УЕБ РАЗРАБОТКА

JS функции

Съдържание

- Деклариране и създаване на функции в JS
- Извикване на функции
- Функции с параметри
- return
- Scope & context
- Overloading
- Влагане на функции

Деклариране и създаване на функции в JS

Деклариране и създаване на функции

- Какво представляват функциите?

- Защо използваме функции? –
  - *Избройте поне 5 причини да използваме функции.*

Деклариране и създаване на функции

- Разрешаваме големи проблеми като ги разделяме на по-малки
- По-добра организация на програмата
- Кодът се чете по-лесно и е по-разбираем
- Избягваме повторения
- Подобряваме поддръжката на кода – променяме само на едно място.
- Преизползване на кода – използваме функциите много пъти.

Деклариране и създаване на функции

- Всяка функция има име, което се използва за да я извикваме и тяло – body
- Името трябва да описва нейното предназначение.

```
function myFunctionName() {

        //body code goes here

        }
```

Деклариране и създаване на функции

- Два от начините да декларираме функции в JS –
  - Декларация на функция – function declaration

```
function print() { console.log('Hello') };
```

  - Израз – function expression – без име на функцията

```
var print = function() { console.log('Hello') };
```

с име на функцията

```
var print = function printFunc() { console.log('Hello') };
```

Деклариране и създаване на функции

- Извикване на функцията
  - Името на функцията и ();

  print();
- Функцията може да бъде извиквана от друга функция

```
function anotherPrint(){
    print();
    }
```

Demo 1

Функции с параметри

Функции с параметри

- Подаваме данни на функцията чрез параметри /аргументи/
  - Може да подадем 0 или няколко стойности
  - Всеки параметър, който подаваме има име
  - Функцията може да има толкова параметри, колкото са й необходими*
  - Всеки параметър, при извикване на функцията има конкретна стойност
  - Подаваните параметри, могат да бъдат от всякакъв тип – число, стринг, обект, масив, дори функция
- В зависимост от подаваните стойности, параметрите могат да променят поведението на функцията.

Demo 2

return

return

Всяка функция в JS връща стойност

- може изрично да посочим какво да върне функцията* - използваме **return**

- в противен случай – функцията връща **undefined**

***стринг, масив … всякакъв тип

Дефиниране на функции, които връщат някаква стойност

Demo 3

return

RETURN

- Веднага прекратява изпълнението на функцията
- Връща конкретен израз/стойност при извикване на функцията
- Може да бъде използвано няколко пъти в една функция, за да върне различна стойност/резултат при различни случаи.

- **Demo 4**

Scope & context

Scope & context

Всяка променлива има свой scope

Той определя къде е достъпна/живее тази променлива

Съществува глобален и локален scope.

Demo 5

Глобалните променливи – тези, които са декларирани без var са различни от променливите, декларирани в глобалния scope.

Scope & context

Контекст на функция *или как функциите виждат променливите, декларирани в скоупа, в който е декларирана функцията.*

Demo *function_context*

IIFE / как да скрием променливи

IIFE

IIFE – immediately invoked function expression

Създава се анонимна функция, която се извиква незабавно.

JS блок код, който скрива променливите, които са декларирани в него.

Предотвратява замърсяването на глобалния scope/енкапсулация.

```
(function() {
        //block of code here
})();
```

Demo 7, demo IIFE

IIFE

Основно предназначение –

създават scope/на функцията/

избягва се проблем с именуването на променливи и функции

Demo 8

Overloading

Overloading

- JavaScript не поддържа overloading – само една функция с дадено име може да съществува в един и същ scope.

Demo 81

Влагане на функции

Влагане на функции

- Възможно е влагането на функции една в друга
- Променливите, дефинирани в родителската функция са видими за вътрешната.
- Вътрешната функция не е видима извън родителския scope.

- **Demo 9**