SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

JavaScript

SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

Front-End Development

Упражнение: смело напред!

- Отворете следният уебсайт и се регистрирайте (ако все още не сте го направили): https://repl.it/
- Вече регистрираните се, могат да се изстрелят с 200 към конзолата: https://repl.it/languages/javascript
- Конзолата е нещо, което ни помага да пишем JavaScript код
- Тя прави по едно изчисление на ред, ето така:

Програмист	Конзола
➡ пише 1 ред код и натиска "Enter"	➡ изчислява кода и показва резултата ➡ очаква ново въвеждане на код

JavaScript (съответно и JS конзолата) може:

(1) Да смята

a. напишете `2+2` и натиснете "`Enter`"

b. може и по-сериозни сметки: `4*(3-1)/2`

(2) Може да сравнява

a. опитайте: `1 < 2`

b. също и: `3 > 2 + 1`

c. а сега: `3 > (2 + 1)`

(3) Дори може и да говори (изпишете с кавичките): `"Hello!"`

Може още:

(1) Да сравнява текст

```
a."Ivan" > "Pesho"

b."Ivan" == "Vanio"
```

(2) Да събира текст

```
a."So" + " cool"

b."Az" + " " + "sam" + " " + "nomer" + " " + 1
```

(3) И да ни казва когато не сме прави

```
a."Nomer " - 1

b.a + b
```

Задача

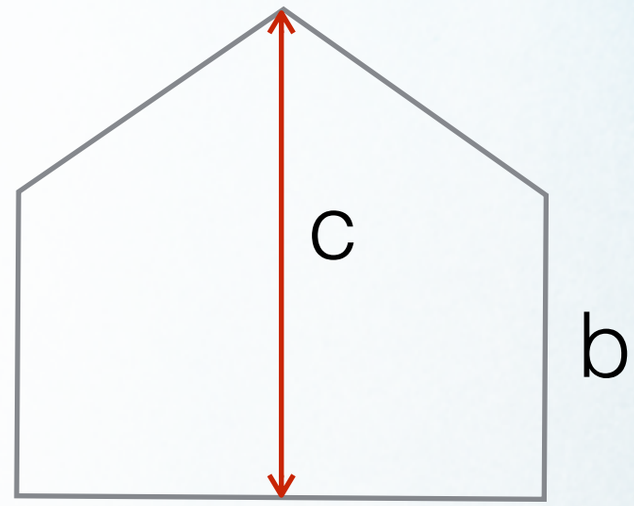
(1) Сметнете площта на следната фигура:

(2) Където:

(1) a = 10

(2) b = 6

(3) c = 10



Формулата за площ на правоъгълник е едната страна по другата. На триъгълник: едната страна по височината, делено на 2

Съхраняване на стойности

(1) В Javascript, можем да си съхраняваме стойностите в т.нар. "именувани променливи"

(2) Напишете: `var promenliva`

(3) Сега напишете само: `promenliva`

(4) Това, което се случи е, че си създадохме именувана променлива. В нея можем да съхраняваме различни стоности:

```
promenliva = 1
promenliva = "Gosho"
```

(5) И също можем да правим изчисления:

```
promenliva + " e nomer 1"
```

Задача 2

(1) Решете отново задача 1, но този път използвайте променливи: a, b и c

(2) Създайте променлива result, на която да зададете резултата от сметката

(3) Накрая, накарайте конзолата да изпише следното:

```
"Площта на дадената фигура е: xx"
```

където на мястото на "xx" трябва да излиза числото, което сте сметнали

Съхранени изчисления

(1) Освен стойности, в Javascript можем да съхраняваме и нашите изчисления

(2) Наричаме ги функции, и ги създаваме ето така:

```
function findArea(a, b, c) {
  var result;
   // изчисленията на за получаване на резултата
  return result;
}
```

(3) След като веднъж вече сме си съхранили дадена функция, можем да я използваме когато и колкото си пъти искаме, по следният начин: **`findArea(10, 6, 10)`**

Задача 3

(1) Като използвате решението на задача 2, направете функция, която да смята площта на дадената фигура

- ✳ използвайте редактора отляво на конзолата в repl.it за писане на многоредов код. След това натиснете "run" от навигацията горе

(2) Изпълнете функцията няколко пъти с различни стойности на a, b и c, за да се уверите, че наистина работи

> ★ *Изпълняването на функцията се нарича още: "извикване"*
>
> ★ *Променливите, които стоят в скобите след името на функцията се наричат: "параметри" на функцията*
>
> ★ *Кодът, който стои в кърдавите скоби, се нарича: "тяло"*

ES6

Функциите вече са още по опростени като запис:

```
(a,b,c) => { return ((b+c)/2)*a; }
```

(това е така наречената: *arrow* функция)

Нагледно:

```
 : (a,b,c) => { return ((b+c)/2)*a; }
=> [Function]
 : var area = (a,b,c) => { return ((b+c)/2)*a; }
=> undefined
 : area(10, 6, 10)
=> 80
```

Впрочем в Javascript, функциите са *стойност* и могат да бъдат задавани като стойност на променливи.

Това не е част от ES6, а си е едно от най-фундаменталните неща в JS!

One does not simply learn JavaScript

Интерпретативни езици за програмиране

- Всяка програма, написана на език за програмиране представлява последо-вателност от команди (инструкции), изчислителни и логически операции

- JavaScript е интерпретативен език, което означава, че командите от нашата програма, се интерпретират (изчисляват) директно

- Разликата между интерпретативните езици и тези, които изискват предварителна компилация (C++, Java, etc..), е че интерпретативните езици се изпълняват директно от source кода, а при компилаторните source кода трябва първо да се компилира (преобразува) до изпълним код

- При компилаторните езици, ако има грешка, изпълнението е невъзможно, тъй като грешката възниква още по време на компилация. При интерпретативните - грешката възниква в хода на изпълнението

Интерпретатори

- Браузърът интерпретира JavaScript кода "ред по ред" и така на всяка стъпка прави някакво изчисление или изпълнява логическа операция

- Т.е. браузърът има вграден JavaScript интерпретатор

- Същият този интерпретатор се използва и от JavaScript конзолата на браузъра

- Именно конзолата ни позволява да въвеждаме нашата програма (набор от инструкции) ред по ред (т.е. инструкция по инструкция), като за всеки ред ни показва междинните резултати от изпълнението

Синтаксис

- Както обикновенните езици, така и тези за програмиране, си имат синтаксис
- Това са правилата за формиране на думите и изразите от езика в разбираема за интерпретаторите последователност
- Синтаксиса на JS ще взимаме в движение, но като за начало можем да кажем следните 2 основни неща:
  - всяка команда (ред) завършва с ; (точка и запетая)
  - всяка "дума", която не е наименование, дефинирано от нас, не е дума от езика и не е коментар - води до грешка

Коментари

- Както във всеки друг език, така и в JavaScript, можем да пишем прозиволен текст като коментар, който се игнорира от интерпретатора

- Има два вида коментари:
  - едноредови:

```
alert('hi!'); // от тук до края на реда, всичко се игнорира
```

  - многоредови:

```
/* Този коментар би могъл да бъде поставен навсякъде
   както и да се разпростира на няколко реда */
```

Типове данни

- Програмата работи с данни. Те са обикновенно *числа*, *текст* или пък са по-сложни структури като *списъци* и *обекти*

- Данните са това, което програмата ползва за да направи съответните изчисления:

  Например ако въведа някъде рожденната си дата - програма може да изчисли на каква възраст съм

- Данните също могат да се използват за логически операции:

  Например ако изчислената възраст е под 18 години, може да ми бъде отказан достъп до някакво съдържание

Прости и сложни типове данни

- Прости типове данни са:
  - числа:
    ```
    typeof(1) // 'number'
    ```
  - текст:
    ```
    typeof("text") // 'string'
    ```
  - булеви (логически):
    ```
    typeof(true) // 'boolean'
    ```
- Сложните типове са: списъци, обекти и функции

Числа (Numbers)

- Цели числа: например 2 или -10
- Числа с плаваща запетая: 3.5
- С числата можем да извършваме основните аритметични операции:
  - събиране: 2+3
  - изваждане: 10-4
  - умножение: 2*3
  - деление: 10/5

Числа - особености

- операции със скоби - както в математиката

- при деление резултата понякога е число с плаваща запетая, ако искаме да го направим цяло, можем да използваме методът **round** на класа **Math**, ето така:

```
Math.round(8/3)   // => 3
```

- Понякога искаме да разберем какъв е остатъка при деление (модул), това става по следния начин:

```
8%3 // => 2
```

Текстов низ (String)

- Всеки набор от символи, заграден в двойни или единични кавички е String
  - "hello"
  - "1"
  - "Some very long text inputs are also strings"
  - 'a'
  - ""

Основни операции с текст

- Съединяване (concat):

```
"hello" + ", world!" // "hello, world!"
```

- Дължина:

```
"hello, world!".length // 13
```

- Символ на определена позиция:

```
"hello, world!".charAt(4) // 'o'
```

- Индекс (позицията) на определен символ:

```
"hello, world!".indexOf('l') // ‘2'
"hello, world!".indexOf('x') // ‘-1'
```

Булеви стойности (Boolean)

- **true** и **false**
- Използват се за да кажат дали нещо е вярно или невярно
- Използват се и се получават при изпълнение на логически операции
- Примери:

```
1 > 2                  // false
true == false          // false
"hello" != "world" // true
```

Списъци (Arrays)

- Това е колекция от стойности
- Стойностите в списъка, се наричат елементи. Те могат да бъдат числа, текст, обекти
- Синтаксис (елементите се изреждат между квадратни скоби, разделени със запетайки):

```
['a', 1, 'w']  // => ["a", 1, "w"]
```

- Важно: елементите в списъка имат ред! (първи, втори и т.н.)
- Някои хора ги наричат: "масиви"

Списъци - особености

- Дължина:

```
[1, 2, 3].length // 3
```

- Достъп до елемент

```
["hello", "world"][0] // "hello"
```

- Индекс (позицията) на определен елемент:

```
["hello", “world"].indexOf("world") // 1
```

- *Първият елемент на string или на масив, е винаги с индекс 0!*

Обекти

- Обекта е колекция от именувани стойности
- За разлика от списъците, елементите в обекта са именувани ("ключ/стойност") и нямат ред
- Синтаксис (елементите се изреждат между къдрави скоби и разделени със запетайка):

```
{ first_name: "Chuck", last_name: "Norris", age: Infinity }
```

- Отделните елементи се наричат полета или свойства (properties)
- Реално списъците също са обекти: `typeof([1, 2]) // 'object'`

Обекти - особености

- Празен обект: `var obj = {}`

- Задаване на нова двойка "ключ/стойност" за обекта:

```
obj.key = "value";
console.log(obj) // { key: 'value' }
```

- Достъп до стойност:

```
obj.key // "value"
obj["key"] // "value"
```

- Проверка дали има даден ключ:

```
obj.hasOwnProperty("key") // true
obj.hasOwnProperty("baba") // false
```

42

Number

"Yesterday, all my troubles seemed so far away .."

String

```
{
  race:   "minion",
  colour: "yellow",
  height: "30cm",
  iq:      3
}
```

Object

Променливи

- Променливите са именувани *указатели*, които използваме, за да съхраняваме в тях данни

- Както се подразбира от името, те могат да имат най-различни стойности, които не са постоянни

- Представяйте си ги като имена или етикети на дадени стойности/обекти

- Синтаксиса за създаване на променлива е следният:

```
var pi = 3.14;
```

- Може и така:

```
var myVar;
```

Именуване

Приета е следната конвенция за именуване на променливи в JavaScript:

1. само латински букви + цифри + "_"

2. имената да са максимално екплицитни (описателни)

3. използват се думи от английския език

4. когато името на променливата се състои от няколко думи, то ги изписваме "слято", като първата дума започва с малка буква, а всяка следваща с главна (CamelCase):

   var myAwesomeVar = 1;

Undefined

- Когато нещо няма стойност:

```
var myVar;     // undefined
[1, 2, 3][4]; // undefined
```

- Дефинирани се водят променливите, на които сме задали конкретна стойност

- Грешката:
  `ReferenceError: variable is not defined`
  се получава, когато променливата ***`variable`*** не е била създадена със специялната дума: **var**

Логически и аритметични операции

- Аритметичните операции са тези, които се извършват върху числови данни и резултатът от тях е пак число (+, -, *, /, % и т.н.)
- Логическите операции се използват за сравнение на данни или за формиране на условие и те могат да се използват за всякакви видове данни. При тях резултата е **true** или **false**
- Логически оператори: <, >, ==, !=, ===, !==, !, &&, || и т.н.
- Логически изрази:
  - `(2 > 1) && (3< 5)  // true`
  - `(age > 18) || (parent !== undefined)`

Логически сравнения и оператори

> , >=	"по-голямо", "по-голямо или равно" (за числа)
<, <=	"по-малко", "по-малко или равно" (за числа)
&&	логическо "и"
||	логическо "или"
!	логическо отрицание

Логически сравнения и оператори

==	сравнение за "еднаквост" по стойност
===	сравнение за "еднаквост" по стойност и по тип
!=	сравнение за "различност" по стойност
!==	сравнение за "различност" по стойност и по тип

http://www.w3schools.com/js/js_comparisons.asp

Въпроси?


KEEP
CALM
AND
LEARN
JAVASCRIPT

Полезни връзки

- Codecademy: https://www.codecademy.com/learn/javascript
- Интерактивен онлайн интерпретатор: https://repl.it/languages/javascript
- Загубалка на време: https://blockly-demo.appspot.com/static/demos/interpreter/index.html

Примери

http://swift-academy.zenlabs.pro/lessons/lesson12/examples/download.zip

https://repl.it/ETeK/0

https://repl.it/ETdb/2

Домашно

http://swift-academy.zenlabs.pro/lessons/lesson12/homework