Slovenska univerza za tretje življenjsko obdobje

Sofinancira program
Evropske unije
Erasmus+

JavaScript za uporabo na spletnih straneh – funkcije in objekti

Funkcije

- Funkcije so bloki kode, namenjeni izvedbi določene naloge.

- Funkcija se izvede, ko jo prikličemo.

- Primer zapisa funkcije:

```
function myFunction(p1, p2) {
return p1 * p2;              // funkcija izpiše rezultat množenja
}
```

- Na naslednji strani si poglejmo kako to funkcijo prikličemo:

```
<!DOCTYPE html>
<html>
<body>
	<h2>Funkcije v JavaScriptu</h2>
	<p>Ta primer prikliče funkcijo, ki izvede računsko operacijo in izpiše rezultat.:</p>
	<p id="demo"></p>

	<script>
		function mojaFunkcija(p1, p2) {
		 return p1 * p2;
	}
		document.getElementById("demo").innerHTML = mojaFunkcija(4, 3);
	</script>
</body>
</html>
```

Sintaksa funkcije

- Funkcijo definiramo s ključno besedo function, ki ji sledi ime (ki ga sami določimo) in nato oklepaj in zaklepaj.
- Za imena funkcij velja enako pravilo kot za spremenljivke.
- V oklepaja zapišemo imena parametrov, ki jih ločimo z vejicami. Vrednost parametrov določimo, ko funkcijo prikličemo in tem vrednostim pravimo argumenti.
- Kodo, ki naj jo funkcija izvede, ko je priklicana, zapišemo v zavite oklepaje:

```
function ime(parameter1, parameter2, parameter3) {
        koda, ki naj se izvede
}
```

Priklic funkcije

- Funkcija se izvede, ko se nekaj zgodi (ko npr. uporabnik klikne na gumb),
- ko je priklicana ali
- avtomatično (sama se prikliče)

Funkcija return (slov. vrni)

- Trditev return, da kodi navodilo, naj se preneha izvajati.
- Če je bila funkcija priklicana s trditvijo, potem bo funkcija return povzročila, da se koda izvaja naprej od trditve, ki je priklicala funkcijo.
- Funkcije pogosto vrnejo (angl. return) vrednost. Vrnjena vrednost je vrnjena klicalcu.
- Primer:

```
var x = mojaFunkcija(4, 3);    // Funkcija je priklicana, vrnjena vrednost
                                   bo prikazana kot vrednost x


function mojaFunkcija(a, b) {
return a * b;                  // Funkcija vrne izračun vrednosti a in b
}
```

Rezultat spremenljivke x bo 12.

Zakaj funkcije?

- Kodo lahko ponovno uporabimo: napišemo jo enkrat in jo potem znova in znova uporabimo.
- Isto kodo lahko uporabimo z različnimi argumenti, da dobimo različne rezultate.

- Primer za pretvorbo Fahrenheitov v stopinje Celzija:

```
function toCelsius(fahrenheit) {
return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

```
<!DOCTYPE html>
<html>
<body>
	<h2>Funkcije v JavaScriptu</h2>
	<p>Ta primer prikliče funkcijo, da pretvori Fahrenheite v stopnije Celzija:</p>
	<p id="demo"></p>
	<script>
		function toCelsius(f) {
		 return (5/9) * (f-32);
	}
		document.getElementById("demo").innerHTML = toCelsius(77);
	</script>
</body>
</html>
```

- Znotraj oklepajev, ko prikličemo funkcijo, argumentiramo vrednost. Če te vrednosti ne argumentiramo, bomo dobili nazaj zapis funkcije, namesto njen rezultat:
- Priklic funkcije:

```
function toCelsius(fahrenheit) {
 return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

- Rezultat:

function toCelsius(f) { return (5/9) * (f-32); }, namesto 25.

Funkcije lahko uporabimo kot vrednosti spremenljivk

- Funkcije lahko uporabljamo na mnoge načine, v vseh možnih formulah, nalogah in računskih operacijah.
- Namesto, da vrnjeno vrednost pripišemo spremenljivki (primer a), lahko funkcijo uporabimo direktno, kot vrednost spremenljivke (primer b):
- Primer a:
- var x = toCelsius(77);
  var besedilo = „Temperatura je " + x + " Celsius";
- Primer b:
- var besedilo = „Temperatura je " + toCelsius(77) + " Celsius";

Lokalne spremenljivke

- Spremenljivke, ki jih deklariramo znotraj funkcije, postanejo lokalne spremenljivke, kar pomeni, da do njih lahko dostopamo le znotraj funkcije.
- Primer:

```
// tukaj koda ne more dostopati do spremenljivke avto

function mojaFunkcija() {
var avto = „Fičo";
// tukaj koda lahko dostopa do spremenljivke avto
}

// tukaj koda ne more dostopati do spremenljivke avto
```

Lokalne spremenljivke

- Lokalne spremenljivke so prepoznane le znotraj funkcije, v kateri so bile deklarirane, zato lahko ista imena uporabimo znotraj drugih funkcij.

- Lokalne variable so ustvarjene, ko se funkcija začne izvajati in so izbrisane, ko je funkcija izvedena.

Objekti

- V resničnem življenju, je avto objekt.
- Avto ima lastnosti, kot so teža in barva in metode, kot sta spelji in ustavi.
- Vsi avtomobili imajo iste vrste lastnosti, vendar se vrednosti teh lastnosti od avta do avta razlikujejo.
- Vsi avtomobili uporabljajo iste metode, vendar jih ne izvajajo nujno ob istem času.

Objekt	Lastnosti	Metode
	avto.ime = Fiat avto.model = 500 avto.teza = 850kg avto.barva = bela	car.start() car.drive() car.brake() car.stop()

Objekti

- Objekti so tudi spremenljivke. Toda ti objekti lahko vsebujejo več vrednosti ali lastnosti.
- Primer:

```
var avto = {tip:"Fiat", model:"500", barva:„bela"};
```

- Zgornjo trditev lahko zapišemo tudi sledeče:

```
var avto = {
        tip:"Fiat",
        model:"500",
        barva:„bela„
};
```

Lastnosti objektov

- Lastnost objekta je ime:vrednost, ki smo jo definirali znotraj spremenljivke:

```
var oseba = {ime:„Neznana", priimek:„oseba", starost:50,
barva_oci:„modra"};
```

- Do vrednosti lastnosti objektov dostopamo na dva načina:
  - imeobjekta.imelastnosti
  - imeobjekta[„imelastnosti“]

- Primer:
- oseba.ime
- oseba[„ime“]

Metode objektov

- Metode objektov so dejanja, ki jih lahko izvajamo na objektu.
- Metode so shranjene v lastnostih kot definicije funkcij, ali drugače povedano, metoda je funkcija, ki je shranjena kot lastnost.

Lastnost	Vrednost lastnosti
ime	Neznana
Priimek	oseba
starost	50
barva_oci	modra
polnoIme	function() {return this.ime + " " + this.priimek;}

Ključna beseda this

- Po definiciji se **this** nanaša na **lastnika** funkcije.
- V prejšnjem primeru se **this** nanaša na objekt **oseba**.
- Objekt **oseba** je lastnik metode **polnoIme**.
- Primer uporabe this metode:
- var **oseba** = {
  ime: „Neznana",
  priimek : „oseba",
  id : 5566,
  polnoIme : function() {
  return **this**.ime + " " + **this**.priimek;
  }
  };

```
<!DOCTYPE html>
<html>
<body>
        <h2>Ključna beseda <b>this</b></h2>
        <p>V tem primeru <b>this</b>predstavlja objekt<b>oseba</b>.</p>
        <p>Ker je objekt oseba lastnik metode polnoIme.</p>
        <p id="demo"></p>
        <script>
                // Ustvari objekt:
                var oseba = {
                 ime: „Neznana",
                 priimek : „oseba",
                 id      : 5566,
                 polnoIme : function() {
                                return this.ime + " " + this.priimek;
                                 }
                };
                // Prikaži podatke objekta
                document.getElementById("demo").innerHTML = person.polnoIme();
        </script>
</body>
</html>
```

- Do metode objekta dostopamo z naslednjo sintakso:

imeobjekta.imemetode()

- Primer:
- Ime = oseba.polnoIme();

- Če dostopamo do metode brez oklepajev (ime = oseba.polnoIme), nam ta vrne definicijo funkcije namesto rezultata.

Povezovanje funkcij

- Metodi **call()** in **appy()** sta vnaprej definirani metodi JavaScripta.
- Obe lahko uporabimo za priklic metode objekta, pri kateri za argument uporabimo drug objekt.
- V naslednjem primeru bomo priklicali metodo objekta oseba1, pri kateri bomo za argument uporabili objekt oseba2. Funkcija **this** se bo nanašala na objekt oseba2, četudi je metoda pripisana objektu oseba1:

Povezovanje funkcij

```
var oseba1 = {
    polnoIme: function() {
        return this.ime + " " + this.priimek;
    }
}
var oseba2 = {
    ime:„Neznana",
    priimek: „oseba",
}
person1.polnoIme.call(oseba2);  // Rezultat bo „Neznana oseba"
```

Povezovanje funkcij

```
var oseba1 = {
    polnoIme : function() {
        return this.ime + " " + this.priimek;
    }
}
var oseba2 = {
    ime:„Neznana",
    priimek: „oseba",
}
person1.polnoIme.call(oseba2);  // Rezultat bo „Neznana oseba"
```