JavaScript - uvod

Adaptirana verzija slajdova.
Zahvaljujemo koleginici Viktoriji Kirst

Plan rada

U ovoj lekciji:

- Uvod u jezik JavaScript
  - Pregled mogućnosti jezika
  - Osnovi obrade događaja (Basic event handling)

Promjenljive: `var`, `let`, `const`

Deklarisanje promjenljive pomoću jedne od tri ključne riječi :

```
// Function scope variable
   var x = 15;
// Block scope variable
   let fruit = 'banana';
// Block scope constant; cannot be reassigned
   const isHungry = true;
```

Nije potrebno navesti tip promjenljive prije njene upotrebe ("dynamically typed")

Parametri funkcija

```
function printMessage(message, times) {
  for (var i = 0; i < times; i++) {
    console.log(message);
  }
}
```

Parametri funkcija se ne deklarišu pomoću `var`, `let` ili `const`

Razumijevanje var

```
function printMessage(message, times) {
  for (var i = 0; i < times; i++) {
    console.log(message);
  }
  console.log('Value of i is ' + i);
}
printMessage('hello', 3);
```

Q: Što se dešava ako pokušamo da štampamo "i" na kraju ciklusa?

Razumijevanje var

```
function printMessage(message, times) {
  for (var i = 0; i < times; i++) {
    console.log(message);
  }
  console.log('Value of i is ' + i);
}
printMessage('hello', 3);
```

```
3 hello
  Value of i is 3
>
```

Vrijednost "i" je čitljiva i van for-ciklusa jer promjenljive deklarisane sa var imaju opseg važenja funkcije.

Opseg funkcije sa var

```
var x = 10;
if (x > 0) {
  var y = 10;
}
console.log('Value of y is ' + y);
```

```
Value of y is 10
>
```

- Opseg važenja promjenljiveih deklarisanih sa "var" je funkcija i one ne izlaze iz opsega sve do kraja funkcije
- Dakle, možete referencirati promjenljivu i poslije bloka (npr. poslije bloka ili if-naredbe u kojoj je deklarisana)

Opseg funkcije sa var

```
function meaningless() {
  var x = 10;
  if (x > 0) {
    var y = 10;
  }
  console.log('y is ' + y);
}
meaningless();
console.log('y is ' + y); // error!
```

```
y is 10
▶Uncaught ReferenceError: y is not defined
    at script.js:9
```

Ne možemo pristupiti promjenljivoj van funkcije u kojoj je deklarisana.

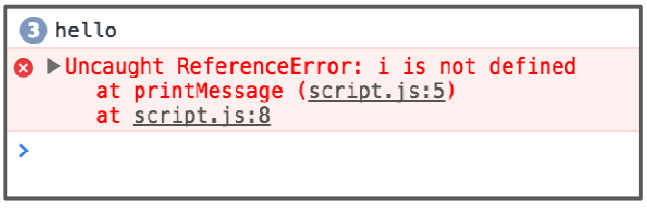
Razumijevanje `let`

```
function printMessage(message, times) {
  for (let i = 0; i < times; i++) {
    console.log(message);
  }
  console.log('Value of i is ' + i);
}
printMessage('hello', 3);
```

Q: Što ako štampamo "i" na kraju ciklusa?

Razumijevanje `let`

```
function printMessage(message, times) {
  for (let i = 0; i < times; i++) {
    console.log(message);
  }
  console.log('Value of i is ' + i);
}
printMessage('hello', 3);
```



Opseg važenja `let` je blok, pa je ovo greška

Razumijevanje const

```
let x = 10;
if (x > 0) {
  const y = 10;
}
console.log(y);  // error!
```

Kao i za let, const takođe ime opseg važenja nivo bloka, pa dobijamo grešku

Razumijevanje const

```
const y = 10;
y = 0;          // error!
y++;            // error!
const list = [1, 2, 3];
list.push(4);   // OK
```

Promjenljive deklarisane sa const ne mogu dobiti novu vrijednost (cannot be reassigned).

Međutim, nije implementirano pravo const ponašanje, pa se može promijeniti objekat

- (ponaša se kao Java's final a ne kao C++'s const)

Poređenje sa `let`

```
let y = 10;
y = 0;          // OK
y++;            // OK
let list = [1, 2, 3];
list.push(4);   // OK
```

`let` dopušta dodjeljivanje nove vrijednosti, što je razlika između `const` i `let`

Što je blok?

Blok je grupa 0 ili više naredbi (obično unutar velikih zagrada). (wiki / mdn)

- Drugi naziv je složena naredna(**compound statement**)
- Ne samo u jeziku JavaScript – postoji i u većini drugij jezika (C++, Java, Python, itd)
- Nema nikave veze sa HTML/CSS pojmom "blok", tj. blok elementima

Što je blok?

Definicija naredbe if:

```
if (expression) statement
```

A blok može biti

```
{
  console.log('Hello, world!');
  console.log('Today is a good day.');
}
```

"Block" ili složena naredba je tip naredbe, pa možemo izvršiti više naredbi ako je uslov tačan.

Blokovi i opsezi

U većini jezika, svaki blok ima svoj opseg važenja tzv. "block scope":

```
// C++ code, not JS:
if (…) {
   int x = 5;
        ...
}
// can't access x here
```

Ovako se ponašaju Java, C++, C, itd.

Blokovi i opsezi

Na sisti način se ponaša i JavaScript, ako se pri deklaraciji promjenljivih koriste `const` i `let`:

```
if (…) {
    let x = 5;
          ...
}
// can't access x here
```

Blokovi i opsezi

Ali kao koristimo `var`, promjenljiva postoji na nivou funckije i potpuno je nezavisna od bloka ukojem je deklarisana :

```
if (…) {
    var x = 5;
          ...
}
// x is 5 here
```

Isto ponašanje kao u jeziku Python, koji takože ima opseg važenja na novou funckije (function scope).

* Hoisting i function scope nisu isto.

Promjenljive – dobra praksa

- Koristite `const` kad je god moguće.
- Ako želite nova dodjeljivanja, koristite `let`.
- **Ne koristite `var`.**
  - Na internetu ima mnogi primjera sa `var` jer su `const` i `let` relativno novi.
  - Međutim, `const` i `let` imaju dobru podršku (well-supported), pa nema razloga da ih ne koristite.

(Preporuke Google i AirBnB o stilu kodiranja JavaScript programa.)

Promjenljive – dobra praksa

- Koristite `const` kad je god moguće.
- Ako želite nova dodjeljivanja, koristite `let`.
- **Ne koristite `var`.**
  - Na internetu ima mnogi primjera sa `var` jer su `const` i `let` relativno novi.

(Pr

Napomena: Internet sadrži tone pogrešnih informacija o jeziku JavaScript!

Uključujući nekoliko "accepted" odgovora na StackOverflow, tutorijala, itd. Dosta materijala je zastarjelo.
Budite pažljivi.

Tipovi

JS **promjenljive** nemaju tipove, ali **vrijednosti** ih imaju.
Postoji šest primitivnih tipova (mdn):

- Boolean : `true` i `false`
- Number : svi su tipa double (nema integer-a)
- String: `u'single'` ili `"double-quotes"`
- Symbol: Novi tip za simbole
- Null: `null`: vrijednost koja znači "this has no value"
- Undefined: vrijednost promjenljive bez dodjele

Postoji i tip Object, uključuje Array, Date, String (object wrapper za primitivni tip), itd.

Brojevi

```
const homework = 0.45;
const midterm = 0.2;
const final = 0.35;
const score =
    homework * 87 + midterm * 90 + final * 95;
console.log(score);     // 90.4
```

Brojevi

```
const homework = 0.45;
const midterm = 0.2;
const final = 0.35;
const score =
    homework * 87 + midterm * 90 + final * 95;
console.log(score);     // 90.4
```

- Svi brojevi su floating point realni brojevi.
- Operatori su kao u jezicima Java ili C++.
- Prioritet je kao u jezicima Java ili C++.
- Specijalne vrijednosti: NaN (not-a-number), +Infinity, -Infinity
- Postoji klasa Math: `Math.floor`, `Math.ceil`, itd.

Stringovi

```
let snack = 'coo';
snack += 'kies';
snack = snack.toUpperCase();
console.log("I want " + snack);
```

Stringovi

```
let snack = 'coo';
snack += 'kies';
snack = snack.toUpperCase();
console.log("I want " + snack);
```

- Mogu se koristiti jednostruki ili dvostruki navodnici
  - Mnogi style guides preferiraju jednostruke, iako nema razlike
- Ne mogu se mijenjati (immutable)
- Nema tipa `char`: slova se stringovi dužine 1
- Nadovezivanje se može uraditi sa simbolom +
- Veličina stringa: svojstvo `length` (nije funkcija)

Boolean

- Postoje dva literala za boolean: `true` i `false`
- Standardni logički operatori: `&& || !`

```
let isHungry = true;
let isTeenager = age > 12 && age < 20;

if (isHungry && isTeenager) {
  pizza++;
}
```

Boolean

- Non-boolean vrijednosti se mogu koristiti u kontrolnim naredbama i tada se konvertuju u njihove "truthy" ili "falsy" verzije:
  - `null`, `undefined`, `0`, `NaN`, `''`, `""` se prevode u `false`
  - Sve ostale se pretvara u `true`

```
if (username) {
  // username is defined
}
```

Equality

JavaScript == i != u suštini ne rade dobro: odrađuje se implicitna konverzija tipa prije poređenja.

```
'' == '0'  // false
 '' == 0  // true
 0 == '0'  // true
   NaN == NaN  // false
 [''] == ''  // true
false == undefined  // false
  false == null  // false
null == undefined  // true
```

Equality

Umjesto promjene ponašanja operatora == i != , ECMA Script standard je zadržao ponašanje i dodao === i !==

```
'' === '0'  // false
 '' === 0  // false
0 === '0'  // false
NaN == NaN  // still weirdly false
 [''] === ''  // false
false === undefined  // false
  false === null  // false
null === undefined  // false
```

Poželjno je koristiti === i !== a ne == ili !=

Null i Undefined

Koja je razlika?

- `null` predstavlja nedostatak vrijednosti, slično null u Java i nullptr u C++.
- `undefined` je vrijednost koju ima promjenljiva kojoj nije dodijeljena vrijednost.

```
let x = null;
let y;
console.log(x);
console.log(y);
```

```
null
undefined
>
```

Null i Undefined

Koja je razlika?

- `null` predstavlja nedostatak vrijednosti, slično null u Java i nullptr u C++.
- `undefined` je vrijednost koju ima promjenljiva kojoj nije dodijeljena vrijednost.
  - … ali, može se postaviti da vrijednosti promjenljive bude undefined

```
let x = null;
let y = undefined;
console.log(x);
console.log(y);
```

```
null
undefined
>
```

Nizovi

Nizovi imaju tip Object – kreira se lista podtaka.

```
// Creates an empty list
let list = [];
let groceries = ['milk', 'cocoa puffs'];
groceries[1] = 'kix';
```

- Indeksi počinju od 0 (0-based indexing)
- Mogu se mijenjati (Mutable)
- Broj elemenata niza: svojstvo `length` (nije funkcija)

Ciklusi i nizovi

Tradicionalni for-ciklus za prolazak kroz listu:

```
let groceries = ['milk', 'cocoa puffs', 'tea'];
for (let i = 0; i < groceries.length; i++) {
  console.log(groceries[i]);
}
```

Ili koristite for-each ciklus `for...of` (mdn):
(intuicija: za svaki objekat iz liste za kupovinu - groceries list)

```
for (let item of groceries) {
      console.log(item);
}
```

Operacije sa listom

Metod	Opis
`list.push(`*element*`)`	Add ***element*** to back
`list.unshift(`*element*`)`	Add ***element*** to front

Metod	Opis
`list.pop()`	Remove from back
`list.shift()`	Remove from front

Metod	Opis
`list.indexOf(`*element*`)`	Returns numeric index for ***element*** or -1 if none found

splice

Dodaje/uklanja element na poziciji: splice
`list.splice(`***startIndex***`, `***deleteCount***`, `***item1***`, `***item2***`, ...)`

Uklanja jedan element sa indeksom 3:
`list.splice(3, 1);`

Dodaje ***element*** na poziciju 2:
`list.splice(2, 0, `***element***`);`

Map sa objektima

- Svaki JavaScript objekat ije skupa parova oblika svojstvo-vrijednost (property-value pairs).
- Možemo definisati map kreiranjem objekta:

```
// Creates an empty object
const prices = {};
const scores = {
  'peach': 100,
  'mario': 88,
  'luigi': 91
};
console.log(scores['peach']);   // 100
```

Map sa objektima

Ključevi tipa string ne moraju imati navodnike.
Bez navodnika, ključevi su i dalje tipa string.

```
// This is the same as the previous slide.
const scores = {
  peach: 100,
  mario: 88,
  luigi: 91
};
console.log(scores['peach']);   // 100
```

Map sa objektima

Postoje dva načina da se dobije vrijednost nekog svojstva :

1. ***objectName*[*property*]**
2. ***objectName.property***

```
const scores = {
  peach: 100,
  mario: 88,
  luigi: 91
};
console.log(scores['peach']);   // 100
console.log(scores.luigi);      // 91
```

Map sa objektima

Postoje dva načina da se dobije vrijednost nekog svojstva:

1. ***objectName*[*property*]**
2. ***objectName.property***

(2 radi samo ključevima tipa string.)

Preporučuje se upotreba načina (2), osim ako se svojstvo čuva u promjenljivoj ili ako svojstvo nije string.

```
const scores = {
  peach: 100,
  mario: 88,
  luigi: 91
};
console.log(scores['peach']);   // 100
scores.luigi = 87;
console.log(scores.luigi);      // 91
```

Map sa objektima

Dodavanje svojstva objektu: dajte ime svojstvu i zadajte vrijednost:

```
const scores = {
  peach: 100,
  mario: 88,
  luigi: 91
};
scores.toad = 72;
let name = 'wario';
scores[name] = 102;
   console.log(scores);
```

▶ *Object {peach: 100, mario: 88, luigi: 91, toad: 72, wario: 102}*

Map sa objektima

Za uklanjanje svojstva objekta koristite **`delete`**:

```
 const scores = {
  peach: 100,
  mario: 88,
  luigi: 91
};
scores.toad = 72;
let name = 'wario';
scores[name] = 102;
delete scores.peach;
console.log(scores);
```

```
▶ Object {mario: 88, luigi: 91, toad: 72, wario: 102}
```

Iteracija kroz Map

Iteracija kroz map primjenom ciklusa for...in (mdn):
(intuicija: for svaki ključ in object)

```
for (key in object) {
    // … do something with object[key]
}
```

```
for (let name in scores) {
  console.log(name + ' got ' + scores[name]);
}
```

- Ne možete koristiti for...in na listama; samo za objekte
- Ne možete koristiti for...of na objektima; samo za liste