JavaScript

SPARK.
SCHOOL

1.JavaScript uvod

JavaScript je jedan od najpopularnijih i najrasprostranjenijih skriptnih programskih jezika danas. Pomoću JavaScript programskog jezika možemo praviti serverske ili klijentske programe (aplikacije). JavaScript kao programski jezik je ograničen jer sa njim ne možemo praviti desktop aplikacije (C++, Java, …), jer JavaScript funkcionira samo u drugom programu - web preglednik. Svaki moderni web preglednik (Chrome, Mozilla Firefox, Safari, …) ima JavaScript *engine*[1] koji omogućava izvršavanje JavaScript koda.

JavaScript ne može pristupati lokalnim dokumentima, niti spasiti dokument na računalni disk ili pristupiti USB portu; … zato što JavaScript nije dizajniran kao generalni programski jezik već skriptni programski jezik za pravljenje dinamičkih web stranica. U ovoj skripti radit će mo JavaScript kao klijentski programski jezik, što znači da se kod izvršava na našem računalu (i naravno trebamo imati bilo koji od modernih web preglednika koji omogućava izvršavanje JavaScript koda).

1.1 Sintaksa

JavaScript kod pišemo između HTML **script** tagova (<script> … </script>). Sve što se nalazi između script tagova web preglednik će smatrati kao JavaScript program.

Sintaksa:

```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body>
```

[1] *engine* - program ili biblioteka koja izvršava JavaScript kod (npr. Chrome web preglednik - V8 engine)

```
  <script>
    // JavaScript code
  </script>
  </body>
</html>
```

Iz primjera možemo vidjeti da script tagove možemo smjestiti bilo gdje na web stranici. Dvostruka kosa crta je jednolinijski komentar. JavaScript kod možemo pisati u najobičnijem tekst editoru ili bilo kojem modernom tekst editoru namijenjen za web tehnologije (Atom, VS Code, …). JavaScript programe možemo pisati i u odvojenoj datoteci koja treba imati nastavak "js". Da bi uključili tu datoteku u HTML pišemo sljedeće:

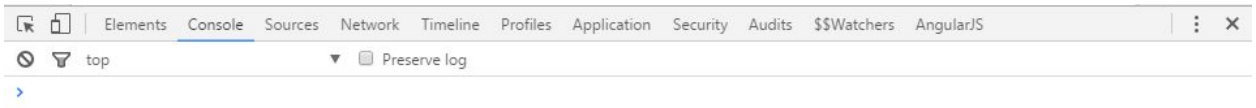
```
<script scr="putanja_do_datoteke.js"></script>
```

Sve primjere iz skripte možemo pisati na "praznoj" web stranici ili koristiti neke od postojećih web stranica koje su namijenjene za pisanje JavaScript koda:

- https://jsfiddle.net/
- http://codepen.io/
- https://jsbin.com/?js,output

Navest će mo par osnovnih funkcija koje se koriste za prikaz podataka i ukratko opisati što svaka funkcija radi, jer će mo ih koristiti u početnim poglavljima.

console.log() - funkcija koja ispisuje određeni tekst u konzolu. Da bi vidjeli konzolu pritisnemo tipku F12 dok se nalazimo u web pretraživaču (Chrome, Mozilla) i prebacimo se na karticu *Console* ili *Settings -> More tools -> Developer tools -> Console*.

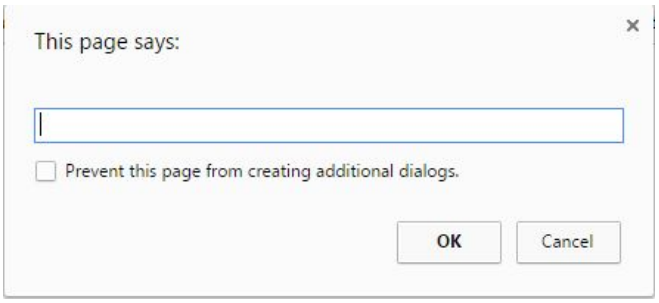


Slika 1. Developer alati

alert() - ispisuje korisniku informaciju u obliku prozorčića koji iskoči, gdje korisnik treba kliknuti "OK" da bi nastavio dalje.

Slika 2. Prikaz praznog alert prozora

prompt() - izbacuje prozor korisniku gdje korisnik može unijeti neku vrijednost koju možemo spremiti u varijablu. Vrijednost koju korisnik unese će biti sprmljena u obliku teksta.



Slika 3. Prikaz praznog prompt prozora

2. Tipovi podataka, varijable i operatori

2.1 Tipovi podataka

JavaScript je *weakly typed* programski jezik, što znači da ne moramo definirati tip podatka već će JavaScript odrediti tip. Čak i kad probamo izvršiti operacije na različitim tipovima podataka, JavaScript će pokušati shvatiti što želimo odraditi (u pozadini postoji set pravila po kojima JavaScript određuje na koji način odraditi konverziju tipova podataka).

2.1.1 Brojevi

Broj kao tip podatka dolazi u dvije forme:

- Cijeli broj (integer), može biti pozitivan ili negativan
- Frakcionalni broj (floating-point), može biti pozitivan ili negativan

JavaScript predstavlja brojeve kao jedan tip, 64 bitne floating brojeve. 1 i 1.0 se u memoriju jednako spremaju. Negativni brojevi se formiraju korištenjem prefix operatora "-" . Vrijednost *NaN* je broj koji je rezultat operacije koja nema valjanu vrijednost. Vrijednost *Infinity* predstavlja sve vrijednosti veće od 1.79769313486231570e+308.

Brojevi imaju metode (poglavlje 5 i 7).

```
Math.floor(number);
```

2.1.2 String

String je jedan ili više znakova koji se nalaze unutar jednostrukih ili dvostrukih navodnika (npr. “Hello World”). Jednostruki i dvostruki navodnici se mogu koristit, sve dok se početni i završni navodnik podudaraju. JavaScript može spremiti prazan string kao vrijednost.
Da bi omogućili prelamanje teksta u novi red, moramo koristiti *backslash* (\ ) u stringu.

- \n - nova linija

“Prva linija i \nDruga linija” će se formatirati u:

Prva linija i
Druga linija

Ako želite da se *backslash* pojavi u stringu, jednostavno koristite 2 *backslash-a* jedan iza drugog. Na tabeli možemo vidjeti detalje najčešće korištenih *escape* sekvenci.

Escape Sekvenca	Znak
\b	Backspace
\n	New line
\r	Carriage return
\t	Tab
\’	Single quote
\’’	Double quote
\\	Backslash

Stringovi imaju *length property* koji vraća broj znakova u stringu (više o propertijima u poglavlju 5 i 6).

```
alert("seven".length); // 5
```

Stringovi su *immutable* tip podatka, tj. jednom napravljen string se ne može mijenjati. Ali je jednostavno napraviti novi string spajanjem stringova korištenjem + operatora.
Dva stringa su jednaka ako imaju isti broj znakova u istom slijedu.

```
alert("h" + "o" + "u" + "s" + "e" === "house"); // true
```

Stringovi imaju metode:

```
alert("house".toUpperCase() === "HOUSE"); // true
```

2.1.3 Boolean

Boolean tip podatka ima dvije moguće vrijednosti: ***true****(istina)* i ***false****(neistina)*.
Falsy vrijednosti:

- false
- null
- undefined
- ' ' (prazan string)
- 0 (broj)
- *NaN*

Sve ostale vrijednosti su *truthy*.

2.1.4 Undefined

Null i *undefined* su dvije specijalne vrijednosti. One su same po sebi vrijednosti, ali ne nose informaciju. U većini slučajeva mogu se tretirati jednako (više o njima kasnije).

2.2 Varijable

Kao mnogi ostali programski jezici, JavaScript ima varijable. Varijable možete zamisliti kao imenovane spremnike koji mogu sadržavati bilo koji tip podatka. Varijable su slične *x* i *y* varijablama u matematici, što znači da su one samo imena koja predstavljaju neke podatka kojima želimo pristupiti. Da bi koristili varijablu u JavaScript programu, potrebno je deklarirati varijablu. Varijabla se deklarira sa ključnom riječi **var**.

```
var name;
var age;
```

Svaka linija koda u JS završava sa točka-zarezom (;). Također možemo deklarirati više varijabli u istoj liniji. Imena varijabli će mo odvajati zarezom.

```
var name, age;
```

Spremanje vrijednosti u variablu se zove **inicijalizacija**. Varijablu možete inicijalizirati prilikom stvaranja varijable ili kasnije u kodu.

```
var name = 'Ante';
var age;
age = 23;
```

Primjetit će mo da smo ključnu riječ var koristili samo jednom prilikom deklaracije odnosno inicijalizacije varijable.

2.2.1 Scope

Scope varijable je dio programa u kojem je varijabla definirana. JavaScript varijable imaju dva *scope*-a:

- *globalne varijable* - ako varijabla ima globalni scope, znači da joj možemo pristupiti bilo gdje u programu

- *lokalne varijable* - vidljive samo u funkciji gdje su definirane. Parametri funkcije su uvijek lokalni toj funkciji.

Ako imamo globalnu i lokalnu varijablu sa istim imenom, lokalna varijabla ima prednost.

Primjer:

```
var myVar = 'global';

function checkScope() {
  var myVar = 'local';
  console.log('In function: ' + myVar); // In function: local
}

checkScope();
console.log('Out of function: ' + myVar); // Out of function: global
```

2.2.2 Imenovanje varijabli

1.Ključne riječi se ne smiju koristiti kao imena varijabli:

- break, boolean, return, function, …

2. Imena varijabli ne smiju počinjati brojem:

- 123test

3. Imena varijabli mogu počinjati sa *underscore* "_":

- _123test

4. Imena varijabli su *case sensitive*

- *Name* i *name* su dvije različite varijable

2.2.3 Ključne riječi

Lista svih ključnih riječi u JavaScript-u se nalaze u tabeli ispod. Ključne riječi se ne mogu koristiti za imena varijabli, funkcija, metoda, labela za petlje ili bilo kakvih imena objekata.

Tablica ključnih riječi:

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throws
case	false	native	transient
catch	final	new	true
char	finally	null	try
class	float	package	typeof
const	for	private	var
continue	function	protected	void
debugger	goto	public	volatile
default	if	return	while
delete	implements	short	with
do	import	static	
double	in	super	

2.2.4 Konverzija tipova podataka

Ako pokušamo zbrojiti string sa brojem, JavaScript prije konkatenacije pravi konverziju broja u string. Automatsku konverziju tipova podataka bi trebali izbjegavati.

```
console.log(1 + 2); // 3
console.log('1' + '2'); // 12
console.log(1 + '2'); // 12
console.log(1 + 2 + 'a'); // 3a
```

Da bi string prebacili u broj koristimo *parseInt()* funkciju.

Sintaksa:

```
parseInt(string, radix);
```

Primjer:

```
console.log(typeof parseInt('23', 10)); // number
```

Više o *typeof* operatoru u sljedećem podpoglavlju.

2.3 Operatori

JavaScript podržava sljedeće tipove operatora:

- aritmetički operatori
- operatori usporedbe
- logički operatori
- operatori pridjele vrijednosti
- uvjetni operatori

2.3.1 Aritmetički operatori

Tablica i opis aritmetičkih operatora:

Operator	**Opis (var A = 7, B = 3;)**
+ (zbrajanje)	Zbraja dva operanda. A + B // 10
- (oduzimanje)	Oduzima drugi operand od prvog A - B // 4
* (množenje)	Množi oba operanda A * B // 21
/ (dijeljenje)	Dijeli prvi operand sa drugim A / B // 2.33333
% (modul)	Daje ostatak cjelobrojnog dijeljenja A % B // 1
++ (inkrement)	Uvećava vrijednost za jedan

	A++ (A = A +1 ili A += 1) // 8
-- (dekrement)	Umanjuje vrijednost za jedan A-- (A = A - 1 ili A -= 1) //6

Primjer:

```
var a = 21;
var b = 10;
var c = 'String';

console.log('a + b = ', a + b); // a + b = 31
console.log('a - b = ', a - b); // a - b = 11
console.log('a / b = ', a / b); // a / b = 2.1
console.log('a % b = ', a % b); // a % b = 1
console.log('a + b + c = ', a + b + c); // a + b + c = 31String
```

2.3.2 Operatori usporedbe

Tablica i opis operatora usporedbe:

Operator	Opis (var A = 7, B = 3;)
== (jednako)	Provjerava je li vrijednost operanada jednaka. A == B // false
!= (nije jednako)	Provjerava vrijednost operanada. Ako vrijednost nije ista onda je izraz istinit A != B // true
> (veće od)	Provjerava je li vrijednost lijevog operanda veća od vrijednosti desnog operanda. A > B // true
< (manje od)	Provjerava je li vrijednost lijevog operanda manja od vrijednosti desnog operanda. A < B // false

>= (veće ili jednako od)	Provjerava je li vrijednost lijevog operanda veća ili jednaka vrijednosti desnog operanda. A >= B // true
<= (manje ili jednako od)	Provjerava je li vrijednost lijevog operanda manja ili jednaka vrijednosti desnog operanda. A <= B // true

Primjer:

```
var a = 10;
var b = 20;

console.log('(a == b) => ', a == b); // (a == b) => false
console.log('(a != b) => ', a != b); // (a != b) => true
console.log('(a > b) => ', a > b); // (a > b) => false
console.log('(a <= b) => ', a <= b); // (a <= b) => true
```

Kod operatora usporedbe == moramo paziti jer:

- == uspoređuje samo vrijednost
- === (!==) uspoređuje tip i vrijednost

Primjer:

```
console.log(5 == '5'); // true
console.log(5 === '5'); // false
```

Operator usporedbe == radi automatsku konvertiju tipa podatka.

2.3.3 Logički operatori

Tablica i opis logičkih operatora:

Operator	Opis (var A = 7, B = 3;)
&& (logičko AND - “i”)	Ako su oba operanda *truthy* vrijednosti rezultat je istinit. (A && B) // true
|| (logičko OR - “ili”)	Ako bilo koji od operanada nije *falsy* rezultat je istinit. (A || B) // true
! (logičko NOT - “ne”)	Prebacuje stanje operanda (true -> false, false -> true) ! (A && B) // false

Tablica istine:

A	B	A && B	A || B	!(A && B)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Primjer:

```
var a = true;
var b = false;

console.log('(a && b) => ', a && b); // (a && b) => false
console.log('(a || b) => ', a || b); // (a || b) => true
console.log('!(a && b) => ', !(a && b)); // !(a && b) => true
console.log('null || "user" => ', null || 'user'); // null || “user” => user
```

2.3.4 Bitwise operatori

Bitwise operatori su operatori koji izvršavaju operacije nad svakim bitom broja.

Tablica i opis *bitwise* operatora:

Operator	Opis (var A = 2, B = 3;)
& (bitwise AND - “i”)	Izvršava AND operaciju nad svakim bitom operanada. A & B // 2
| (bitwise OR - “ili”)	Izvršava OR operaciju nad svakim bitom operanada. A | B // 3
^ (bitwise XOR - ‘ekskluzivno ili’)	Izvršava ekskluzivno OR nad svakim bitom operanada. A ^ B // 1
~ (bitwise not - ‘ne’)	Unarni operator. Okreće sve bitove operanda. ~B // 4
<< (shiftanje u lijevo)	Pomjera sve bitove prvog operanda u lijevo za onoliko mjesta koliko smo definirali drugim operandom. Novi bitovi su popunjeni nulama. Shiftanje vrijednosti u lijevo za jednu poziciju je jednako množenju broja sa 2 (dvi pozicije - 4, …). A << 2 // 8
>> (shiftanje u desno)	Pomjera sve bitove prvog operanda u desno za onoliko mjesta koliko smo definirali drugim operandom. A >> 1 // 1

Primjer:

```
var a = 2;
var b = 3;

console.log('(a & b) => ', a & b); // (a & b) => 2
console.log('(a | b) => ', a | b); // (a | b) => 3
console.log('(a ^ b) => ', a ^ b); // (a ^ b) => 1
console.log('~b => ', ~b); // ~b => -4
console.log('(a << 2) => ', a << 2); // (a << 2) => 8
console.log('(a > 1) => ', a > 1); // (a > 1) => 1
```

2.3.5 Operatori pridjele vrijednosti

Tablica operatora pridjele vrijednosti:

Operator	Opis (var A = 7, B = 3;)
= (jednostavna pridjela)	Pridjeluje vrijednost sa desne strane, lijevoj strani. C = A + B;
+= (dodaj i pridjeli)	Zbraja trenutnu vrijednost lijevog operanda sa desnim i sprema novu vrijednost u lijevi operand. C += A; (ekvivalentno C = C + A)
-= (oduzmi i pridjeli)	Oduzima trenutnu vrijednost lijevog operanda sa desnim i sprema novu vrijednost u lijevi operand. C -= A; (ekvivalentno C = C - A)
*= (pomnoži i pridjeli)	Množi trenutnu vrijednost lijevog operanda sa desnim i sprema novu vrijednost u lijevi operand. C *= A; (ekvivalentno C = C * A)
/= (podijeli i pridjeli)	Dijeli trenutnu vrijednost lijevog operanda sa desnim i sprema novu vrijednost u lijevi operand. C /= A; (ekvivalentno C = C / A)
%= (modul i pridjeli)	Modul trenutne vrijednosti lijevog operanda sa desnom i sprema novu vrijednost u lijevi operand. C %= A; (ekvivalentno C = C % A)

Primjer:

```
var a;
var b = 10;

console.log('(a = b) => ', a = b); // (a = b) => 10
console.log('(a += b) => ', a += b); // (a += b) => 20
console.log('(a -= b) => ', a -= b); // (a -= b) => 10
console.log('(a %= b) => ', a %= b); // (a %= b) => 0
```

2.3.6 Uvjetni operator

Uvjetni operator prvo izračuna uvjetni izraz i izvrši jedno od dva data stanja u ovisnosti o rezultatu uvjetnog izraza.

Tablica uvjetnog operatora:

Operator	Opis (var A = 7, B = 3;)
? :	(uvjet) ? ako je uvijet istinit : ako uvijet nije istinit (A > B) ? 'istinit' : 'neistinit' // istinit

Primjer:

```
var a = 7;
var b = 3;

var result = (a >= b) ? 10 : 20;
console.log('(a >= b) ? 10 : 20 => ', result); // (a >= b) ? 10 : 20 =>  10
```

2.3.7 Typeof operator

Typeof operator je unarni operator koji se stavlja ispred operanda bilo kojeg tipa. Njegova vrijednost je string koji označava tip podatka operanda.

Tablica typeof operatora:

Tip	String
Number	"number"
String	"string"
Boolean	"boolean"
Obejct	"object"
Function	"function"

Undefined	“undefined”
Null	“object”

Primjer:

```
var a = 10;
var b = 'String';
var result;

result = typeof b === 'string' ? 'B is String' : 'B is Numeric';
console.log(result); // B is String

result = typeof a === 'string'? 'A is String' : 'A is Numeric';
console.log(result); // A is Numeric
```

3. Kontrola toka

Kada program sadrži više od jednog izraza, oni se izvršavaju predvidivo, od vrha programa prema dnu programa. Pogledajmo primjer sa dva izraza:

1.Pitaj korisnika da unese broj.

2.Prikaži kvadratnu vrijednost tog broja.

```
var number = parseInt(prompt('Unesite broj: '), 10);
console.log('Kvadratna vrijednost broja je ', number * number);
```

Imamo dva izraza koja se uvijek izvršavaju istim slijedom, pa bi shematski prikaz kontrole toka programa izgledao:

Uvjetno izvršavanje programa daje programskim jezicima inteligenciju, tj. omogućava različite rute kojima će se program izvoditi u ovisnosti o *boolean* vrijednostima:

Uvjetno izvršavanje u JavaScript programskom jeziku je moguće preko **if** uvjetnog izraza. U jednostavnom slučaju želimo da se dio programa izvrši samo ako je određeni uvjet zadovoljen.

Primjer:

```
var number = parseInt(prompt('Unesite broj: '), 10);
if (number === 5) {
    console.log('Kvadrat broja 5 je 25.');
}
```

Console funkcija će se izvršiti samo ako je *if* uvjet zadovoljen, odnosno ako je korisnik upisao broj 5.

3.1 if - else

JavaScrip podržava sljedeće forme **if-else** izraza:

- **if** izraz
- **if-else** izraz
- **if - else if** izraz

Već smo vidjeli na koji način funkcionira *if* izraz. **If-else** funkcionira slično, gdje se *if* izraz izvrši ako je uvjet istinit, u suprotnom se izvrši *else* izraz.

Sintaksa:

```
if (uvjet) {
  // Blok koda
} else {
  // Blok koda
}
```

Vitičaste zagrade "{}" predstavljaju **blok koda** u kojima se izvršava određeni dio koda. Uvjet je zadovoljen samo ako je logička istina (true). Podsjetimo se koje su sve *falsy* vrijednostu:

- false
- 0 (nula)
- '' (prazan string)
- null
- undefined
- NaN

Primjer:

```
var a = 5;
if (a) {
   console.log(a); // 5
}
// ----------------------
if (!a) {
```

```
    console.log(a); // nema ispisa
}
// ----------------------
a = 0;
if (a) {
    console.log(a); // nema ispisa
}
// ----------------------
if (!a) {
    console.log(a); // 0
}
```

1. Zadatak:

Inicijalizirajte dvije varijable, *a* i *b*. Varijabli *a* pridjelite vrijednost 10, a varijabli *b* vrijednost 5. Koristeći if else izraz i operator usporedbe manje "<" ispišite odgovarajuću vrijednost.

Rješenje:

```
var a = 10;
var b = 5;

if (a < b) {
    console.log(a);
} else {
    console.log(b);
}
```

Ispis: 5

U izrazu možemo koristiti više uvijeta koje povezujemo logičkim operatorima.

Primjer:

```
var a = 10;
var b = 5;

if (a === 10 && b === 5) {
  console.log(a);
} else {
  console.log(b);
}
```

Ispis: 10

Treći if-else if izraz je ulančavanje else if izraza koje će mo pokazati na jednostavnom primjeru.

Primjer:

```
var a = 15;

if (a > 0 && a < 10) {
  console.log('Low');
} else if (a >= 10 && a <= 20) {
  console.log('Average');
} else if (a >= 20 && a <= 30) {
  console.log('High');
} else {
  console.log('Out of range');
}
```

Ispis: Average

2. Zadatak:

Učitati dva različita broja (koristiti funkciju *prompt*). Ako su oba parna, podijeliti veći broj sa manjim. Ako su oba neparna, od većeg broja oduzeti manji. Ispisati rezultat svake operacije.

Rješenje:

```
var num1 = parseInt(prompt('Unesite prvi broj:'), 10);
var num2 = parseInt(prompt('Unesite drugi broj:'), 10);
var result;

if (num1 % 2 === 0 && num2 % 2 === 0) {
  result = num1 > num2 ? num1 / num2 : num2 / num1;
} else if (num1 % 2 !== 0 && num2 % 2 !== 0) {
  result = num1 > num2 ? num1 - num2 : num2 - num1;
}

console.log('Rezultat operacije je ' + result);
```

3. Zadatak:

Zatražiti od korisnika unos godine rođenja (koristiti funkciju *prompt*). Izračunati starost korisnika i ispisati odgovarajuću poruku (‘Punoljetni ste’ / ‘Niste punoljetni’). Ne uzeti u obzir ako korisnik unese nevaljanu godinu, u tom slučaju ispisati ‘Pogrešan unos’.

Rješenje:

```
var birthYear = parseInt(prompt('Unesite godinu rodjenja:'), 10);
var age;

if (birthYear > 1900 && birthYear <= 2016) {
  age = 2016 - birthYear;

  if (age >= 18) {
    console.log('Punoljetni ste.');
  } else {
    console.log('Niste punoljetni.');
  }
} else {
  console.log('Pogresan unos.');
}
```

4. Zadatak

Unesite dva cijela troznamenkasta broja. Ukoliko uneseni broj nije troznamenkast, odbaciti ga. Zbrojiti zadnje znamenke svakog unesenog broja i ispisati broj.

Rješenje:

```
var num1 = parseInt(prompt('Unesite prvi broj:'));
var num2 = parseInt(prompt('Unesite drugi broj:'));
var digitSum = 0;

if (num1 > 99 && num1 < 1000) {
  digitSum += num1 % 10;
}
if (num2 > 99 && num2 < 1000) {
  digitSum += num2 % 10;
}
console.log('Zbroj zadnjih znamenki iznosi ' + digitSum);
```

5. Zadatak:

Napraviti jednostavan kalkulator. Omogućiti korisniku da pomoću prompt funkcije unese dva broja i operaciju (+, -, *, /). Ispišite odgovarajuću poruku sa rezultatom, a ako operacija nije podržana ispišite ‘Operacija ne postoji’. (Pripaziti kod operacije dijeljenja).

Rješenje:

```
var num1 = parseInt(prompt('Unesite prvi broj:'));
var num2 = parseInt(prompt('Unesite drugi broj:'));
var operation = prompt('Unesite znak operacije:');
var result;

if (operation === '+') {
  result = num1 + num2;
} else if (operation === '-') {
  result = num1 - num2;
} else if (operation === '*') {
  result = num1 * num2;
} else if (operation === '/') {
  if (num2 !== 0) {
    result = num1 / num2;
  } else {
    result = 'Infinity';
    console.log('Nije moguće dijeliti s nulom.');
  }
} else {
  result = 'error';
  console.log('Operacija ne postoji.');
}

console.log(num1 + ' ' + operation + ' ' + num2 + ' = ' + result);
```

3.2 switch - case

switch - case izraz je dosta sličan *if - else* izrazu. *Switch - case* izraz koristimo u slučaju izvršavanja akcija baziranih na različitim uvjetima.

Sintaksa:

```
switch (izraz) {
  case condition1:
    // Blok koda
    break;
  case condition2:
    // Blok koda
```

```
    break;
  default:
    // Blok koda
}
```

break izraz se navodi na kraju određenog *case*-a. *Break* također koristimo kod kontrole petlji (iduće poglavlje).

6. Zadatak

Napraviti kalkulator koristeći *switch* - *case* strukturu.

Rješenje:

```
var num1 = parseInt(prompt('Unesite prvi broj:'), 10);
var num2 = parseInt(prompt('Unesite drugi broj:'), 10);
var operation = prompt('Unesite znak operacije:');
var result;

switch (operation) {
  case '+':
    result = num1 + num2;
    break;
  case '-':
    result = num1 - num2;
    break;
  case '*':
    result = num1 * num2;
    break;
  case '/':
    if (num2 !== 0) {
      result = num1 / num2;
    } else {
      result = 'Infinity';
      console.log('Nije moguće dijeliti s nulom.');
    }
    break;
  default:
    result = 'error';
    console.log('Operacija ne postoji.');
}

console.log(num1 + ' ' + operation + ' ' + num2 + ' = ' + result);
```

Zadaci za ponavljanje:

1. Zatražiti od korisnika unos dva broja. Ako su oba parna ispisati poruku *Oba broja su parna*, ako su oba neparna ispisati poruku *Oba su neparna*, ako je jedan broj paran a jedan neparan ispisati poruku *Jedan broj je paran, a jedan neparan.*

2. Napisati program koji će pitati korisnika želi li ispisati temperaturu u celzijevim stupnjevima ili fahrenheit-ima. Ako korisnik upiše *C* ispisati u cezijevima, odnosno ako upiše *F* ispisati u fahrenheit-ima. Vrijednost temperature koju treba pretvoriti je 30 °C.
(F = C * (9/5) +32)

4. Petlje

Dok pišemo program, možemo se naći u situaciji gdje moramo ponavljati određene akcije. U ovakvim situacijama pišemo petlje da bi smanjili broj linija koda. Petlja predstavlja blok koda koji se ponavlja sve dok je uvijet istinit.

4.1 while petlja

while petlja je jedna od osnovnih petlji u JavaScript programu. Svrha while petlje je izvršavanje bloka koda sve dok je uvjet istinit. Kada izraz postane neistinit, petlja se završava.

Sintaksa:

```
while (uvjet) {
  // Blok koda
}
```

1. Zadatak

Program treba ispisati brojeve od 0 do 9.

Rješenje:

```
var count = 0;

while (count < 10) {
  console.log('Trenutni broj je: ', count);
  count++;
}
```

Primjetit će mo liniju koda "*count*++" gdje uvećavamo vrijednost brojača. Bez ove linije imali bi beskonačnu petlju koja bi srušila program.

2. Zadatak

Napisati program koji ispisuje sve parne brojeve između 1 i 20.

Rješenje:

```
var count = 1;

while (count <= 20) {
  if (count % 2 === 0) console.log(count);
  count++;
}
```

3. Zadatak

Napisati program koji od korisnika traži unos 5 pozitivnih brojeva, zatim ispisuje njihov umnožak. Ukoliko ih u više od 10 pokušaja nije uspio unijeti, ispisati poruku o grešci i završiti program.

Rješenje:

```
var positiveCount = 0;
var tryCount = 0;
var product = 1;
var input;

while (positiveCount < 5 && tryCount < 10) {
  input = parseInt(prompt('Enter positive number: '), 10);

  if (input > 0) {
    product *= input;
    positiveCount++;
  }

  tryCount++;
}

if (positiveCount === 5) {
  console.log('Product: ', product);
} else {
  console.log('Error');
}
```

4.2 do-while petlja

do-while petlja je slična **while** petlji. Uvjet se provjerava na kraju petlje, što znači da se ova petlja mora izvršiti najmanje jednom.

Sintaksa:

```
do {
  // Blok koda
} while (uvjet);
```

4. Zadatak

Napisati program u kojem korisnik unosi tri pozitivna troznamenkasta broja. Ako broj nije pozitivan, odbaciti ga. Ispisati zbroj prvih znamenki unesenih brojeva. (koristiti do-while petlju)

Rješenje:

```
var tryCount = 0;
var sum = 0;
var input;

do {
  input = parseInt(prompt('Enter 3 digit number: '), 10);

  if (input > 99 && input < 1000) {
    sum += parseInt(input / 100, 10);
    tryCount++;
  }
} while (tryCount < 3)

console.log('Sum: ', sum);
```

4.3 for petlja

for petlja je najčešće korištena petlja u JavaScriptu. Sastoji se od 3 osnovna dijela:

- **inicijalizacija** - inicijaliziramo brojač na početnu vrijednost. Inicijalizacijski izraz se izvrši prije izvođenja petlja.
- **ispitivanje uvjeta** - ispituje je li uvijet istinit ili neistinit. Ako je uvijet istinit, blok koda će se izvršiti, u suprotnom petlja će se završit.
- **iteracijski izraz** - uvećavanje/umanjivanje brojača

Sintaksa:

```
for (inicijalizacija; ispitivanje uvjeta; iteracijski izraz) {
 // Blok koda
}
```

5. Zadatak

Ispisati brojeve od 0 do 9.

Rješenje:

```
var len = 10;
var i;

for (i = 0; i < len; i++) {
  console.log('Current count: ', i);
}
```

6. Zadatak

Ispisati brojeve od 10 do 1.

Rješenje:

```
var i;
var len = 0;

for (i = 10; i > len; i--) {
  console.log('Current count: ' + i);
}
```

7. Zadatak

Učitati dva broja, zatim ispisati svaki drugi broj između njih koji je djeljiv sa 2 i sa 3.

Rješenje:

```
var num1 = parseInt(prompt('Unesite prvi broj:'), 10);
var num2 = parseInt(prompt('Unesite drugi broj:'), 10);
var temp;

if (num1 > num2) {
      temp = num2;
  num2 = num1;
  num1 = temp;
}

for (var i = num1; i <= num2; i += 2) {
  if (i % 2 === 0 && i % 3 === 0) {
    console.log(i);
  }
}
```

4.4 break izraz

break izraz smo spominjali kod *switch* izraza. Koristi se za prekid petlje.

Primjer:

Dopustiti korisniku da unosi brojeve sve dok ne unese broj 5.

Rješenje:

```
var input;

while (true) {
  input = parseInt(prompt('Enter number: '), 10);
  if (input === 5) {
    break;
  }
}
```

U primjeru smo napravili beskonačnu *while* petlju (uvijet izvršavanja uvijek istinit). Blok koda u *while* petlji ima **break** izraz što nam omogućava prekid petlje.

4.5 Continue izraz

continue izraz govori interpreteru da odmah starta sa sljedećom iteracijom petlje i da preskoči preostalio dio bloka koda.

Primjer:
Ispisati brojeve od 1 do 10. Preskočiti ispis broja 5.

Rješenje:

```
var i = 0;

while (i < 10) {
  i++;
  if (i === 5) {
    continue;
  }

  console.log(i);
}
```

Zadaci za ponavljanje:

1. Zatražiti od korisnika unos dva troznamenkasta broja. Ispisati sve parne brojeve koji se nalaze u rasponu između ta dva broja.

2. Zatražiti od korisnika unos brojeva sve dok im zbroj ne bude 150 (ili veći). Zatim ispisati brojeve od 150 do 1 koristeći *while* petlju.

5. Funkcije

Funkcija je grupa *reusable* koda koju možemo koristiti u programu, što znači da ne moramo pisati isti kod više puta već samo jednom. Funkcije pomažu programerima da pišu modularniji kod, odnosno da podijele veliki program u manje funkcije.

Do sada smo koristili funkcije **alert()** i **prompt()** koje su napise u core JavaScript-u.

5.1 Definiranje i pozivanje funkcije

Prije korištenja funkcije, moramo je definirati. Najjednostavniji način definiranja funkcije je pomoću ključne riječi **function** nakon čega pišemo jedinstveno ime funkcije, kojoj opcionalno proslijeđujemo listu parametara.

Sintaksa:

```
function functionName(parameterList) {
  // Blok koda
}
```

Primjer:
Napisati *sayHello()* funkciju koja će ispisati poruku ‘Good day!’. Koristiti alert funkciju za ispis poruke.

Rješenje:

```
function sayHello() {
  alert('Good day!');
}
```

Funkciju također možemo definirati kao function literal, tj. izraz koji definira bezimenu funkciju:

```
var sayHello = function() {
  alert('Good day!');
}
```

Da bi izvršili blok koda neke funkcije, moramo pozvati tu funkciju. Funkciju pozivamo na način da navedemo ime funkcije: *functionName();*
Poziv funkcije iz prethodnog primjera: *sayHello();*

5.2 Parametri funkcije

Prilikom poziva, funkciji možemo proslijediti parametre. Parametri se ponašaju kao obične varijable. U funkciji možemo izvršiti bilo kakvu promjenu na parametrima. Funkcija može primiti više parametara koji su odvojeni zarezom.
Važno svojstvo funkcija je da varijable napravljene u funkciji, uključujući i parametre, su lokalne toj funkciji. To znači da će se varijable uvijek iznova stvarati svaki put kada pozovemo funkciju.

Primjer:
Dodati parametre “name” i “age” na funkciju *sayHello:*

Rješenje:

```
function sayHello(name, age) {
  alert(name + ' is ' + age + ' years old.');
}

sayHello('Mate', 26);
```

5.3 return izraz

JavaScript funkcije mogu imati opcionalni **return** izraz. Potreban je ako želimo da funkcija vrati neku vrijednost. Obično je **return** izraz zadnji izraz u funkciji (ne nužno).

Primjer:
Napisati funkciju koja vraća zbroj dva broja. Brojeve proslijediti kao parametre.

Rješenje:

```
function add(a, b) {
  return a + b;
}

var c = add(5, 4);
console.log(c);
```

5.4 Scope

Scope u prijevodu znači doseg, odnosno u programskom smislu koji dio koda vidi određene varijable. JavaScript je *function scope* jezik, što znači da sve varijable definirate u nekoj funkciji su vidljive samo u toj funkciji - **lokalne varijable**. Varijable definirane izvan funkcije su vidljive u cijelom programu - **globalne varijable**.

Primjer:

```
var x = 5;

function change(num) {
  num = 10;
  console.log(num); // 10
}
console.log(x); // 5
change(x);
console.log(x); // 5
```

Proslijedili smo varijablu x funkciji change. Spomenuli smo da su parametri i lokalne varijable u funckiji uvijek iznova naprave prilikom poziva funkcije, što znači da se orginalna vrijednost varijable x neće promijeniti.

Primjer:

```
var x = 5;

function change() {
  var x = 10;
  console.log(x); // 10
}

change();
console.log(x); // 5
```

Iz ovog primjera možemo zaključiti da lokalna varijabla prepisuje globalnu varijablu (samo u funkciji), ali lokalna x varijabla (10) nije vidljiva izvan funkcije, niti će promijenuti vrijednost globalne varijable x (5).

Primjer:

```
var x = 5;

function change() {
  x = 10;
  console.log(x); // 10
}

change();
console.log(x); // 10
```

Iz ovoga primjera možemo zaključiti da varijable koje nisu definirane u nekoj funkciji, JavaScript će pretraživati u scope-u koji je izvan te funkcije sve dok ne nađe scope gdje je definirana određena varijabla. Bilo koja promjena na vrijednosti ove varijable u funkciji, će rezultirati i promjeni varijable izvan funkcije, jer su to iste varijable. Ovo svojstvo u JavaScript-u se zove **closure** gdje funkcija ima pristup varijablama definiranim izvan funkcije.

1. Zadatak

Napisati funkciju *min()* koja prima dva broja kao argumente i vraća manji od njih.

Rješenje:

```
function min(a, b) {
  return (a <= b) ? a : b;
}

console.log(min(5, 3));
console.log(min(4, 4));
```

2. Zadatak

Napišite funkciju za računanje broja na zadanu potenciju. Funkcija prima dva parametra, broj i potenciju, te vraća rezultat. Zatražiti od korisnika unos broja i potencije, te ispisati rezultat.

Rješenje:

```
function calculate(num, pot) {
  if (pot === 0) return 1;

  var base = num;
  for (var i = 1; i < pot; i++) {
    num *= base;
  }

  return num;
}
var num = parseInt(prompt('Unesite broj:'), 10);
var pot = parseInt(prompt('Unesite potenciju:'), 10);

console.log('Broj ' + num + ' na potenciju ' + pot + ' iznosi ' +
calculate(num, pot) + '.');
```

5.5 Ugnježđivanje funkcija

Funkcije se mogu ugnježđivati i za njih vrijede ista pravila koja smo naveli u prijašnjim primjerima.

Primjer:

Napisati funkciju za izračun hipotenuze pravokutnog trokuta.

Rješenje:

```
function hypotenuse(a, b) {
  function square(x) {
    return x * x;
  }
  return Math.sqrt(square(a) + square(b));
}

console.log(hypotenuse(3, 4));
```

5.6 Rekurzivne funkcije

Rekurzija je proces u kojem funkcija poziva samu sebe, tj. rekurzija omogućava da funkcije pišemo u drukčijem stilu. Ako funkcija poziva samu sebe tada JS engine mora napraviti novi stack[2] . Moramo biti oprezni kod rekurzivnih funkcija jer postoje ograničenja zbog memorije.

Primjer:

Zbrajanje

```
function sum(x, y) {
  return (y > 0) ? sum(x + 1, y - 1) : x;
}

console.log(sum(1, 6));
```

Detaljan pregled:

- proslijedimo dva broja (parametri x i y)
- provjerimo vrijednost y (mora biti veća od 0)

[2] dio memorije alociran da bi sačuvao trenutne informacije za izvršavanje funkcije

- ako je (y > 0), rekurzivno pozivamo **sum** ali ovaj put vrijednost **x** je uvećana za 1, a **y** umanjena za 1
- priliko sljedećeg poziva funkcije vrijednosti su **x=2, y=5**
- prvi poziv **sum** funkcije još nije završio
- u ovom trenutku JavaScript engine je napravio 2 stack-a, jedan za x=1, y=6, drgi za x=2, y=5
- JavaScript engine mora napraviti stack za svaki rekurzivni poziv

3. Zadatak

Napisati funkciju *countdown* (rekurzivno) koja prima jedna pozitivan broj i ispisuje sve brojeve od tog broja do 0.

Rješenje:

```
var countdown = function(value) {
  if (value <= 0) {
    return value;
  } else {
    console.log(value);
    return countdown(value - 1);
  }
};
countdown(10);
```

4. Zadatak

Napraviti funkciju *factorial* koja računa faktorijele. (primjer: 5! -> 1*2*3*4*5)

Rješenje:

```
function factorial(num) {
  if (num < 0) {
    return -1; // reject
  } else if (num === 0) {
    return 1;
  } else {
    return (num * factorial(num - 1));
  }
```

```
}

var rez = factorial(5);
console.log(rez); // 120
```

5.7 Callback funkcije i timeri

Callback funkcija je funkcija koja je proslijeđena drugoj funkciji kao parametar i izvršena u drugoj funkciji.

Primjer:

```
function log(callback) {
  if (callback) callback();
}
function a() {
  console.log('a');
}

log(a);
```

Iz primjera vidimo da funkcija *log()* prima drugu funkciju *a()* kao parametar. Uvijek je dobro provjeriti da li je callback funkcija stvarno proslijeđena, a zatim izvršiti callback funkciju.

Timere koristimo za simuliranje asinkronosti. Koristimo **setTimeout()** i **setInterval()** funkcije koje odgađaju izvršavanje nekog dijela koda.

Sintaksa:

```
setTimeout(function() {
  // Blok koda
}, miliseconds);

var interval = setInterval(function() {
  // Blok koda
}, miliseconds);
```

Anonimne funkcije su callback funkcije koje će se izvršiti nakon definiranog broja milisekundi. *setTimeout()* funkcija će odgoditi izvršavanje koda jednom, dok će *setInterval()* odgađati

izvršavanje koda neprestano, sve dok ne očistimo interval koristeći funkciju **clearInterval(interval)**.

Primjer:

```
var a = 2;
var b = 4;

function test1(callback) {
  setTimeout(callback, 1000);
}

function test2() {
  a = 10, b = 12;
  console.log(a + ' - ' + b);
  // Nakon što se pozove callback a i b se ažuriraju i ispiše se: 10-12
}

test1(test2);
console.log(a + ' - ' + b); // Ispisati će se 2 - 4 jer se test2 funkcija još nije izvršila
```

Zadaci za ponavljanje:

1. Napisati funkciju koja prima broj a vraća sumu svih prostih brojeva do tog broja.

2. Napisati funkciju koja računa b^n, gdje je b baza a n eksponent.

6. Nizovi

Da bi radili sa nekim setom podataka, moramo naći način kako da ih reprezentiramo. Uzmimo primjer neke kolekcije brojeva: 1 3 5 7 9 11. Ove brojeve možemo predstaviti kao string, budući da string može biti bilo koje dužine : "1 3 5 7 9 11". Ovakav pristup nije dobar, jer bi morali naći način da nekako konvertiramo podatke ponovo u brojeve.

JavaScript pruža tip podatka koji je specifičan za spremanje sekvenci vrijednosti - **niz (array).** Nizovi su skupovi vrijednosti najčešće istog tipa, što znači da niz može sadržavati više varijabli (istog ili različitog tipa podatka).

Deklaracija niza:

```
var arrayName = [];
```

Inicijalizacija niza:

```
var arrayName = [item1, item2, item3, ...];
```

Pristupanje elementima niza:

```
var n = [1, 3, 5, 7, 9, 11];
// n[0] je prvi element
// n[1] je drugi element
// n[2] je treći element
// …
```

Indeksiranje elemenata počinje od 0. Svaki element ima jedinstvenu poziciju.

Elemente niza možemo modificirati:

```
n[0] = 'first item';
```

U ovom slučaju prvi element niza će biti string ‘first item’.

Property **length** vraća 32-bitni integer koji definira broj elemenata u nizu.

Sintaksa:

array.length

Primjer:

```
var arr = [1, 5, 10];
console.log(arr.length); // 3
```

6.1 Metode

Ukratko metode će mo predstaviti kao obične funkcije u ovom poglavlju, a više o samim metodama u idućem poglavlju. U tablici se nalaze osnovne metode koje koristimo za manipuliranje elementima u nizu. Primjeri navedeni u tablici su osnovni primjeri i ako želite naučiti nešto više o ovim metodama, možete pronaći na :

- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array#Methods_2

METODA	OPIS
concat()	Vraća novi niz sastavljen od trenutnog niza i proslijeđenih nizova (ili vrijednosti). `var str1 = 'Hello ';` `var str2 = 'World!';` `var str3 = str1.concat(str2);` `console.log(str3); // Hello World!`
every()	Vraća istinu ako svaki element u nizu zadovoljava funkcijski uvijet. `function checkAdult(age) {`

	`` return age >= 18; } console.log([19, 22, 17].every(checkAdult)); // false ```
filter()	Vraća novi niz sa svim elementima koji zadovoljavaju funkcijski uvijet. ``` function checkAdult(age) { return age >= 18; } console.log([19, 22, 17].filter(checkAdult)); // [19, 22] ```
forEach()	Poziva funkciju za svaki element u nizu. ``` [19, 22, 17].forEach(function (val) { console.log(val); }); ```
indexOf()	Vraća prvu poziciju proslijeđenog parametra u nizu ili -1 ako parametar ne postoji u nizu. ``` var arr = [1, 3, 5, 7]; console.log(arr.indexOf(5)); // 2 ```
join()	Pretvara sve elemente niza u string. ``` var fruits = ['Banana', 'Orange', 'Apple']; console.log(fruits.join()); // Banana,Orange,Apple ```
map()	Vraća novi niz koji je rezultat poziva funkcije za svaki element u niz. ``` var numbers = [4, 9, 16]; console.log(numbers.map(Math.sqrt)); // [2, 3, 4] ```
pop()	Uklanja zadnji element u nizu i vraća taj element. ``` var numbers = [4, 9, 16]; console.log(numbers.pop()); // 16 console.log(numbers); // [4, 9] ```

push()	Stavlja element na kraj niza i vraća novu dužinu niza. `var numbers = [4, 9, 16];` `console.log(numbers.push(25)); // 4` `console.log(numbers); // [4, 9, 16, 25]`
reduce()	Reducira niz na jednu vrijednost. `var sum = [1, 2, 3, 4, 5].reduce(function(total, num) {` `  return total + num;` `});` `console.log(sum); // 15`
reverse()	Okreće redoslijed elemenata u nizu. `var arr = [1, 3, 5, 7];` `console.log(arr.reverse()); // [7, 5, 3, 1]`
shift()	Uklanja prvi element u nizu i vraća taj element. `var arr = [1, 3, 5, 7];` `console.log(arr.shift()); // 1` `console.log(arr); // [3, 5, 7]`
slice()	Vraća izdvojeni dio niza. `var arr = [1, 3, 5, 7, 9, 11];` `console.log(arr.slice(1, 3)); // [3, 5]`
some()	Vraća istinu ako najmanje jedan element zadovoljava funkcijski uvijet. `var arr = [15, 17, 19];` `function checkAdult(age) {` `  return age >= 18;` `}` `console.log(arr.some(checkAdult)); // true`

sort()	Vraća sortirani niz. `var arr = ['c', 'a', 'b'];` `console.log(arr.sort()); // ["a", "b", "c"]`
splice()	Dodaje i/ili uklanja element iz niza. `var fruits = ['Banana', 'Orange', 'Apple', 'Mango'];` `fruits.splice(2, 0, 'Lemon', 'Kiwi');` `console.log(fruits); // ["Banana", "Orange", "Lemon", "Kiwi", "Apple", "Mango"]`
toString()	Pretvara broj(eve) u string. `var arr = [1, 3, 5, 7];` `console.log(arr.toString()); // 1,3,5,7`

1. Zadak:

Deklarirati prazan niz i popuniti ga sa 50 cijelih brojeva (koristiti for petlju). Zatim u drugoj petlji uraditi računicu prosječne vrijednosti članova niza.

Rješenje:

```
var arr = [];
var sum = 0;

for(var i = 1; i <= 50; i++){
  arr.push(i);
}
for(i = 0; i < arr.length; i++){
  sum += arr[i];
}
console.log('Prosjecna vrijednost iznosi ' + (sum/arr.length)); // 25.5
```

2. Zadatak:

Učitati brojeve u niz sve dok im zbroj ne prijeđe 150. Ispisati niz sortiran od najvećeg prema najmanjem.

Rješenje:

```
var arr = [];
var sum = 0;
var num, temp, len;

while (sum <= 150) {
  num = parseInt(prompt('Unesite broj:'));
  arr.push(num);
  sum += num;
}

len = arr.length;
for (var i = 0; i < len; i++) {
  for (var j = i+1; j < len; j++) {
      if(arr[i] < arr[j]) {
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
      }
  }
}
console.log('Niz poredan od većeg prema manjem: ', arr);
```

2. Rješenje:

```
var arr = [];
var sum = 0;
var num, temp;

while (sum <= 150) {
  num = parseInt(prompt('Unesite broj:'));
  arr.push(num);
  sum += num;
}

arr.sort(function (a, b) {
  return b-a;
});
```

```
console.log('Niz poredan od većeg prema manjem: ', arr);
```

3. Zadatak:

Zatražiti od korisnika unos nekog stringa. Ispisati koliko ima samoglasnika u tom stringu.

Rješenje:

```
var text = prompt('Unesite tekst:');
var vowels = ['a', 'e', 'i', 'o', 'u'];
var counter = 0;
var i, len;

text = text.toLowerCase();
len = text.length;
for (i = 0; i < len; i++) {
  if (vowels.indexOf(text[i]) > -1) counter++;
}

console.log('Broj samoglasnika: ' + counter);
```

4. Zadatak:

Zadan je niz cijena kupljenih proizvoda: prices = [‘1.00 KM’, ‘1.85 KM’, ‘19.99 KM’]; Zbrojiti ukupan iznos koji treba platiti.

Rješenje:

```
var prices = ['1.00 KM', '1.85 KM', '19.99 KM'];
var sum = 0;
var price;

for(var i = 0; i < prices.length; i++) {
  price = prices[i].split(' ');
  sum += parseFloat(price[0]);
}
console.log('Iznos za platiti: ' + sum); // Iznos za platiti: 22.84
```

5. Zadatak:

Napravi funkciju koja će usporediti dva niza i vratiti novi niz sa elementima koji se nalaze u jednom ili drugom nizu (jedinstveni elementi). Ignorirati elemente koji su isti i u prvom i u drugom nizu.

Rješenje:

```
function diffArray(arr1, arr2) {
  var newArr = [];

  newArr = arr1.filter(function(val) {
    return arr2.indexOf(val) === -1;
  }).concat(arr2.filter(function(val) {
    return arr1.indexOf(val) === -1;
  }));


  return newArr;
}

var result = diffArray([1, 'cat', 3, 'dog'], [4, 'dog', 3, 1]);
console.log(result); // ["cat", 4]
```

6. Zadatak

Napraviti funkciju koja prima dva ili više nizova i vraća novi niz od jedinstvenih vrijednosti u slijedu kao što se nalaze u orginalnim nizovima.

Rješenje:

```
function uniteUnique(arr) {
  var concatenated = [];
  var union = [];
  var i, len;

  len = arguments.length;
  for (i = 0; i < len; i++) {
    concatenated = concatenated.concat(arguments[i]);
  }

  concatenated.map(function(num) {
```

```
    if (union.indexOf(num) === -1) union.push(num);
  });


  return union;
}

var result = uniteUnique([1, 3, 2], [5, 2, 1, 4], [2, 1]);
console.log(result); // [1, 3, 2, 5, 4]
```

Zadaci za ponavljanje:

1. Napisati funkciju koja prima string a vraća string u obrnutom slijedu. (Npr: “danas” -> “sanad”).

2. Napisati funkciju koja prima rečenicu (string), a vraća broj slova najduže riječi u rečenici.

3. Napisati funkciju koja prima rečenicu (string), a vraća istu rečenicu sa svakom riječi napisanom velikim slovom. (Npr. “Danas je lijep dan za programiranje” -> “Danas Je Lijep Dan Za Programiranje”).

7. Objekti

Objekti, kao i nizovi, mogu spremati više vrijednosti. Kod nizova smo koristili indeks elementa za pristup ili spremanje elementa u niz, a kod objekata koristimo **ključeve**. *Ključ:vrijednost* par nazivamo **property**. Dakle, objekt je neuređena lista primitivnih tipova podataka koji su spremljeni kao parovi *ključ - vrijednost.* Vrijednost može biti bilo kojeg tipa: broj, string, niz, funkcija, drugi objekt, …
Funkcija koja se nalazi u objektu se zove **metoda**.

Inicijalizacija praznog objekta:

```
var obj = {};
```

7.1 Property objekta

Spomenuli smo da je property naziv za parove *ključ:vrijednost*. Postoje tri načina dodavanja propertija.

Dodavanje propertija 1:

```
var person = {
  name: 'Ante',
  age: 15
};
```

U primjeru smo dodali dva propertija čiji su ključevi *name* i *age*. Propertiji se u objektu odvajaju zarezom. Da bi pristupili vrijednosti nekog propertija koristimo operator **točka**.

```
person.name; // Ante
person.age; // 15
```

Dodavanje propertija 2:

```
var person = {};
person.name = 'Ante';
person.age = 15;
```

Kod drugog načina dodavanja propertija, prvo inicijaliziramo prazan objekt, zatim dodajemo propertije koje želimo.

Dodavanje propertija 3:

```
var person = {};
person['name'] = 'Ante';
person['age'] = 15;
```

Kod trećeg načina dodavanja propertija koristimo uglate zagrade gdje navodimo ime propertija.

1. Zadatak:

Inicijalizirati prazan objekt *auto*. Pomoću for petlje dodati 5 propertija koji počinju nazivom *vlasnik*... (ne smije biti *vlasnik0*). Dodijeliti im proizvoljne vrijednosti.

Rješenje:

```
var auto = {};
var imena = ['Ante', 'Mate', 'Nikola', 'Josip', 'Ivan'];

for(var i = 0; i < 5; i++){
  auto['vlasnik' + (i+1)] = imena[i];
}
console.log(auto);
```

Brisanje propertija:

Za brisanje propertija koristimo ključnu riječ **delete**.

Sintaksa:

```
delete obj.property;
```

7.2 Metode

Metoda je funkcija spremljena kao property objekta i ponaša se kao obična (anonimna) funkcija.

Primjer:

```
var obj = {
  zbroj: function (a, b) {
      return a + b;
  }
}

console.log(obj.zbroj(1,2)); // 3
```

2. Zadatak:

Inicijalizirati objekt *auto* sa propertijem *stanje*. Inicijalizirati u objektu dvije metode *vozi()* i *stani()* koje će mijenjati property *stanje* i vraćati novu vrijednost propertija *stanje*. Ispisati stanje vozila na početku, te nakon poziva svake metode.

Rješenje:

```
var auto = {
  stanje: 'pocetno',
  vozi: function () {
      auto.stanje = 'vozi';
      return auto.stanje;
  },
  stani: function () {
      auto.stanje = 'stani';
      return auto.stanje;
  }
};

console.log(auto.stanje); // pocetno
console.log(auto.vozi()); // vozi
console.log(auto.stani()); // stani
```

Da bi pristupili propertiju *stanje*, uvijek smo navodili naziv objekta pa zatim naziv ključa (npr. *auto.stanje*). Umjesto naziva objekta možemo koristiti ključnu riječ **this**. **this** se odnosi na

kontekst objekta u kojem se nalazi i ima dva važna svojstva (koja će mo objasniti u ovom poglavlju):

- *this* vrijednost može biti dinamička, ovisno kako je funkcija pozvana
- kontekst možemo promijeniti preko metoda *.call()*, *.bind()* i *.apply()*

Window objekt:

Po defaultu, *this* bi trebao biti **window** objekt, referenca na glavni globalni scope.

```
console.log(this); // [object Window]
console.log(typeof this); // object
```

Objekt literal:

U objektu, *this* vrijednost će uvijek referencirati vlastiti objekt.

```
var obj = {};

// Definiraj metodu na objektu
obj.logThis = function () {
  console.log(this); // obj
}

// Pozovi metodu
obj.logThis();
```

3. Zadatak:

Inicijalizirati objekt *test* sa propertijima *bodovi, maxBodovi* i *ocjena* te im pridjeliti neke vrijednosti (*ocjenu* postaviti na nulu, *maxBodove* na 100, a *bodove* između 0 i 100). Dodati metodu *izracunajOcjenu* koja mijenja iznos ocjene s obzirom na iznos ostvarenih bodova za taj test.

Rješenje:

```
var test = {
  bodovi: 80,
  maxBodovi: 100,
  ocjena: 0
};
```

```
// Definiraj metodu
test.izracunajOcjenu = function() {
  var maxOcjena = 5; // Privatna varijabla
  this.ocjena = Math.round((this.bodovi / this.maxBodovi) * maxOcjena);
}

console.log(test.ocjena); // 0
test.izracunajOcjenu();
console.log(test.ocjena); // 4
```

Dinamička promjena **this** vrijednosti:

Vrijednost *this* objekta se može mijenjati u ovisnosti kako je funkcija pozvana.

```
var logThis = function () {
  console.log(this);
};

var elem = document.querySelector('.elem');

// Kada je element kliknut, 'this' će postati element
elem.addEventListener('click', logThis);

// 'this' će postati window objekt
logThis();
```

Promjena **this** konteksta:

Postoji mnogo razloga zašto nam je potrebna promjena konteksta neke funkcije, a glavni je da nam *this* referencira na nešto drugo a ne na scope u kojem se nalazi. Za promjenu konteksta postoje tri metode: *.call()*, *.apply()* i *.bind()*.

call() sintaksa:

```
someMethod.call(anotherScope, arg1, arg2);
```

Primjer:

```
function logName() {
  console.log(this.name);
};
```

```
var persons = [{
  name: 'Ante'
}, {
  name: 'Marko'
}, {
  name: 'Ivan'
}];

var i;
var len = persons.length;
for (i = 0; i < len; i++) {
  logName.call(persons[i]); // Promjena konteksta preko call metode
}
```

apply() sintaksa:

```
someMethod.call(anotherScope, [arg1, arg2]);
```

Jedina razlika između *call()* i *apply()* metode je način na koji se prosljeđuju argumenti.

bind() sintaksa:

```
someMethod.bind(anotherScope);
```

bind() ima isti efekt kao i *call()*, jedina razlika je što *bind()* neće pozvati funkciju, već će samo promijeniti kontekst.

4. Zadatak:

Inicijalizirati objekt *time* sa propertijem *seconds* i dodijeliti neku numeričku vrijednost. Objektu dodati metodu *convertToMin()* koja će preračunati sekunde u minute i sekunde. Metoda treba objektu dodati novi property *minutes*, u njega pohraniti broj minuta te ažurirati broj sekundi u property *seconds*. Dodati metodu *convertToSec()* koja preračunava minute i sekunde natrag u sekunde te potom ažurira objekt. Dodati metodu *addSeconds()* koja prima broj sekundi. Metoda treba (koristeći metode *convertToSec()* i *convertToMin()*) ažurirati stanje minuta i sekundi. U glavnom programu zatražiti unos broja sekundi i poslati ga metodi.

Rješenje:

```
var time = {
  seconds: 112
};

time.convertToMin = function() {
  this.minutes = parseInt(this.seconds / 60);
  this.seconds = this.seconds % 60;
};

time.convertToSec = function() {
  this.seconds = this.seconds + (this.minutes * 60);
  this.minutes = 0;
};

time.addSeconds = function(sec) {
  this.convertToSec();
  this.seconds += sec;
  this.convertToMin();
};

console.log(time.seconds);
time.convertToMin();
console.log(time.minutes + ':' + time.seconds);
time.addSeconds(parseInt(prompt('Unesite broj sekundi:')));
console.log(time.minutes + ':' + time.seconds);
```

7.3 Kopiranje po vrijednosti/referenci

Glavna razlika između referentnih tipova podataka (niz, objekt) i primitivnih tipova podataka je u tome što se referentni tipovi podataka u memoriji spremaju po referenci, a primitivni tipovi podataka po vrijednosti. Razliku možemo vidjeti iz sljedeća dva primjera.

Primjer kopiranja po vrijednosti:

```
var num1 = 5;
var num2 = num1;
num1 = 10;
```

```
console.log('num1: ', num1); // num1: 10
console.log('num2: ', num2); // num2: 5
```

Primjer kopiranja po referenci:

```
var num1 = { val: 5 };
var num2 = num1;
num1.val = 10;

console.log('num1.val: ', num1.val); // num1.val: 10
console.log('num2.val: ', num2.val); // num2.val: 10
```

7.4 Iteracija kroz propertije objekta

Za iteraciju kroz propertije objekta koristimo **for...in** petlju. Ova petlja je najsporija petlja u JavaScript-u.

Sintaksa:

```
var key, value;
for (key in object) {
  value = object[key];
}
```

5. Zadatak:

Inicijalizirati prazan objekt *knjiga*. Dodijeliti mu propertije *autor*, *ime, godina*. Koristeći for … in petlju, ispisati propertije na sljedeći način "ključ - vrijednost".

Rješenje:

```
var knjiga = {};
knjiga.autor = 'Mato Lovrak';
knjiga.ime = 'Vlak u snijegu';
knjiga.godina = 1976;

var key, value;
for (key in knjiga) {
  value = knjiga[key];
  console.log(key + ' - ' + value);
}
```

Osim *for...in* petlje moćemo koristiti **Object.keys(objectName).forEach()** strukturu za interaciju propertija nekog objekta. *Object.keys(objectName)* nam vraća niz ključeva nekog objekta, a pomoću *.forEach()* prolazimo kroz svaki element niza.

Sintaksa:

```
Object.keys(objectName).forEach(function(key) {
  var value = objectName[key];
});
```

Primjer:

```
var tv = {
  size: '119in',
  brand: 'LG',
  type: 'LED'
};
var keys = Object.keys(tv);
console.log(keys); // ["size", "brand", "type"]

keys.forEach(function(key) {
  var value = tv[key];
  console.log(key + ' - ' + value);
});
```

6. Zadatak:

Zadan je objekt *movie*. Ispisati sve propertije objekta u formatu "ključ - vrijednost". Potrebno ispisati sve glumce čije prezime počinje sa slovom 'c'.

Rješenje:

```
var movie = {
  name: 'The Godfather',
  duration: 175,
  rating: 9.2,
  actors: ['Marlon Brando', 'Al Pacino', 'James Caan', 'Richard Castellano']
};

Object.keys(movie).forEach(function(key) {
  console.log(key + ' - ' + movie[key]);
```

```
});

var lastName, i, len;

len = movie.actors.length;
for (i = 0; i < len; i++) {
  lastName = movie.actors[i].split(' ')[1];

  if (lastName.toLowerCase().slice(0, 1) === 'c') {
    console.log(movie.actors[i]);
  }
}
```

7. Zadatak:

Zadan je niz objekata *movies*. Ispisati imena svih filmova. Usporediti ocjene svih filmova i prema tome ispisati ime filma sa najvišom ocjenom.

Rješenje:

```
var movies = [{
  name: 'The Godfather',
  duration: 175,
  rating: 9.2,
  actors: ['Marlon Brando', 'Al Pacino', 'James Caan', 'Richard Castellano']
}, {
  name: 'The Shawshank Redemption',
  duration: 142,
  rating: 9.3,
  actors: ['Tim Robbins', 'Morgan Freeman', 'Bob Gunton']
}];

var highestRated = movies[0];
var movie, i, len;

len = movies.length;
for (i = 0; i < movies.length; i++) {
  movie = movies[i];

  if (movie.rating > highestRated.rating) {
    highestRated = movie;
  }

  console.log(movie.name);
```

```
}

console.log('Film s najvišom ocjenom je ' + highestRated.name);
```

7.5 Objekt konstruktor i paterni

Najčešće korišten način za stvaranje objekata je preko objekt konstruktora. Konstruktor je funkcija korištena za inicijaliziranje novog objekta, i koristimo ključnu riječ **new** da pozovemo konstruktor.

Primjer:

```
var person = new Object();
person.name = 'Anna';
person.age = 24;

console.log(person); // { name: "Anna", age: 24 }
```

Često se koristi **konstruktor patern** za stvaranje objekata. Stvaramo konstruktor funkcije čije prvo slovo je veliko slovo iz razloga da znamo da se radi o konstruktor funkciji (klasi).

Primjer:

```
function Person(name, age) {
  this.name = name;
  this.age = age;
};

var p1 = new Person('Anna', 24);
var p2 = new Person('Ema', 23);

console.log(p1); // { name: "Anna", age: 24 }
console.log(p2); // { name: "Ema", age: 23 }
console.log(p1.name) // Anna

p1.name = 'Mike';
console.log(p1.name); // Mike
console.log(p1); // { name: "Mike", age: 24 }
```

Pomoću ključne riječi *new* i poziva konstruktora, stvorili smo dva neovisna objekta. Konstruktor opisuje objekt koji će biti napravljen. *this* varijabla uvijek referencira na novi objekt napravljen od konstruktora i preko *this* objekta pravimo **public** propertije. Konstruktor može imati i privatne varijable, kojima ne možemo pristupiti izvan objekta.

Primjer:

```
function Car () {
  // Privatna varijabla
  var speed = 10;

  // Public metoda
  this.accelerate = function(change) {
    speed += change;
  };

  this.decelerate = function() {
    speed -= 5;
  };

  this.getSpeed = function() {
    return speed;
  };
};

var c = new Car();
console.log(c.speed); // undefined
console.log(c.getSpeed()); // 10

c.accelerate(2);
console.log(c.getSpeed()); // 12
```

Propertije možemo dodavati koristeći **prototype patern**.

Primjer:

```
function Car() {
  this.speed = 10;
};

Car.prototype.accelerate = function(change) {
  this.speed += change;
};
```

```
Car.prototype.decelerate = function() {
  this.speed -= 5;
};

var c = new Car();
console.log(c.speed); // 10

c.accelerate(2);
console.log(c.speed); // 12
```

8. Zadatak:

Napraviti algoritam koji će generirati špil karata i nasumično ga podijeliti na dva jednaka dijela, te izvući jednu kartu s vrha iz oba dijela špila i odrediti koji dio ima veću kartu. U slučaju da je karta ista, izvlači se nova sve dok jedan dio špila ne bude jači. As je najjača karta.

Rješenje:

```
// Konsstruktor za kartu
function Card(rank, suit) {
  this.rank = rank;
  this.suit = suit;
}

// Konstruktor za špil
function Deck() {
  this.deck = [];

  this.create();
}

// Napravi špil
Deck.prototype.create = function() {
  var self = this;
  var ranks = ['A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q',
'K'];
  var suits = ['Club', 'Diamond', 'Heart', 'Spade'];

  ranks.map(function(rank) {
    suits.map(function(suit) {
      self.deck.push(new Card(rank, suit));
```

```
    });
  });
};

// Promiješaj špil
Deck.prototype.shuffle = function() {
  var i, j, len, temp;

  len = this.deck.length;
  for (i = 0; i < len; i++) {
    j = Math.floor(Math.random() * i);
    temp = this.deck[i];
    this.deck[i] = this.deck[j];
    this.deck[j] = temp;
  }
};

// Izvuci jaču kartu
Deck.prototype.getHighestCard = function() {
  // Izračunaj vrijednost karte
  function getValue(simbol) {
    return {
      'A': 15,
      '2': 2,
      '3': 3,
      '4': 4,
      '5': 5,
      '6': 6,
      '7': 7,
      '8': 8,
      '9': 9,
      '10': 10,
      'J': 12,
      'Q': 13,
      'K': 14
    }[simbol];
  }

  var len = this.deck.length;
  var half = len / 2;
  var halfs = {};
  var i, left, right;

  // Razdoji špilove
  halfs.left = this.deck.slice(0, half);
  halfs.right = this.deck.slice(half);
  console.log('Lijevi špil: ', halfs.left);
```

```
  console.log('Desni špil: ', halfs.right);

  // Ispisati jaču kartu
  for (i = 0; i < len; i++) {
    left = getValue(halfs.left[i].rank);
    right = getValue(halfs.right[i].rank);

    if (left > right) {
      console.log('Pobijeđuje karta iz lijevog špila: ' + halfs.left[i].rank +
' - ' + halfs.right[i].rank);
      break;
    } else if (left < right) {
      console.log('Pobijeđuje karta iz desnog špila: ' + halfs.left[i].rank +
' - ' + halfs.right[i].rank);
      break;
    } else {
      console.log('Karte su iste: ' + halfs.left[i].rank + ' - ' +
halfs.right[i].rank);
    }
  }
};

var deck = new Deck();
console.log('Špil: ', deck); // { deck: Array[52] }
deck.shuffle();
deck.getHighestCard();
```

Zadaci za ponavljanje:

1. Napravite niz objekata, gdje svaki objekt definira knjigu sa propertijima *naslov*, *autor* i *procitana* (boolean koji definira je li knjiga pročitana). Za svaku knjigu ispišite naslov i autora u obliku "*The Hobbit od J.R.R. Tolkein"*, a ako je knjiga pročitana ispisati u obliku *'Već ste čitali "The Hobbit od J.R.R. Tolkein".*

8. AJAX

AJAX dozovoljava da preko JavScript-a dohvatimo podatke sa servera, a da ne osvježimo stranicu. U početku *AJAX* je označavaio *asinkroni JavaScript i XML* (XML je format kojim su komunicirali server i browser). Danas *AJAX* označava patern za slanje i primanje podataka, a da korisnik ne osvježi čitavu stranicu. Danas imamo mnogo interaktivnije web aplikacije koje se ponašaju slično kao nativne aplikacije.

8.1 XMLHttpRequest

XMLHttpRequest objekt omogućava developerima da jednostavno otvore HTTP konekciju i dohvate podatke sa servera. Kada napravimo XMLHttpRequest objekt, spremni smo da dohvatamo podatke. Prvi korak je da pozovemo *open()* metodu da inicijalizira objekt.

```
var xhttp = new XMLHttpRequest();
xhttp.open(requestType, url, async);
```

- *requestType* - string vrijednost koja predstavlja tip zahtjeva kojeg želimo napraviti (GET, POST)
- *url* - adresa na koju šaljemo zahtjev
- *async* - sinkroni ili asinkroni mod (true, false)

Sljedeći korak je da pošaljemo request koristeći metodu *send()*. Ova metoda prima jedan argument, string koji predstavlja *body* poruke.

Svaki XMLHttpRequest objekt ima *status* property. Ovaj property sadržava HTTP statusni kod poslan sa serverskom porukom. Server će vratiti 200 za uspješan zahtjev ili 404 ako ne može pronaći datoteku.

Primjer sinkronog zahtjeva:

```
var xhttp = new XMLHttpRequest();
xhttp.open('GET', 'http://localhost/textFile.txt', false);
xhttp.send();

if (xhttp.status === 200) {
  console.log('Pronadjen tekstni dokument!');
} else if (xhttp.status === 404) {
  console.log('Tekstni dokument nije pronadjen!');
} else {
  console.log('Server je vratio' + xhttp.status + 'statusni kod!');
}
```

Prošli primjer je bio primjer sinkonog zahtjeva. Kod asinkonog zahtjeva moramo hendlati *readystatechange* event. U asinkronom zahtjevu moramo gledati i na *readyState* property koji sadržava numeričku vrijednost koja odgovara određenom stanju.

readyState:

- 0 - objekt je napravljen ali *open()* metoda nije pozvana
- 1 - *open()* metoda je pozvana ali zahtjev nije poslan
- 2 - zahtjev je poslan; zaglavlja i status su dostupni
- 3 - odgovor je primljen sa servera
- 4 - zahtjevani podaci su uspješno primljeni

Readystatechange event se mijenja svaki put kada *readyState* promijeni stanje.

Primjer asinkronog zahtjeva:

```
var xhttp = new XMLHttpRequest();
xhttp.open('GET', 'http://localhost/textFile.txt', true);

xhttp.onreadystatechange = function() {
  if (xhttp.status === 200 && xhttp.readyState === 4) {
    console.log(xhttp.responseText); // Odgovor sa servera u tekst formatu
  }
}

xhttp.send();
```

8.2 JSON

JSON (**J**ava**S**cript **O**bject **N**otation) je format za spremanje i prijenos podataka i često korišten format za slanje podataka sa servera.

Primjer jednostavnog JSON podatka:

```
{
  "students": [{
    "name": "John",
    "age": 25
  }, {
    "name": "Anna",
    "age": 24
  }, {
    "name": "Peter",
    "age": 24
  }]
}
```

JSON format je sintaktički identičan kodu za stvaranje JavScript objekata. Zbog sličnosti JavaScript program može jednostavni konvertirati JSON podatke u nativne JavScript objekte koristeći **JSON.parse(request.responseText)**.

1. Zadatak:

Napišite funkciju za asinkroni dohvat podataka sa servera. Dohvatite JSON sa linka: http://output.jsbin.com/yoluyoqowa.json . Dinamički pronađite prosječan rejting igre i najrjeđe platforme za koju je igra objavljena. Dinamički provjerite koji se dani u tjednu puštanja igrice ponavljaju. Koji je najčešći mjesec puštanja igrice?

Primjer JSON podatka:

```
[
  {
    "name": "Grand Theft Auto V",
    "rating": 9.77,
    "platforms": ["PlayStation", "Xbox", "Windows"],
    "released": "2015-04-14"
```

```
  },
  {
    "name": "Life is strange",
    "rating": 8.31,
    "platforms": ["PlayStation", "Xbox", "Windows", "OS X", "Linux"],
    "released": "2015-10-20"
  },
  {
    "name": "Fallout 4",
    "rating": 9.44,
    "platforms": ["PLAYSTATION", "XBOX", "WINDOWS"],
    "released": "2015-11-10"
  },
  {
    "name": "Civilization VI",
    "rating": 9.40,
    "platforms": ["Windows"],
    "released": "2016-10-21"
  },
  {
    "name": "The Witcher 3",
    "rating": 9.88,
    "platforms": ["PlayStation", "XBox", "Windows"],
    "released": "2015-05-19"
  }
]
```

Rješenje:

```
// Funkcija za dohvat podataka
function load(url, callback) {
    var xhttp = new XMLHttpRequest();

    xhttp.onreadystatechange = function (response) {
        if (xhttp.readyState === 4 && xhttp.status === 200) {
            if (callback) callback(JSON.parse(xhttp.response));
        }
    };

    xhttp.open('GET', url, true);
    xhttp.send();
}

// Obrada podataka
function parse(data) {
    console.log('igre: ', data);
```

```
    calculateAvgRating(data);
    checkReleaseDays(data);
}

load('https://output.jsbin.com/yoluyoqowa.json', parse);


// Dinamički pronađite prosječan rejting igre i najrjeđe platforme
// za koju je igra objavljena
function calculateAvgRating(data) {
    var avg = 0;
    var platforms = {};

    // Izračunaj rejting
    data.map(function (game) {
        avg += game.rating;

        // Napravi/ažuriraj platforms objekt
        game.platforms.map(function (platform) {
            var p = platform.toLowerCase();

            if (!platforms[p]) platforms[p] = 0;
            platforms[p]++;
        });
    });

    // Postavi prosječan rejting
    avg /= data.length;
    console.log('Prosječan rejting igre: ', avg);

    // Izračunaj najrijeđu platformu
    var rarest = [];
    var min;
    Object.keys(platforms).forEach(function (platform) {
        var value = platforms[platform];

        if (!min) {
            min = value;
            rarest.push(platform);
        } else if (value < min) {
            min = value;
            rarest = [];
            rarest.push(platform);
        } else if (min === value) {
            rarest.push(platform);
        }
```

```
    });
    console.log('Najrjeđe platforme za koju je igra objavljena: ', rarest);
}

// Dinamički provjerite koji se dani u tjednu puštanja igrice ponavljaju.
// Koji je najčešći mjesec puštanja igrice?
function checkReleaseDays(data) {
    var months = {
        0: 'Siječanj',
        1: 'Veljača',
        2: 'Ožujak',
        3: 'Travanj',
        4: 'Svibanj',
        5: 'Lipanj',
        6: 'Srpanj',
        7: 'Kolovoz',
        8: 'Rujan',
        9: 'Listopad',
        10: 'Studeni',
        11: 'Prosinac'
    };
    var days = {
        0: 'Nedjelja',
        1: 'Ponedjeljak',
        2: 'Utorak',
        3: 'Srijeda',
        4: 'Četvrtak',
        5: 'Petak',
        6: 'Subota'
    };
    var repetitiveDays = {};
    var repetitiveMonths = {};

    // Izračunja ponavljajuće dane/mjesece
    data.map(function (game) {
        var d = new Date(game.released);
        var day = days[d.getDay()];
        var month = months[d.getMonth()];

        if (!repetitiveDays[day]) repetitiveDays[day] = 0;
        repetitiveDays[day]++;

        if (!repetitiveMonths[month]) repetitiveMonths[month] = 0;
        repetitiveMonths[month]++;
    });
    console.log('Dani puštanja igrice koji se ponavljaju: ', repetitiveDays);
```

```
    // Izračunaj najčešće ponavljajući mjesec
    var frequent = [];
    var max;
    Object.keys(repetitiveMonths).forEach(function (month) {
        var value = repetitiveMonths[month];

        if (!max) {
            max = value;
            frequent.push(month);
        } else if (value > max) {
            max = value;
            frequent = [];
            frequent.push(month);
        } else if (max === value) {
            frequent.push(month);
        }
    });
    console.log('Najučestaliji mjeseci: ', frequent);
}
```

Zadaci za ponavljanje:

1. 1. Napišite funkciju za asinkroni dohvat podataka sa servera. Dohvatite JSON sa linka: http://output.jsbin.com/xorezehoye.json. 2. Napravite funkciju koja prima 2 parametra: podatke koje ste dohvatili sa servera i callback. Funkcija treba sortirati dobijene podatke ("amounts") po id-u, i ispisati tekst, nakon sortiranja, (za svaki podatak) u obliku "id - ime" ("ime" se nalazi u propertiju meta), na kraju pozvati callback i vratit se u glavni program. 3. Napravite novu funkciju koja prima 2 parametra: podatke koje ste dohvatili sa servera i callback. Funkcija treba sortirati dobijene podatke ("amounts") po "totalAmount", nakon sortiranja ispisati top 3 podatka i pozvati callback. 4. Napraviti funkciju koja će sumirati uplatu po danu uplate (izvući dan iz timestampa -> datetime) i ispisati ih.

Primjer JSONa:

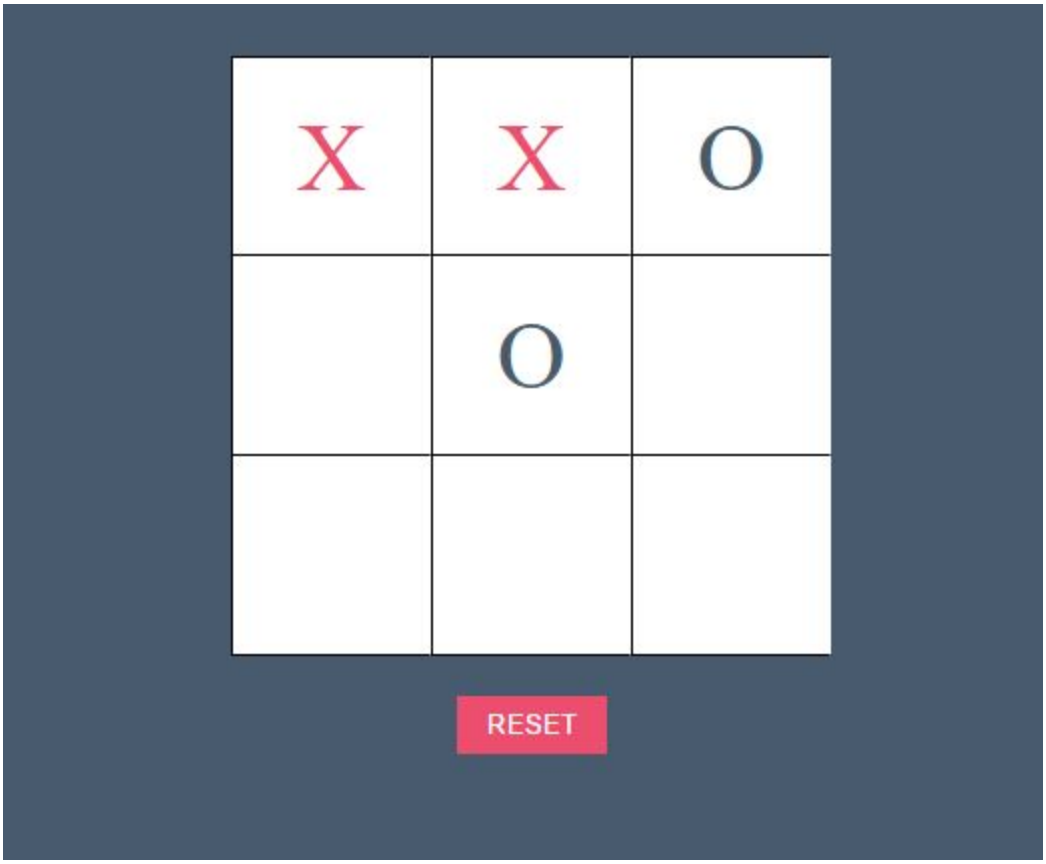
```
{
      "amounts": [
            {
                  "id": 1,
                  "datetime": 1477387516826,
                  "totalAmount": "11234.58",
```

```
            "tickets": [11111, 22222, 77777],
            "totalPayout": 587.21
        },
        {
            "id": 6,
            "datetime": 1477327516826,
            "totalAmount": "5878.67",
            "tickets": [77777, 33333],
            "totalPayout": 12.33
        },
        {
            "id": 4,
            "datetime": 1477136516826,
            "totalAmount": "489897.22",
            "tickets": [44444, 88888, 11111],
            "totalPayout": 8797.00
        },
        {
            "id": 2,
            "datetime": 1476036516826,
            "totalAmount": "2.56",
            "tickets": [99999]
        },
        {
            "id": 8,
            "datetime": 1472365166826,
            "totalAmount": "12.57",
            "tickets": [12345, 67890],
            "totalPayout": 4.00
        },
        {
            "id": 3,
            "datetime": 1476536516826,
            "totalAmount": "102.57",
            "tickets": [10203, 40506, 70809],
            "totalPayout": 23.30
        },
        {
            "id": 11,
            "datetime": 1477336516826,
            "totalAmount": "55.78",
            "tickets": [11213, 21314],
            "totalPayout": 21.00
        },
        {
            "id": 7,
            "datetime": 1476236516826,
```

```
                  "totalAmount": "878.24",
                  "tickets": [11213, 89764],
                  "totalPayout": 157.00
            },
            {
                  "id": 5,
                  "datetime": 1477036516826,
                  "totalAmount": "0",
                  "tickets": []
            }
      ],
      "meta": {
            "1": "Soccer",
            "2": "Voleyball",
            "3": "Snooker",
            "4": "Tennis",
            "5": "Table tennis",
            "6": "Basketball",
            "7": "Ice Hockey",
            "8": "Handball",
            "11": "Baseball"
      }
}
```

9. Aplikacija

Aplikacija će biti X-O igrica, koja će izgledati kao na slici ispod. Koristit će mo do sada usvojeno znanje o JavaScript-u, i jednostavan kod za HTML i CSS.



Slika 4. Prikaz X-O igrice

game.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>X-O igrica</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
```

```
    <div class="container">
        <ul id="game">
            <li data-pos="0,0"></li>
            <li data-pos="0,1"></li>
            <li data-pos="0,2"></li>
            <li data-pos="1,0"></li>
            <li data-pos="1,1"></li>
            <li data-pos="1,2"></li>
            <li data-pos="2,0"></li>
            <li data-pos="2,1"></li>
            <li data-pos="2,2"></li>
        </ul>
        <button id="reset-game">Reset</button>
    </div>

    <script src="game.js"></script>
</body>

</html>
```

Prikazana je jednostavna HTML struktura gdje smo koristili **ul** element za prikaz kvadrata. Na **li** elementima se nalazi **data-pos** atribut koji će nam pomoći za popunjavanje kliknutih kvadrata.

style.css

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    background-color: #485b6e;
}

.container {
    width: 300px;
    margin: 50px auto;
}

#game {
    margin-bottom: 20px;
    background-color: #fff;
    overflow: hidden;
```

```
}

#game > li {
    list-style: none;
    float: left;
    width: 100px;
    height: 100px;
    line-height: 100px;
    border: 1px solid #000;
    border-right: 1px solid #fff;
    cursor: pointer;
    font-size: 3em;
    text-align: center;
    text-transform: uppercase;
}

#game > li:nth-child(n + 4) {
    border-top: 0;
}

#game > li:hover {
    background: #f1f1f1;
}

#game > li.x {
    color: #ed4e6e
}

#game > li.o {
    color: #485b6e;
}

#reset-game {
    display: block;
    margin: 0 auto;
    border: 0;
    padding: 7px 15px;
    color: #fff;
    background-color: #ed4e6e;
    text-transform: uppercase;
    text-align: center;
    cursor: pointer;
    outline: 0;
}

#reset-game:hover {
    background: #2c3e52;
}
```

Osnovna CSS struktura.

game.js

```
// Definiraj globalne varijable
var elems; // Spremi DOM elemente
var board; // Ploča
var turns; // Trenutni potez
var context; // Kontekst igre (x/o)
var currentContext; // Trenutni kontekst
var message; // Poruka o statusu igre

// Inicijaliziraj igru
var init = function () {
    // Postavi početne vrijednosti
    message = '';
    turns = 0;
    board = set().board(); // Definiraj 3x3 ploču
    context = set().context(); // Postavi kontekst igre

    currentContext = get().currentContext(); // Postavi trenutni kontekst
    elems = get().elements(); // Spremi DOM elemente

    bindClickEvents(); // Zakači click evente
};

// Postavi ploču/kontekst
var set = function () {
    // Postavi ploču
    function board() {
        return [
          [undefined, undefined, undefined],
          [undefined, undefined, undefined],
          [undefined, undefined, undefined]
        ];
    }

    // Postavi kontekst
    function context() {
        return {
            player1: 'x',
            player2: 'o'
        };
    }

    return {
```

```
        board: board,
        context: context
    };
};

// Dohvati elemente/trenutni kontekst
var get = function () {
    function elements() {
        return {
            game: document.getElementById('game'),
            boxes: document.querySelectorAll('li'),
            resetGame: document.getElementById('reset-game')
        };
    }

    // Izračunaj koji igrač je na potezu
    function currentContext() {
        return (turns % 2 === 0) ? context.player1 : context.player2;
    }

    return {
        elements: elements,
        currentContext: currentContext
    };
}

// Zakači click evente
var bindClickEvents = function () {
    Object.keys(elems.boxes).forEach(function (key) {
        var box = elems.boxes[key];
        box.addEventListener('click', clickHandler);
    });

    // Zakači click event na reset dugme
    elems.resetGame.addEventListener('click', resetGameHandler);
};

// Definiraj click handlere
var clickHandler = function () {
    this.removeEventListener('click', clickHandler);

    this.className = currentContext;
    this.innerHTML = currentContext;

    var pos = this.getAttribute('data-pos').split(',');
    board[pos[0]][pos[1]] = currentContext === 'x' ? 1 : 0;
```

```
    // Provjeri “won” status igre
    if (checkStatus()) {
        gameWon();
        return;
    }

    // Osvježi varijable
    turns++;
    currentContext = get().currentContext();
};

// Definiraj handler za resetiranje igre
var resetGameHandler = function () {
    // Clear board
    Object.keys(elems.boxes).forEach(function (key) {
        var box = elems.boxes[key];
        box.className = '';
        box.innerHTML = '';
    });

    clearEvents();
    init();
};

// Očisti click eventove
var clearEvents = function () {
    Object.keys(elems.boxes).forEach(function (key) {
        var box = elems.boxes[key];
        box.removeEventListener('click', clickHandler);
    });
};

// Provjeri status igre
var checkStatus = function () {
    var status = false; // Status igre (true -> won)
    var usedBoxes = 0;

    // Izračunaj poene redaka, stupaca (X = 1; O = 0;)
    board.map(function (row, i) {
        var rowTotal = 0;
        var colTotal = 0;

        row.map(function (col, j) {
            rowTotal += board[i][j];
            colTotal += board[j][i];

            if (typeof board[i][j] !== 'undefined') usedBoxes++;
```

```
        });

        // Provejri horizontalno/vertikalno
        if (rowTotal === 0 || rowTotal === 3 || colTotal === 0 || colTotal ===
3) status = true;

        // Provjeri dijagonalno
        var diagonalLeft = board[0][0] + board[1][1] + board[2][2];
        var diagonalRight = board[0][2] + board[1][1] + board[2][0];

        if (diagonalLeft === 0 || diagonalLeft === 3 || diagonalRight === 0 ||
diagonalRight === 3) status = true;
    });

    // Provejri “draw” status igre
    if (!status && usedBoxes === 9) {
        gameDraw();
        return;
    }

    return status;
};

// Igra dobivena
var gameWon = function () {
    switch (currentContext) {
        case 'x':
            message = 'Player 1 won the game!';
            break;
        case 'o':
            message = 'Player 2 won the game!';
            break;
    }

    alert(message);
    clearEvents();
};

// Igra neriješena
var gameDraw = function () {
    message = 'Game draw!';
    alert(message);
    clearEvents();
};

init();
```

Rješenja zadataka

3. Kontrola toka:

1. Zadatak:

```
var a = parseInt(prompt('Upišite prvi broj:'));
var b = parseInt(prompt('Upišite drugi broj:'));

if (a % 2 === 0 && b % 2 === 0) {
  console.log('Oba broja su parna');
} else if (a % 2 === 1 && b % 2 === 1) {
  console.log('Oba broja su neparna');
} else {
  console.log('Jedan broj je paran, a jedan neparan');
}
```

2. Zadatak:

```
var temp = 30;
var choice = prompt('Ispis temperature u ');

if (choice === 'C') {
  console.log(temp); // 30
} else if (choice === 'F') {
  console.log(temp * (9 / 5) + 32); // 86
}
```

4. Petlje:

1. Zadatak:

```
var a, b;

do {
  a = parseInt(prompt('Unesite prvi troznamenkasti broj:'));
  b = parseInt(prompt('Unesite drugi troznamenkasti broj:'));
} while (((a / 100) < 1 || (a / 100) > 10) && ((b / 100) < 1 || (b / 100) >
10))

// Provjeri i zamijeni ako je prvi broj veći od drugog
if (a > b) {
  var temp = a;
```

```
  a = b;
  b = a;
}

// Ispis parnih brojeva
for (var i = a; i < b; i++) {
  if (i % 2 === 0) console.log(i);
}
```

2. Zadatak:

```
var sum = 0;

while (sum < 150) {
  sum += parseInt(prompt('Unesite broj'));
}
console.log('Zbroj: ', sum);

sum = 150;
while(sum) {
      console.log(sum--);
}
```

5. Funkcije:

1. Zadatak:

```
function sumPrimes(num) {
  var sum = 0;

  // Provjeri je li broj prost
  var isPrime = function(val) {
    for (var i = 2; i < val; i++) {
      if (val % i === 0) return false;
    }

    return true;
  };

  while (num > 1) {
    sum += isPrime(num) ? num : 0;
    num--;
  }

  return sum;
```

```
}

console.log(sumPrimes(100)); // 1060
```

2. Zadatak:

```
function exp(b, n) {
  var rez = 1;

  for (var i = 1; i <= n; i++) {
    rez = b * rez;
  }

  return rez;
}
console.log(exp(2, 3)); // 8
```

6. Nizovi:

1. Zadatak:

```
function reverseString(str) {
  return str.split('').reverse().join('');
}

console.log(reverseString("danas")); // sanad
```

2. Zadatak:

```
function findLongestWord(str) {
  var words = str.split(' ');
  var maxLen = words[0].length;
  var i;

  for (i = 1; i < words.length; i++) {
    maxLen = words[i].length > maxLen ? words[i].length : maxLen;
  }
  return maxLen;
}

console.log(findLongestWord("Danas je lijep dan za programiranje.")); // 14
```

3. Zadatak:

```
function titleCase(str) {
  str = str.split(' ').map(function(word) {
    word = word.toLowerCase();
    return word.charAt(0).toUpperCase() + word.slice(1);
  }).join(' ');
  return str;
}

console.log(titleCase('Danas je lijep dan za programiranje.'));
```

7. Objekti:

1. Zadatak:

```
// Book konstruktor
function Book(title, author, alreadyRead) {
  this.title = title;
  this.author = author;
  this.alreadyRead = alreadyRead;
}

var books = [];
var title, author, alreadyRead;

for (var i = 0; i < 5; i++) {
  title = prompt('Naslov knjige:');
  author = prompt('Autor:');
  alreadyRead = prompt('Da li ste pročitali knjigu? (DA/NE)').toLowerCase()
=== 'da';

  books.push(new Book(title, author, alreadyRead));
}

books.map(function(book) {
  var prefix = 'Već ste čitali ';
  var message = book.title + ' od ' + book.author;

  if (book.alreadyRead) {
    console.log(prefix + message);
  } else {
    console.log(message);
  }
});
```

8. AJAX:

1. Zadatak:

```
// 1. dio
function load(url, callback) {
  var xhttp = new XMLHttpRequest();

  xhttp.onreadystatechange = function(response) {
    if (xhttp.readyState === 4 && xhttp.status === 200) {
      if (callback) callback(JSON.parse(xhttp.response));
    }
  }

  xhttp.open('GET', url, true);
  xhttp.send();
}

var amountCounter = 0;

function parse(data) {
  console.log('data: ', data);

  if (data) {
    sortById(data, parse);
    sortByTotalAmount(data);
    sumByDays(data);
  }
}

load('https://output.jsbin.com/xorezehoye.json', parse);

// 2. dio
function sortById(data, callback) {
  console.log('-------------> 2. Sortiranje podataka (id) ...');

  // Sortiraj amounts po id
  data.amounts.sort(function(a, b) {
    return a.id - b.id;
  });

  // Ispis u formatu "id - ime"
  data.amounts.map(function(amount) {
    var id = amount.id;
    console.log(id + ' - ' + data.meta[id]);
  });
```

```
  if (callback) callback();
}

// 3. dio
function sortByTotalAmount(data, callback) {
  console.log('-------------> 3. Sortiranje podataka (totalAmount) ...');

  // Sortiraj amounts po totalAmount
  data.amounts.sort(function(a, b) {
    return parseFloat(a.totalAmount) - parseFloat(b.totalAmount);
  });

  // Ispis prva tri
  var id;
  var len = 3;
  for (var i = 0; i < len; i++) {
    amount = data.amounts[i];
    console.log(data.meta[amount.id] + ' - ' + amount.totalAmount);
  }

  if (callback) callback();
}

// 4. dio
function sumByDays(data) {
  console.log('-----------> 4. Napraviti funkciju koja će sumirati uplatu po
danu uplate ...');
  var days = {
    0: 'Nedjelja',
    1: 'Ponedjeljak',
    2: 'Utorak',
    3: 'Srijeda',
    4: 'Četvrtak',
    5: 'Petak',
    6: 'Subota'
  };
  var payIns = {};

  // Sumiraj uplatu po danu uplate
  data.amounts.map(function(amount) {
    var d = new Date(amount.datetime);
    var day = d.getDay();

    if (!payIns[day]) payIns[day] = 0;
    payIns[day] += parseFloat(amount.totalAmount);
  });
```

```
  // Ispis
  Object.keys(payIns).forEach(function(key) {
    var value = payIns[key];
    console.log(days[key] + ' - ' + value);
  });
}
```