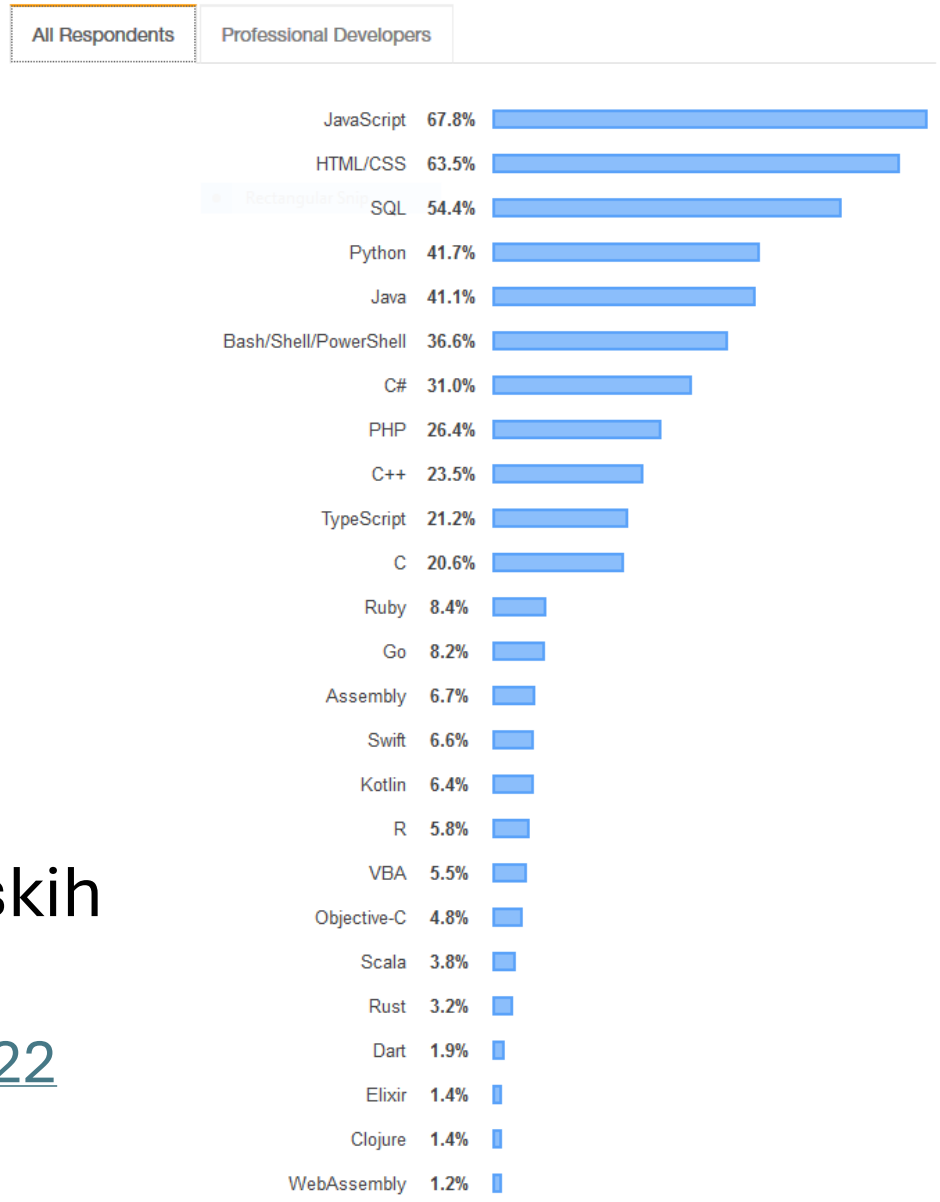
JavaScript

Web programiranje

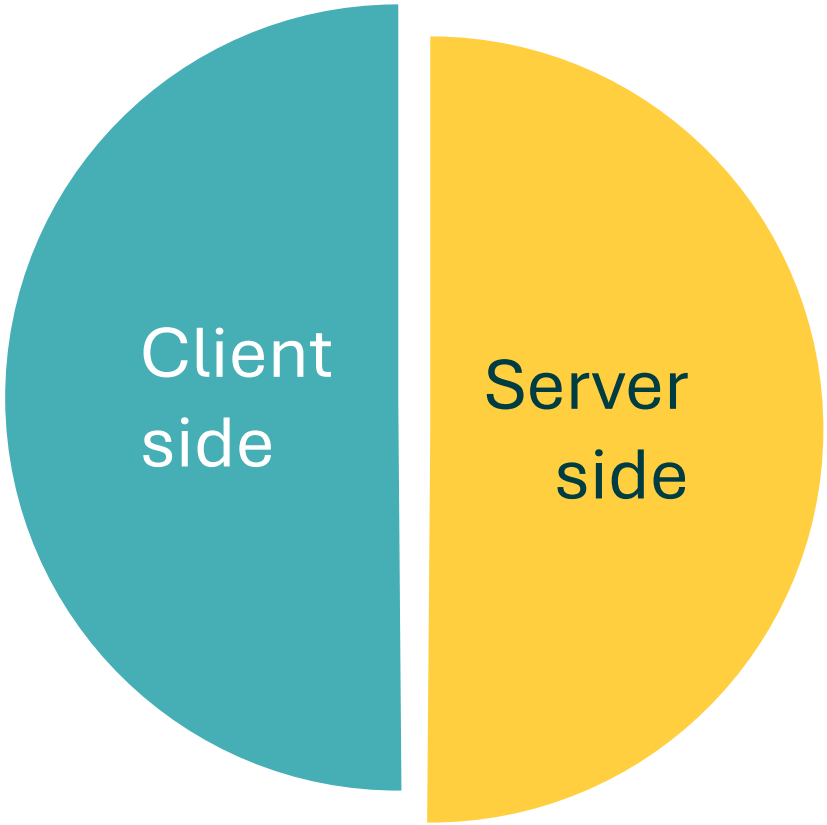
Jurica Maltar

JavaScript



- Brendan Eich, 1995., Netscape
- Jedan od najpopularnijih programskih jezika
  - Stack Overflow Developer Survey 2022
  - Github Language Stats

Gdje se nalazi JavaScript



Uvod

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Document</title>
        <!--JavaScript can be placed here-->
</head>
<body>
        ...

        <!--JavaScript can be placed here-->
</body>
</html>
```

Uvod

- Sam JavaScript kod smješta se unutar `<script>` elementa, ili se atributu `src` pridružuje putanja do vanjske datoteke
  - Ekstenzija vanjske datoteke je `*.js`
  - Gdje može biti smještena vanjska JavaScript datoteka?

- Ispis: `console.log(…)`

- Tri osnovne naredbe koje izbacuju „popup” prozor su:
  1. `alert(…)`
  2. `confirm(…)`
  3. `prompt(…)`

- **Zadatak**: stvorite web stranicu koja pri učitavanju otvara upozorenje s porukom „Ovo je prazna stranica”

- **Zadatak**: Što vraća `confirm`, a što `prompt`?

Uvod

- Komentari: `//` i `/**/`

- Uz `<script>`, JavaScript se može izvršavati i u developer tools konzoli
    - Developer tools (F12)

- Usporedba Pythona i JavaScripta?
- Usporedba C++-a i JavaScripta?

Tipovi

- JavaScript je *dinamički* jezik

```
let foo = 42; // foo is now a number
foo = "bar"; // foo is now a string
foo = true; // foo is now a boolean
```

- JavaScript je *slabo tipizirani (eng. weakly typed)* jezik

```
const foo = 42; // foo is a number
const result = foo + "1";
console.log(result); // 421
```

Tipovi

- Primitivni tipovi

Type	`typeof` return value	Object wrapper
Null	`"object"`	N/A
Undefined	`"undefined"`	N/A
Boolean	`"boolean"`	`Boolean`
Number	`"number"`	`Number`
BigInt	`"bigint"`	`BigInt`
String	`"string"`	`String`
Symbol	`"symbol"`	`Symbol`

- Složeni tipovi – *objekti*

Tipovi

- S `typeof` provjeravamo tip podatka spremljene varijable
- **Zadatak:** Provjerite tip sljedećih podataka u ovisnosti o spremljenoj vrijednosti

```
var x = 1000;
x = 3.14;
x = 4e-2;
x = "Everything's fine";
x = true;
```

- Aritmetički operatori kao i u C-u:

```
y++;
++y;
y *= 1;
y += 2;
...
```

Grananja & petlje

```
var c1 = false;
var c2 = false;

if (c1) {
    console.log("Hi1");
} else if (c2) {
    console.log("Hi2");
} else {
    console.log("Hi3");
}

for (var i = 0; i < 10; ++i) {
    console.log(i);
}
```

Za unutarnji scope s jednim retkom koda, { } nisu potrebne

Napomena: Kao što je sintaksa `for` petlje identična C/C++ jeziku, tako će biti i sa `while` te `do-while` petljom. Ključne riječi `continue` i `break` koriste se na identičan način.

Prijetvorba (cast) tipova

```
parseInt(x); // x to int
parseFloat(x); // x to float
String(x); // x to string
Boolean(x); // x to boolean
```

- **Zadatak**: Isprobajte ove funkcije

Operatori

- Logički operatori `&&`, `||`, `!` identični C-u
- Bitwise operatori `&`, `|`, `^`, `~`, `>>`, `<<` identični C-u
- Uspoređivanje: `==` i `===` (`!=` i `!==`)
- `NaN` nije jednak ničemu

- Dva su objekta (TBA) jednaka ako referiraju na isti podatak

```
let a = { x, y };
let b = { x, y };
console.log(a == b);
```

- `null == undefined` vs `null === undefined`

Globalne i lokalne varijable

- Vrijede slična pravila kao i u drugim programskim jezicima
- Unutar funkcije možemo pristupiti lokalnim varijablama, kao i globalnim definiranim izvan funkcije

```
var a = 5;
var fn = function(b) {
        return a * b;
}
```

- Napomena: Ukoliko ne stavimo `var` ispred imena lokalne varijable, ona postaje globalna

Globalne i lokalne varijable

- Ako umjesto `var` koristimo `let`, prema varijabli se ponašamo kao prema svakoj lokalnoj varijabli u C/C++-u

- Za nepromijenjive veličine umjesto `var` i `let` koristimo `const`

- var vs let

Globalne i lokalne varijable

- `var` vs `let`, scoping rules:

```
var foo = "Foo";
let bar = "Bar";
console.log(foo, bar); // Foo Bar
{
    var moo = "Mooo"
    let baz = "Bazz";
    console.log(moo, baz); // Mooo Bazz
}
console.log(moo); // Mooo
console.log(baz); // ReferenceError
```

Globalne i lokalne varijable

- `var` vs `let`, hoisting:

```
console.log(foo); // undefined
var foo = "Foo";
console.log(foo); // Foo
```

Globalne i lokalne varijable

- `var` vs `let`, nanovo deklariranje:

```
var foo = "foo1";
var foo = "foo2"; // No problem, 'foo1' is replaced with 'foo2'.

let bar = "bar1";
let bar = "bar2"; // SyntaxError: Identifier 'bar' has already been declared
```

Funkcije u JavaScript-u

- Funkcije možemo pisati na različite načine
- “najklasičniji” način:

```
function something() {
// code goes here
}

// call function
something();

//use result
var result = something();
```

- **Zadatak:** Napišite funkciju koja vraća string dohvaćen pomoću `prompt`

Funkcije u JavaScript-u

- Funkcija se može spremiti kao varijablu (function expression)

```
var myFunction = function(a, b) {
        return a * b;
}
var x = myFunction(5, 7);
```

- Ovakav način će omogućiti “lakše” redefiniranje funkcije

```
myFunction = function(a, b) {
        return a * b * 2;
}
```

Funkcije u JavaScript-u

- Sve “obične” (one koje nisu *lambda*) funkcije imaju ugrađeni objekt `arguments` (koji je „polje” TBA) :

```
function findMax() {
        let x = -Infinity;
        for(var i = 0; i < arguments.length; ++i)
                if(arguments[i] > x)
                        x = arguments[i];
        return x;
}
```

- **Zadatak**: Koristeći objekt `arguments`, definirajte metodu `sum` (za proizvoljan broj brojeva)

Pozivanje funkcija

- Funkcije se pozivaju
  - Pri pozivu
  - Automatski! (self-invoked function)

```
(function() {
        console.log("Hello World");
})();
```

  - Pri određenom događaju (npr. klik na gumb, pri unosu, pri prelasku miša...).

Array

- Struktura u JavaScriptu koja nalikuje listi u Pythonu

```
let myArray = [1, 2, 3, 4];
let mySecondArray = [
        "Hi",
        5.6,
        false,
        "Hello World"
];
```

- Uz to što sadrži niz vrijednosti, Array je i objekt s atributima (npr. `length` ili metoda `push`)
    - Isprobajte metodu `push` nad array-om
    - Što ako pristupamo nečemu izvan duljine?

Array

- Neke od metoda: `push`, `pop`, `shift`, `unshift`, `sort`, `reverse`, `indexOf` …
- Za detaljni popis pogledati http://www.w3schools.com/js/js_array_methods.asp
- `delete` operator briše elemente iz array-a (oni postaju `undefined`)
- Klasični prolaz kroz array

```
for(let i = 0; i < mySecondArray.length; ++i)
        console.log(mySecondArray[i]);
```

Array – for/of i for/in petlja

- Elementima array-a možemo prolaziti direktno koristeći for/of i for/in petlju
- Primjer

```
for(let i in mySecondArray)
        console.log(i)

for(let i of mySecondArray)
        console.log(i)
```

Array – još neke funkcije

- `forEach, map, reduce…`

```
arr.forEach(function(element) {
      console.log(element);
});
```

- Za neko polje koje sadrži brojeve, korištenjem `reduce` izračunajte sumu

Lambda funkcije

- Umjesto uobičajne sintakse za definiranje funkcija, u ovakvim slučajevima uglavnom koristimo lambda funkcije.

```
arr.forEach((element) => {
      console.log(element);
});
```

Object

- Objekt je asocijativno polje (rječnik)
- Primjer:

```
let person = { firstName: "Jurica", age: 31 };
```

  - `firstName` i `age` se nazivaju svojstva (properties) objekta
- Objekt možemo stvoriti koristeći i ključnu riječ `new`. Npr.

```
let person = new Object();
person.firstName = "Jurica";
person.age = 31;
console.log(person);

let person = new Object({
        firstName: "Jurica",
        age: 29
});
```

Object

- Možemo iterirati kroz key vrijednosti objekta jednako kao i u slučaju kod array-a

```
for(let i in person)
        console.log(i);
```

- Objekt je dinamička struktura – “u letu” možemo dodavati nova svojstva. Npr.

```
person.lastName = "Maltar";
```

- Ili obrisati neko svojstvo

```
delete(person.age);
```

- **Zadatak**: Uvjerite se da se objekt prosljeđuje po referenci, a string po vrijednosti.
- Prosljeđivanje objekta po vrijednosti/kopiranje: `{...object}`

Object

- Property-iji objekta mogu biti i funkcije

```
var person = new Object({
        firstName: "Jurica",
        age: 29,
        print: function() {
                console.log(this.firstName + ", " + this.age);
        }
});
```

- Pokušajte napraviti `print` kao lambda funkciju
- Napravite funkciju `printScope` koja ispisuje `this`

Object

- Ili generaliziraniji primjer:

```
var objectWithFunctionWithObjectWithoutLambda = {
        value: 255,
        innerScope: function() {
                var object = {
                        value: 256,
                        printValue: function() {
                                console.log(this.value);
                        }
                }
                return object;
        }
}
```

Lambda funkcije

- Sintaksa:

```
(...args) => {
        // nešto
}
```

- Kod lambda funkcija `this` “pokazuje” na djelokrug u kojem je objekt s lambda funkcijom inicijaliziran
- Kod uobičajenih funkcija `this` “pokazuje” na djelokrug samog objekta
- U drugu ruku, lambde ne podržavaju built-in-ani objekt `arguments`
- Ukoliko se ne susrećemo s gore navedenim problemima, možemo koristiti lambde
- Više o lambda funkcijama i ključnoj riječi `this` možete pronaći ovdje i ovdje.

Callback funkcije

- Callback funkcija: funkcija koju prosljeđujemo drugoj funkciji kao argument

- Mogućnosti je mnogo, jedan od tipičnih primjera je prosljeđivanje lambda funkcija funkcijama koje rade nad strukturom array, ili npr. dohvaćanje elemenata iz baze podataka (to ćemo vidjeti jednom).

- **Zadatak:** Implementirajte funkciju koja nalazi najveću vrijednost iz polja i prosljeđuje ju callback funkciji

Callback funkcije

- Još jedan primjer:

```
function encCallb(callb) {
        this.item = "Hello World 1";
        var obj = { item: "Hello World 2", callb }
        obj.callb();
}

encCallb(() => {
        console.log(this.item);
});
```

Class

- Počevši od standarda ECMAScript 2015, u JavaScript-u se pojavljuje klasa
- Sintaksa:

```
class Rectangle {
        constructor(height, width) {
                this.height = height;
                this.width = width;
        }
}

let r = new Rectangle(5, 3.5);
```

Class - konstruktor

- Konstruktor ne možemo preopterećivati

```
// ERROR
class Rectangle {
        constructor(height, width) {
                this.height = height;
                this.width = width;
        }

        constructor() {
                this.height = 0;
                this.width = 0;
        }
}
```

Class – funkcije unutar klase

- Uz članove koji su varijable ili objekti, možemo enkapsulirati i funkcije

```
class Rectangle {
        constructor(height, width) {
                this.height = height;
                this.width = width;
        }

        area() {
                return this.height * this.width;
        }
}
```

Class – funkcije unutar klase

- Dodatno, možemo imati statične funkcije
  - Koja je razlika između statične i obične funkcije?

```
class Rectangle {
        constructor(height, width) {
                this.height = height;
                this.width = width;
        }

        area() {
                return this.height * this.width;
        }
        static equals(r1, r2) {
                if (r1 instanceof Rectangle && r2 instanceof Rectangle)
                        return r1.height == r2.height && r1.width == r2.width;
                else
                        return false;
        }
}
```

Class - nasljeđivanje

- Neku baznu klasu nasljeđujemo u izvedenu klasu koristeći ključnu riječ `extends`
- Konstruktor izvedene klase poziva konstruktor bazne klase u svome djelokrugu

```
class Square extends Rectangle {
        constructor(side) {
                super(side, side);
        }
}
```

- funkcijama bazne klase pristupamo koristeći ključnu riječ `super`

Class – get/set

- Kao i u nekim drugim programskim jezicima, možemo definirati atribute koji nisu varijable ali se tako ponašaju

```
class Square extends Rectangle {
        constructor(side) {
                super(side, side);
        }
        get area() {
                return this.height ** 2;
        }
        set area(a) {
                let aSqrt = Math.sqrt(a);
                this.height = aSqrt;
                this.width = aSqrt;
        }
}
```