**Классы Вебкадеми**

**//ES5**

Используем **функцию - конструктор** и **метод в прототипе**:

function **PersonES5**(name, yearOfBirth, job) {

this.name = name;

this.yearOfBirth = yearOfBirth;

this.job = job;

}

PersonES5.**prototype**.**calculateAge** = function () {

var age = new Date().getFullYear() - this.yearOfBirth;

console.log("age: ", age);

};

var johnES5 = new PersonES5("Jhon", 1992, "Designer").calculateAge();

**//ES6**

Определяем **класс**  - объект **без запятых и без точек с запятой**

**class PersonES6 {**

*// определяем конструктор (как происходит заполнение новых экземпляров класса)*

**constructor**(name, yearOfBirth, job) {

this.name = name;

this.yearOfBirth = yearOfBirth;

this.job = job;

}

*// метод*

calculateAge() {

const age = new Date().getFullYear() - this.yearOfBirth;

console.log("age: ", age);

}

}

*Использование:*

const jhonES6 = new PersonES6("Jhon", 1992, "Designer");

jhonES6.**calculateAge**();

**Статический метод** - это метод, который приснадлежит **классу**, а не объекту (не её "prototype"). В классе такие методы обозначаются ключевым словом static. Нам не обязательно создавать для этого метода объект. Он вызывается непосредственно от класса.

class PersonES6 {

constructor(name, yearOfBirth, job) {

this.name = name;

this.yearOfBirth = yearOfBirth;

this.job = job;

}

calculateAge() {

const age = new Date().getFullYear() - this.yearOfBirth;

console.log("age: ", age);

}

**static** greeting() {

console.log("Hello! How are you?");

}

}

**PersonES6.greeting();** // Hello! How are you?

**!** Для объявления классов не работает **hoisting.** Если сначала попробуем создать объект, а потом объявить класс, то будет ошибка. // Cannot access 'PersonES6' before initialization

**!** Внутри класса могут находиться только **методы**, но не **свойства**. **?**

## Наследование

Сделаем наследование. Создаем класс Person. Далее создаем класс Athlete и наследуем его от класса Person. Создаем объект с помощью класса Athelete.

**//ES5**

function **PersonES5**(name, yearOfBirth, job) {

*this*.name = name;

*this*.yearOfBirth = yearOfBirth;

*this*.job = job;

}

PersonES5.**prototype.calculateAge** = function () {

  var age = new Date().getFullYear() - *this*.yearOfBirth;

  console.log(age);

};

function **AthleteES5**(name, yearOfBirth, job, olympicGames, medals) {

*// Вызываем конструктор родительского класса  чтобы он сработал для нового объекта класса Athlete,  а не для класса Person*

  PersonES5.call(*this*, name, yearOfBirth, job);

*this*.olympicGames = olympicGames;

*this*.medals = medals;

}

Чтобы создать корректную цепочку прототипирования, нужно в прототип потомка сначала записать прототип родителя. Для этого необходимо использовать **Object.create().** Object.create позволяет нам вручную задать протип для создаваемого объекта. И нам нужно в прототипе у класса AthleteES5 установить прототип класса PersonES5. Чтобы они были соединены.

AthleteES5.prototype = **Object.create(PersonES5.prototype**);

Потом уже допишем потомку свои методы:

AthleteES5.prototype.wonMedal = function () {

*this*.medals++;

  console.log(*this*.medals);

};

*//Использование*

var johnAthleteES5 = new AthleteES5("John", 1992, "athlete", "swimming", 10);

console.log(johnAthleteES5);

**//ES6**

class PersonES6 {

constructor(name, yearOfBirth, job) {

this.name = name;

this.yearOfBirth = yearOfBirth;

this.job = job;

}

calculateAge() {

const age = new Date().getFullYear() - this.yearOfBirth;

console.log("age: ", age);

}

}

*//* ***Класс - наследник***

class AthleteES6 **extends** PersonES6 {

*// конструктор*

constructor(name, yearOfBirth, job, olympicGames, medals){

*// сначала обязательно вызвать* ***конструктор родительского класса***

**super**(name, yearOfBirth, job);

this.olympicGames = olympicGames;

this.medals = medals;

}

wonMedal() {

this.medals++;

console.log(this.medals);

}

}

*//Использование*

const johnAthleteES6 = new AthleteES6("John", 1992, "athlete", "swimming", 10);

console.log(johnAthleteES6);

johnAthleteES6.calculateAge();

johnAthleteES6.wonMedal();

**Из учебника**: **Базовый синтаксис для классов** выглядит так:

class MyClass {

prop = value; // свойство

constructor(...) { // конструктор

// ...

}

method(...) {} // метод

get something(...) {} // геттер

set something(...) {} // сеттер

[Symbol.iterator]() {} // метод с вычисляемым именем (здесь - символом)

// ...

}

Затем использовать вызов new MyClass(). При этом автоматически вызывается метод constructor(), в нём мы можем инициализировать объект.

class User {

constructor(name) {

this.name = name;

}

sayHi() {

alert(this.name);

}

}

// Использование:

let user = new User("Иван");

user.sayHi();

Вот что на самом деле делает конструкция class User {...}:

* Создаёт функцию с именем User, которая становится результатом объявления класса. Код функции берётся из метода constructor (она будет пустой, если такого метода нет).
* Сохраняет все методы, такие как sayHi, в User.prototype.

При вызове метода объекта new User он будет взят из прототипа. Таким образом, объекты new User имеют доступ к методам класса. На картинке показан результат объявления class User: