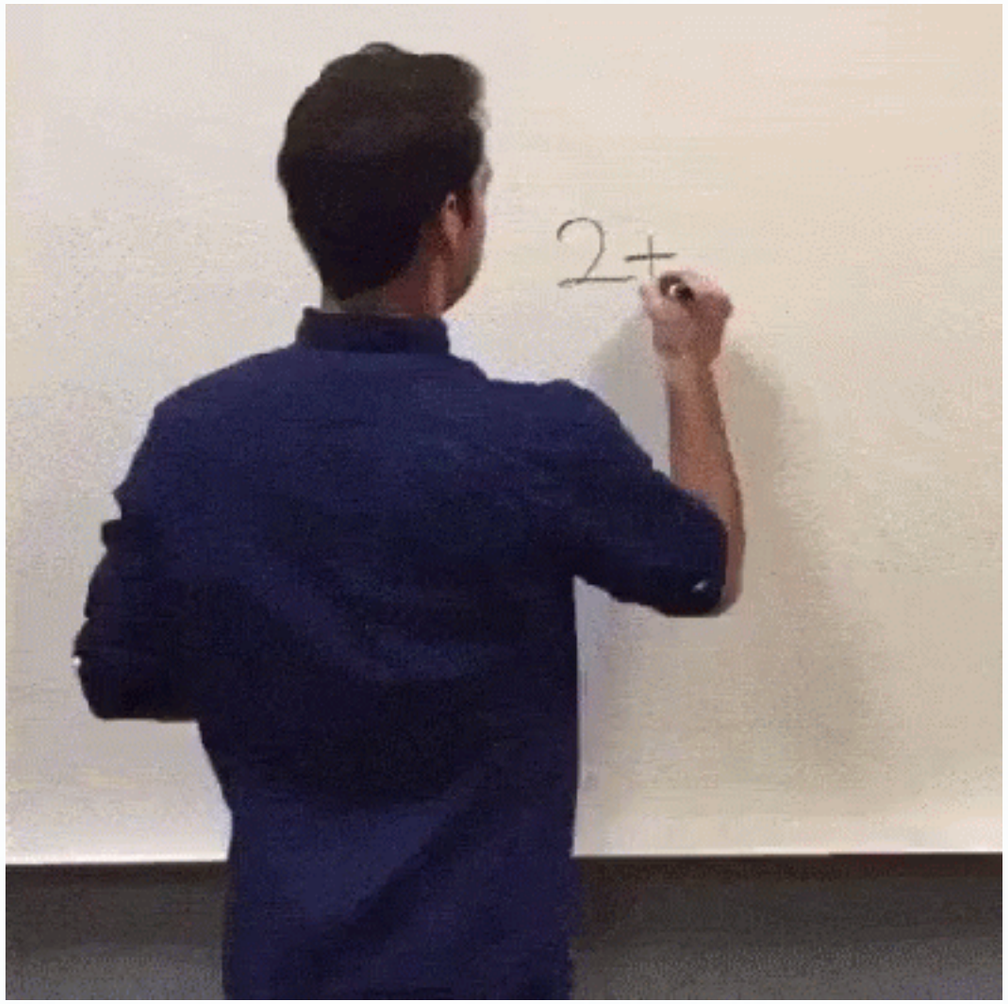
SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

Преглед на наученото до момента



http://swift-academy.zenlabs.pro/misc/the_process.gif

Разгрявка

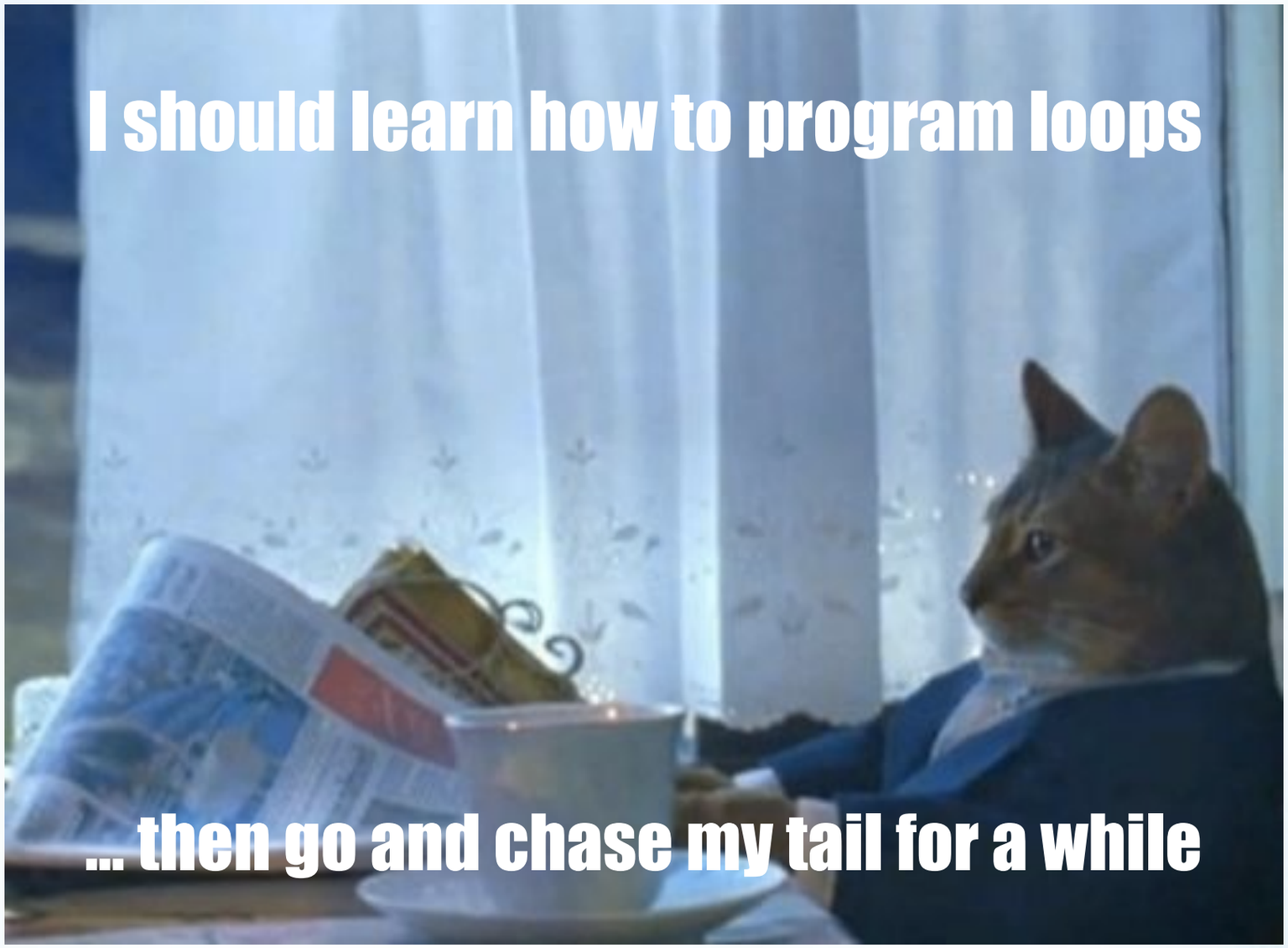
(Какво ще върне следният код?)

- `"this is awesome!".indexOf("some");`
- `"high 5".split(" ")[0] == 5`
- `"My way".split("").reverse().join("").charAt(2)`
-
```
for (let i=0; i<10; i++) {
   i -= 1;
}
```
- `["this", "is", "javascript"].join("! ").charAt(8);`

.. и малко теория за тонус

- Какво е променлива и как се ползва?
- Какво е функция? Какво знаете за функциите?
- Какво знаете за списъците? С какво се различават от простите типове данни и кога ги ползваме?
- Как създаваме списък?
- Какво са обектите и как се ползват?
- Как създаваме обект?

Какво още трябва да знаем



Още за числата в JavaScript

- Числата в JS са прост тип данни (primitive) от класа **Number**
- С тях извършваме математически аритметични операции и сравнение
- Числата могат да се превръщат в **String** по следният начин:
  - `5 + "";` // "5"
  - `String(5);` // "5"
  - `var number = 5;`
    `number.toString();` // "5"
- Любопитно: стойността NaN е число:
  - typeof NaN // Number

Кога две числа са наистина равни?

- Примери:
  - `5 == 5; // true`
  - `5 === 5; // true`
  - `5 == "5"; // true`
  - `5 === "5"; // false`
- С оператора "==" сравняваме по стойност
- С оператора "===" сравняваме по стойност и тип

Още за стринговете

- Стринговете (текстът) в JS са прост тип данни (primitive) от класа **String**
- С тях можем да извършваме следните операции:
  - `String.prototype.indexOf(word) // => Number`
  - `String.prototype.split(separator) // => Array`
  - `String.prototype.charAt(index) // => String`
- Конвертиране на **String** в **Number**
  - `"5" * 1; // 5`
  - `Number("5"); // 5`

Escape characters

- Стринговете могат да бъдат зададени с единични или двойни кавички
- Ако искаме стринг да съдържа в себе си символа за двойна кавичка, има 2 начина да го направим:

  Using single quotes: `'This string contains "double quotes" inside'`

  Escaping: `"This string contains \"double quotes\" inside"`

- При текст, съдържащ единична кавичка нещата са аналогични
- Текста, който е ограден с двойни кавички има свойството да преобразува последователности като: "\n" и "\t", съответно в нов ред и табулация.
  Това обаче е приложимо само за текста изписан в конзолата.
- За да използваме символа "\" в текст, трябва да напишем: "\\"
- https://www.w3schools.com/js/js_strings.asp

Още за Boolean

- Логическите стойности в JS са прост тип данни (primitive) от класа **Boolean**
- Тези стойности са точно 2: **true** и **false** , където **true === !false** и обратното
- Конвертиране на друг тип данни в Boolean:
  - `!!myVariable; // true or false`
  - `!!"text"; // true`
  - `!!""; // false`
  - `!!5; // true`
  - `!!0; // false`

Още за Array

- Подредените списъци в JS са сложен тип данни (object) от класа **Array**
- Списъка се използва за съхраняване на много на брой, еднотипни данни, които могат да бъдат числа, стрингове, булеви или обекти
- Често използвани методи на класа (обекта) Array:
  - `Array.prototype.indexOf(element); // Number`
  - `Array.prototype.join(separator); // String`
  - `Array.prototype.reverse(); // Array`
  - `Array.prototype.sort(function); // Array`
  - `Array.prototype.filter(function); // Array`
  - http://swift-academy.zenlabs.pro/misc/Lists.pdf

допълни-
телен слайд

Сортиране на списък

- Сортирането на списък означава подреждане на елементите му по определен критерий
- Стандартно можем да сортираме елементите по *азбучен ред* с метода **sort** на Array обектите:

```
var a = ["grape", "apple", "kiwi", "banana"]
a.sort() // => ['apple', 'banana', 'grape', 'kiwi']
```

- Ако искаме да сортираме числа, обаче, трябва да напишем функция за сравнение, която да подадем на функцията sort

```
var a = [2, 1, 16, 3, 9]
a.sort() // => [ 1, 16, 2, 3, 9 ]
```

допълни-
телен слайд

Сортиране на списък чрез сравняваща функция

```
/* Сравняващата функция ще получи като аргументи всеки 2 елемента от списъка (т.е. ще бъде извикана от сортиращата функция много пъти). От нея се очаква да върне положително число, ако първият елемент е по-голям, отрицателно число, ако е по-малък и нула ако двата елемента са равни */
```

```
function compare(a, b) {
  if (a > b) { return 1; }
  if (a < b) { return -1; }
  return 0;
}

var numbers = [2, 1, 16, 3, 9];
numbers.sort() // => [ 1, 16, 2, 3, 9 ]
numbers.sort(compare) // => [ 1, 2, 3, 9, 16 ]
```

допълни-
телен слайд

Сортиране на списък чрез вложени цикли

```
/* Най-популярният алгоритъм за сортиране е метода на мехурчето.
По принцип вече не се ползва, но е добре човек да го знае от
обща култура (може да ви питат за него на интервю за работа) */

function bubleSort(array) {
   for (var i=0; i<array.length; i++) {
     for (var j=0; j<i; j++) {
        if (array[i] < array[j]) {
           var tmp = array[i];
           array[i] = array[j];
           array[j] = tmp;
        }
     }
   }
   return array;
}

bubleSort([2, 1, 16, 3, 9]) // => [ 1, 2, 3, 9, 16 ]
```

допълни-
телен слайд

Списък от обекти

- Същото като списък от числа или стрингове, само че елементите на списъка са обекти

```
var fruits = [];
fruits.push({ name: "grape", price: 2 });
fruits.push({ name: "apple", price: 1 });
fruits.push({ name: "kiwi", price: 5 });
fruits.push({ name: "banana", price: 2 });

console.log(fruits[0]);
                // => { name: "grape", price: 2 }
console.log(fruits[2].name); // => "kiwi"
console.log(fruits[3].price); // => 2
```

Референтни типове данни

- Референтни се наричат онези данни, чиято стойност е само указател към мястото във виртуалната памет на програмата, където е съхранена същинската стойност
- В JavaScript имаме прости и сложни типове данни.
- Простите типове (т.нар. примитиви или primitive) са числа, Boolean, Undefined, String, Null
- Сложните типове данни са обекти (в това число и списъци) и функции
- Сложните типове данни в JS са референтни

Обектите

- Обектите в JS са сложен тип данни (object) от класа **Object**
- Използваме ги за съхраняване на различни характеристики, които се отнасят за обекта. Например:

```
var car1 = {}; // => Object
car1.brand = "Ford";
car1.speed = 230;
car1.color = "red";

var car2 = {
  brand: "Ford",
  speed: 230,
  color: "red"
}; // => Object
```

Сравняване на обекти

- Два обекта могат да имат абсолютно еднакви полета (характеристики) и стойности за тях, но при при обикновенно сравнение те няма да са еднакви:
  `car1 == car2 // false`
- Това е защото истинската стойност на обекта е само указател към мястото в паметта, където всъщност се съхраняват неговите полета
- Единственият начин два обекта да бъдат еднакви по стойност, е ако създадем единия обект от другия. Ето така:

```
var car3 = car2;
car3 == car2 // true
```

- Това означава, че и двата обекта сочат към едно и също място в паметта и към едни и същи съхранени стойности:

```
car2.color; // => "red"
car3.color = "black";
car2.color; // => "black"
```

Сравняване на обекти II

- Има 2 начина да разберем дали 2 обекта имат еднакви полета и стойности.
- Първият е да ги сравним по отделно:

```
car1.brand === car2.brand &&
car1.speed === car2.speed &&
car1.color === car2.color
```

- Вторият е да използваме функцията JSON.stringify, която преобразува обекта в текст:

```
JSON.stringify(car1) === JSON.stringify(car2);
                                                // true
```

допълни-
телен слайд

for x in object

- Обектите не са списъци и не могат да се обхождат така както списъците защото нямат наредба на полетата
- За да обходим всички полета на обекта, можем да използваме една специална форма на оператора **for**, която е само за обекти:

```
bikes.forEach(function (bike) {
  for (var key in bike) {
    console.log(key + ":", bike[key]);
  }
  console.log("\n");
});
```

Често допускани грешки - 2

- Презаписване на променливи или функции когато използваме същото име:

```
var apple = { type: "golden" };
function apple() {}
apple.type; // undefined
```

- Създаване на променлива без да използваме var:

```
apple = { type: "golden" }; // creates a global object
```

- Присвояване на стойност в логическо сравнение

```
if (var1 = var2) {} // overrides var1 with the value of var2
```

- Сравняване на обекти по стойност

- Как да се пазим от грешки: **`"use strict";`** + внимаваме!

Упражнение

Задача 1 (5мин)

- Направете обект **student**
- Помислете какви характеристики (полета) може да има този обект
- Помислете също как да ги именувате и какви ще са тяхните стойности
- Пример:

```
// Рзсъждение: "Ученикът има име"
var student = {
  name: "Иван Петров"
}
```

Задача 2

- Отворете следния repl.it: https://repl.it/EZix
- Ще видите списък с обекти, които са велосипеди
- Направете така, че функцията logBikes да принтира в конзолата името на велосипеда и цената му, за всеки велосипед
- Подсказка:
  Започнете като си създадете една променлива bike и зададете да е равна на първият елемент от списъка. След това логнете в конзолата името и цената на този обект.
  За да повторите това и за останалите обекти, трябва да използвате цикъл за обхождане (например **for**)

Задача 3

- Отворете отново задачата за велосипедите: https://repl.it/EZix
- Филтрирайте всички колела, чиято цена е над 300 като използвате метода filter (MDN filter)
- Напишете функция `filterByPrice`, която получава като аргументи минимална и максимална цена и връща списък с имената на колела, които имат цена между 2-те стойности
- Пример:

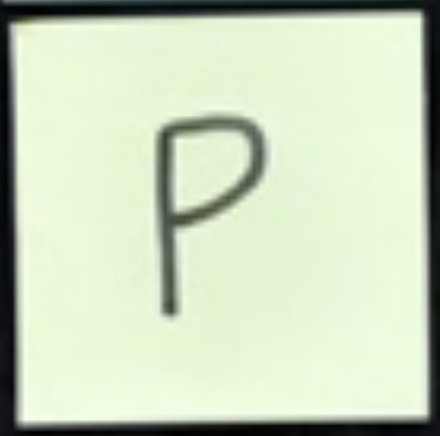
```
var filtered = filterByPrice(100, 600);
                                // => ["Drag", "Ram"]
```

Задача 1 - за домашно

- Отворете следният линк: https://repl.it/CPaV/0
- Имплементирайте двете функции, които смятат количеството на артикулите в пазарската кошница.
- Функцията **quantity()** трябва да сметне общото количество от всички артикули
- Функцията **veganQuantity()** , трябва да сметне само колко бройки са общо плодовете и зеленчуците.

Задача 2 - за домашно

- Използвайте кода от предната задача.
- Решете отново задачата, но този път използвайте функции от висок ред
- Подсказка:
  - Използвайте функцията **filter()**, за да филтрирате всички плодове и зеленчуци
  - Използвайте функцията **forEach()**, за да изчислите общото количество



Object Oriented Programming

ООП като стил на програмиране

- Обектно ориентираното програмиране (както и обектно ориентираният модел) е подход, при който основната част от програмата е изградена от обекти, които си взаимодействат
- В ООП модела обектите са нещо повече от структура от данни. Там те са основна градивна единица за програмата
- Обекта е програмна абстракция, която има за цел да представи дадено нещо като набор от характеристики и функционалност
- JavaScript не е обектно-ориентиран език, но е съвместим с ООП модела (до някаква степен)

История

- ООП се появява след функционалното и логическото програмиране
- Някои гледат на него като на революция в програмния подход, тъй като предлага интуитивен модел за представянето на данните
- Например - имаме база от данни, съхраняваща данните на онлайн магазин (таблици с продукти, потребители, поръчки)
- Функционалният модел, разполага само с ограничен набор от типове данни (числа, текст, списък, именуван списък) и затова при него е по-трудно представянето на данните от базата като програмни компоненти
- ООП модела, позволява на програмиста да създаде свои типове данни като например: Продукт, Потребител или Поръчка

Пример:

- Обект от тип (клас) Object, представящ велосипед:

```
var bike = {
  name: "Drag",
  price: 300
};
```

- Обект от тип (клас) Bike в Javascript:

```
function Bike(name, price) {
  this.name = name;
  this.object = object;
}
var bike = new Bike("Drag", 300);
```

Клас

- Класовете в JavaScript са функции, които конструират обекти
- В обектно-ориентирания подход всичко е обект, а всеки обект е инстанция на даден клас.
- Следователно класовете в JS са функции, които могат да "произвеждат" свои обекти-инстанции
- За да създадем обект от даден клас, използваме името на класа и ключовата дума **new**:

```
var text = new String("this is a Sting instance")
text // => [String: 'this is a Sting instance']
var arr = new Array() // => []
```

Класа ни служи, за да можем лесно да конструираме обекти от един и същи тип!

Инстанции

- Както казахме, за да създадем нова променлива - обект, използваме името на класа и ключовата дума **new**
- Когато създадем обект по този начин, казваме че той е инстанция (представител) на съответния клас
- Когато създадем обект без експлицитно да използваме **new** (например: `var object = {}`), то той по подразбиране става инстанция на класа Object
- Заключение: всички обекти са инстанции на даден клас

Конструктор

- Създаването на нов клас в JS е изключително лесно*
  Единственото, което трябва да направим, е да създадем функция, чието име започва с главна буква:

```
function Bike () {
  // constructing a new bike …
}
var bike = new Bike();
```

- Тази функция се нарича ***конструктор*** и реално тя е отговорна за произвеждането на новите обекти-инстанции

**забележка: в ES6 има нов подход, който няма да разглеждаме в курса, но можете да си прочетете допълнително*

this

- Конструирането на обект в конструктора става посредством ключовата дума **this**
- В контекста на конструктора **this** е референция към текущия обект (този който в момента се създава)

```
function Bike (name, price) {
  this.name = name;
  this.value = price;
}
var bike = new Bike("Scott", 560);

  // => { name: "Scott", value: 560 }
```

this

- Реално this съществува навсякъде и винаги представлява референция към текущия обект
- Ако няма текущ обект, this е референция към window (сещате ли се защо?)

instance методи
или: поведение на обекта

- Основното качество на обектите е, че част от тяхните полета могат да бъдат функции
- Тогава казваме че тези функции са instance методи или просто методи на обекта
- Instance методите ни служат за да зададем поведение на обектите, които са от даден клас
- Например - всички обекти в JS имат метода toString() - той е наследен от класа Object, където този метод е дефиниран

Пример:

```
function Hero(name, knownAs) {
  this.name = name;
  this.nickname = knownAs || name;
  this.kickAss = function (enemy) {
    // actions to kick the enemy's ass
  };
  this.saveWorld = function () {
    // actions to save the world
    console.log("Once again " + this.nickname + " saved the world!");
  };
}

var superman = new Hero("Clark Kent", "Superman");
var chuck = new Hero("Chuck Norris");

superman.saveWorld();
```

Прототип на класа

- Прието е методите на обектите от един клас, да се създават посредством т.нар. *прототип* на класа, вместо в конструктора

  Пример:

```
Bike.prototype.toHTML = function () {
  var html = "<h3>" + this.name + "</h3>";
  html += "<p>price: " + this.value + "$</p>";
  return html;
}
var bike = new Bike("Scott", 560);
document.write(bike.toHTML());
```

Прототип на класа - 2

- Всеки клас има прототип (ние няма нужда да го създаваме, той си е там)
- Прототипа се достъпва като поле на самия клас: ClassName.prototype
- Разликата между конструктор и прототип е:
  - когато създаваме нови обекти-инстанции, методите и полетата от конструктора се копират за всеки обект
  - докато методите и полетата от прототипа, са достъпни за всички обекти от класа, без да се копират в тях

Прототип на класа - 3

- Когато опитаме да изпълним метод на обекта, той го търси първо в себе си (т.е. дали е зададен от конструктора посредством this)
- Ако метода не съществува директно в обекта, тогава се търси в прототипа на класа
- Ако метода го няма и в прототипа, той се търси в прототипа на класа Object (Object.prototype)
- Например метода toString()
- https://www.w3schools.com/js/js_object_prototypes.asp - тази статия дава добри насоки, но съдържа леко подвеждаща информация: прототип и конструктор не са едно и също!

Още малко преговор

- Какво е `hoisting` ?
- Какво връщат функциите без `return`?
- Как работи тройния оператор `?`
- Кои типове данни са референтни `?`
- `typeof NaN ?`
- Какво е `IIFE ?`

Задача 3 - за домашно

- Направете клас за колело Bike с instance метод toHTML(), който да генерира представяне на колелото в HTML-а
- Като използвате списъка с колелата от предните задачи:
  1. създайте аналогичен списък с колела-обекти, които са инстанции на Bike. Например ето така:

     ```
     var newList = bikes.map(…);
     ```
  2. изведете всички колела в свързана HTML страница
  3. задайте приятно оформление на резултата чрез CSS

Задача 4 - за домашно

- Използвайте кода от примера за класа Hero
- Създайте клас BadGuy(name, knownAs)
- Създйте 1-2 инстанции от класа BadGuy, след което извикайте метода kicksAss на Hero с аргумент - обект от тип BadGuy
- Пример:

```
var batman = new Hero("Bruce Wane", "Batman");
var joker = new BadGuy("Joker");
batman.kicksAss(joker);
  // => "Batman just kicked Joker's ass"
```

KEEP CALM AND LEARN JAVASCRIPT

Полезни връзки

- This: https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/this

- Prototype: https://www.w3schools.com/js/js_object_prototypes.asp

- ES6 Classes: https://www.sitepoint.com/object-oriented-javascript-deep-dive-es6-classes/

- Code wars https://www.codewars.com/dashboard

Примери

http://swift-academy.zenlabs.pro/lessons/lesson16/examples/download.zip

Домашно

http://swift-academy.zenlabs.pro/lessons/lesson16/homework