.log(this.model)   // obj.model
 }
};


obj.show();
```

THIS

```
function showMeThis() {
 console.log(this);
}

let eagle = {
 name: 'Oleg',
 show: showMeThis // { name: 'Oleg', show: Function }
};

let owl = {
   name: 'Igor',
   show: showMeThis // { name: 'Igor', show: Function }
};
```

CALL / APPLY / BIND

CALL / APPLY

Метод call() вызывает функцию с указанным значением this и индивидуально предоставленными аргументами.

```
function showMeThis(surname) {
 console.log(this.name);
 console.log(surname);
}

let Owl = {
   name: 'Igor',
};

showMeThis.call(Owl,'FamilyName')
```

Метод apply() вызывает функцию с указанным значением this и аргументами, предоставленными в виде массива

```
function showMeThis(surname) {
 console.log(this.name);
 console.log(surname);
}

let Owl = {
   name: 'Igor',
};

showMeThis.apply(Owl,['FamilyName'])
```

Bind

```
function showMeThis() {
 console.log(this.name);
}

let Eagle = {
 name: 'Oleg',
};

let showNameOfEagle =
showMeThis.bind(Eagle)
showNameOfEagle()
```

Метод bind() создаёт новую функцию, которая при вызове устанавливает в качестве контекста выполнения this предоставленное значение. В метод также передаётся набор аргументов, которые будут установлены перед переданными в привязанную функцию аргументами при её вызове.

Перерыв

PROTOTYPE

getPrototypeOf(), setPrototypeOf()

Два встроенных метода класса Object, позволяющих, соответственно, получить ссылку на прототип текущего объекта, а также вручную заменить ссылку на новую. С помощью первого, можно проследить всю иерархию наследования любого объекта.

`Object.getPrototypeOf(obj)` - возвращает прототип указанного объекта

`Object.setPrototypeOf(obj, prototype)` - устанавливает прототип указанного объекта в другой объект или null.

PROTOTYPE

```
function Student(name) {
 this.name = name;
}

 Student.prototype.sayHello = function() {
  console.log(this.name)
 };
```

Определение “функции-конструктора”

Создаем методы конструктора

Наследование

```
function Person(name) {
  this.name = name;
}

function Student(name, id) {
  Person.call(this, name);
  this.id = id;
}

Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

const fedor = new Student('Fedor', 'ABC-123');
```

вызываем конструктор родителя

Создаем новый объект
‘Student.prototype’ связанный с ‘Person’

Восстанавливаем значение
конструктора

Композиция

```
function Group(groupName) {
  this.groupName = groupName;
}

function Student(name) {
  this.name = name;
}

Student.prototype.setGroup = (group) => {
  this.group = group;
};
```

```
const jsGroup = new Group('JS lovers');
const fedor = new Student('Fedor');
fedor.setGroup(jsGroup);
```