

PHASE 1

WEEK 2

DAY 1



План

1. Объекты (objects)
2. Объектно-ориентированное программирование
3. `this` (ключевое слово указывающее контекст)
4. Prototype (прототип)

Объекты

Объекты

Объект — неупорядоченный набор пар "ключ: значение".

```
const emptyObject1 = {}; // создание через литерал
```

```
const emptyObject2 = new Object(); // создание через конструктор
```

```
const myObject = {  
  key2: "value 2",  
  key1: "value 1",  
};
```

```
console.log(myObject); // { key2: 'value 2', key1: 'value 1' }
```

Получение значения по ключу

```
const color = {  
  red: "rgb(255, 0, 0)",  
  green: "#00FF00",  
  blue: "#00f",  
};  
  
// доступ по ключу через точку  
color.red; // "rgb(255, 0, 0)"  
  
const key = "green";  
  
// доступ по строковому ключу  
color[key]; // #00FF00
```

Получение ключей и/или значений

```
const color = { red: "rgb(255, 0, 0)", green: "#00FF00", blue:
"#00f" };

// список всех ключей
Object.keys(color); // [ 'red', 'green', 'blue' ]

// список всех значений
Object.values(color); // ["rgb(255, 0, 0)", "#00FF00", "#00f"]

// список всех полей (пар ключ-значение)
Object.entries(color);
// [["red", "rgb(255, 0, 0)"], ["green", "#00FF00"], ["blue",
"#00f"]]
```

Проверка свойств объекта

```
const myObject = { x: "first", y: "second" };
```

```
// оператор in показывает, есть ли данное свойство у объекта
```

```
// или в его цепочке прототипов
```

```
"x" in myObject; // true
```

```
"z" in myObject; // false
```

```
myObject.z = "third";
```

```
"z" in myObject; // true
```

Удаление свойства из объекта

```
const myObject = { x: "first", y: "second" };
```

// оператор delete удаляет свойство и возвращает true, возвращает false при попытке удаления переменной из области видимости

```
delete myObject.x; // удаление свойства из объекта через точку  
"x" in myObject; // false
```

```
delete myObject["y"]; // удаление свойства из объекта через строку  
"y" in myObject; // false
```


Проверка свойств объекта

```
const myObject = { x: "first", y: "second" };
```

// Метод `hasOwnProperty` показывает, есть ли свойство на самом объекте

```
myObject.hasOwnProperty("x"); // true
```

```
myObject.hasOwnProperty("z"); // false
```

```
myObject.z = "third";
```

```
myObject.hasOwnProperty("z"); // true
```

`delete myObject.x;` // Удаление свойства из объекта

```
myObject.hasOwnProperty("x"); // false
```

Перебор ключей

```
const myObject = {  
  x: "first",  
  y: "second",  
  z: "third",  
};
```

```
for (let key in myObject) {  
  // key - текущий ключ  
  // myObject[key] - значение текущего ключа  
}
```

ООП

Объектно-ориентированное
программирование

Парадигма программирования, основанная на использовании **объектов**, которые могут содержать:

- данные (поля, свойства),
- код (методы).

Методы, привязанные к объекту, могут использовать/изменять данные (свойства) этого объекта или переиспользовать другие методы.

Для доступа к свойствам/методам объекта внутри внутри методов используется ключевое слово `this` — ссылка на текущий объект.

this

Ключевое слово, указывающее контекст

Кастомные методы объекта

```
const car = {  
  model: "Toyota",  
  year: 2017,  
  
  show: function () {  
    console.log(`${car.model}, год выпуска: ${car.year}`);  
  },  
};  
  
car.show(); // Toyota, год выпуска: 2017
```

this

Переменная с ссылкой на тот объект, в котором к ней обратились.

```
const car = {  
  model: "Toyota",  
  year: 2017,  
  
  show: function () {  
    console.log(`${this.model}, год выпуска: ${this.year}`);  
  },  
};
```

```
car.show(); // Toyota, год выпуска: 2017
```


this

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

```
const oleg = { name: "Олег", show: showMeThis};  
const igor = { name: "Игорь", show: showMeThis};
```

```
oleg.show(); // { name: 'Олег', show: Function }  
igor.show(); // { name: 'Игорь', show: Function }
```

this и контекст

При описании любой функции, мы всегда добавляем фигурные скобки `{ }`, а затем выполняем её через круглые скобки `()`.

При нахождении фигурных скобок движок JavaScript создаёт контекст выполнения, их может быть любое количество.

В первую очередь, движок JavaScript создаёт **глобальный контекст** выполнения, после чего нам будет доступен: глобальный объект и ключевое слово `this`.

- в браузере глобальный объект и `this` являются `Window`
- в Node.js глобальный объект и `this` является `global`

Разница this в функциях

```
const obj = {  
  arrowFunc: () => this, // нет контекста, ссылается на глобальный  
  объект  
  classicFunc: function () { return this } // ссылается на текущий  
  объект  
}
```

```
obj.arrowFunc(); // {} - global в Node.js или Window в браузере  
obj.classicFunc(); // { arrowFunc: [Function: arrowFunc],  
classicFunc: [Function: classicFunc] }
```

Метод call, привязка контекста

Вызывает функцию с указанным контекстом `this` и *отдельными аргументами*.

```
function showFullName(lastName, patronymic) {  
  console.log(`${lastName} ${this.name} ${patronymic}`);  
}
```

```
const igor = { name: "Игорь" };
```

```
showFullName.call(igor, "Иванов", "Сергеевич"); // Иванов Игорь  
Сергеевич
```

Метод `apply`, привязка контекста

Вызывает функцию с указанным значением контекстом `this` и *массивом аргументов*.

```
function showFullName(surname, patronymic) {  
  console.log(`${surname} ${this.name} ${patronymic}`);  
}
```

```
const igor = { name: "Игорь" };
```

```
showFullName.apply(igor, ["Иванов", "Сергеевич"]);  
// Иванов Игорь Сергеевич
```

Метод bind, привязка контекста

Создаёт *новую функцию*, привязанную к предоставленному контексту `this`.

```
function showFullName(patronymic, surname) {  
  console.log(`${surname} ${this.name} ${patronymic}`);  
}
```

```
const igor = { name: "Игорь" };  
const showIgor = showFullName.bind(igor);
```

```
showIgor("Сергеевич", "Иванов"); // Иванов Игорь Сергеевич
```

Прототипы

Прототип

Прототип — это шаблон объекта. Объект наследует свойства прототипа.

```
const myProto = { x: 1 };
```

```
// создать объект на основе шаблона
```

```
const myObject = Object.create(myProto);
```

```
myProto.x = 2;
```

```
console.log(myObject.x); // 2
```


Методы `getPrototypeOf`, `setPrototypeOf`

```
const myObject = Object.create({ x: 1 });  
  
// получить прототип объекта  
Object.getPrototypeOf(myObject); // {x: 1}  
  
// установить новый прототип объекта  
Object.setPrototypeOf(myObject, { y: 2 });  
Object.getPrototypeOf(myObject); // {y: 2}
```

Конструктор, prototype

```
// функция-конструктор
```

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

```
// создать метод sayHi для прототипа на основе конструктора Person
```

```
Person.prototype.sayHi = function () {  
  console.log(`Привет, я ${this.name}`);  
};
```

```
const ivan = new Person("Иван"); // создание объекта через конструктор  
ivan.sayHi(); // Привет, я Иван
```

Наследование

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

```
function Student(name, group) {  
  Person.call(this, name); // вызываем конструктор прототипа-родителя  
  this.group = group;  
}
```

```
// задать Person.prototype как родителя для Student.prototype  
Object.setPrototypeOf(Student.prototype, Person.prototype);
```

```
const ivan = new Student("Иван", "Еноты 2021 СПб");  
console.log(ivan); // Student {name: "Иван", group: "Еноты 2021 СПб"}  
console.log(ivan instanceof Student && ivan instanceof Person); // true
```

Композиция*

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

```
const ivan = new Person("Иван");
```

```
const canSayHi = {  
  sayHi: function () {  
    console.log(`Привет! Я ${this.name}`);  
  },  
};
```

```
Object.assign(ivan, canSayHi); // скопировать свойства из canSayHi в ivan  
ivan.sayHi(); // Привет! Я Иван
```