# JavaScript interview questions

1. **What is prototype chain?**

**Prototype chaining** (ulančavanje) se koristi za kreiranje novih tipova objekata na osnovu postojecih. Slicno je nasledjivanju u class based jezicima.

```
// kreiramo object constructor
function Student(name,age){
   this.name = name;
   this.age = age;
}
// kreiramo objekat tipa student
let student1 = new Student('John',32)
let student2 = new Student('Mary',32)

// dodajemo property na instancu  student1
student1.sports = 'Cricket';

// dodajemo property na Student prototype;
// dodace se gender property na sve postojece
// instance objekta od Student objekta i
// inicijalizovace se na null
Student.prototype.gender = null
```

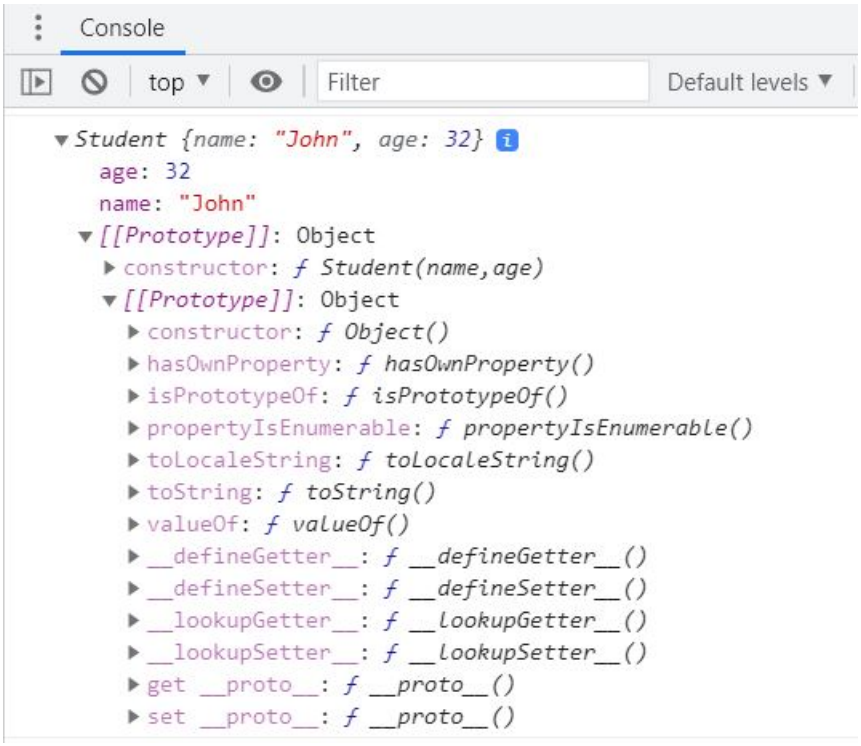
Propertiji definisani u prototype objektu su dostupni preko instance objekta, zbog cega mozemo da pristupimo propertijima koje nismo eksplicitno definisali na objektu, posto su dostupni nasledjivanjem kroz prototype chaining. Kada pristupamo nekom propertiju, prvo se proverava da li postoji u propertijima tog objekta – ako ne, onda se trazi preko prototipa objekta. To traje dokle god ne pronadjemo trazeni properti ili dok ne dostignemo kraj prototype chain-a koji vraca undefined.

Ako imamo vise levela prototype chaining-a, to moze uticati negativno na performanse i vreme izvrsenja. Da bismo izbegli ove situacije, mozemo koristiti **hasOwnProperty()** metod, koji je nasledjen od svih objekata iz object.prototype i koji ogranicava trazenje na odredjenom nivou.

```
let duck = new Dog("Donald");
duck.hasOwnProperty("name"); // yields true
```

Prototype konstruktorske funkcije je dostupan preko **Object.prototype**, a na instanci objekta je dostupan preko **Object.getPrototypeOf(object)** ili **proto** property. Svi objekti u JS imaju prototype. Prototip objekta je takodje objekat. Posto je prototype objekat, on takodje ima svoj prototype. U tom slucaju, prototype od Dog.prototype je Object.prototype.

```
function Dog(name) {
        this.name = name;
}
// Returns true
Object.prototype.isPrototypeOf(Dog.prototype);
```




Na slici vidimo 2 levela prototype chain-a. Prvi level je prototype objekta od Student konstruktora, koji takodje ima prototype objekta od Object konstruktora, koji nema prototype objekta jer je Object top level prototype chain-a. Prototype-u student objekta mozemo pristupiti preko:

- Object.getPrototypeOf(student)
- student._proto_
- Student.prototype

Opcije 2 i 3 ce dati isti rezultat, ali opcija 1 ce dati prototype konstruktorske funkcije - Student, a ne instance objekta - student.

https://www.sudshekhar.com/blog/prototype-and-prototype-chain-in-javascript

**2. What is difference between prototype and proto?**

**Prototip** je objekat iz kog je izveden drugi objekat. Novokreirani objekat ima pristup svim propertijima i metodama prototipa od kojeg je objekat kreiran, a moze imati i neke svoje dodatne propertije i metode.
**Proto** je property preko kojeg pristupamo prototipu instance objekta. The `__proto__` object is the actual object that is used in the lookup chain (lanac trazenja) to resolve methods, etc. whereas `prototype` is the object that is used to build `__proto__` when you create an object.

Razlika između prototipa i __proto__ je u tome što je prvi svojstvo konstruktora klase, dok je drugi svojstvo instance klase.

**3. What is the purpose of array slice method?**

slice()
slice(start)
slice(start, end)

The **slice()** method returns the selected elements in an array as a new array object. It selects the elements starting at the given start argument, and ends at the given optional end argument

without including the last element. **If you omit the second argument then it selects till the end.** Some of the examples of this method are,

```
let arrayIntegers = [1, 2, 3, 4, 5];
let arrayIntegers1 = arrayIntegers.slice(0,2); // returns [1,2]
let arrayIntegers2 = arrayIntegers.slice(2,3); // returns [3]
let arrayIntegers3 = arrayIntegers.slice(4); //returns [5]
```

**Note:** Slice method won't mutate the original array but it returns the subset (podskup) as new array.

### 4. What is the purpose of array splice (spajanje) method?

splice(start)
splice(start, deleteCount)
splice(start, deleteCount, item1)
splice(start, deleteCount, item1, item2, itemN)

The **splice()** method is used either adds/removes items to/from an array, and then returns the removed item. The first argument specifies the array position for insertion (umetanje) or deletion whereas the option second argument indicates the number of elements to be deleted.

```
let array = [1, 2, 3, 4, 5];

let newArray = array.splice(0,2); // returns [1, 2]; original array: [3, 4,
5]
let newArray = array.splice(3); // returns [4, 5]; original array: [1, 2, 3]
let newArray = array.splice(3, 1, "a", "b", "c"); //returns [4]; original
array: [1, 2, 3, "a", "b", "c", 5]

const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb')’ // inserts at index 1: ["Jan", "Feb", "March",
"April", "June"]
months.splice(4, 1, 'May'); // replaces 1 element at index 4: ["Jan", "Feb",
"March", "April", "May"]
```

**Note:** Splice method modifies the original array and returns the deleted array.

### 5. What is the difference between slice and splice?

| Slice | Splice |
|---|---|
| Doesn't modify the original array(immutable) | Modifies the original array(mutable) |
| Returns the subset of original array | Returns the deleted elements as array |
| Used to pick the elements from array | Used to insert or delete elements to/from array |

### 6. How do you compare Object and Map?

**Object** je slican **Mapu** po tome sto oboje omogucuju dodeljivanje kljuceva vrednostima, vracanje tih vrednosti, brisanje kljuceva i ispitivanju da li se nesto nalazi u kljucu. Zbog svega toga Objekti su se ranije koristili kao Mape. Ali Mape su pozeljnije u odredjenim slucajevima:

i. Kljucevi Objekta su stringovi i simboli, dok u Mapi mogu biti bilo koje vrednosti (zajedno sa funkcijama, objektima i any primitive).
ii. Kljucevi u Mapi su sortirani, dok u Objektu nisu, stoga prilikom iteracije kroz kljuceve – Map vraca kljuceve po redosledu umetanja.

iii. Velicinu Mape mozemo dobiti preko size propertija, dok broj propertija u Objektu moramo odrediti rucno.

iv. Kroz Mapu moze direktno da se iterira (preko mapa1.forEach((value, key) => console.log(value,key)), dok Objekat zahteva dobijanje svih kljuceva pa tek iteraciju kroz njih (preko for..in, object.keys, object.entities):

```
// iterate over the user object
for (const key in user) {
   console.log(`${key}: ${user[key]}`);          // name: John Doe

}
```

ili

```
// convert object to key's array
const keys = Object.keys(courses);
console.log(keys);                              // [ 'java', 'javascript', 'nodejs', 'php' ]
```

ili

```
const entries = Object.entries(animals);
console.log(entries);                           // [ [ 'tiger', 1 ], [key, value]
```

v. Objekat ima prototip, tako da postoje podrazumevani ključevi na mapi koji bi mogli da se sudare sa vašim ključevima ako niste pažljivi. This can be skipped by using map = Object.create(null), but this is seldom done.

vi. Mapa moze biti bolja kada treba cesto da se dodaju i uklanjaju parovi kljuceva.

**7. What is the difference between == and === operators?**

JavaScript provides both strict(===, !==) and type-converting(==, !=) equality compariso. The strict operators follow the below conditions for different types:

i. Two strings are strictly equal when they have the same sequence of characters, same length, and same characters in corresponding positions.

ii. Two numbers are strictly equal when they are numerically equal. i.e, Having the same number value. There are two special cases in this,

iii. NaN is not equal to anything, including NaN.

iv. Positive and negative zeros are equal to one another.

v. Two Boolean operands are strictly equal if both are true or both are false.

vi. Two objects are strictly equal if they refer to the same Object.

vii. Null and Undefined types are not equal with ===, but equal with ==. i.e, null===undefined --> false but null==undefined --> true

Some of the example which covers the above cases

```
0 == false   // true
0 === false  // false
1 == "1"     // true
1 === "1"    // false
null == undefined // true
null === undefined // false
'0' == false // true
'0' === false // false
[]==[] or []===[] //false, refer different objects in memory
{}=={} or {}==={} //false, refer different objects in memory
```

**8. What are lambda or arrow functions?**

An arrow function is a shorter syntax for a function expression and does not have its own **this, arguments, super, or new.target**. These function are best suited for non-method functions, and they cannot be used as constructors.

let str1 = () => "this is geeksforgeeks"; // arrow func. without parameters

**9. What is a first class function?**

U JS, funkcije su first class objekti. Programski jezik ima first-class funkcije ako su one tretirane kao bilo koja druga promenljiva (variable). To znaci da su funkcije ustvari vrednosti. Npr funkcije mogu biti prosledjene kao argument drugoj funkciji, mogu biti vracene od strane druge funkcije i moze biti dodeljena kao vrednost nekoj varijabli. U ovom primeru handler f-ja je dodeljena listeneru.

```
const handler = () => console.log ('This is a click handler function');
document.addEventListener ('click', handler);
```

**10. What is a first order function?**

First-order function je funkcija koja ne prihvata druge f-je kao argument i ne vraca f-ju kao return vrednost.

```
const firstOrder = () => console.log ('I am a first order function!');
```

**11. What is a higher order function?**

Higher-order function je funkcija koja prihvata druge f-je kao argument ili vraca f-ju kao return vrednost.

```
const firstOrderFunc = () => console.log ('Hello I am a First order function');
const higherOrder = ReturnFirstOrderFunc => ReturnFirstOrderFunc ();
higherOrder (firstOrderFunc);
```

**12. What is a unary function?**

Unary function (i.e. monadic) je funkcija koja prihvata tacno 1 argument.

```
const unaryFunction = a => console.log (a + 10);
```

**13. What is the difference between let and var?**

| var | let |
|---|---|
| It has function scope | It has block scope |
| Variables will be hoisted | Hoisted but not initialized |

**Hoisting** je default behavior gde se sve deklaracije (var x;) pomeraju ka vrhu trenutnog scope-a (ka vrhu skripte ili funkcije). Pomeraju se samo deklaracije, ne i inicijalizacije (x=5;). Varijable definisane sa let i const su hoistovane ka vrhu bloka, ali nisu inicijalizovane.

```
function userDetails(username) {
if(username) {
          console.log(salary); // undefined(due to hoisting)
          console.log(age); // error: age is not defined

          var salary = 10000;
          let age = 30;
}
```

```
console.log(salary);   //10000 (accessible to due function scope)
console.log(age);       //error: age is not defined(due to block scope)
}
```

**14. How do you redeclare variables in switch block without an error?**

If you try to redeclare variables in a `switch block` then it will cause errors because there is only one block.

```
let counter = 1;
switch(x) {
  case 0:
    let name;
    break;

  case 1:
    let name; // SyntaxError for redeclaration.
    break;
}
```

To avoid this error, you can create a nested block inside a case clause:

```
let counter = 1;
    switch(x) {
      case 0: {
        let name;
        break;
      }
      case 1: {
        let name; // No SyntaxError for redeclaration.
        break;
      }
    }
```

**15. What are modules?**

Moduli nam omogucuju da podelimo kod u odvojene fajlove koji mogu biti reusable i nezavisni jedni od drugih. Oslanjaju se na import i export statements.

**16. What is the benefit of using modules?**

i. Odrzavanje (maintainability)
ii. Reusability
iii. Namespacing (each namespace is just a big variable, that has many properties and methods)

**17. What is memoization?**

Memoizacija je tehnika za ubrzavanje app kesiranjem rezultata skupih poziva funkcija i njihovim vracanjem kada se isti inputi ponovo koriste. Ako se koristi isti input u buducnosti, nece morati da se izvrsava cela f-ja opet, vec ce se vratiti kesovan odgovor iz memorije, sto ce smanjiti vreme izvrsavanja. Skup function call je onaj koji koristi mnogo memorije i vremena prilikom izvrsavanja. Kes memorija je privremeno skladiste podataka.

**18. What are closures?**

Closure je unutrasnja funkcija koja pamti spoljasnje stvari koje se koriste u njoj. Closure ima 3 scope lanca:

i. Own scope, gde su varijable definisane izmedju { }
ii. Outer function's variables
iii. Global variables

```
// Define the closure
function multFn() {
        var mult = 9;
        return function(val) {
                mult = mult * val;
                return mult;
}}

// Use the closure
var mult = multFn();
console.log(mult(18));
```

U ovom primeru, unutrasnja f-ja ima pristup varijablama u spoljasnjem function skopu(multFn).

**19. What is scope in javascript?**

Scope predstavlja pristupacnost varijabli, f-ja i objekata u nekom delu koda u toku runtime-a, tj odredjuje vidljivost varijabli i drugih resursa naseg koda. JS ima 3 tipa scope-a:

- **Block scope** (kreira se pomocu let i const, varijable deklarisane unutar { } nisu dostupne van njih)
- **Function scope** (varijable definisane unutar funkcije nije vidljiva van nje)
- **Global scope** (sve skripte i f-je na web stranici im mogu pristupiti)

**20. What is web storage?**

Web storage je API pomocu kojeg *browseri* skladiste key/value vrednosti lokalno (u sklopu browsera), na bolji nacin nego da se koriste cookie. To znaci da cuvaju podatke bez ukljucivanja servera u pricu. Omogucuje 2 mehanizma cuvanja podataka na klijentu:

i. **Local storage:** cuva podatke za trenutni origin (domen+protokol+port) bez expiration date-a
ii. **Session storage:** cuva podatke za 1 sesiju i podaci se gube kada se tab browsera zatvori

**21. What is a Cookie?**

Podatak koji omogucuje browseru da nas jedinstveno identifikuje. Cuvaju se kao key/value parovi. Primer: `document.cookie = "username=John".`

- Http protokol je **stateless** jer se svaki par zahtev-odgovor tretira nezavisno -> to dalje znaci da server ne cuva info o korisniku koji je poslao zahtev. Ali posto cuva podatke, oni se mogu menjati u zavisnosti od zahteva klijenta.
- *Primer:* ako klijent posalje zahtev za logovanje sa svojim kredencijalima, a nakon toga zahtev za izvrsenje neke funkcionalnosti, ta dva zahteva ce biti nezavisni jedan od drugog i server nece znati da li korisnik ima dozvolu (pristup) da izvrsi drugi zahtev, jer on prethodno zahteva logovanje na sistem. Tu nastupaju cookie gde klijent salje serveru svoje kredencijale prilikom prijave na sistem.
- Cookie je string koji server generise pri prijavi na sistem i klijentu se vrati kao odgovor na prijavu. Ima ogranicen rok vazenja (jedna sesija).
- Server skladišti parove cookie-korisnik, jer na taj način server moze da utvrdi na kojeg korisnika se cookie odnosi.

- Mana cookie-ja jeste ta sto server cuva informacije o sesiji korisnika, a problem nastaje ako aplikaciji pristupa veliki broj korisnika. Resenje ovoga su tokeni. Jos jedna mana je sto su cookie upravljani od strane pregledaca, sto uvodi ogranicenja pri upravljanju komunikacijom.
- Vrste cookie-a: **session**, **persistent** (pamte podesavanja kako smo ih namestili prilikom prve posete sajtu, ugl imaju dugacak vek trajanja) i **third-part cookies** (prate nasa interesovanja, lokacije, pretrage i prosledjuju im oglasivacima kako bi mogli da nam sponzorisu one reklame sto iskacu na instagramu npr)

### 22. Why do you need a Cookie?

Cookies are used to remember information about the user profile(such as username). It basically involves two steps,

i. When a user visits a web page, user profile can be stored in a cookie.
ii. Next time the user visits the page, the cookie remembers user profile.

### 23. What are the options in a cookie?

There are few below options available for a cookie,

i. By default, cookie se brise kada se browser zatvori, ali ovo moze da se promeni tako sto se stavi expiry date:

```
document.cookie = "username=John expires=Sat, 8 Jun 2019 12:00:00 UTC";
```

ii. By default, cookie pripada trenutnoj stranici. Ali mozemo reci browser-u kojoj putanji cookie pripada koristeci path parametar:

```
document.cookie = "username=John path=/services";
```

### 24. How do you delete a cookie?

You can delete a cookie by setting the expiry date as a passed date. You don't need to specify a cookie value in this case. For example, you can delete a username cookie in the current page as below.

```
document.cookie = "username=; expires=Fri, 07 Jun 2019 00:00:00 UTC; path=/;";
```

**Note:** You should define the cookie path option to ensure that you delete the right cookie. Some browsers doesn't allow to delete a cookie unless you specify a path parameter.

### 25. What are the differences between cookie, local storage and session storage?

| Feature | Cookie | Local storage | Session storage |
|---|---|---|---|
| Accessed on client or server side | Both server-side & client-side | client-side only | client-side only |
| Lifetime | As configured using Expires option | until deleted | until tab is closed |
| SSL support | Supported | Not supported | Not supported |
| Maximum data size | 4KB | 5 MB | 5MB |

SSL (Secure Sockets Layer) – tehnologija za safe internet konekciju, sprecavaju hakere da citaju transferovane podatke

1MB = 1024KB

### 26. What is the main difference between localStorage and sessionStorage?

LocalStorage je isti kao i SessionStorage samo sto cuva podatke i kada se browser zatvori i opet otvori (nema expiration time), dok se podaci u sessionStorage obrisu cim se sesija stranice zavrsi.

**27. How do you access web storage?**

Window objekat implementira `WindowLocalStorage` i `WindowSessionStorage` objekte koji imaju `localStorage i sessionStorage propertije (window.sessionStorage).` Oni kreiraju instancu Storage objekta, preko koje se data mogu setovati, preuzeti i obrisati za odredjeni domen i storage type (session ili local). For example, you can read and write on local storage objects as below.

```
localStorage.setItem('logo', document.getElementById('logo').value);
localStorage.getItem('logo');
```

**28. What are the methods available on session storage?**

The session storage provided methods for reading, writing and clearing the session data

```
// Save data to sessionStorage
sessionStorage.setItem('key', 'value');

// Get saved data from sessionStorage
let data = sessionStorage.getItem('key');

// Remove saved data from sessionStorage
sessionStorage.removeItem('key');

// Remove all saved data from sessionStorage
sessionStorage.clear();
```

**29. Why do you need web storage?**

Web storage je bezbedniji i mogu da se skladiste velike kolicine podataka LOKALNO, bez uticaja na performanse sajta. Podaci se nikad ne transferuju ka serveru i preporucuje se vise nego Cookie.

**30. What is a promise?**

Promise je objekat koji moze proizvesti neku vrednost u buducnosti kao resenu vrednost ili kao razlog zasto nije resena (npr network error). Moze biti u 3 stanja: fulfilled, rejected, ili pending.

```
const promise = new Promise(function(resolve, reject) {
  // promise description
})
```

**31. Why do you need a promise?**

Promises hendluju asinhrone operacije. Pruzaju alternativu callback-u smanjujuci callback hell i pisuci clean code.

**32. What are the three states of promise?**

ii. **Pending:** inicijalno stanje promisa pre pocetka operacije

iii. **Fulfilled:** pokazuje da je odredjena operacija zavrsena, completed

iv. **Rejected:** pokazuje da operacija nije zavrsena i dobice se error

**32. What is a callback function?**

Callback je funkcija koja je kao argument prosledjena drugoj funkciji. Ta funkcija je pozvana unutar spoljasnje funkcije da bi zavrsila akciju.

```
function callbackFunction(name) {
  console.log('Hello ' + name);
}

function outerFunction(callback) {
  let name = prompt('Please enter your name.');
  callback(name);
}

outerFunction(callbackFunction);
```

### 33.Why do we need callbacks?

Callback je potreban zato sto je JS event driven jezik. To znaci da on ne ceka response vec nastavlja izvrsavanje dok slusa druge dogadjaje.

```
function firstFunction(){

  // Simulate a code delay
  setTimeout( function(){
    console.log('First function called');
  }, 1000 );
}
function secondFunction(){
  console.log('Second function called');
}
firstFunction();
secondFunction();

Output
// Second function called
// First function called
```

U primeru vidimo da JS nije cekao response od prve f-je. Callback bi ovde imao ulogu da se uveri da se drugi deo koda nece izvrsiti dokle god prvi deo koda ne zavrsi svoje izvrsavanje..

### 34.What is a callback hell?

Callback hell je anti-pattern sa vise ugnjezdenih callback-a, koji cine kod teskim za citanje i debagovanje kada se radi sa asinhronom logikom. Asinhroni kod se ne izvrsava redom kojim je kod ispisan, vec se stvaraju thredovi koji se izvrsavaju istovremeno tako da se izvrsavanje ne pauzira zbog funkcija kojima treba duze vremena za izvrsenje.

```
async1(function(){
    async2(function(){
        async3(function(){
            async4(function(){
                ....
            });
        });
    });
});
```

### 35.What are the main rules of promise?

v. Promise je objekat koji ima .then( ) metodu – ona se pokrece nakon sto promise bude resolve-ovan

vi. Pending promise je tranzicija ili u fulfilled ili u rejected state

vii. Fulfilled/rejected promise nema tranziciju u neki drugi state jer se deep rooted, ukorenjen

viii. Jednom kada je promise ukorenjen, njegova vrednost se ne menja

### 36.What is promise chaining?

Proces izvrsavanja niza asinhronih taskova jedan za drugim koristeci promises.

```
new Promise(function(resolve, reject) {

  setTimeout(() => resolve(1), 1000);

}).then(function(result) {

  console.log(result); // 1
  return result * 2;

}).then(function(result) {

  console.log(result); // 2
  return result * 3;

}).then(function(result) {

  console.log(result); // 6
  return result * 4;

});
```

In the above handlers, the result is passed to the chain of .then() handlers with the below work flow,

a. The initial promise resolves in 1 second,
b. After that `.then` handler is called by logging the result(1) and then return a promise with the value of result * 2.
c. After that the value passed to the next `.then` handler by logging the result(2) and return a promise with result * 3.
d. Finally the value passed to the last `.then` handler by logging the result(6) and return a promise with result * 4.

### 37.What is promise.all?

Promise.all je promise koji uzima niz promisa kao input i resolvuje se kada su svi promisi resolvovani ili kada je jedan od njih rejected. Koristi se kada postoji vise povezanih asinhronih taskova koje zelimo da ispunimo pre nego sto se izvrsavanje koda nastavi. Ako nam treba result od svakog promisa, cak i ako je neki rejectovan, onda treba koristiti promise.allSettled( ), jer se promise.all( ) zavrsava kao rejectovan ako je bar jedan input rejectovan.

```
Promise.all([Promise1, Promise2, Promise3]) .then(result) =>
{   console.log(result) }) .catch(error => console.log(`Error in promises ${error}
`))
```

**Note:** Redosled promises-a (izlaz rezultata) se održava prema redosledu unosa.

### 38.What is the purpose of race method in promise?

**Promise.race()** method vraca instancu promisa koji je prvi resolved ili rejectovan. Let's take an example of race() method where promise2 is resolved first

```
var promise1 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 500, 'one');
});
var promise2 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 100, 'two');
});

Promise.race([promise1, promise2])
    .then(function(value) {
          console.log(value);     // "two"
                                  // Both promises will resolve, but promise2 is
faster
});
```

### 39. What is a strict mode in javascript?

Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a "strict" operating context. This way it prevents certain actions from being taken and throws more exceptions. The literal expression `"use strict";` instructs the browser to use the javascript code in the Strict mode.

### 40. Why do you need strict mode?

Strict mode is useful to write "secure" JavaScript by notifying "bad syntax" into real errors. For example, it eliminates accidentally creating a global variable by throwing an error and also throws an error for assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object.

### 41. What is the purpose of double exclamation?

The double exclamation or negation(!!) ensures the resulting type is a boolean. If it was falsey (e.g. 0, null, undefined, etc.), it will be false, otherwise, true. For example, you can test IE version using this expression as below,

```
let isIE8 = false;
isIE8 = !! navigator.userAgent.match(/MSIE 8.0/);
console.log(isIE8); // returns true or false
```

If you don't use this expression then it returns the original value.

```
console.log(navigator.userAgent.match(/MSIE 8.0/));  // returns either an Array or
null
```

**Note:** The expression !! is not an operator, but it is just twice of ! operator.

### 42. What is the purpose of delete operator?

The delete keyword is used to delete the property as well as its value.

```
var user= {name: "John", age:20};
delete user.age;

console.log(user); // {name: "John"}
```

### 43. What is typeof operator?

You can use the JavaScript typeof operator to find the type of a JavaScript variable. It returns the type of a variable or an expression.

```
typeof "John Abraham"     // Returns "string"
```

```
typeof (1 + 2)        // Returns "number"
```

### 44.What is undefined property?

The undefined property indicates that a variable has not been assigned a value, or not declared at all. The type of undefined value is undefined too.

```
var user;    // Value is undefined, type is undefined
console.log(typeof(user)) //undefined
```

Any variable can be emptied by setting the value to undefined.

```
user = undefined
```

### 45.What is null value?

The value null represents the intentional absence of any object value. It is one of JavaScript's primitive values. The type of null value is object. You can empty the variable by setting the value to null.

```
var user = null;
console.log(typeof(user)) //object
```

### 46.What is the difference between null and undefined?

Below are the main differences between null and undefined,

| Null | Undefined |
|---|---|
| It is an assignment value which indicates that variable points to no object. | It is not an assignment value where a variable has been declared but has not yet been assigned a value. |
| Type of null is object | Type of undefined is undefined |
| The null value is a primitive value that represents the null, empty, or non-existent reference. | The undefined value is a primitive value used when a variable has not been assigned a value. |
| Indicates the absence of a value for a variable | Indicates absence of variable itself |
| Converted to zero (0) while performing primitive operations | Converted to NaN while performing primitive operations |

### 47.What are the javascript data types?

Below are the list of javascript data types available

a. Number
b. String
c. Boolean
d. Object
e. Undefined

### 48.What is an event flow?

Event flow is the order in which event is received on the web page. When you click an element that is nested in various other elements, before your click actually reaches its destination, or target element, it must trigger the click event each of its parent elements first, starting at the top with the global window object. There are two ways of event flow

a. Top to Bottom(Event Capturing)

b. Bottom to Top (Event Bubbling)

### 49. What is event bubbling?

Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same nesting hierarchy till it reaches the outermost DOM element.

### 50. What is event capturing?

Event bubbling is a type of event propagation where the event is first captured by the outermost element and , and then successively triggers on the descendants (children) of the target element in the same nesting hierarchy till it reaches the inner DOM element.

### 51. Is JavaScript a compiled or interpreted language?

JavaScript is an interpreted language, not a compiled language. An interpreter in the browser reads over the JavaScript code, interprets each line, and runs it. Nowadays modern browsers use a technology known as Just-In-Time (JIT) compilation, which compiles JavaScript to executable bytecode just as it is about to run.

### 52. Is JavaScript a case-sensitive language?

Yes, JavaScript is a case sensitive language. The language keywords, variables, function & object names, and any other identifiers must always be typed with a consistent capitalization of letters.

### 53. What are events?

Events are "things" that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can `react` on these events. Some of the examples of HTML events are,

a. Web page has finished loading
b. Input field was changed
c. Button was clicked

Let's describe the behavior of click event for button element,

```
<!doctype html>
<html>
 <head>
   <script>
     function greeting() {
       alert('Hello! Good morning');
     }
   </script>
 </head>
 <body>
   <button type="button" onclick="greeting()">Click me</button>
 </body>
</html>
```

### 54. What is the use of preventDefault method?

The preventDefault() method cancels the event if it is cancelable, meaning that the default action or behaviour that belongs to the event will not occur. For example, prevent form submission when clicking on submit button and prevent opening the page URL when clicking on hyper link are some common usecases.

```
document.getElementById("link").addEventListener("click", function(event){
 event.preventDefault();
});
```

**Note:** Remember that not all events are cancelable.

### 55. What is the use of stopPropagation method?

The stopPropagation method is used to stop the event from bubbling up the event chain. For example, the below nested divs with stopPropagation method prevents default event propagation when clicking on nested div(Div1)

```
<p>Click DIV1 Element</p>
<div onclick="secondFunc()">DIV 2
  <div onclick="firstFunc(event)">DIV 1</div>
</div>

<script>
function firstFunc(event) {
  alert("DIV 1");
  event.stopPropagation();
}

function secondFunc() {
  alert("DIV 2");
}
</script>
```

### 56. What is the use of setTimeout?

The setTimeout() method is used to call a function or evaluates an expression after a specified number of milliseconds. For example, let's log a message after 2 seconds using setTimeout method,

```
setTimeout(function(){ console.log("Good morning"); }, 2000);
```

### 57. What is the use of setInterval?

The setInterval() method is used to call a function or evaluates an expression at specified intervals (in milliseconds). For example, let's log a message after 2 seconds using setInterval method,

```
setInterval(function(){ console.log("Good morning"); }, 2000);
```

### 58. Why is JavaScript treated as Single threaded?

JavaScript is a single-threaded language. Because the language specification does not allow the programmer to write code so that the interpreter can run parts of it in parallel in multiple threads or processes. Whereas languages like java, go, C++ can make multi-threaded and multi-process programs.

### 59. What is an event delegation?

Event delegation is a technique for listening to events where you delegate a parent element as the listener for all of the events that happen inside it. For example, if you wanted to detect field changes in inside a specific form, you can use event delegation technique,

```
var form = document.querySelector('#registration-form');

// Listen for changes to fields inside the form
form.addEventListener('input', function (event) {

  // Log the field that was changed
  console.log(event.target);

}, false);
```

---

**60. What is ECMAScript?**

ECMAScript is the scripting language that forms the basis of JavaScript.

**61. What is JSON?**

JSON (JavaScript Object Notation) is a lightweight format that is used for data interchanging (razmenu podataka). It is based on a subset of JavaScript language in the way objects are built in JavaScript.

**62. What are the syntax rules of JSON?**

a. The data is in name/value pairs
b. The data is separated by commas
c. Curly braces hold objects
d. Square brackets hold arrays

**63. What is the purpose JSON stringify?**

When sending data to a web server, the data has to be in a string format. You can achieve this by converting JSON object into a string using stringify() method.

```
var userJSON = {'name': 'John', age: 31}
var userString = JSON.stringify(user);
console.log(userString);  //"{"name":"John","age":31}"
```

**64. How do you parse JSON string?**

When receiving the data from a web server, the data is always in a string format. But you can convert this string value to javascript object using **parse()** method.

```
var userString = '{"name":"John","age":31}';
var userJSON = JSON.parse(userString);
console.log(userJSON);   // {name: "John", age: 31}
```

**65. Why do you need JSON?**

When exchanging data between a browser and a server, the data can only be text. Since JSON is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

**66. What are PWAs?**

Progressive web applications (PWAs) are a type of mobile app delivered through the web, built using common web technologies including HTML, CSS and JavaScript. These PWAs are deployed to servers, accessible through URLs, and indexed by search engines.

### 67. What is the purpose of clearTimeout method?

The clearTimeout() function is used in javascript to clear the timeout which has been set by setTimeout()function before that. i.e, The return value of setTimeout() function is stored in a variable and it's passed into the clearTimeout() function to clear the timer. For example, the below setTimeout method is used to display the message after 3 seconds for just one time. This timeout can be cleared by clearTimeout() method.

```
<script>
var msg;
function greeting() {
   alert('Good morning');
}
function start() {
  msg =setTimeout(greeting, 3000);

}

function stop() {
    clearTimeout(msg);
}
</script>
```

### 68. What is the purpose of clearInterval method?

The clearInterval() function is used in javascript to clear the interval which has been set by setInterval() function. i.e, The return value returned by setInterval() function is stored in a variable and it's passed into the clearInterval() function to clear the interval. For example, the below setInterval method is used to display the message for every 3 seconds. This interval can be cleared by clearInterval() method.

```
<script>
var msg;
function greeting() {
   alert('Good morning');
}
function start() {
  msg = setInterval(greeting, 3000);

}

function stop() {
    clearInterval(msg);
}
</script>
```

**Note:** we can use clearTimeout and clearInterval interchangeably (naizmenicno). This is because both functions will return a random ID which gets saved in the browser's memory but there's no separate group of IDs that are allocated to setTimeout versus (naspram) setInterval; they're shared.

```
const myIntervalFunc = setInterval(() => {
   console.log('Hello World');
  }, 100);
```

```
clearTimeout(myIntervalFunc); // clearTimeout works!
```

### 69. How do you redirect new page in javascript?

In vanilla javascript, you can redirect to a new page using `location` property of window object. The syntax would be as follows,

```
function redirect() {
   window.location.href = 'newPage.html';
}
```

### 70. How do you check whether a string contains a substring?

There are 3 possible ways to check whether a string contains a substring or not

a. **Using includes:** ES6 provided `String.prototype.includes` method to test a string contains a substring

```
var mainString = "hello", subString = "hell";
mainString.includes(subString)
```

ii. **Using indexOf:** In an ES5 or older environments, you can use `String.prototype.indexOf` which returns the index of a substring. If the index value is not equal to -1 then it means the substring exist in the main string.

```
var mainString = "hello", subString = "hell";
mainString.indexOf(subString) !== -1
```

iii. **Using RegEx:** The advanced solution is using Regular expression's test method(`RegExp.test`), which allows for testing for against regular expressions

```
var mainString = "hello", regex = "/hell/";
regex.test(mainString)
```

### 71. How do you validate an email in javascript?

You can validate an email in javascript using regular expressions. It is recommended to do validations on the server side instead client side. Because the javascript can be disabled on the client side.

```
function validateEmail(email) {
    var re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\
[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]
{2,}))$/;
    return re.test(String(email).toLowerCase());
}
```

The above regular expression regular accepts unicode characters. Unicode provides a unique number for every character across languages and scripts, making almost all characters accessible across platforms, programs, and devices.

### 72. How do you get the current url with javascript?

You can use `window.location.href` expression to get the current url path and you can use the same expression for updating the URL too. You can also use `document.URL` for read-only purpose but this solution has issues in FF.

```
console.log('location.href', window.location.href); // Returns full URL
```

### 73.What are the various url properties of location object?

The below `Location` object properties can be used to access URL components of the page,

iv. href - The entire URL
v. protocol - The protocol of the URL
vi. host - The hostname and port of the URL
vii. hostname - The hostname of the URL
viii. port - The port number in the URL
ix. pathname - The path name of the URL
x. search - The query portion of the URL
xi. hash - The anchor portion of the URL

### 74.How do get query string values in javascript?

You can use URLSearchParams to get query string values in javascript. Let's see an example to get the client code value from URL query string,

```
const urlParams = new URLSearchParams(window.location.search);
const clientCode = urlParams.get('clientCode');
```

### 75.How do you check if a key exists in an object?

You can check whether a key exists in an object or not using two approaches,

a. **** Using in operator:**** You can use the in operator whether a key exists in an object or not

```
"key" in obj
```

and If you want to check if a key doesn't exist, remember to use parenthesis (zagradu),

```
!("key" in obj)
```

ii. **** Using hasOwnProperty method:**** You can use `hasOwnProperty` to particularly test for properties of the object instance (and not inherited properties)

```
obj.hasOwnProperty("key") // true
```

### 76.How do you loop through or enumerate javascript object?

You can use the `for-in` loop to loop through javascript object. You can also make sure that the key you get is an
actual property of an object, and doesn't come from the prototype using `hasOwnProperty` method.

```
var object = {
    "k1": "value1",
    "k2": "value2",
    "k3": "value3"
};

for (var key in object) {
    if (object.hasOwnProperty(key)) {
        console.log(key + " -> " + object[key]); // k1 -> value1 ...
    }
}
```

### 77.How do you test for an empty object?

There are different solutions based on ECMAScript versions

iii. **Using Object entries(ECMA 7+):** You can use object entries length along with constructor type.

```
Object.entries(obj).length === 0 && obj.constructor === Object // Since date object length is 0, you need to check constructor check as well
```

ii. **Using Object keys(ECMA 5+):** You can use object keys length along with constructor type.

```
Object.keys(obj).length === 0 && obj.constructor === Object // Since date object length is 0, you need to check constructor check as well
```

iii. **Using for-in with hasOwnProperty(Pre-ECMA 5):** You can use for-in loop along with hasOwnProperty.

```
function isEmpty(obj) {
  for(var prop in obj) {
    if(obj.hasOwnProperty(prop)) {
      return false;
    }
  }

  return JSON.stringify(obj) === JSON.stringify({});
}
```

### 78. What is an arguments object?

The arguments object is an Array-like object accessible inside functions that contains the values of the arguments passed to that function.

```
function func1(a, b, c) {
  console.log(arguments[0]);  // expected output: 1
  console.log(arguments[1]);  // expected output: 2
  console.log(arguments[2]);  // expected output: 3
}

func1(1, 2, 3);
```

### 79. How do you make first letter of the string in an uppercase?

You can create a function which uses chain (lanac) of string methods such as charAt, toUpperCase and slice methods to generate a string with first letter in uppercase.

```
function capitalizeFirstLetter(string) {
    return string.charAt(0).toUpperCase() + string.slice(1);
}
```

**Note:** charAt() is a method that returns the character from the specified index.

### 80. What are the pros and cons of for loop?

The for-loop is a commonly used iteration syntax in javascript. It has both pros and cons

**Pros**

iv. Works on every environment
v. You can use break and continue flow control statements
vi. You loop through every single element for a given string or array

**Cons**

vi. You cannot skip any element as the concept of index is not there. You cannot choose to traverse (prelaziti) to odd or even indexed elements too.
vii. It's very restricting - you can't determine where to start or how long you want to go on for. Incrementing is always set to one at a time.
viii. You might face one-by-off (pojedinacni) errors

### 81.How do you display the current date in javascript?

You can use new Date() to generate a new Date object containing the current date and time. For example, let's display the current date in mm/dd/yyyy

```
var today = new Date();
var dd = String(today.getDate()).padStart(2, '0');
var mm = String(today.getMonth() + 1).padStart(2, '0'); //January is 0!
var yyyy = today.getFullYear();

today = mm + '/' + dd + '/' + yyyy;
document.write(today);
```

**Note:** The padStart(length of the resulting string, string to pad the current with) method pads (prosiruje) the current string with another string (multiple times, if needed) until the resulting string reaches the given length.

```
const str1 = '5';
console.log(str1.padStart(2, '0'));  // expected output: "05"
```

### 82.How do you compare two date objects?

You need to use date.getTime() method to compare date values instead comparision operators (==, !=, ===, and !== operators)

```
var d1 = new Date();
var d2 = new Date(d1);
console.log(d1.getTime() === d2.getTime()); //True
console.log(d1 === d2); // False
```

### 83.How do you check if a string starts with another string?

You can use `String.prototype.startsWith()` method to check a string starts with another string or not. But it is not yet supported in all browsers.

```
"Good morning".startsWith("Good"); // true
"Good morning".startsWith("morning"); // false
```

### 84.How do you trim a string in javascript?

JavaScript provided a trim method on string types to trim any whitespaces present at the begining or ending of the string.

```
"  Hello World   ".trim(); //Hello World
```

If your browser(<IE9) doesn't support this method then you can use below polyfill.

```
if (!String.prototype.trim) {
    (function() {
        // Make sure we trim BOM and NBSP
        var rtrim = /^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g;
        String.prototype.trim = function() {
            return this.replace(rtrim, '');
```

```
        };
    })();
```

**85.How do you add a key value pair in javascript?**

There are two possible solutions to add new properties to an object. Let's take a simple object to explain these solutions.

```
var object = {
    key1: value1,
    key2: value2
};
```

a. **Using dot notation:** This solution is useful when you know the name of the property

```
object.key3 = "value3";
```

ii. **Using square bracket notation:** This solution is useful when the name of the property is dynamically determined.

```
obj["key3"] = "value3";
```

**86.Is the !-- notation represents a special operator?**

No,that's not a special operator. But it is a combination of 2 standard operators one after the other,

iii. A logical not (!)

iv. A prefix decrement (--)

At first, the value decremented by one and then tested to see if it is equal to zero or not for determining the truthy/falsy value.

**87.How do you assign default values to variables?**

You can use the logical or operator || in an assignment expression to provide a default value. The syntax looks like as below,

```
var a = b || c;
```

As per the above expression, variable 'a 'will get the value of 'c' only if 'b' is falsy (if is null, false, undefined, 0, empty string, or NaN), otherwise 'a' will get the value of 'b'.

**88.How do you define multiline strings?**

You can define multiline string literals using '' character followed by line terminator.

```
var str = "This is a \
very lengthy \
sentence!";
```

But if you have a space after the '' character, the code will look exactly the same, but it will raise a SyntaxError.

**89.What is an app shell model?**

An application shell (or app shell) architecture is one way to build a Progressive Web App that reliably and instantly loads on your users' screens, similar to what you see in native applications. It is useful for getting some initial HTML to the screen fast without a network.

**90.Can we define properties for functions?**

Yes, We can define properties for functions because functions are also objects.

```
fn = function(x) {
   //Function code goes here
}

fn.name = "John";

fn.profile = function(y) {
  //Profile code goes here
}
```

### 91. What is the way to find the number of parameters expected by a function?

You can use `function.length` syntax to find the number of parameters expected by a function. Let's take an example of `sum` function to calculate the sum of numbers

```
function sum(num1, num2, num3, num4){
    return num1 + num2 + num3 + num4;
}
sum.length       // 4 is the number of parameters expected.
```

### 92. What is a polyfill?

A polyfill is a piece of JS code used to provide modern functionality on older browsers that do not natively support it. For example, Silverlight plugin polyfill can be used to mimic the functionality of an HTML Canvas element on Microsoft Internet Explorer 7.

### 93. What are break and continue statements?

The break statement is used to "jumps out" of a loop. i.e, It breaks the loop and continues executing the code after the loop.

```
for (i = 0; i < 10; i++) {
  if (i === 5) { break; }
  text += "Number: " + i + "<br>";
}
```

The continue statement is used to "jumps over" one iteration in the loop. i.e, It breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
for (i = 0; i < 10; i++) {
    if (i === 5) { continue; }
    text += "Number: " + i + "<br>";
}
```

### 94. What are js labels?

The label statement allows us to name loops and blocks in JavaScript. We can then use these labels to refer back to the code later. For example, the below code with labels avoids printing the numbers when they are same,

```
var i, j;

loop1:
for (i = 0; i < 3; i++) {
   loop2:
   for (j = 0; j < 3; j++) {
      if (i === j) {
         continue loop1;
```

```
        }
        console.log('i = ' + i + ', j = ' + j);
    }
}

// Output is:
//   "i = 1, j = 0"
//   "i = 2, j = 0"
//   "i = 2, j = 1"
```

### 95.What are the benefits of keeping declarations at the top?

It is recommended to keep all declarations at the top of each script or function. The benefits of doing this are,

a. Gives cleaner code
b. It provides a single place to look for local variables
c. Easy to avoid unwanted global variables
d. It reduces the possibility of unwanted re-declarations

### 96.What are the benefits of initializing variables?

It is recommended to initialize variables because of the below benefits,

a. It gives cleaner code
b. It provides a single place to initialize variables
c. Avoid undefined values in the code

### 97.What are the recommendations to create new object?

It is recommended to avoid creating new objects using `new Object()`. Instead you can initialize values based on it's type to create the objects.

a. Assign { } instead of new Object()
b. Assign "" instead of new String()
c. Assign 0 instead of new Number()
d. Assign false instead of new Boolean()
e. Assign [ ] instead of new Array()
f. Assign /( )/ instead of new RegExp()
g. Assign function ( ) { } instead of new Function()

You can define them as an example,

```
var v1 = {};
var v2 = "";
var v3 = 0;
var v4 = false;
var v5 = [];
var v6 = /()/;
var v7 = function(){};
```

### 98.How do you define JSON arrays?

JSON arrays are written inside square brackets and array contain javascript objects. For example, the JSON array of users would be as below,

```
"users":[
  {"firstName":"John", "lastName":"Abrahm"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Shane", "lastName":"Warn"}
]
```

**99.How do you generate random integers?**

You can use Math.random() with Math.floor() to return random integers. For example, if you want generate random integers between 1 to 10, the multiplication factor should be 10,

```
Math.floor(Math.random() * 10) + 1;     // returns a random integer from 1 to 10
Math.floor(Math.random() * 100) + 1;     // returns a random integer from 1 to 100
```

**Note:** Math.random() returns a random number between 0 (inclusive), and 1 (exclusive).
The Math.floor() function returns the largest integer less than or equal to a given number.

```
console.log(Math.floor(5.05));              // expected output: 5
console.log(Math.floor(5));          // expected output: 5
console.log(Math.floor(-5.05));             // expected output: -6
```

100.**Can you write a random integers function to print integers with in a range?**

Yes, you can create a proper random function to return a random number between min and max (both included)

```
function randomInteger(min, max) {
  return Math.floor(Math.random() * (max - min + 1) ) + min;
}
randomInteger(1, 100); // returns a random integer from 1 to 100
randomInteger(1, 1000); // returns a random integer from 1 to 1000
```

**101.What is tree shaking?**

Tree shaking is a form of dead code elimination. It means that unused modules will not be included in the bundle during the build process ( i.e. import and export).

**102.What is the need of tree shaking?**

Tree Shaking can significantly reduce the code size in any application. i.e, The less code we send over the wire the more performant the application will be.

**103.What is a Regular Expression?**

A regular expression is a sequence (niz) of characters that forms a search pattern. You can use this search pattern for searching data in a text. These can be used to perform all types of text search and text replace operations. Let's see the syntax format now,

```
/pattern/modifiers;
```

For example, the regular expression or search pattern with case-insensitive username would be,

```
/John/i
```

**104.What are the string methods available in Regular expression?**

Regular Expressions has two string methods: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

```
var msg = "Hello John";
var n = msg.search(/John/i); // 6
```

The replace() method is used return a modified string where the pattern is replaced.

```
var msg = "Hello John";
var n = msg.replace(/John/i, "Buttler"); // Hello Buttler
```

**105.What are regular expression patterns?**

Regular Expressions provided group of patterns in order to match characters. Basically they are categorized into 3 types,

a. **Brackets:** These are used to find a range of characters. For example, below are some use cases,
    i. [abc]: Used to find any of the characters between the brackets(a,b,c)
    ii. [0-9]: Used to find any of the digits between the brackets
    iii. (a|b): Used to find any of the alternatives separated with |
b. **Metacharacters:** These are characters with a special meaning For example, below are some use cases,
    i. \d: Used to find a digit
    ii. \s: Used to find a whitespace character
    iii. \b: Used to find a match at the beginning or ending of a word
c. **Quantifiers:** These are useful to define quantities For example, below are some use cases,
    i. n+: Used to find matches for any string that contains at least one n
    ii. n*: Used to find matches for any string that contains zero or more occurrences of n
    iii. n?: Used to find matches for any string that contains zero or one occurrences of n

**106.What is a RegExp object?**

RegExp object is a regular expression object with predefined properties and methods. It's used for matching text with a pattern.

```
var regexp = new RegExp('\\w+');
console.log(regexp);
// expected output: /\w+/
```

**107.How do you search a string for a pattern?**

You can use test( ) method of regular expression in order to search a string for a pattern, and returns true or false depending on the result.

```
var pattern = /you/;
console.log(pattern.test("How are you?")); //true
```

**108.What is the purpose of exec method?**

The purpose of exec method is similar to test method but it returns a founded text as an object instead of returning true/false.

```
var pattern = /you/;
console.log(pattern.test("How are you?")); //you
```

### 109.How do you change style of a HTML element?

You can change inline style or classname of a HTML element using javascript

a. **** Using style property:****

```
document.getElementById("title").style.fontSize = "30px";
```

ii. **** Using ClassName property:**** It is easy to modify element class using className property

```
document.getElementById("title").style.className = "custom-title";
```

### 110.What would be the result of 1+2+'3'?

The output is going to be `33`. Since `1` and `2` are numeric values, the result of first two digits is going to be a numeric value `3`. The next digit is a string type value because of that the addition of numeric value `3` and string type value `3` is just going to be a concatenation value `33`.

### 111.What is a debugger statement?

The debugger statement invokes (priziva) any available debugging functionality, such as setting a breakpoint. If no debugging functionality is available, this statement has no effect. For example, in the below function a debugger statement has been inserted. So execution is paused at the debugger statement just like a breakpoint in the script source.

```
function getProfile() {
// code goes here
debugger;
// code goes here
}
```

### 112.What is the purpose of breakpoints in debugging?

You can set breakpoints in the javascript code once the debugger statement is executed and debugger window pops up. At each breakpoint, javascript will stop executing, and let you examine (ispitati) the JavaScript values. After examining values, you can resume the execution of code using play button.

### 113.Can I use reserved words as identifiers?

No, you cannot use the reserved words as variables, labels, object or function names. Let's see one simple example,

```
var else = "hello"; // Uncaught SyntaxError: Unexpected token else
```

### 114.How do you detect a mobile browser?

You can use regex which returns a true or false value depending on whether or not the user is browsing with a mobile.

```
window.mobilecheck = function() {
  var mobileCheck = false;
```

### 115.How do you detect a mobile browser without regexp?

You can detect mobile browser by simply running through a list of devices and checking if the useragent matches anything. This is an alternative solution for RegExp usage,

```
function detectmob() {
 if( navigator.userAgent.match(/Android/i)
 || navigator.userAgent.match(/webOS/i)
 || navigator.userAgent.match(/iPhone/i)
 || navigator.userAgent.match(/iPad/i)
 || navigator.userAgent.match(/iPod/i)
 || navigator.userAgent.match(/BlackBerry/i)
 || navigator.userAgent.match(/Windows Phone/i)
 ){
    return true;
  }
 else {
    return false;
  }
}
```

### 116.How do you get the image width and height using JS?

You can programmatically get the image and check the dimensions(width and height) using Javascript.

```
var img = new Image();
img.onload = function() {
  console.log(this.width + 'x' + this.height);
}
img.src = 'http://www.google.com/intl/en_ALL/images/logo.gif';
```

### 117.How do you make synchronous HTTP request?

Browsers provide an XMLHttpRequest object which can be used to make synchronous HTTP requests from JavaScript

```
function httpGet(theUrl)
{
    var xmlHttpReq = new XMLHttpRequest();
    xmlHttpReq.open( "GET", theUrl, false ); // false for synchronous request
    xmlHttpReq.send( null );
    return xmlHttpReq.responseText;
}
```

### 118.How do you make asynchronous HTTP request?

Browsers provide an XMLHttpRequest object which can be used to make asynchronous HTTP requests from JavaScript by passing 3rd parameter as true. If you use an asynchronous XMLHttpRequest, you receive a callback when the data has been received. This lets the browser continue to work as normal while your request is being handled.

```
function httpGetAsync(theUrl, callback)
{
    var xmlHttpReq = new XMLHttpRequest();
    xmlHttpReq.onreadystatechange = function() {
        if (xmlHttpReq.readyState == 4 && xmlHttpReq.status == 200)
            callback(xmlHttpReq.responseText);
    }
    xmlHttp.open("GET", theUrl, true); // true for asynchronous
    xmlHttp.send(null);
}
```

### 119.How do you convert date to another timezone in javascript?

You can use toLocaleString() method to convert date in one timezone to another. For example, let's convert current date to British English timezone as below,

```
console.log(event.toLocaleString('en-GB', { timeZone: 'UTC' })); //29/06/2019,
09:56:00
```

### 120.What are the properties used to get size of window?

You can use innerWidth, innerHeight, clientWidth, clientHeight properties of windows, document element and document body objects to find the size of a window. Let's use them combination of these properties to calculate the size of a window or document,

```
var width = window.innerWidth
|| document.documentElement.clientWidth
|| document.body.clientWidth;

var height = window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;
```

### 121.What is a conditional operator in javascript?

The conditional (ternary) operator is the only JavaScript operator that takes three operands which acts as a shortcut for if statement.

```
var isAuthenticated = false;
console.log(isAuthenticated ? 'Hello, welcome' : 'Sorry, you are not
authenticated'); //Sorry, you are not authenticated
```

### 122.Can you apply chaining on conditional operator?

Yes, you can apply chaining on conditional operator similar to if … else if … else if … else chain. The syntax is going to be as below,

```
function traceValue(someParam) {
    return condition1 ? value1
         : condition2 ? value2
         : condition3 ? value3
         : value4;
}

// The above conditional operator is equivalent to:

function traceValue(someParam) {
    if (condition1) { return value1; }
    else if (condition2) { return value2; }
    else if (condition3) { return value3; }
    else { return value4; }
}
```

### 123.What are the ways to execute javascript after page load?

You can execute javascript after page load in many different ways,

iii. **** window.onload:****

```
window.onload = function ...
```

ii. **document.onload:**

```
document.onload = function ...
```

iii. **** body onload:****

```
<body onload="script();">
```

### 124.Give an example where do you really need semicolon?

It is recommended to use semicolons after every statement in JavaScript. For example, in the below case it throws an error ".. is not a function" at runtime due to missing semicolon.

```
// define a function
var fn = function () {
    //...
} // semicolon missing at this line

// then execute some code inside a closure
(function () {
    //...
})();
```

and it will be interpreted as

```
var fn = function () {
    //...
}(function () {
    //...
})();
```

In this case, we are passing second function as an argument to the first function and then trying to call the result of the first function call as a function. Hence, the second function will fail with a "... is not a function" error at runtime.

### 125.What is a freeze method?

The freeze() method is used to freeze an object. Freezing an object doesn't allow adding new properties to an object,prevents from removing and prevents changing the enumerability, configurability, or writability of existing properties. i.e, It returns the passed object and does not create a frozen copy.

```
const obj = {
  prop: 100
};

Object.freeze(obj);
obj.prop = 200; // Throws an error in strict mode

console.log(obj.prop); //100
```

**Note:** It causes a TypeError if the argument passed is not an object.

### 126.What is the purpose of freeze method?

Below are the main benefits of using freeze method,

iv. It is used for freezing objects and arrays.

v. It is used to make an object immutable.

### 127.Why do I need to use freeze method?

In Object-oriented paradigm, an existing API contains certain elements that are not intended to be extended, modified, or re-used outside of their current context. Hence it works as `final` keyword which is used in various languages.

### 128.How do you detect a browser language preference?

You can use navigator object to detect a browser language preference as below,

```
var language = navigator.languages && navigator.languages[0] || // Chrome / Firefox
               navigator.language ||   // All browsers
               navigator.userLanguage; // IE <= 10

console.log(language);
```

### 129.How to convert string to title case with javascript?

Title case means that the first letter of each word is capitalized. You can convert a string to title case using the below function,

```
function toTitleCase(str) {
    return str.replace(
        /\w\S*/g,
        function(txt) {
            return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();
        }
    );
}
toTitleCase("good morning john"); // Good Morning John
```

### 130.How do you detect javascript disabled in the page?

You can use **<noscript>** tag to detect javascript disabled or not. The code block inside `<noscript>` get executed when JavaScript is disabled, and are typically used to display alternative content when the page generated in JavaScript.

```
<script type="javascript">
    // JS related code goes here
</script>
<noscript>
    <a href="next_page.html?noJS=true">JavaScript is disabled in the apge. Please
click Next Page</a>
</noscript>
```

### 131.What are various operators supported by javascript?

An operator is capable of manipulating(mathematical and logical computations) a certain value or operand. There are various operators supported by JavaScript as below,

a. **Arithmetic Operators:** Includes + (Addition),– (Subtraction), * (Multiplication), / (Division), % (Modulus), + + (Increment) and – – (Decrement)
b. **Comparison Operators:** Includes = =(Equal),!= (Not Equal), ===(Equal with type), > (Greater than),> = (Greater than or Equal to),< (Less than),<= (Less than or Equal to)
c. **Logical Operators:** Includes &&(Logical AND),||(Logical OR),!(Logical NOT)
d. **Assignment Operators:** Includes = (Assignment Operator), += (Add and Assignment Operator), – = (Subtract and Assignment Operator), *= (Multiply and Assignment), /= (Divide and Assignment), %= (Modules and Assignment)
e. **Ternary Operators:** It includes conditional(: ?) Operator

f. **typeof Operator:** It uses to find type of variable. The syntax looks like `typeof variable`

### 132.What is a rest parameter?

Rest parameter allows a function to accept an indefinite number of arguments as an array. The syntax would be as below:

```
function f(a, b, ...theArgs) {
  // ...
}
```

For example, let's take a sum example to calculate on dynamic number of parameters,

```
function total(…args){
  let sum = 0;

  for(let i of args){
          sum+=i;
  }
  return sum;
}

console.log(fun(1,2)); //3
console.log(fun(1,2,3)); //6
console.log(fun(1,2,3,4)); //13
console.log(fun(1,2,3,4,5)); //15
```

**Note:** Rest parameter is added in ES2015 or ES6

### 133.What happens if you do not use rest parameter as a last argument?

The rest parameter should be the last argument, as its job is to collect all the remaining arguments into an array. For example, if you define a function like below it doesn't make any sense and will throw an error.

```
function someFunc(a,…b,c){
  //You code goes here
  return;
}
```

### 134.What are the bitwise operators available in javascript?

Below are the list of bit-wise logical operators used in JavaScript

a. Bit-wise AND ( & )
b. Bit-Wise OR ( | )
c. Bit-Wise XOR ( ^ )
d. Bit-Wise NOT ( ~ )
e. Left Shift ( << )
f. Sign Propagating Right Shift ( >> )
g. Zero fill Right Shift ( >>> )

### 135.What is a spread operator?

Spread operator allows iterables( arrays / objects / strings ) to be expanded into single arguments/ elements. Let's take an example to see this behavior, Spread syntax can be used when all elements from an object or array need to be included in a new array or object.

```
function calculateSum(x, y, z) {
  return x + y + z;
}

const numbers = [1, 2, 3];

console.log(calculateSum(...numbers)); // 6
```

### 136.How do you determine whether object is frozen or not?

Object.isFrozen() method is used to determine if an object is frozen or not. An object is frozen if all of the below conditions hold true,

a. If it is not extensible.
b. If all of its properties are non-configurable.
c. If all its data properties are non-writable. The usage is going to be as follows,

```
const object = {
   property: 'Welcome JS world'
};
Object.freeze(object);
console.log(Object.isFrozen(object));
```

### 137.How do you determine two values same or not using object?

The Object.is() method determines whether two values are the same value. For example, the usage with different types of values would be,

```
Object.is('hello', 'hello');     // true
Object.is(window, window);   // true
Object.is([], []) // false bcs their pointer (or reference) are being compared - we have created two separate objects, so they aren't equal. -> list are objects!
```

Two values are the same if one of the following holds:

a. both undefined
b. both null
c. both true or both false
d. both strings of the same length with the same characters in the same order
e. both the same object (means both object have same reference)
f. both numbers and both +0 both -0 both NaN both non-zero and both not NaN and both have the same value.

### 138.What is the purpose of using object is method?

Some of the applications of Object's `is` method are follows,

a. It is used for comparison of two strings.
b. It is used for comparison of two numbers.
c. It is used for comparing the polarity of two numbers.
d. It is used for comparison of two objects.

### 139.How do you copy properties from one object to other?

You can use Object.assign() method which is used to copy the values and properties from one or more source objects to a target object (ciljni obj.). It returns the target object which has properties and values copied from the target object. The syntax would be as below,

```
Object.assign(target, ...sources)
```

Let's take exampple with one source and one target object,

```
const target = { a: 1, b: 2 };
const source = { b: 3, c: 4 };

const returnedTarget = Object.assign(target, source);

console.log(target); // { a: 1, b: 3, c: 4 }

console.log(returnedTarget); // { a: 1, b: 3, c: 4 }
```

As observed in the above code, there is a common property(`b`) from source to target so it's value is been overwritten.

### 140.What are the applications of assign method?

Below are the some of main applications of Object.assign() method,

a. It is used for cloning an object.
b. It is used to merge object with same properties.

### 141.What is a proxy object?

The Proxy object enables you to create a proxy for another object, which can intercept (presresti) and redefine fundamental operations for that object.The syntax would be as follows,

```
var p = new Proxy(target, handler);
```

Let's take an example of proxy object,

```
const target = {
  message1: "hello",
  message2: "everyone"
};

const handler2 = {
  get(target, prop, receiver) {
    return "world";
  }
};

const proxy2 = new Proxy(target, handler2);

console.log(proxy2.message1); // world
console.log(proxy2.message2); // world
```

### 142.What is the purpose of seal method?

The Object.seal() method is used seal an object, by preventing new properties from being added to it and marking all existing properties as non-configurable. But values of present properties can still be changed as long as they are writable. Let's see the below example to understand more about seal() method

```
 const object = {
    property: 'Welcome JS world'
 };
```

```
Object.seal(object);
object.property = 'Welcome to object world';
console.log(Object.isSealed(object)); // Welcome to object world
delete object.property; // You cannot delete when sealed
console.log(object.property); //Welcome to object world
```

143.**What are the applications of seal method?**

a. It is used for sealing objects and arrays.
b. It is used to make an object immutable.

**144.What are the differences between freeze and seal methods?**

If an object is frozen using the Object.freeze() method then its properties become immutable and no changes can be made in them whereas if an object is sealed using the Object.seal() method then the changes can be made in the existing properties of the object.

**145.How do you determine if an object is sealed or not?**

The Object.isSealed() method is used to determine if an object is sealed or not. An object is sealed if all of the below conditions hold true

a. If it is not extensible.
b. If all of its properties are non-configurable.
c. If it is not removable (but not necessarily non-writable). Let's see it in the action

```
const object = {
  property: 'Hello, Good morning'
};

Object.seal(object);     // Using seal() method to seal the object

console.log(Object.isSealed(object));      // checking whether the object is sealed
or not
```

**146.How do you get enumerable (nabrojive) key and value pairs?**

The Object.entries() method returns an array whose elements are arrays corresponding to the enumerable string-keyed property [key, value] pairs, in the same order as that provided by a for...in loop.

```
const object = {
  a: 'Good morning',
  b: 100
};

for (let [key, value] of Object.entries(object)) {
  console.log(`${key}: ${value}`); // a: 'Good morning'
                                   // b: 100
}
```

**Note:** The order is not guaranteed as object defined.

**147.What is the main difference between Object.values and Object.entries method?**

The Object.values() method's behavior is similar to Object.entries() method but it returns an array of values instead [key,value] pairs.

```
const object = {
```

```
  a: 'Good morning',
  b: 100
};

for (let value of Object.values(object)) {
  console.log(`${value}`); // 'Good morning'
                              100
}
```

### 148.How can you get the list of keys of any object?

You can use `Object.keys()` method which is used return an array of a given object's own property names, in the same order as we get with a normal loop.

```
const user = {
  name: 'John',
  gender: 'male',
  age: 40
};

console.log(Object.keys(user));  //['name', 'gender', 'age']
```

### 149.How do you create an object with prototype?

The Object.create() method is used to create a new object with the specified prototype object and properties. i.e, It uses existing object as the prototype of the newly created object. It returns a new object with the specified prototype object and properties.

```
const user = {
  name: 'John',
  printInfo: function () {
    console.log(`My name is ${this.name}.`);
  }
};

const admin = Object.create(person);

admin.name = "Nick"; // Remember that "name" is a property set on "admin" but not
on "user" object

admin.printInfo(); // My name is Nick
```

### 150.What is a WeakSet?

WeakSet is used to store a collection of weakly(weak references) held objects. The syntax would be as follows,

```
new WeakSet([iterable]);
```

Let's see the below example to explain it's behavior,

```
var ws = new WeakSet();
var user = {};
ws.add(user);
ws.has(user);    // true
ws.delete(user); // removes user from the set
ws.has(user);    // false, user has been removed
```

### 151.What are the differences between WeakSet and Set?

The main difference is that references to objects in Set are strong while references to objects in WeakSet are weak. i.e, An object in WeakSet can be garbage collected if there is no other reference to it. Other differences are,

a. Sets can store any value whereas WeakSets can store only collections of objects
b. WeakSet does not have size property unlike Set
c. WeakSet does not have methods such as clear, keys, values, entries, forEach.
d. WeakSet is not iterable.

**152.List down the collection of methods available on WeakSet?**

a. add(value): A new object is appended with the given value to the weakset
b. delete(value): Deletes the value from the WeakSet collection.
c. has(value): It returns true if the value is present in the WeakSet Collection, otherwise it returns false.
d. length(): It returns the length of weakSetObject

```
var weakSetObject = new WeakSet();
var firstObject = {};
var secondObject = {};
// add(value)
weakSetObject.add(firstObject);
weakSetObject.add(secondObject);
console.log(weakSetObject.has(firstObject)); //true
console.log(weakSetObject.length()); //2
weakSetObject.delete(secondObject);
```

**153.What is a WeakMap?**

The WeakMap object is a collection of key/value pairs in which the keys are weakly referenced. In this case, keys must be objects and the values can be arbitrary (proizvoljan) values.

```
new WeakMap([iterable])
```

```
 var ws = new WeakMap();
 var user = {};
 ws.set(user);
 ws.has(user);    // true
 ws.delete(user); // removes user from the map
 ws.has(user);    // false, user has been removed
```

**154.What are the differences between WeakMap and Map?**

The main difference is that references to key objects in Map are strong while references to key objects in WeakMap are weak. i.e, A key object in WeakMap can be garbage collected if there is no other reference to it. Other differences are,

a. Maps can store any key type Whereas WeakMaps can store only collections of key objects
b. WeakMap does not have size property unlike Map
c. WeakMap does not have methods such as clear, keys, values, entries, forEach.
d. WeakMap is not iterable.

155.**List down the collection of methods available on WeakMap?**

a. set(key, value): Sets the value for the key in the WeakMap object. Returns the WeakMap object.

**b.** delete(key): **Removes any value associated (povezana) to the key.**

c. has(key): Returns a Boolean if a value has been associated to the key in the WeakMap object or not.

d. get(key): Returns the value associated to the key, or undefined if there is none.

```
var weakMapObject = new WeakMap();
var firstObject = {};
var secondObject = {};
// set(key, value)
weakMapObject.set(firstObject, 'John');
weakMapObject.set(secondObject, 100);
console.log(weakMapObject.has(firstObject)); //true
console.log(weakMapObject.get(firstObject)); // John
weakMapObject.delete(secondObject);
```

### 156.Is it recommended to use eval( )?

No, it allows arbitrary (proizvoljnom) code to be run which casues a security problem. As we know that the eval() function is used to run text as code. In most of the cases, it should not be necessary to use it.

console.log(eval('2 + 2'));  // expected output: 4

### 157.What is the purpose of uneval?

The uneval() is an inbuilt function which is used to create a **string** representation of the source code of an Object. It is a top-level function and is not associated with any object.

```
var a = 1;
uneval(a);                // returns a String containing 1
uneval(function user() {});     // returns "(function user(){})"
```

### 158.What is the difference between uneval and eval?

The `uneval` function returns the source of a given object; whereas the `eval` function does the opposite, by evaluating that source code in a different memory area.

```
var msg = uneval(function greeting() { return 'Hello, Good morning'; });
var greeting = eval(msg);
greeting();      // returns "Hello, Good morning"
```

### 159.How do you encode an URL?

The encodeURI() function is used to encode complete URI which has special characters except (, / ? : @ & = + $ #) characters. URLs can only be sent over the Internet using the ASCII character-set. Since URLs often contain characters outside the ASCII set, the URL has to be converted into a valid ASCII format. URL encoding replaces unsafe ASCII characters with a "%" followed by two hexadecimal digits.

```
var uri = 'https://mozilla.org/?x=шеллы';
var encoded = encodeURI(uri);
console.log(encoded);     // https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B
```

### 160.How do you decode an URL?

The decodeURI() function is used to decode a Uniform Resource Identifier (URI) previously created by encodeURI().

```
 var uri = 'https://mozilla.org/?x=шеллы';
```

```
 var encoded = encodeURI(uri);
 console.log(encoded); // https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B
try {
  console.log(decodeURI(encoded)); // "https://mozilla.org/?x=шеллы"
} catch(e) { // catches a malformed URI
  console.error(e);
}
```

### 161.How do you print the contents of web page?

The window object provided print() method which is used to prints the contents of the current window. It opens Print dialog box which lets you choose between various printing options.

```
<input type="button" value="Print" onclick="window.print()" />
```

**Note:** In most browsers, it will block while the print dialog is open.

### 162.What is an anonymous function?

An anonymous function is a function without a name! Anonymous functions are commonly assigned to a variable name or used as a callback function. The syntax would be as below,

```
function (optionalParameters) {
  //do something
}

const myFunction = function(){  //Anonymous function assigned to a variable
  //do something
};

[1, 2, 3].map(function(element){        //Anonymous function used as a callback
function
  //do something
});
```

Let's see the above anonymous function in an example,

```
var x = function (a, b) {return a * b};
var z = x(5, 10);
console.log(z); // 50
```

---

### 163.What are the different ways to access object properties?

There are 3 possible ways for accessing the property of an object.

- **Dot notation:** It uses dot for accessing the properties, choose it when the property name is known ahead of time (poznato unapred)

  *objectName.property*

  ```
  const hero = {
    name: 'Batman'
  };
  hero.name; // => 'Batman'
  ```

  hero.name reads the property name of the object hero.

- **Square brackets notation:** It uses square brackets for property access, choose it when the property name is dynamic, i.e. it can be changed during program execution.

*objectName["property"]*

```
const property = 'name';
const hero = {
  name: 'Batman'
};
hero['name'];   // => 'Batman'
hero[property]; // => 'Batman'
```

`hero['name']` and `hero[property]` both read the property `name` by using the square brackets syntax.

- **Object destructuring:** It uses expression in the square brackets, choose it when you'd like to create a variable having the property value.

*const { property } = expression;*

```
const hero = {
  name: 'Batman'
};
const { name } = hero;
name; // => 'Batman'
```

The destructuring defines a variable name with the value of property name.

To access properties in **objects or Arrays**, as we've seen you can do like this:
const title = myObject.title;
const name = myObject.name;

const title = myObject['title'];
const name = myObject['name'];
Now using destructuring syntax to access properties on objects, the same code above would look like this:

const { title, name } = myObject;

**Note:** Primary Expressions are:
this in JavaScript ( this always holds the reference to a single object, that defines the current line of code's execution context. )
Async/Await Function in JavaScript
JavaScript Object initializer
JavaScript Regular Expressions
JavaScript Function Expression
JavaScript class expression

### 164.What is an error object?

An error object is a built in error object that provides error information when an error occurs. It has two properties: *name* and *message*.

```
try {
  greeting("Welcome");
}
catch(err) {
  console.log(err.name + "<br>" + err.message);
```

}

### 165.What are the different error names from error object?

There are 6 different types of error names returned from error object:

| Error Name | Description |
|---|---|
| EvalError | An error has occurred in the eval() function |
| RangeError | An error has occurred with a number "out of range" |
| ReferenceError | An error due to an illegal reference * |
| SyntaxError | An error due to a syntax error |
| TypeError | An error due to a type error (when a value is not of the expected type) |
| URIError | An error due to encodeURI() |

*when a variable that doesn't exist (or hasn't yet been initialized) in the current scope is referenced; the references are pointers to values stored in variables and NOT pointers to other variables or references

### 166.What are the various (razne) statements in error handling?

- try: This statement is used to test a block of code for errors
- catch: This statement is used to handle the error
- throw: This statement is used to create custom errors.
- finally: This statement is used to execute code after try and catch regardless (bez obzira) of the result.

### 167.What are the two types of loops in javascript?

Entry Controlled loops: In this kind of loop type, the test condition is tested before entering the loop body. For example, For Loop and While Loop comes under this category.

Exit Controlled Loops: In this kind of loop type, the test condition is tested or evaluated at the end of loop body. i.e, the loop body will execute atl east once irrespective of test condition true or false. For example, do-while loop comes under this category.

### 168.What is nodejs?

Node.js is a server-side platform for easily building fast network applications. It is an event-based, non-blocking, asynchronous runtime that uses Google's V8 JavaScript engine and libuv library.

### 169.What is an Intl object?

The Intl object is the namespace for the ECMAScript Internationalization API, which provides language sensitive string comparison, number formatting, and date and time formatting. It provides an access to several constructors and language sensitive functions.

**170.What are the properties of Intl object?**

| | |
|---|---|
| Collator: | These are the objects that enable language-sensitive string comparison. |
| DateTimeFormat: | Objects that enable language-sensitive date and time formatting. |
| ListFormat: | Objects that enable language-sensitive list formatting. |
| NumberFormat: | Objects that enable language-sensitive number formatting. |
| PluralRules: | Objects that enable plural-sensitive formatting and language-specific rules for plurals. |
| RelativeTimeFormat: | Objects that enable language-sensitive relative time formatting. |

**171.How do you perform language specific date and time formatting?**

You can use **Intl.DateTimeFormat** object which is constructor for objects that enable language-sensitive date and time formatting.

```
var date = new Date(Date.UTC(2019, 07, 07, 3, 0, 0));
console.log(new Intl.DateTimeFormat('en-GB').format(date)); // 07/08/2019
console.log(new Intl.DateTimeFormat('en-AU').format(date)); // 07/08/2019
```

**172.What is an Iterator?**

An iterator is an object which defines a sequence(niz) and a return value upon its termination(po njegovom prestanku). It implements *the Iterator protocol* with a next() method which returns an object with two properties: value (the next value in the sequence) and done (which is true if the last value in the sequence has been consumed). An object becomes an iterator when it implements a next() method.

```
function makeRangeIterator(start = 0, end = Infinity, step = 1) {
  let nextIndex = start;
  let iterationCount = 0;

  const rangeIterator = {
    next() {
      let result;
      if (nextIndex < end) {
        result = { value: nextIndex, done: false };
        nextIndex += step;
        iterationCount++;
        return result;
      }
      return { value: iterationCount, done: true };
    }
  };
  return rangeIterator;
}
```

**173.What is an event loop?**

The event loop is the secret behind JavaScript's asynchronous programming. JS executes all operations on a single thread, but using a few smart data structures, it gives us the illusion of

multi-threading. The Event Loop is a queue* (an ordered(uredjena) list of elements) of callback functions. When an async function executes, the callback function is pushed into the queue. The JavaScript engine doesn't start processing the event loop until async function has finished executing the code. Note: It allows Node.js to perform non-blocking I/O operations eventhough JavaScript is single-threaded.

* The name *queue* comes from the analogy to a queue of customers at a bank. The customer who comes first will be served first, and the one who comes later is queued at the end of the queue and will be served later.

Note: use useCallback is to prevent a component from re-rendering unless its props have changed.

**174.What is an Unary operator?**

The unary(+) operator is used to convert a variable to a number.If the variable cannot be converted, it will still become a number but with the value NaN.

```
var x = "100";
var y = + x;
console.log(typeof x, typeof y);       // string, number

var a = "Hello";
var b = + a;
console.log(typeof a, typeof b, b);  // string, number, NaN
```

**175.How do you sort elements in an array?**

The **sort()** method is used to sort the elements of an array in place and returns the sorted array. The sort() overwrites the original array.

```
var months = ["Aug", "Sep", "Jan", "June"];
months.sort();
console.log(months); //  ["Aug", "Jan", "June", "Sep"]
```

**176.What is the purpose of compareFunction while sorting arrays?**

The **compareFunction** is used to define the sort order. If omitted, the array elements are converted to strings, then sorted according to each character's Unicode code point value.

```
let numbers = [1, 2, 5, 3, 4];
numbers.sort((a, b) => b - a);
console.log(numbers); // [5, 4, 3, 2, 1]
```

**177.How do you reversing an array?**

You can use **reverse()** method is used reverse the elements in an array. This method is useful to sort an array in descending order. The first array element now becoming the last, and the last array element becoming the first.

```
let numbers = [1, 2, 5, 3, 4];
numbers.sort((a, b) => b - a);
numbers.reverse();
console.log(numbers); // [1, 2, 3, 4 ,5]
```

### 178.How do you find min and max value in an array?

You can use **Math.min.apply** and **Math.max.apply** methods on array variable to find the minimum and maximum elements with in an array.

```
let num = [2, 3, 4, 5, 6, 7, 8, 9];
Math.min.apply(null, num);  // output: 2
Math.max.apply(null, num); // output: 9
```

### 179.How do you find min and max values without Math functions?

You can write functions which loops through an array comparing each value with the lowest value or highest value to find the min and max values.

```
const array = [-1, 2, -5, 8, 16];

let max = array[0], min = array[0];
for (let i = 0; i < array.length; i++) {

   // If we get any element in array greater
   // than max, max takes value of that
   // larger number
   if (array[i] > max) {
      max = array[i];
    }

   if (array[i] < min) {
     min = array[i];
  }
}
console.log("max = " + max);    // max = 16
console.log("min = " + min);     // min = -5
```

### 180.What are the differences between javascript and typescript?

| feature | Typescript | JavaScript |
|---|---|---|
| Language paradigm | Object oriented programming language | Scripting language |
| Typing support | Supports static typing | It has dynamic typing |
| Modules | Supported | Not supported |
| Interface | It has interfaces concept | Doesn't support interfaces |
| Optional parameters | Functions support optional parameters | No support of optional parameters for functions |

Cetiri osnovna pojma OOP:

**Apstrakcija -** proces skrivanja detalja implementacije i pokazivanja korisniku samo funkcionalnosti

**Enkapsulacija** (nivo zastite atributa/f-ja u okviru jedne klase iliti access modifiers):

- Public – javni pristup bez ogranicenja
- Protected – clanovima se pristupa okviru klase i njenih izvedenih klasa, mogu se naslediti
- Private – clanovima se pristupa samo u okviru klase, ne mogu se naslediti
- Defaultni je kada se nijedan ne navede i drugacije se tretira

**Nasledjivanje**
**Polimorfizam** - omogucava da osnovna klasa definise f-je koje ce biti zajednicke za sve izvedene klase, s tim da izvedene klase mogu na svoj nacin da implemetiraju te f-je

Skript je programski kod koji se izvršava u aplikacijama i pisan je u različitim programskim jezicima. Kao takav, nije kompajliran, nego interpretiran. To dalje znači da se izvorni kod ne prevodi u izvršni kod, već se izvorni kod direktno izvršava. Time se stvara mogućnost da se program izvršava zadajući komandu po komandu, što se najčešće koristi u toku razvoja programa za probavanje komandi.

Dynamically-typed languages perform type checking at runtime, while statically typed languages perform type checking at compile time.

### 181.What are the advantages of typescript over javascript?

TypeScript is able to find compile time errors at the development time only and it make sures less runtime errors. Whereas javascript is interpreted language.

TypeScript is is strongly-typed or supports static typing which allows for checking type correctness at compile time. This is not available in javascript.

TypeScript compiler can compile the .ts files into ES3,ES4 and ES5 unlike ES6 features of javascript which may not be supported in some browsers.

### 182.What is an object initializer?

An object initializer is an expression that describes the initialization of an Object. The syntax for this expression is represented as a comma-delimited (odvojeno zarezima) list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}). This is also known as literal notation. It is one of the ways to create an object.

```
var obj = {a: 'John', b: 50, c: {}};
console.log(obj.a); // John
```

### 183.What is a constructor method?

The constructor method is a special method for creating and initializing an object instance of a class. If you do not specify a constructor method, a default constructor is used. The purpose of a constructor is to create a new object and set values for any existing object properties.

```
class Employee {
  constructor() {
    this.name = "John";        // this. refers to created new object
  }
}
var obj = new Employee();
console.log(obj.name); // John
```

### 184.What happens if you write constructor more than once in a class?

The "constructor" in a class is a special method and it should be defined only once in a class. i.e, If you write a constructor method more than once in a class it will throw a SyntaxError error.

```
 class Employee {
   constructor() {
     this.name = "John";
```

```
  }
  constructor() {   //  Uncaught SyntaxError: A class may only have one constructor
    this.age = 30;
  }
}

var employeeObject = new Employee();
console.log(employeeObject.name);
```

**185.How do you call the constructor of a parent class?**

You can use **super** keyword to call the constructor of a parent class. Remember that super() must be called before using 'this' reference. Otherwise it will cause a reference error.

```
class Square extends Rectangle {
  constructor(length) {
    super(length, length);
    this.name = 'Square';
  }
  get area() {
    return this.width * this.height;
  }
  set area(value) {
    this.area = value;
  }
}
```

**186.What Is Obfuscation in javascript?**

Obfuscation is the deliberate(namerno) act of creating obfuscated(zamucen) javascript code that is difficult for humans to understand. It is something similar to encryption, but a machine can understand the code and execute it. Let's see the below function before Obfuscation,

```
function greeeting() {
   console.log('Hello, welcome to JS world');
}
```

And after the code Obfuscation, it would be appeared as below,

```
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--)
{d[c]=k[c]||c}k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c])
{p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('2 1(){0.3(\'4, 7 6 5
8\')}',9,9,'console|greeeting|function|log|Hello|JS|to|welcome|world'.split('|'),0,{}))
```

**187.Why do you need Obfuscation?**

- The code size will be reduced. So data transfers between server and client will be fast.
- It hides the business logic from outside world and protects the code from others
- Reverse engineering is highly difficult
- The download time will be reduced

**188.What is Minification?**

Minification is the process of removing all unnecessary characters(empty spaces are removed) and variables will be renamed without changing it's functionality. It is also a type of obfuscation.

**189.What are the advantages of minification?**

Normally it is recommend to use minification for heavy traffic and intensive requirements of resources. It reduces file sizes with below benefits, it decreases loading times of a web page and saves bandwidth(protok) usages.

**190.What are the differences between Obfuscation and Encryption?**

Enkripcija je proces u kriptografiji pomocu kojeg su podaci necitljivi za osobe koje ne poseduju odredjeni kljuc. Da bi podaci bili citljivi, potrebno ih je dekodirati procesom suprotnim od enkripcije, a to je dekripcija.

| Feature | Obfuscation | Encryption |
|---|---|---|
| Definition | Changing the form of any data in any other form | Changing the form of information to an unreadable format by using a key |
| A key to decode | It can be decoded without any key | It is required |
| Target data format | It will be converted to a complex form | Converted into an unreadable format |

**191.How do you perform form validation using javascript?**

JavaScript can be used to perform HTML form validation. For example, if form field is empty, the function needs to notify, and return false, to prevent the form being submitted. Lets' perform user login in an html form,

```
<form name="myForm" onsubmit="return validateForm()" method="post">
User name: <input type="text" name="uname">
<input type="submit" value="Submit">
</form>
```

And the validation on user login is below:

```
function validateForm() {
  var x = document.forms["myForm"]["uname"].value;
  if (x == "") {
    alert("The username shouldn't be empty");
    return false;
  }
}
```

**192.How do you perform form validation without javascript?**

You can perform HTML form validation automatically without using javascript. The validation enabled by applying required attribute to prevent form submission when the input is empty.

```
<form method="post">
  <input type="text" name="uname" required>
  <input type="submit" value="Submit">
</form>
```

Note: Automatic form validation does not work in Internet Explorer 9 or earlier.

**193.Is enums feature available in javascript?**

No, javascript does not natively support enums. But there are different kind of solutions to simulate them even though they may not provide exact equivalent. For example, you can use freeze or seal on object,

var DaysEnum = Object.freeze({"monday":1, "tuesday":2, "wednesday":3, ...})

**194.What is an enum?**

An enum is a type restricting variables to one value from a predefined set of constants. JavaScript has no enums but typescript provides built-in enum support.

```
enum Color {
        RED, GREEN, BLUE
}
```

**195.How do you list all properties of an object?**

You can use **Object.getOwnPropertyNames()** method which returns an array of all properties found directly in a given object. Let's the usage of it in an example,

```
const newObject = {
  a: 1,
  b: 2,
  c: 3
};

console.log(Object.getOwnPropertyNames(newObject));  ["a", "b", "c"]
```

## 196.How do you extend classes?

The extends keyword is used in class declarations/expressions to create a class which is a child of another class. It can be used to subclass custom classes as well as built-in objects. The syntax would be as below,

class ChildClass extends ParentClass { ... }

Let's take an example of Square subclass from Polygon parent class,

```
 class Square extends Rectangle {
   constructor(length) {
     super(length, length);
     this.name = 'Square';
   }

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

   set area(value) {
     this.area = value;
   }
```

```
}
```

**197.How do you check whether an array includes a particular value or not?**

The Array#includes() method is used to determine whether an array includes a particular value among its entries by returning either true or false. Let's see an example to find an element(numeric and string) with in array.

```
var numericArray = [1, 2, 3, 4];
console.log(numericArray.includes(3)); // true

var stringArray = ['green', 'yellow', 'blue'];
console.log(stringArray.includes('blue')); //true
```

**198.How do you print numbers with commas as thousand separators?**

You can use Number.prototype.toLocaleString() method which returns a string with a language-sensitive representation such as thousand separator,currency etc of this number.

```
function convertToThousandFormat(x){
  return x.toLocaleString(); // 12,345.679
}

console.log(convertToThousandFormat(12345.6789));
```

**199.Does javascript support namespace?**

JavaScript doesn’t support namespace by default. So if you create any element(function, method, object, variable) then it becomes global and pollute the global namespace. Let's take an example of defining two functions without any namespace,

```
function func1() {
   console.log("This is a first definition");
}
function func1() {
   console.log("This is a second definition");
}
func1(); // This is a second definition
```

It always calls the second function definition. In this case, namespace will solve the name collision problem.

**200.Why do we call javascript as dynamic language?**

JavaScript is a loosely typed or a dynamic language because variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned/re-assigned with values of all types.

```
let age = 50;    // age is a number now
age  = 'old'; // age is a string now
age  = true;  // age is a boolean
```

**201.What is the output of below for loops?**

```
for (var i = 0; i < 4; i++) { // global scope
  setTimeout(() => console.log(i));
}

for (let i = 0; i < 4; i++) { // block scope
  setTimeout(() => console.log(i));
}
```

The output of the above for loops is 4 4 4 4 and 0 1 2 3 Explanation: Due to event queue/loop of javascript, the setTimeout callback function is called after the loop has been executed. Since the variable i is declared with var keyword it became a global variable and the value was equal to 4 using iteration when the time setTimeout function is invoked (bcs var is referring to the same variable address – var je vezan za istu promenljivu van petlje). Hence, the output of the first loop is 4 4 4 4. Whereas in the second loop, the variable i is declared as let keyword it became a block scoped variable and it holds a new value(0, 1 ,2 3) for each iteration. Hence, the output of the first loop is 0 1 2 3.

**202.List down some of the features of ES6?**

- Support for constants or immutable variables
- Block-scope support for variables, constants and functions
- Arrow functions
- Default parameters
- Rest and Spread Parameters
- Template Literals
- Multi-line Strings
- Destructuring Assignment
- Enhanced Object Literals
- Promises
- Classes
- Modules

**203.What is ES6?**

ES6 is the sixth edition of the javascript language and it was released on June 2015. It was initially known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015. Almost all the modern browsers support ES6 but for the old browsers there are many transpilers, like Babel.js etc.

**204.Is const variable makes the value immutable?**

No, the const variable doesn't make the value immutable. The const declaration creates a read-only reference to a value. It does not mean the value it holds is immutable – the variable reference is immutable (ne mozemo dodeliti promeljivu drugom delu memorije).

```
const x = {a: 123};

// This is not allowed.  This would reassign `x` itself to refer to a
// different object.
x = {b: 456};

// This, however, is allowed.  This would mutate the object `x` refers to,
// but `x` itself hasn't been reassigned to refer to something else.
x.a = 456;
```

**205.What are default parameters?**

In E5, we handle default values of function parameters with logical OR operator. Whereas in ES6, Default function parameters feature allows parameters to be initialized with default values if no value or undefined is passed.

```
// Define a cube function with a default value
function cube(x = 5) {
  return x * x * x
}
// Invoke cube function without an argument
cube();         // output: 125

//ES5
var calculateArea = function(height, width) {
   height =  height || 50;
   width = width || 60;

   return width * height;
}
console.log(calculateArea()); //300
```

The default parameters makes the initialization more simpler,

```
//ES6
var calculateArea = function(height = 50, width = 60) {
   return width * height;
}
console.log(calculateArea()); //300
```

Default function parameters allow named parameters to be initialized with default values if no value or undefined is passed.

**206.What are template literals?**

Template literals or template strings are string literals allowing embedded expressions(ugradjene izraze). These are enclosed by the back-tick ` ` character instead of double or single quotes. In E6, this feature enables using dynamic expressions as below,

```
var greeting = `Welcome to JS World, Mr. ${firstName} ${lastName}.`
```

In ES5, you need break string like below,

```
var greeting = 'Welcome to JS World, Mr. ' + firstName + ' ' + lastName.`
```

**207.How do you write multi-line strings in template literals?**

In ES5, you would have to use newline escape character('\n') and concatenation symbol(+) in order to get multi-line strings.

```
console.log('This is string sentence 1\n' +
'This is string sentence 2');
```

Whereas in ES6, You don't need to mention any newline sequence character,

```
console.log(`This is string sentence
'This is string sentence 2`);
```

**208.What are nesting templates?**

The nesting templates is a feature supported with in template literals syntax to allow inner backticks inside a placeholder ${ } within the template. For example, the below nesting template is used to display the icons based on user permissions whereas outer template checks for platform type,

```
const iconStyles = `icon ${ isMobilePlatform() ? '' :
  `icon-${user.isAuthorized ? 'submit' : 'disabled'}` }`;
```

You can write the above usecase without nesting template feature as well. However, nesting template feature is more compact and readable.

```
//Without nesting templates
 const iconStyles = `icon ${ isMobilePlatform() ? '' :
  (user.isAuthorized ? 'icon-submit' : 'icon-disabled'}`;
```

209.**Stack and heap?**

Objekti se cuvaju na heap-u, a primitivni tipovi (string, number, boolean..) na stack-u.

A stack is used for static memory allocation, a heap is used for dynamic memory allocation. Stack memory management follows the LIFO (Last In First Out) order; storing variables creates space for new variable created by a function. The heap is a memory used by programming languages to store global variables. By default, all global variable are stored in heap memory space.

**210.What is destructuring assignment?**

The destructuring assignment is a JavaScript expression that makes it possible to unpack values from arrays or properties from objects into distinct variables.

```
var [one, two, three] = ['JAN', 'FEB', 'MARCH'];
console.log(one); // "JAN"
console.log(two); // "FEB"
console.log(three); // "MARCH"
```
and you can get user properties of an object using destructuring assignment,
```
var {name, age} = {name: 'John', age: 32};
console.log(name); // John
console.log(age); // 32
```

**211.What are default values in destructuring assignment?**

A variable can be assigned a default value when the value unpacked from the array or object is undefined during destructuring assignment. It helps to avoid setting default values separately for each assignment.

```
var x, y, z;
[x=2, y=4, z=6] = [10];
console.log(x); // 10
console.log(y); // 4
console.log(z); // 6
```

Objects destructuring:

```
var {x=2, y=4, z=6} = {x: 10};
console.log(x); // 10
console.log(y); // 4
console.log(z); // 6
```

**212.How do you swap variables in destructuring assignment?**

If you don't use destructuring assignment, swapping(zamena) two values requires a temporary variable. Whereas using destructuring feature, two variables values can be swapped in one destructuring expression.

```
var x = 10, y = 20;
[x, y] = [y, x];
console.log(x); // 20
console.log(y); // 10
```

**213.What are enhanced(unapredjeni) object literals?**

Object literals make it easy to quickly create objects with properties inside the curly braces. For example, it provides shorter syntax for common object property definition as below.

```
//ES6
var x = 10, y = 20
obj = { x, y }
console.log(obj); // {x: 10, y:20}

//ES5
var x = 10, y = 20
obj = { x : x, y : y}
console.log(obj); // {x: 10, y:20}
```

**214.What are dynamic imports?**

The dynamic imports using import() function syntax allows us to load modules on demand(na zahtev) by using promises or the async/await syntax. Currently this features is in stage4 proposal(https://github.com/tc39/proposal-dynamic-import). The main advantage of dynamic imports is smanjenje velicine naseg paketa, the size/payload response of our requests and overall improvements in the user experience. The syntax of dynamic imports would be as below,

```
import('./Module').then(Module => Module.method());
```

**215.What are the use cases for dynamic imports?**

Below are some of the use cases of using dynamic imports over static imports,
Import a module on-demand or conditionally. For example, if you want to load a polyfill on legacy browser

```
if (isLegacyBrowser()) {
   import(···)
   .then(···);
}
```

Compute the module specifier at runtime. For example, you can use it for internationalization.

```
import(`messages_${getLocale()}.js`).then(···);
```

Import a module from within a regular script instead a module.

**216.What is for...of statement?**

The for...of statement creates a loop iterating over iterable objects or elements such as built-in String, Array, Array-like objects (like arguments or NodeList), TypedArray, Map, Set, and user-defined iterables. The basic usage of for...of statement on arrays would be as below,

```
let arrayIterable = [10, 20, 30, 40, 50];
for (let value of arrayIterable) {
  value ++;
  console.log(value); // 11 21 31 41 51
}
```

**217.What is the output of below spread operator array?**

```
[...'John Resig']
```

The output of the array is ['J', 'o', 'h', 'n', '', 'R', 'e', 's', 'i', 'g'] Explanation: The string is an iterable type and the spread operator with in an array maps every character of an iterable to one element. Hence, each character of a string becomes an element within an Array.

**218.What paradigm is Javascript?**

JavaScript is a multi-paradigm language, supporting imperative/procedural programming, Object-Oriented Programming and functional programming. JavaScript supports Object-Oriented Programming with prototypical inheritance.

**219.What is the difference between internal and external javascript?**

Internal JavaScript: It is the source code with in the script tag. External JavaScript: The source code is stored in an external file(stored with .js extension) and referred with in the tag.

**220.Is JavaScript faster than server side script?**

Yes, JavaScript is faster than server side script. Because JavaScript is a client-side script it does require any web server’s help for its computation or calculation. So JavaScript is always faster than any server-side script like ASP, PHP, etc.

**221.How do you get the status of a checkbox?**

You can apply **checked** property on selected checkbox in the DOM. If the value is True means the checkbox is checked otherwise it is unchecked.

```
  <input type="checkbox" name="checkboxname" value="Agree"> Agree the conditions<br>
console.log(document.getElementById(‘checkboxname’).checked); // true or false
```

**222.What is the output of below string expression?**

```
console.log("Welcome to JS world"[0])
```

The output of the above expression is "W". Explanation: The bracket notation with specific index on a string returns the character at a specific location. Hence, it returns character "W" of the string. Since

this is not supported in IE7 and below versions, you may need to use .charAt() method to get the desired result.

**223.What is the purpose of Error object?**

The Error constructor creates an error object and the instances of error objects are thrown when runtime errors occur. The Error object can also be used as a base object for user-defined exceptions. The syntax of error object would be as below,

```
new Error([message[, fileName[, lineNumber]]])
```

You can throw user defined exceptions or errors using Error object in try...catch block as below,

```
try {
  if(withdraw > balance)
  throw new Error('Oops! You don't have enough balance');
} catch (e) {
  console.log(e.name + ': ' + e.message);
}
```

**224.Does all objects have prototypes?**

No. All objects have prototypes except for the base object which is created by the user, or an object that is created using the new keyword.

**225.What is the difference between a parameter and an argument?**

Parameter is the variable name of a function definition whereas an argument represent the value given to a function when it is invoked.

```
function myFunction(parameter1, parameter2, parameter3) {
  console.log(arguments[0]) // "argument1"
  console.log(arguments[1]) // "argument2"
  console.log(arguments[2]) // "argument3"
}
myFunction("argument1", "argument2", "argument3")
```

**226.What is the purpose of some method in arrays?**

The some() method is used to test whether at least one element in the array passes the test implemented by the provided function. The method returns a boolean value.

```
var array = [1, 2, 3, 4, 5, 6 ,7, 8, 9, 10];
var odd = element => element % 2 !== 0;
console.log(array.some(odd)); // true (the odd element exists)
```

**227.How do you combine two or more arrays?**

The **concat()** method is used to join two or more arrays by returning a new array containing all the elements.

```
array1.concat(array2, array3, ..., arrayX)
```

Let's take an example of array's concatenation with veggies and fruits arrays,

```
  var veggies = ["Tomato", "Carrot", "Cabbage"];
```

```
  var fruits = ["Apple", "Orange", "Pears"];
  var veggiesAndFruits = veggies.concat(fruits);
  console.log(veggiesAndFruits); // Tomato, Carrot, Cabbage, Apple, Orange, Pears
```

---

**228.What is the difference between Shallow and Deep copy?**

There are two ways to copy an object:

Shallow Copy

Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

Example

```
var empDetails = {
  name: "John",
  age: 25,
  expertise: "Software Developer"
}
```

to create a duplicate

```
var empCopy = empDetails;    //Shallow copying!
```

if we change some property value in the duplicate one like this:

```
empCopy.name = "Johnson"
```

The above statement will also change the name of empDetails, since we have a shallow copy. That means we're loosing the original data as well.

Deep copy

A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.

Example

```
var empDetails = {
  name: "John", age: 25, expertise: "Software Developer"
}
```

Create a deep copy by using the properties from the original object into new variable

```
var empCopy = {
  name: empDetails.name,
  age: empDetails.age,
  expertise: empDetails.expertise
}
```

Now if you change empCopy.name, it will only affect empCopy & not empDetails

**229.How do you create specific number of copies of a string?**

The **repeat()** method is used to construct and returns a new string which contains the specified number of copies of the string on which it was called, concatenated together. Remember that this

method has been added to the ECMAScript 2015 specification. Let's take an example of Hello string to repeat it 4 times,

```
'Hello'.repeat(4);  // 'HelloHelloHelloHello'
```

**230.How do you return all matching strings against a regular expression?**

The **matchAll()** method can be used to return an iterator of all results matching a string against a regular expression. For example, the below example returns an array of matching string results against a regular expression,

```
let regexp = /Hello(\d?))/g;
let greeting = 'Hello1Hello2Hello3';
let greetingList = [...greeting.matchAll(regexp)];
console.log(greetingList[0]); //Hello1
console.log(greetingList[1]); //Hello2
console.log(greetingList[2]); //Hello3
```

**231.How do you trim a string at the beginning or ending?**

The trim method of string prototype is used to trim on both sides of a string. But if you want to trim especially at the beginning or ending of the string then you can use trimStart/trimLeft and trimEnd/ trimRight methods.

```
var greeting = '   Hello, Goodmorning!   ';
console.log(greeting); // "   Hello, Goodmorning!   "
console.log(greeting.trimStart()); // "Hello, Goodmorning!   "
console.log(greeting.trimLeft()); // "Hello, Goodmorning!   "
console.log(greeting.trimEnd()); // "   Hello, Goodmorning!"
console.log(greeting.trimRight()); // "   Hello, Goodmorning!"
```

**232.What is the output of below console statement with unary operator?**

```
console.log(+ 'Hello');
```

The output of the above console log statement returns NaN. Because the element is prefixed by the unary operator and the JavaScript interpreter will try to convert that element into a number type. Since the conversion fails, the value of the statement results in NaN value.

**233.What is a thunk function?**

A thunk is just a function which delays the evaluation of the value(odlaze procenu vrednosti). It doesn't take any arguments but gives the value whenever you invoke the thunk. i.e, It is used not to execute now but it will be sometime in the future. Let's take a synchronous example,

```
const add = (x,y) => x + y;
const thunk = ( ) => add(2,3);
thunk( ) // 5
```

**234.What are asynchronous thunks?**

The asynchronous thunks are useful to make network requests. Let's see an example of network requests,

```
function fetchData(fn){
```

```
  fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())
  .then(json => fn(json))
}
const asyncThunk = function (){
   return fetchData(function getData(data){
      console.log(data)
  })
}
asyncThunk()
```

The getData function won't be called immediately but it will be invoked only when the data is available from API endpoint. The setTimeout function is also used to make our code asynchronous. The best real time example is redux state management library which uses the asynchronous thunks to delay the actions to dispatch.

**235.How to remove all line breaks from a string?**

The easiest approach is using regular expressions to detect and **replace** newlines in the string. In this case, we use replace function along with string to replace with, which in our case is an empty string.

```
function remove_linebreaks( var message ) {
    return message.replace( /[\r\n]+/gm, "" );
}
```

In the above expression, g and m are for global and multiline flags.

**236.What is the difference between reflow and repaint?**

A repaint occurs when changes are made which affect the visibility of an element, but not its layout. Examples of this include outline, visibility, or background color. A reflow involves changes that affect the layout of a part of the page (or the whole page). Resizing the browser window, changing the font, content changing (such as user typing text), using JavaScript methods involving computed styles, adding or removing elements from the DOM, and changing an element's classes are a few of the things that can trigger reflow. Reflow of an element causes the subsequent reflow of all child and ancestor elements as well as any elements following it in the DOM (Ponovno preoblikovanje elementa uzrokuje naknadno preoblikovanje svih podređenih i pretka elemenata, kao i svih elemenata koji ga prate u DOM-u).

**237.How do you remove falsy values from an array?**

You can apply filter method on array by passing Boolean as parameter. This way it removes all falsy values(0, undefined, null, false and "") from the array.

```
const myArray = [false, null, 1,5, undefined]
myArray.filter(Boolean); // [1, 5] // is same as myArray.filter(x => x);
```

**238.How do you get unique values of an array?**

You can get unique values of an array with the combination of Set and rest expression/spread(...) syntax.

```
console.log([...new Set([1, 2, 4, 4, 3])]); // [1, 2, 4, 3]
```

**239.What is destructuring aliases?**

Sometimes you would like to have destructured variable with a different name than the property name. In that case, you'll use a : newName to specify a name for the variable. This process is called destructuring aliases.

```
const obj = { x: 1 };
const { x: otherName } = obj;  // Grabs obj.x as a { otherName }
```

**240.How do you map the array values without using map method?**

You can map the array values without using map method by just using **from** method of Array. Let's map city names from Countries array,

```
const countries = [
   { name: 'India', capital: 'Delhi' },
   { name: 'US', capital: 'Washington' },
   { name: 'Russia', capital: 'Moscow' },
   { name: 'Singapore', capital: 'Singapore' },
   { name: 'China', capital: 'Beijing' },
   { name: 'France', capital: 'Paris' },
];
const cityNames = Array.from(countries, ({ capital}) => capital);
console.log(cityNames); // ['Delhi, 'Washington', 'Moscow', 'Singapore', 'Beijing', 'Paris']
```

**241.How do you empty an array?**

You can empty an array quicky by setting the array length to zero.

```
let cities = ['Singapore', 'Delhi', 'London'];
cities.length = 0; // cities becomes []
```

**242.How do you rounding numbers to certain decimals?**

You can rounding numbers to a certain number of decimals using **toFixed** method from native javascript.

```
let pi = 3.141592653;
pi = pie.toFixed(3); // 3.142
```

**243.What is the easiest way to convert an array to an object?**

You can convert an array to an object with the same data using spread(...) operator.

```
var fruits = ["banana", "apple", "orange", "watermelon"];
var fruitsObject = {...fruits};
console.log(fruitsObject); // {0: "banana", 1: "apple", 2: "orange", 3: "watermelon"}
```

**244.How do you create an array with some data?**

You can create an array with some data or an array with the same values using **fill** method.

```
var newArray = new Array(5).fill("0");
console.log(newArray); // ["0", "0", "0", "0", "0"]
```

**245.How do you verify that an argument is a Number or not?**

The combination of **IsNaN** and **isFinite(beskonacan)** methods are used to confirm whether an argument is a number or not.

```
function isNumber(n){
   return !isNaN(parseFloat(n)) && isFinite(n);
}
```

The parseFloat() function is used to accept a string and convert it into a floating-point number

**246.What is the shortcut to get timestamp?**

You can use new Date().getTime() to get the current timestamp(vremensku oznaku). There is an alternative shortcut to get the value.

```
console.log(+new Date());
console.log(Date.now());
```

**247.What is the easiest multi condition checking?**

You can use **indexOf** to compare input with multiple values instead of checking each value as one condition.

```
// Verbose approach
if (input === 'first' || input === 1 || input === 'second' || input === 2) {
  someFunction();
}

// Shortcut
if (['first', 1, 'second', 2].indexOf(input) !== -1) {
  someFunction();
}
```

**248.How do you capture browser back button?**

The window.**onbeforeunload** method is used to capture browser back button event. This is helpful to warn user about loosing the current data.

```
window.onbeforeunload = function() {
    alert("You work will be lost");
};
```

**249.Map:**

- Map is a collection of key/value pairs that can use any type of data as a key or value and remember the order of its entries. It is used to iterate over all the elements in an array, which results in a new array. Map will iterate through each item in the array and allow us to transform each item in some way, so the result will be another array with transformed values (it doesn't change the original array). Map calls a function once for each element in an array. Map doesn't execute the function for empty elements.

```
Const names = characters.map((character) => { return character.name; })
                    is same as
Const names = characters.map((character) => character.name )
```

-This method accepts two parameters: *array.map( function(currentValue, index, arr), thisValue)*

- **function(currentValue, index, arr):** Required parameter. A function to be run for each array element. It contains three parameters which are listed below:
  - **currentValue:** Required. The value of the current element.
  - **index:** Optional. The index of the current element.
  - **arr:** Optional. The array of the current element.
- **thisValue:** Optional. Default value undefined. A value passed to the function to be used as its this value.

- `map` is used to transform each element of an array, while `forEach` is used to run a function on each element without changing the array. One of the main differences between forEach() and map() methods is their ability to chain other methods. map() is chainable but forEach isn't. This means that one could use reduce(), sort(), and other methods after map(), but that's not possible with foreach() because it returns undefined.

To map through an object's value in React:

- Use the **Object.values()** method to get an array of the object's values.
- Call the map() method on the array of values.

```
const employee = {
 id: 1,
 name: 'Alice',
 salary: 100,
};
console.log(Object.values(employee));                    //  [1, 'Alice', 100]
```

---

```
export default function App() {
 const employee = {
  id: 1,
  name: 'Alice',
  salary: 100,
 };

 return (
  <div>
   {/* iterate object VALUES */}
   {Object.values(employee).map((value, index) => {
    return (
     <div key={index}>
      <h2>{value}</h2>
      <hr />
     </div>
    ); })}
  </div>
 ); }
```

**250.How to map multiple arrays with JavaScript?**

To map multiple arrays with JavaScript, we can use the map method.

```
const zip = (a1, a2) => a1.map((x, i) => [x, a2[i]]);
const arr1 = ['a', 'b', 'c'];
const arr2 = [1, 2, 3];
console.log(zip(arr1, arr2))
```

To define the zip function that calls a1.map to combine the entries from a1 and a2 with index i into one entry.
Then we call zip with arr1 and arr2 to zip them into one array.
As a result, the array logged is [["a", 1], ["b", 2], ["c", 3]].

**251.Reduce:**

```
//get total height of all characters
Const totalHeight = characters.reduce((acc, cur) => acc + cur.height, 0);  // 0 is starting value of acc

// get total number of characters by eye color
Const charByEyeColor = characters.reduce((acc, cur) => {
    Const color = cur.eye_color;
    If (acc [color]) {              // if we already have eye color set in our object, it'll be increased by 1
        Acc [color])++;
    } else {
        Acc [color] = 1;        // if we saw that eye color for the first time, set it to 1 bcs we just saw
it
    } return acc;
    }, { } );       ---> result: { blue: 2, yellow: 1, brown: 1 }
```

**252.Managed vs unmanaged code:**

**Managed code** je kod koji se direktno izvrsava od strane runtime okruzenja (to je kao mali operativni sistem koji omogucuje programu da izvrsava sve neophodne funkcionalnosti). Runtime biblioteke omogucuju exception handling, garbage collection, type checking.. sve je to omoguceno automatski. Runtime biblioteke upravljaju alokacijom i dealokacijom memorije i garbage kolektorom, sto znaci da ne moramo puno da brinemo o memoriji.

Prednosti:

- Sigurnost koda – jer runtime okruzenje stiti od buffer overflow-a (prelivanja) tako sto proverava memorijske bafere
- Garbage collector je automatski implementiran
- Dynamic type checking
- Proverava da li je referenca na objekat validna i da li ima duplikata medju objektima

Mane:

- Memorija ne moze biti direktno alocirana
- Ne moze se pristupiti CPU arhitekturi

Buffer – memorijski prostor koji trenutno zadrzava podatke dok se transferuju iz jedne lokacije u drugu. Buffer overflow se desi kada velicina podataka prekoraci kapacitet memorijskog bafera. Primer: bafer za log in kredencijale moze biti napravljen tako da ocekuje username i pass od 8 bajta, ali ako mu se prosledi 10 bajta onda ce doci do overflowa.
Garbage collector – je automatsko upravljanje memorije koje vraca memoriju koju je program alocirao ali se vise ne koristi (nije referencirana)
Memory leak – desi se kada programer kreira memoriju u heap-u i zaboravi da je obrise, cime se smanjuje kolicina dostupne memorije i performanse racunara.

Unmanaged code  ne omogucava pristup alokacije, dealokacije memorije. Naglasimo programu da ce upravljanje koda biti uradjeno od strane nas tako sto se napise "unsafe". Ono sto je najgore sto moze da se desi jeste memory leak. Kod se izvrsava direktno od strane operativnog sistema,
Prednosti:

- Unsafe kod povecava performanse programa
- Omogucen nam je low-level access

Mane:

- Security nije omogucena u app
- Programer mora da radi exception handling
- Nema automatske implementacije garbage collector-a

Razlike:

| Managed Code | Unmanaged Code |
|---|---|
| Izvrsen je od strane runtime okruzenja | Izvrsen je od strane operativnog sistema |
| Omogucuje security koda | Ne omogucuje nikakav security |
| Memory buffer overflow se ne desava jer memorijom upravlja runtime okruzenje | Memory buffer overflow may occur. |
| Omogucuje runtime services like Garbage Collection, exception handling.. | It does not provide runtime services like Garbage Collection, exception handling, etc. |
| The source code se kompajlira u intermediate language known as *IL or MSIL or CIL*. | The source code directly compiles into native languages. |
| It does not provide low-level access to the programmer. | It provide low-level access to the programmer. |