SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

Как да пишем работещ JS код?

SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

Front-End Development

Работете итеративно (на малки, последователни стъпки).

След всяка нова стъпка, проверявайте как и дали скрипта ви все още работи.

Използвайте **console.log()**, за да си изведете текущите стойности на променливите и резултатите от функциите ви.

Използвайте дебъгера, за да виждате къде какво точно става.

Често грешката е нещо много дребно като лиспваща запетайка или скоба, но пък чупи цялата програма.


WASTES 2 HOURS DEBUGGING
BROKEN JAVASCRIPT
MISSING A CURLY
BRACE
quickmeme.com

Scope

Контекст

- Обхват (зона на действие) на това за което се говори. Извън тази **зона на действие**, това за което се говори няма смисъл.

- Пример от реалния живот:
  - Видя ли Лора?
  - Да.
  - Как е тя! извън контекста на разговора, това изречение няма смисъл

- От компютърните езици:

```
function calc(x) {
  return x / 2;   // ok, I know x
}
console.log(x);  // who the hell is x?..
```

Контекст в JavaScript

- *Scope-а в JavaScript се определя от функциите*
- Scope е онази част от кода, където нашите променливите и именувани функции имат смисъл. Извън тази зона те не съществуват
- Зоната на действие на една променлива или функция е тялото на функцията, в която те са дефинирани
- Т.е. тази зона започва там където е отварящата скоба на обграждащата функция (“{“) и свършва със затварящата скоба (“}”) или при първия **return**, който ще бъде изпълнен
- Понякога ще казваме “зона на видимост” или “текущ контекст”

Глобален контекст

- Това е най-външния възможен контекст. Той съдържа всички останали.

- Глобалният контекст е закачен за глобалната променлива **window** (т.е. всичко което дефинираме директно в глобалният контекст, отива в **window**)

- Всички променливи и функции, които се декларират в глобалния контекст, са глобални и съответно видими отвсякъде

- Създаването на глобални функции и променливи се счита за лош стил и може да доведе до непредвидени грешки

Локален контекст

- Всичко, което не е глобален контекст е локален контекст
- Един вид - всичко, което се намира в тялото на функция (което е оградено от function() { .. })
- **Локалните функции и променливи не се виждат отвън!**
- Всяка локална функция или променлива, се вижда във всички вложени в нея контексти (scopes)
- Локалният контекст се нарича още *closure*

```
var x = "I'm global";
function level1 () {
    var y = "level 1 variable";
    function level2 (param) {
        var z = "level 2 variable";

        console.log("I can see x and y and z");
        console.log("I can also see param");

        return x + y + z + param;
    }
    console.log("I can see x and y and level2");
    return level2("surprise!");
}
```

Глобални:
x, level1

Локални за level1:
y, level2

Локални за level2:
z, param

return

когато направим return във функция, кодът който се намира след return-а не се изпълнява:

```
function example() {
  console.log(“This text will be printed”);
  return;
  console.log(“This text will NOT be printed”);
}
```

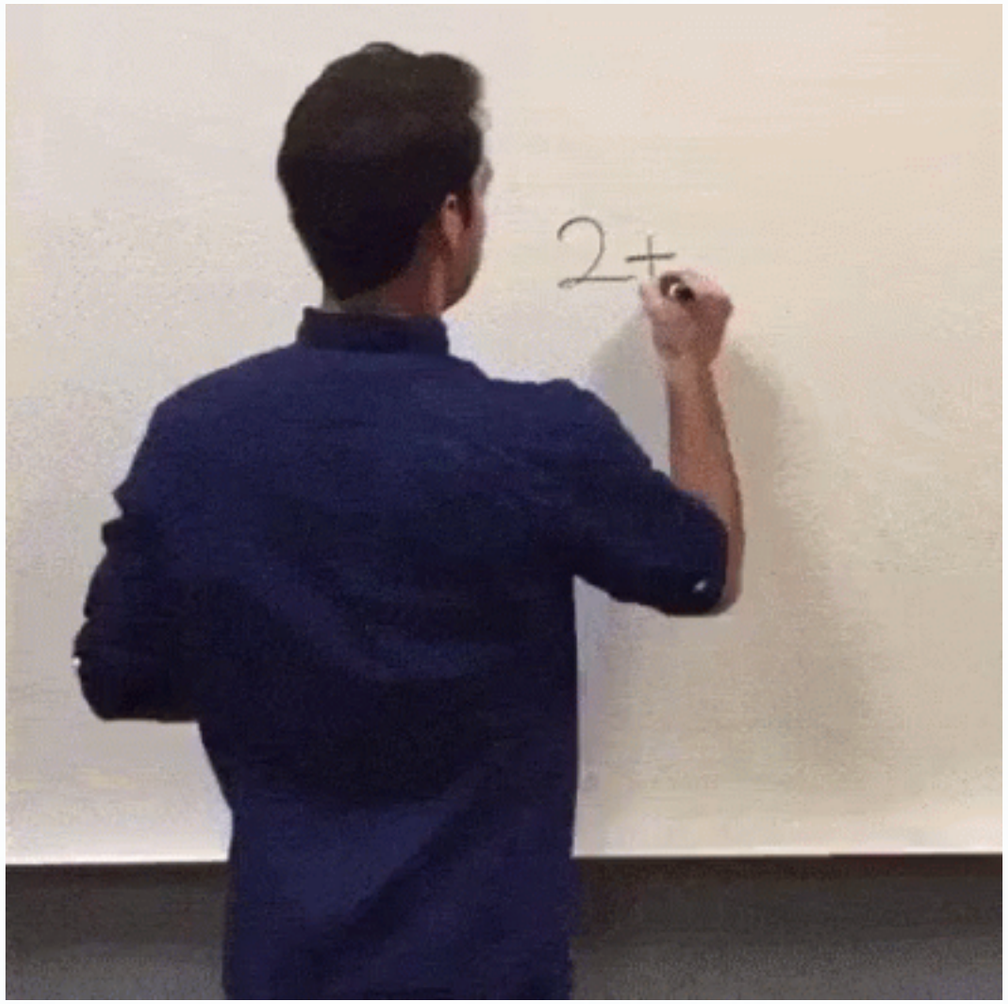
Счита се за лош стил ако има прекалено много returns в една функция. В идеалният случай има само един и той е най накрая във функцията, точно преди затварящата скоба (“}”)

Namespace

- Както и с контекста - има глобален и локален неймспейс
- Неймспейса е начин да си дефинираме глобални функции, които обаче да не *цапат* глобалният контекст
- За целта се ползват глобални обекти. За тях не считаме че цапат глобалният контекст, защото тяхните имена започват с главна буква и те могат да съдържат колкото си искат финкции и променливи, които вече ще са в тяхния локален контекст, но ще са достъпни от вън през името на самият глобален обект
- пример: Array, Date, Math, Boolean

Въпроси?

Преговор на наученото до момента



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

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"
    ```
    ```

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.indexOf(word) // => Number`
  - `String.split(separator) // => Array`
  - `String.charAt(index) // => String`
- Конвертиране на **String** в **Number**
  - `"5" * 1; // 5`
  - `Number("5"); // 5`

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 мого на брой, еднотипни данни, които могат да бъдат числа, стрингове, булеви или обекти
- Често използвани методи на класа (обекта) Array:
  - `Array.indexOf(element); // Number`
  - `Array.join(separator); // String`
  - `Array.reverse(); // Array`
  - `Array.sort(function); // Array`
  - `Array.filter(function); // Array`

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 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
```

Често допускани грешки

- 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 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

- Помислете как да си направите обект **robot**, който да използвате за решаване на задачата за роботчето.
- Трябва да прецените какви характеристики има роботчето, как да ги именувате и какви ще са тяхните стойности
- Например:
  "Роботчето има посока посока, в която е завъртяно. При завъртане наляво или надясно тази посока се сменя"
  "Роботчето има позиция, тъй като след изпълнението на 1 ход, то се намира в различна позиция"

Задача 3

- 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**)

Въпроси?

Функции от по-висок ред

- Конструкцията for:

```
for (let i=0; i<bikesList.length; i++) {
  var bike = bikesList[i];
  console.log(bike.name + ": " + bike.price + "$");
}
```

- и метода Array.forEach():

```
bikesList.forEach(function(bike) {
  console.log(bike.name + ": " + bike.price + "$");
});
```

- Правят едно и също. С разликата, че във вторият пример използваме композиция на функции и за това казваме, че forEach() е функция от по-висок ред

Упражнение

Задача 1

- Направете всички задачи от домашното към лекция 13 (http://swift-academy.zenlabs.pro/lessons/lesson13/homework)
- Свържете ги с html файл и покажете резултатите в браузъра, като използвате обекта **document** и полето **innerHTML**:

```
var element = document.getElementById("result");
element.innerHTML = "the result of the function";
```

Задача 2

- 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 3

- 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

Подготовка за контролното следващият път

- Преговорете си много добре HTTP протокола и HTML формулярите
- Преговорете псевдо-класовете и псевдо-елементите
- Ще има основно въпроси свързани със заявките към бекенд-а
- Ще има няколко въпроса за JS синтаксис
- Ще има няколко много лесни JS задачи за indexOf() и charAt() методите

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(12);`


KEEP
CALM
AND
LEARN
JAVASCRIPT

Пример как да започнем да решаваме задачата за роботчето (ако остане време)

Игра

(по желание - за домашно)

http://zenlabs.pro/api/game

Полезни връзки

- **JavaScript best practices**:
  https://www.w3.org/wiki/JavaScript_best_practices

- MDN документация:
  forEach
  filter
  reduce

- JavaScript онлайн обучения:
  https://www.codeschool.com/search?query=JavaScript
  https://www.codecademy.com/learn/javascript

- Една примерна имплементация на задачата за роботчето:
  http://zenlabs.pro/robot/

Примери

http://swift-academy.zenlabs.pro/lessons/lesson14/examples/download.zip

https://repl.it/E0Ct/13

https://repl.it/E0PZ/0

Домашно

http://swift-academy.zenlabs.pro/lessons/lesson14/homework