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

(1) напишете `2+2` и натиснете "`Enter`"

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

(2) Може да сравнява

(1) опитайте: `1 < 2`

(2) също и: `3 > 2 + 1`

(3) а сега: `3 > (2 + 1)`

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

Аритметичните операции

(1) Събиране: `2+2 // => 4`

(2) Изваждане: `5-3 // => 2`

(3) Умножение: `4*5 // => 20`

(4) Деление: `20/5 // => 4`

(5) Приоритет (скоби):

```
(2+3)*4 // => 20 (което е различно от 2+3*4)
```

(6) Остатък при деление (деление "по модул"):

```
11%2 // => 1
```

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
```

Задачи

(1) Сметнете колко е 50 на квадрат

(2) Сметнете колко е 51 на трета степен

(3) Сметнете колко е периметърът на кръг с радиус 8

Задачи

(1) Сметнете колко е 50 на квадрат

(2) Сметнете колко е 51 на трета степен

(3) Сметнете колко е периметърът на кръг с радиус 8

Решения:

(1) `50*50 // => 2500`

(2) `51*51*51 // => 132651`

(3) Формулата за радиус на окръжност е `2*π*r`*. Задачата може да бъде решена по три начина:*

3.1) `2*3.14*8 // => 50.24`

3.2) `2*(22/7)*8 // => 50.285714285714285`

3.3) `2*Math.PI*8 // => 50.26548245743669`

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 4

(1) Решете отново задача 3, но този път използвайте променливи: pi и r

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

(3) Накрая, накарайте конзолата да изпише следното:

`"Периметърът на кръг с радиус XXX е: YYY"`

където на мястото на XXX трябва да излиза стойността на променливата r, а на мястото на YYY - стойността на result

Решение

```
var r = 8;
var pi = Math.PI;

var result = 2*pi*r;

console.log(
"Периметърът на кръг с радиус " + r + " е: " +
result);
```

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 5

(1) Като използвате решението на задача 4, направете функция, която да смята периметъра на кръг

- ✳ 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 за радиус, за да се уверите, че наистина работи

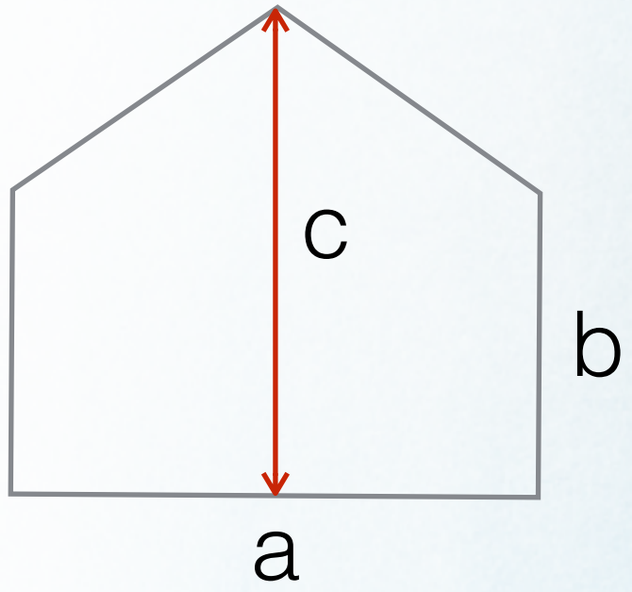
★ *Изпълняването на функцията се нарича още: "извикване"*

★ *Променливите, които стоят в скобите след името на функцията се наричат: "параметри" на функцията*

★ *Кодът, който стои в кърдавите скоби, се нарича: "тяло"*

Задача за домашно

(1) Напишете функция ***area()***, която получава три аргумента (a, b и c) и смята площта на следната фигура:



(2) Функцията трябва да връща правилният резултат при всяко извикване.

(3) Примери:

```
(1)area(10, 6, 10); // => 80
(2)area(2, 1, 3); // => 4
(3)area(10, 6, 10); // => 80
```

*Формулата за площ на правоъгълник е едната страна по другата (a*b), а на триъгълник е едната страна по височината, делено на 2*(a*(c-b)/2)*

Връзка с HTML-а

- За да включим JavaScript кода, който пишем в нашият HTML, трябва да изпълним следните стъпки:
  - да запазим JS кода във файл с разширение **.js**, който сме създали в нашият HTML проект (в същата папка)
  - да свържем този файл посредством **script** таг-а:

```
<script src="script.js"></script>
```

  - Да изведем резултатите от нашите изчисления директно в HTML-а, посредством **document.wite()** или друга функция:

```
document.write("The result is:", result);
```

ES6

EcmaScript е спецификация за езика JavaScript:

> *ECMAScript (or ES) is a trademarked scripting-language specification standardized by the European Computer Manufacturers Association in ECMA-262 and ISO/IEC 16262. It was created to standardize JavaScript, so as to foster multiple independent implementations.*

Текущата версия, която се ползва масово е ES5 от 2009г.

През 2015г. беше публикуван нов стандарт: ES6, който все още не се поддържа от всички браузъри, тъй като стандарта представи няколко генерални промени и нововъведения в езика.

http://kangax.github.io/compat-table/es6/
http://caniuse.com/#search=es6

The arrow function

В **EC6** 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 *и това не е част от ES6*, а си е едно от най-фундаменталните неща в JS.

Сравняване на стойности

- Както вече видяхме, можем да сравняваме 2 числа, за да видим кое е по-голямо и кое по-малко (за целта използваме ***операторите*** за сравнение: **>** и **<**)

- Можем също така да сравним дали две числа са равни или не:

```
(2 + 3) == 5 // => true
2 == 3       // => false
2 != 1       // => true
```

Условен преход (if - else конструкция)

- Често искаме да проверим някакво условие и ако то е вярно да направим едно действие, а ако не е вярно - друго
- За целта има конструкция, която ни позволява да направим точно това
- Синтаксис:

```
if (условие) {
   // действия
} else {
   // действия
}
```

Задача 6

(1) напишете функция calc(), която получава един аргумент - някакво число

(2) функцията трябва да може да извърши следното изчисление: ако подаденият аргумент е четно число, връща квадрата на това число; ако пък е нечетно - връща числото на трета степен

Задача 6

(1) напишете функция calc(), която получава един аргумент - някакво число

(2) функцията трябва да може да извърши следното изчисление: ако подаденият аргумент е четно число, връща квадрата на това число; ако пък е нечетно - връща числото на трета степен

★ *Решение:*

```
function calc(x) {
  if (x%2 == 0) {
    return x*x;
  } else {
    return x*x*x;
  }
}
```

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
```

Текст (String)

- Всеки набор от символи, заграден в двойни или единични кавички е текст (наричан още текстов низ или String)
  - "hello"
  - "1"
  - "Some very long text inputs are also strings"
  - празен стринг: ""
  - прието е да се използват двойни кавички, а не единични

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
- Получават при изпълнение на логически операции и се използват в конструкциите за условен преход
- 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):

```
["text", 1, true]  // => ["text", 1, true]
```

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

Списъци - особености

- Дължина

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

- Добавяне на елемент

```
var array = [1, 2, 3]; array.push(4) // [1, 2, 3, 4]
```

- 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 тип: **ключ: стойност**
- За разлика от списъците, елементите в обекта са именувани посредством тяхния ключ 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