**1. Šta je nginx?**
**2. Kakve funkcionalnosti nginx omogućava?**

- Open-source web server koji se, nakon inicijalnog uspeha kao web server, u zadnje vreme koristi i kao reverse proxy, HTTP cache te load balancer. Recimo kada neko poseti sajt on prvo pristupa Nginx serveru koji keširani sadržaj sajta odmah isporuči klijentu bez čekanja ili bilo kakve obrade.

**3. Šta je ECMAScript i koje verzije postoje?**

- Specifikacija skriptnog jezika, standardizovana od strane Ecma International. Kreirana da bi se standardizovao JavaScript koji je ostao najpoznatija implementacija ECMAScript standarda. Postoje verzije od 1 do 12, poslednja objavljena 2021.

**4. Šta označava "use strict"?**

- Strict mode-om se uvode brojna ograničenja koja pomažu da kod bude sigurniji, uglavnom tako što će ranije prihvaćena "loša sintaksa" sada izbacivati greške. Ideja je sprečiti sve greške preko kojih interpreter može da pređe.

**5. Šta je forEach i kako funkcioniše?**

- Predstavlja petlju koja prolazi kroz sve elemente niza i izvršava kod u telu petlje.

```
ourArray.forEach( element => /* telo petlje */);
```

**6. Getters & setters?**

- Funkcije koje služe za upisivanje i čitanje nekog svojstva (property), kako se ne bi pristupalo svojstvima kao "golim" podacima.

```
let user = {
  firstName: "John",
  lastName: "Smith",

  get firstName() {
    return this.firstName;
  }

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

**7. JS prototype?**

- Objekti u JavaScriptu imaju ugrađeni mehanizam nasleđivanja kroz takozvano prototipsko nasleđivanje. To je poseban stil objektno-orijentisanog programiranja koje se zasniva na delegiranju, gde svaki objekat ima svoja svojstva i metode, ali i posebnu vezu ka roditeljskom objektu od kojeg nasleđuje svojstva i metode.

**8. Object.create?**

- Object.create() je metoda koja kreira novi objekat, koristeći neki postojeći objekat kao njegov prototip.

```
const newObject = Object.create(existingObject);
```

**9. Var, let i const?**

- Ključne reči kojima se deklarišu varijable. Neke od najbitnijih razlika:
  - let i const varijable se ne mogu deklarisati globalno, već samo u okviru lokalnog domena tj. ako su deklarisane u okviru vitičastih zagrada onda je njihov domen ograničen tim zagradama
  - const varijablama se ne može ponovo dodeliti nov sadržaj
  - let i const se ne mogu redeklarisati ponovo u istom domenu, dok var može

```
if(1) {
    let x;
    let x = 2; //SyntaxError
}
```

**10. For, for in, for of?**

- For je najčešće korišćena petlja, koja prati "iteration statement" strukturu tj. u jednom redu se navode ključne tačke petlje i odvojene su sa tačka-zarezom.

```
for( [početna vrednost]; [uslov]; [izraz za povećavanje]){
    /* telo petlje */
}

for( var i = 0; i<10; i++) {
    console.log(i);
}
//ispis brojeva od 0 do 9
```

- For in je specifična petlja koja prolazi kroz sva svojstva nekog objekta kod kojih je deskriptor svojstva enumerable, setovan na true. Može se koristi i u radu sa nizovima ali postoji dosta neželjenih sporednih efekata, tako da nije preporučljivo.

```
var element = { property1: "property1", property2: "property2"};

for (var key in element) {
    console.log(element[key] + " ");
}
//ispis svojstava objekta 'element'
```

- For of prolazi kroz vrednosti bilo koje iterabilne kolekcije (Array, Map, Set,

String, ...), ali nije planiran za iteraciju kroz svojstva objekata.

```
let numArray = [1, 2, 3];

for (let arrayElement of numArray) {
    console.log(arrayElement)
}
//ispis elemenata niza
```

**11. JS iterables?**

- Iterabilni objekat kroz koji želimo da vršimo iteracije kako bismo u svakoj iteraciji dolazili do njegovog novog podatka. Iterator ima obaveznu metodu next(), koja nam vraća novu vrednost tj. podatak ali spakovan unutar objekta koji ima dva svojsta. Ta svojstva su value koje predstavlja sam podatak i done koje predstavlja logičku vrednost, gde je false ako ima još podataka ili true ukoliko smo iterisali kroz sve podatke tog objekta.

```
/* ovde treba kreirati objekat i dodati iteratorsku funkciju */
let iterator = obj[Symbol.iterator]();
while(true) {
    const element = iterator.next();
    console.log(element.value);
    if(element.done)
        break;
}
//ispis podataka nekog objekta
```

**12. JS Map & Set?**

- Map je kolekcija parova ključ/vrednost, gde ključ može biti bilo koji tip podataka za razliku od objekta gde su ključevi string ili symbol. Ima ceo set izgrađenih metoda za rad sa elementima kolekcije: set, delete, get, keys, values, clear, entries i ima svojstvo size.

```
var ourMap = new Map();
```

- Set je tip podataka sličan nizovima, sa jednom razlikom, a to je da ne dozvoljava dupliranje vrednosti. Metode koje su dostupne su: add, delete, has - ispituje postojanje neke vrednosti, values, entries, clear, forEach, a svojstva size i constructor.

```
var ourSet = new Set();
```

**13. JS Class?**

- Klase u JavaScript su samo drugačiji način predstavljanja konstruktorske funkcije i metoda iz "prototype objekta". Rezervisana reč "class" ukazuje na to da je u pitanju specijalna vrsta funkcije, koja za razliku od obične ne može da se pozove već se jedino može koristiti uz rezervisanu reč "new".

```
class Parent {
```

```
        constructor() {
            console.log("Parent");
            this.luckyNumber = 43;
        }

        get getLuckyNumber() {
            console.log(this.luckyNumber);
        }

        set setLuckyNumber(newLuckyNumber) {
            this.luckyNumber = newLuckyNumber;
            this.getLuckyNumber;
        }
    }

    class Child extends Parent {
        constructor() {
            super(); //uvek moramo pozvati konstruktor klase koju nasleđujemo
            console.log("Child");
        }
    }

    var person = new Child();
    // Izlaz će biti:
    // Parent
    // Child
    var parent = new Parent();
    parent.getLuckyNumber; //vraća vrednost srećnog broja tj. izlaz je 43
    parent.setLuckyNumber = 7; //setuje srećni broj na novu vrednost i ispisuje
ga tj. izlaz je 7
```

**14. JS Promise?**

- Promise je objekat koji pri asinhronoj operaciji, umesto krajnje vrednosti, "daje obećanje" da će dostaviti tu vrednost u nekom trenutku u budućnosti. Može biti u tri stanja, a to su pending (radnja se još izvršava), fulfill (radnja završena uspešno) i reject (radnja neuspešno završena tj. greška).

```
    function asyncFunction() {
        return new Promise(function(resolve, reject) {
            /* kod za asinhronu operaciju */

            if(success) {
                resolve(resultValue);
            }
            else {
                reject(error);
            }
        });
    }

    asyncFunction.then( //čeka dok se asinhrona operacija ne završi
```

```
        function(resultValue) {/* deo koda kada je uspešna asinhrona operacija */},
        function(error) {/* deo koda kada je neuspešna asinhrona operacija */}
    )
```

**15. Fetch API?**

- Fetch API je sistem za slanje asinhronih zahteva serveru i obradu primljenih podataka, a implementiran je preko promise-a. U osnovi ima funkciju fetch(file) koja kao argument prihvata naziv datoteke, a vraća promise koji sadrži rezultat.

```
    fetch(file)
        .then(result => result.json()) //.text() ako je rez običan tekst, .blob() ako je binarna datoteka (npr. slika)
        .catch( error => console.log(error)) //reaguje u slučaju greške

    fetch(file, params) //poziv sa dodatnim parametrima params kada radimo POST zahtev
```

**16. Symbol type?**

- Predstavlja jedinstveni token i kreira se koristeći funkciju Symbol().

```
    newSymbol = Symbol();
```

- Uglavnom se koristi kada želimo da zaštitimo neko svojstvo.

```
    let id = Symbol();

    person[id] = 1;
```

- Svojstvo id ignoriše većina metoda, ali privatnost nije 100% i postoje metode koje imaju i dalje pristup kao .getOwnPropertySymbols() i .ownKeys().

**17. Function default & rest?**

- Funkcije dozvoljavaju da imamo default parametre.

```
    const sum = function(a, b = 1) => a + b;
    sum(3); //vraća 4
```

- Rest sintaksa dozvoljava da predstavimo neograničen broj argumenata kao niz. Grupiše preostale argumente u niz.

```
    function sumFunction(...numbers) {
        var sum = 0;
        numbers.forEach(number => sum += number);
        return sum;
    }
```

**18. NJS Async non blocking vs Sync blocking?**

- Blokirajuće metode se izvršavaju sinhrono, dok se neblokirajuće izvršavaju

asinhrono. Mana prvog pristupa je što se zaustavlja svako dalje izvršavanje JavaScript-a dok se ne završi radnja metode u celosti, a takođe moramo da "hvatamo" greške, što je u asinhronom pristupu ostavljeno programeru da odluči.

**19. NJS Async event queue loop?**

- Svi pozivi idu u event queue. Neblokirajuće operacije se odmah izvršavaju i šalje se odgovor â€" response. Za pozive blokirajućih operacija koje se ne mogu odmah izvršiti i za koje se ne zna koliko će trajati registruje se callback funkcija.

**20. Implementacija blokirajućih operacija - Worker Pool?**

- Node.js koristi Worker Pool za baratanje "skupim" zadacima. Ovo uključuje I/O za koje operativni sistem ne nudi neblokirajuću verziju, a takođe i za naročito procesorski zahtevne zadatke. Worker Pool je implementiran u biblioteci libuv.

**21. NJS Async callback funkcija?**

- Callback je mehanizam koji omogućava da se funkcija prosledi kao parametar, da bi kasnije bila pozvana po potrebi. U najvećem broju slučajeva callback funkcije koristimo kao funkcije za obradu događaja. Nakon poziva neke asinhrone operacije, da ne bismo čekali odgovor čime blokiramo izvršavanje ostatka koda, mi prosleđujemo callback funkciju koja će po prijemu odgovora da nastavi izvršenje željenih akcija sa dobijenim odgovorom.

**22. process.nextTick & setImmediate?**

- process.nextTick(callback) - zakazuje izvršavanje callback funkcije pre početka sledeće iteracije Event loop-a.

- setImmediate(callback) - ima manji prioritet od process.nextTick jer se izvršava tek pri sledećoj iteraciji Event loop-a.

**23. EventEmitter?**

- Modul koji olakšava komunikaciju između objekata u Node-u, predstavlja jedan od najbitnijih delova Node-ove asinhrone arhitekture vođene događajima. Emitter objekti emituju imenovane događaje koji izazivaju pozivanje prethodno registrovanih slušalaca (listener).

```
    const ourEmitter = new EventEmitter();

    function listenerFunction() {
        console.log("Listener function called");
    }

    ourEmitter.on("eventOccured", listenerFunction); //kada god se emituje
'eventOccured' pozvaće se listenerFunction
    /* da smo umesto .on stavili .once posle prvog emitovanja 'eventOccured',
listener bi prestao da hvata nova emitovanja ovog događaja */
```

```
ourEmitter.emit("eventOccured");
ourEmitter.emit("eventOccured");
//Izlaz:
//Listener function called
//Listener function called

ourEmitter.off("eventOccured", listenerFunction); //odjava slušaoca
```

**24. JS closure?**

- Closure je unutrašnja funkcija koja je referencirana van svoje nadfunkcije ali i dalje ima pristup lokalnim promenljivama nadfunkcije.

```
function incrementCounter() {
    var counter = 0;
    return function () {
        return counter++;
    };
}

var count = incrementCounter();

count(); //Izlaz: 0
count(); //Izlaz: 1
count(); //Izlaz: 2
```

**25. Callback chain?**

- Kada želimo da lančano pozovemo više funkcija jednu za drugom (kada se prva završi pozivamo drugu, kada se druga završi treću, ...) to možemo uraditi tako što ćemo sve te funkcije smestiti u niz i pozvati ih rekurzivno chain metodom.

```
let functionArray = [funcOne, funcTwo, funcThree, funcFour]; //niz funkcija

function chain(func) {
    if(func) {
        func(() => chain(functionArray.shift()));
    }
}

chain(functionArray.shift()); //počinjemo ulančavanje pozivom za prvi element niza
```

**26. NJS Buffer?**

- Buffer se koristi u radu sa binarnim podacima. Postoji sličnost sa nizovima ali je glavna razlika što elementi niza mogu da budu bilo kog tipa, dok buffer radi isključivo sa binarnim podacima, a takođe veličina niza je promenljiva, dok buffer ima fiksnu veličinu koju određujemo pri njegovom kreiranju.

```
/* načini kreiranja bafera */
var ourBuffer1 = Buffer.alloc(100);
```

```
var ourBuffer2 = new Buffer("ABC");
var ourBuffer3 = Buffer.from([1,2,3,4]);
```

**27. NJS Stream?**

- Tokovi su metod za rukovanje podacima i koriste se za sekvencijalno čitanje ili upisivanje unosa u izlaz. Umesto da program upisuje podatke u memoriju odjednom kao na tradicionalan način, tokovi čitaju delove podataka deo po deo, obrađujući njegov sadržaj bez da drže sve podatke u memoriji.

**28. NJS kompresija (zlib modul)?**

- Zlib modul omogućava kompresiju i dekompresiju fajlova, koristeći Gzip, Deflate/Inflate i Brotli. Svaki metod za kompresiju (deflate, deflateRaw, gzip) ima odgovarajući metod za dekompresiju (inflate, inflateRaw, gunzip). Svi metodi rade asinhrono, tj. koriste callback, što se može videti u ispisu koji ne prati redosled poziva u listingu.

**29. NJS File read/write (fs modul)?**

- Fs (file system) modul najčešće se koristi za čitanje, fs.readFile, i pisanje, fs.writeFile, u fajl ili zamena fajla ukoliko on već postoji sa novim. Ponuđene su nam opcije sinhrone i asinhrone verzije. Sinhrona verzija najviše ima primenu na početku programa ako učitavamo neki konfiguracioni fajl, dok se u ostatku obično koristi asinhrona verzija.

**30. File system promise?**

- fs/promises API obezbeđuje asinhrone metode sistema datoteka koje vraćaju obećanja. Ovaj API koristi osnovni skup niti Node.js-a za obavljanje operacija sistema datoteka izvan niti petlje događaja. Ove operacije nisu sinhronizovane ili bezbedne za niti, tako da treba biti oprezan kada se vrše istovremene modifikacije na istom fajlu jer može doći do oštećenja podataka.

**31. NJS http server?**
**32. NJS http client?**

- Node.js ima ugrađeni modul HTTP, pomoću kojeg možemo da šaljemo podatke preko Hyper Text Transfer Protocol-a (HTTP-a). uključujemo ga metodom require() tj. sa:

```
var http = require('http');
```

- HTTP modul može kreirati HTTP server koji sluša portove servera i vraća odgovor klijentu. Takođe može da šalje HTTP zahteve drugim serverima.

**33. NJS http predaja podataka? *******

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
```

```
    }).listen(8080);

    var options = {
      host: 'www.nodejitsu.com',
      path: '/',
      port: '1338',
      headers: {'custom': 'Custom Header Demo works'} //objekat sa zaglavljima
(header-ima) za traženje
    };
    http.request();
```

**34. HTTPS public/private key?**

- HTTPS je protokol za šifrovanu komunikaciju između klijenta i servera. HTTPS koristi SSL (Secure Sockets Layer) / TLS (Transport Layer Security) protokol. Komunikacija se šifruje korišćenjem tajnog ključa koji se razmenjuje preko PKI (Public Key Infrastrucure). PKI zamenjuje sigurnog kurira za razmenu simetričnog kratkoročnog tajnog ključa kojim se šifruje HTTPS komunikacija. Javni ključ tj. public key predstavlja sertifikat koji dokazuje identitet pošiljaoca. Tajni i javni ključ izdaje CA (Certifiate Authority) koji potpisuje izdate ključeve.

**35. NJS sockets (net moduli)?**

- Soketima se obezbeđuje višesmerna komunikacija (duplex) tj. istovremena komunikacija u oba smera. Postoji stalna veza klijenta i servera, gde obe strane mogu istovremeno da salju poruke jedni drugima.

- Net modul obezbeđuje asinhroni mrežni API za kreiranje TCP ili IPC servera (net.createServer()) i klijenata (net.createConnection()) zasnovanih na stream-u.

**36. NJS Express?**

- NJS okvir (framework) koji omogućava razne dodatne funkcionalnosti, u odnosu na HTTP, kao što su rutiranje i serviranje statičkih fajlova.

```
    const express = require("express");
    const app = express();
```

**37. Express static files (css, slike)?**

- Za serviranje statičkih datoteka kao što su slike, CSS datoteke i JavaScript datoteke, koristi se express.static middleware funkcija.

```
      expres.static(root, [options]) //root predstavlja root direktorijum odakle
treba da serviramo statičke datoteke

      app.use(express.static('public')); //sada možemo da učitamo fajlove koji su
u 'public' direktorijumu
```

**38. Express rutiranje method -> get, post, put, delete, itd?**

```
    app.method(path, callback [,callback ...])
```

=> method - naziv metode tj. get, post, itd.
=> path - putanja za koju se poziva middleware funkcija
=> callback - može biti middleware funkcija, više middleware funkcija odvojenih zarezom, niz middleware funkcija ili kombinacija navedenog

```
app.get("/user", function(req, res) {
    res.send("Handled GET request");
});
```

**39. Express predaja podataka? *******

- Parametri rute su imenovani delovi putanje (URL-a) koji se koriste da prihvate vrednosti navedene na poziciji gde se nalaze. Vrednosti parametara se prihvataju iz koda sa req.params i nazivom parametra kao ključem: req.params.userId, req.params.bookId i slično.

**40. Express callback next?**

- Pozivanjem next() funkcije izvršava se middleware koji je naslednik trenutnog middleware-a. Ako trenutni middleware ne završi ciklus zahtev-odgovor, mora pozvati next() da bi preneo kontrolu narednom middleware-u. U suprotnom zahtev će ostati da "visi".

```
const express = require("express")
const app = express()

app.use("/", (req, res, next) => {
    console.log("Hello");
    next();
})

app.get("/", (req, res, next) => {
    console.log("World");
})

app.listen(3000, () => {
    console.log("Server is running");
})

//Izlaz:
//Server is running
//Hello
//World
```

**41. Express modularni ruteri?**

```
/* index.js */
const express = require('express');
const app = express();
const albumsRouter = require('./routers/albums');
```

```
    //...

    app.use('/albums', albumsRouter); // prosledjuje bilo koji zahtev /albums
putanje nasem albums router-u

    //...

    /* albums.js */
    const albums = require('express').Router();

    //...

    // app.use prosleđuje zahtev našem albums router-u
    // tako da ova ruta obrađuje zahtev /albums
    albums.get('/', function(req, res, next) {
      // res.send() nas odgovor
    });


    // ruta za neki specifičan album
    albums.get('/:albumId', function(req, res, next) {
      let myAlbumId = req.params.albumId;
      // vraćamo album sa zadatim albumId i šaljemo odgovor sa res.send()
    });

    //...

    module.exports = albums;
```

**42. Express middleware?**

- Middleware funkcije su funkcije koje imaju pristup objektu zahteva (req), objektu odgovora (res) i sledećoj funkciji middleware-a u zahtev-odgovor ciklusu aplikacije. Sledeća middleware funkcija se obično označava promenljivom koja se zove next.

- Middleware funkcije mogu da obavljaju sledeće zadatke:
  - Izvršite bilo koji kod
  - Izmenite objekte zahteva i odgovora
  - Završite ciklus zahtev-odgovor
  - Pozovite sledeću middleware funkciju u steku

- Ako trenutna middleware funkcija ne završi ciklus zahtev-odgovor, mora pozvati next() da bi prenela kontrolu sledećoj middleware funkciji. U suprotnom, zahtev će ostati da "visi".

**43. Šta je TypeScript (TS)?**

- TypeScript (TS) objektno-orijentisani programski jezik i kompajler su nastali 2012 godine. Osnovna ideja TypeScript-a je da se uvedu brojna ograničenja kao kod strogo tipiziranih jezika (C#, JAVA) koja će omogućiti razvoj velikih i složenih

projekata u JavaScript-u bez izmene samog JavaScript jezika.

**44. TS deklaracije tipova?**

```
var name: string;
var age: number;
var luckyCharm: any;
```

**45. TS izvedeno tipiziranje?**

- Složeni tipovi (tipovi izvedeni iz osnovnih) su object i function.

```
function funcName(param1: param1Type, param2: param2Type,...): returnType {}

function add(firstNumber: number, secondNumber: number): number {
    return firstNumber + secondNumber;
}
```

- Object tip je identičan kao u JS.

```
let user {
    firstName: "Vasa",
    lastName: "Tajcic",
    age: 43
}
```

**46. TS deklaracije - *.d.ts file?**

```
/* my-module.js */
const randNum = 43;

module.exports = {randNum};

/* my-module.d.ts */
export declare const randNum: number;

/* index.ts */
import { randNum } from "./my-module"

console.log(randNum); // 43, uzima iz .js fajla
console.log(typeof randNum); // number, ovo uzima iz .d.ts fajla
```

- File-ovi tipa *.d.ts se koriste za povezivanje postojećeg JS koda sa novim TS kodom. Bez dodatne specifikacije postojećeg JS koda za novi TS, to nije moguće, TS prevodilac će prijaviti greške da objekti/promenljive iz postojećeg JS koda nisu definisani. Deklaracijom svih objekata/promenljivih u postojećem JS kodu sa declare, kao u prikazanom kodu iznad, TS prevodilac zna kog tipa su ti objekti/promenljive i može da prevede takav kod, kombinaciju postojećeg JS koda koji se koristi sa novim TS kodom.

**47. Any & enum?**

- Enum je nabrojivi tip, omogućava definisanje skupa imenovanih konstanti:

```
enum Color {Red, Green, Blue}
let c: Color = Color.Green;
```

- Any menja bilo koji tip, ako nismo sigurni kog tipa će biti neka promenljiva koristimo any:

```
let joker: any = 43;
joker = "Let this be string instead.";
joker = true;
```

**48. TS default & rest parametri?**

```
function buildName( firstName: string, lastName = "X"): string {
    return firstName + " " + lastName;
}

buildName("Nikola") // Nikola X
buildName("Nikola", undefined) //Nikola X
buildName("Nikola", "Rnjak") //Nikola Rnjak
```

- Kao što se vidi iz primera, lastName je default parametar i ukoliko korisnik ne prosledi taj parametar pri pozivu funkcije automatski mu se dodeljuje vrednost "X"

```
function buildName(firstName: string, ...restOfName: string): string {
   return firstName + " " + restOfName.join(" ");
}

buildName("Nikola", "Z." , "Rnjak", "Junior") //Nikola Z. Rnjak Junior
buildName("Nikola", "Rnjak") //Nikola Rnjak
```

- Ukoliko ne znamo koliko parametara možemo da dobijemo koristimo ..., tako da naša funkcija sad prihvata bilo koji broj parametara nakon parametra firstName koji je obavezan.

**49. Deklaracija funkcija?**

- Kao i u JS funkcije mogu biti imenovane ili anonimne:

```
/* anonimna */
let myFunc = function(x: number, y:number): number {
    return x + y;
};

/* imenovana */
function add(x: number, y: number): number {
    return x + y;
}
```

- Mogu imati opcione, default i rest parametre, a svi ostali su obavezni parametri:

```
function add(x: number, y?: number, z = 0, ...restOfNums): number {
    //...
}
//x obavezan parametar, y opcioni, z default i restOfNums rest parametar
```

**50. Union & type?**

- TypeScript nam dozvoljava da koristimo više tipova za jednu promenljivu (union):

```
let code: (string | number);
code = 43;
code = "abc";
code = true //Kompajlerska greška, samo dozovljeni string i number
```

- Sa typeof možemo da saznamo kog je tipa neka promenljiva:

```
function displayType(code: (string | number)): void {
    if(typeof(code) === "number")
        console.log("Code is number.");
    else if(typeof(code) === "string")
        console.log("Code is string.");
}

displayType(43); //Code is number
displayType("abc"); //Code is string
displayType(true); //Kompajlerska greška
```

**51. Object res & spread?**

- Res objekat predstavlja HTTP odgovor koji Express aplikacija šalje nakon obrađenog HTTP zahteva.

```
albums.get('/', function(req, res, next) {
  // res.send() naš odgovor
});
```

- Spread operator (...) nam dozvoljava da neku kolekciju elemenata raspakujemo u pojedinačne elemente.

```
const arr = ["element1", "element2", "element3"];
const newArr = ["element0", ...arr];
console.log(newArr);
//Izlaz:
//['element0', 'element1', 'element2', 'element3']
```

**52. Class & interface?**

- Class je nacrt kako objekat treba da izgleda i kako da se ponaša. Kreiranjem instance ove klase, dobijamo objekat koji ima metode koje se mogu izvršiti i definisana svojstva.

```
class PizzaMaker {
```

```
  static create(event: { name: string; toppings: string[] }) {
    return { name: event.name, toppings: event.toppings };
  }
}

const pizza = PizzaMaker.create({
  name: 'Inferno',
  toppings: ['cheese', 'peppers'],
});
```

- Interface je virtuelna struktura i postoji samo unutar TypeScript konteksta.

```
interface Pizza {
  name: string;
  toppings: string[];
}

class PizzaMaker {
  static create(event: Pizza) {
    return { name: event.name, toppings: event.toppings };
  }
}
//ne možemo kreirati instancu interfejsa
```

**53. TS decorators?**

- Dekorator je posebna vrsta deklaracije koja se može priložiti deklaraciji klase, metodu, pristupniku, svojstvu ili parametru. Dekoratori koriste formu @expression, gde izraz (expression) mora da se proceni u funkciji koja će biti pozvana u toku izvršavanja sa informacijama o ukrašenoj deklaraciji.

**54. TS generyc types?**

- Primer jedne generičke funkcije:

```
function identity<Type>(arg: Type): Type {
  return arg;
}
```

- Umesto da koristimo any, ovime prihvatamo i dalje bilo koji tip, ali i čuvamo informaciju o tom tipu.

**55. TS promise?**

- Promise je objekat koji pri asinhronoj operaciji, umesto krajnje vrednosti, "daje obećanje" da će dostaviti tu vrednost u nekom trenutku u budućnosti.

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});
```

```
myPromise
  .then(callbackResolved, callbackRejected);
```

**56. TS async/await?**

- Async/await uvek vraćaju Promise. Async i await uvek idu u paru tj. await se jedino može pozvati unutar async funkcije.

```
async function sumFunction(...numbers) {
    var sum = 0;
    numbers.forEach(number => sum += number);
    return sum;
}

async function getSum() {
  let result = await sumFunction(numbers); // Blokirano na ovoj liniji
  useThatResult(result); // Neće se izvršiti sve dok se sumFunction() ne bude završila
}
```

**57. Šta je angular?**

- AngularJS je frontend framework koji je razvio Google. To je struktuiran framework napravljen na JavaScriptu i služi za pravljenje dinamičkih web aplikacija. Svaka angular aplikacija se sastoji od modula koji su osnovni gradivni blok svake angular aplikacije. Moduli se sastoje od komponenti. Angular ne pokriva backend deo.

**58. Kako se startuje angular aplikacija?**

```
ng new ime-projekta //kreira novi projekat

cd ime-projekta //uđemo u direktorijum projekta

ng serve //pokretanje projekta
```

**59. Struktura angular aplikacije?**

- Folder e2e - sadrži konfiguracione fajlove za testiranje aplikacije.
- Folder node_modules - sadrži preuzete pakete.
- Folder src sadrži izvor kod i ima 3 podfoldera:
  - app - fajlove angular projekta tj. komponente, sablone, poglede,...
  - assets - sadrži statičke fajlove kao što su npr. slike
  - environments - sadrži fajlove vezane za okruženje, postoji razvojno i produkciono okruženje
- Fajl tsconfig.json - sadrži TS konfiguraciju
- Fajl package.json - sadrži informacije o potrebnim paketima u projektu
- Fajl angular.json - sadrži konfiguracione opcije za kompajliranje i razmeštanje projekta

**60. Angular main.ts, index.html, index.css?**

- Fajl index.html - predstavlja glavnu html stranu koja se učitava pri poseti sajta, a index.css je css deo te strane.
- Fajl main.ts - predstavlja glavnu ulaznu tačku aplikacije, tu se specificira koreni modul aplikacije koji se pokreće pri pokretanju aplikacije.

**61. Angular assets?**

- Folder assets nalazi se unutar src foldera i sadrži statičke fajlove kao što su slike, json fajlovi i slično.

**62. Angular app.module.ts?**

- Glavni (koreni) modul aplikacije, prilikom kreiranja nove angular aplikacije automatski se generiše. Da bi neka klasa bila modul mora da ima @NgModule anotaciju. Prilikom izrade aplikacije, učitavamo različite eksterne module u postojeći modul. Dekoratoru @NgModule se prosleđuje objekat koji sadrži sledeća svojstva:
  - declarations - niz specificira koje komponente i direktive pripadaju modulu
  - imports - niz služi da se importuju funkcionalnosti drugih modula
  - providers - niz služi da se specificiraju servisi koji obezbeđuju podatke
  - bootstrap - niz, govori angularu koje komponente da kreira i ubaci u DOM browsera

**63. Angular components?**

- Komponenta kontroliše deo ekrana koji se naziva pogled. Svaka komponenta obezbeđuje deo funkcionalnosti za aplikaciju. Sastoji se od:
  - Klase
  - Metapodataka koji opisuju klasu i proširuju njenu funkcionalnost
  - Šablona koji se koristi za definisanje html pogleda

**64. Angular interpolacija?**

- Jedan od načina povezivanja podataka u Angularu.

```
/* klasa komponente */
export class AppComponent {
    naslov = 'Data binding';
    osoba = {ime: 'Marko', prezime: 'Markovic', grad: 'Beograd'};

    citajIme(): string {
    return this.osoba.ime + ' ' + this.osoba.prezime;
    }
}

/* pogled */
<b>Vezivanje za svojstvo</b>
<h2>{{naslov}}</h2>

<p>
```

```
    Ime: {{osoba.ime}}
    </p>
    <p>
    Prezime: {{osoba.prezime}}
    </p>

    <b>Vezivanje za metodu</b>
    <p>{{citajIme()}}</p>
```

**65. Angular pipes?**

- Cevi (pipe) koristimo da transformišemo nizove, iznose valute, datume i druge podatke za prikaz. Cevi su jednostavne funkcije koje se koriste u template izrazima za prihvatanje ulazne vrednosti i vraćanje transformisane vrednosti.

```
    {{ datum | date }}
    {{ iznos | currency:'EUR' }}
```

**66. Angular property binding?**

- Jednosmerna komunikacija između komponente i pogleda, gde se podaci prosleđuju od komponente do pogleda komponente. Pristupa se svojstu html elementa i postavlja mu se vrednost na osnovu nekog svojstva klase komponente. Koriste se srednje zagrade [] da se specificira svojstvo html elementa, svojstvo klase komponente se navodi unutar navodnika.

```
    export class AppComponent {
        imageUrl = '/assets/angular.jpg';
        isDisabled = true;
    }

    <img [src]="imageUrl">
    <button [disabled]="isDisabled"> Button name </button>
```

**67. Angular attribute, style & class binding?**

- Attribute - [attr.atribute-name]:
```
    <button [attr.aria-label]> {{actionName}} with Aria </button>
```

- Style - [style.style-property-name]:
```
    <nav [style.background-color]="expression"></nav>
```

- Class:
```
    [class.sale]="onSale"
    [class]="classExpression" //kada navodimo više klasa
```

**68. Angular event binding?**

- Podaci se prosleđuju od pogleda komponente ka komponenti. Izvršava akciju u komponenti kada korisnik izvršava akciju u korisničkom interfejsu pogleda npr. klikne dugme u pogledu. Ime događaja koji treba da se desi za izvršenje akcije se navodi unutar malih zagrada npr. (click), a zatim mu se dodeljuje funkcija iz

komponente.

```
export class AppComponent {
    brojac = 0;

    uvecaj(): void{
        this.brojac++;
        console.log('Vrednost brojaca: ', this.brojac);
    }
}

<button (click)="uvecaj()"> Uvecaj brojac </button>
```

**69. ngClass & ngStyle?**

- ngClass - dodaje i uklanja CSS klase na HTML elementu.

```
<some-element-tag [ngClass]="'firstClass
secondClass'">...</some-element-tag>
```

- ngStyle - Direktiva atributa koja ažurira stilove za HTML element. Postavlja jedno ili više svojstava stila, navedenih kao par ključ-vrednost razdvojenih sa dve tacke.

```
<some-element-tag [ngStyle]="{'font-style':
styleExp}">...</some-element-tag>
```

**70. ngModel (direktiva)?**

- Kreira FormControl instancu iz modela domena i povezuje je sa kontrolnim elementom obrasca.

```
<input [(ngModel)]="name" #ctrl="ngModel" required>
```

**71. ngIf, ngFor & ngSwitch?**

- Direktive ngIf i ngFor počinju sa '*' pa naziv direktive tj. *ngIf i *ngFor. U okviru istog elementa ne možemo da imamo dve direktive koje počinju sa '*'.

- ngIf - koristi se kada hoćemo da prikažemo ili sklonimo neki element u zavisnosti od nekog uslova.

```
<div *ngIf="condition">
    //Sadržaj za prikazivanje kada je uslov istinit.
</div>
```

- ngSwitch - kada imamo više opcija koje su moguće, ne samo true i false, onda koristimo ngSwitch kako bi ispitali koja opcija je trenutno aktivna i kako bi prikazali odgovarajuci sadržaj.

```
<ul *ngFor="let person of people"
    [ngSwitch]="person.country">
```

```
      <li *ngSwitchCase="'UK'"
          class="text-success">
        {{ person.name }} ({{ person.country }})
      </li>

      <li *ngSwitchCase="'USA'"
          class="text-primary">
        {{ person.name }} ({{ person.country }})
      </li>

    </ul>
```

- ngFor - koristimo za pravljenje tabela i listi za prikaz podataka.

```
    <tr *ngFor="let hero of heroes">
        <td>{{hero.name}}</td>
    </tr>
```

**72. Angular Input/Output?**

- Uobičajeni obrazac u Angular-u je deljenje podataka između roditeljske komponente i jedne ili više podređenih komponenti. Ovaj obrazac se implementira pomoću dekoratora @Input() i @Output(). @Input() omogućava roditeljskoj komponenti da ažurira podatke u podređenoj komponenti. Nasuprot tome, @Output() dozvoljava detetu da pošalje podatke roditeljskoj komponenti.

```
    <app-input-output [item]="currentItem"
(deleteRequest)="crossOffItem($event)"></app-input-output>
```

```
    => selektor podređene klase
    => item - @Input() svojstvo iz podređene klase
    => currentItem - svojstvo iz roditeljske koje se prosleđuje
    => deleteRequest - @Output događaj ka roditeljskoj klasi
    => crossOffItem($event) - metoda iz roditeljske klase
```

**73. Angular dependency injection / service?**

- Mehanizam koji nam omogućava da povezujemo delove aplikacije koji su odvojeni npr. controller sa service-om.

```
    /* hero.service.ts */
    import { Injectable } from '@angular/core';

    @Injectable({ //ovime specificiramo da angular može da koristi ovu klasu za
DI sistem
      providedIn: 'root',
    })
    export class HeroService {
      getHeroes() { return HEROES; }
    }

    /* hero-list.component.ts */
```

```
    //...

    constructor(heroService: HeroService) //dodavanjem ovog servisa u konstruktor
on je sada vidljiv toj komponenti

    //...

    this.heroService.getHeroes(); //ovako koristimo metode hero servisa
```

**74. Angular rutiranje?**

- Router je poseban Angular modul koji omogućava navigaciju od jedne do druge komponente kada korisnik klikne na link, dugme ili dugmad za navigaciju u browser-u.

- Ruta govori Angular-u koji pogled da prikaze kada korisnik klikne na link ili unese adresu u adresnoj liniji. Ruta se sastoji od putanje i komponente u koju je ta putanja mapirana.

```
    { path: 'home', component: HomeComponent}
```

- Direktiva <router-outlet> služi da specificira mesto gde će ruter da prikaže pogled.
- Direktiva routerLink povezuje html element sa rutom:

```
    <a [routerLink]="['home']"> Home </a>
```

**75. Ng build?**

- Sa ng build <ime-projekta> [options] dobija se prevedena verzija projekta, tako što se generišu statički fajlovi *.html, *css i *.js u folderu dist(ribution). Ovi fajlovi se mogu servirati bilo kojim web serverom.

**76. Angular HttpClient/Observable/subscribe - slanje zahteva serveru/prijem podataka?**

```
    /* login.service.ts */
    export class LoginService {
        constructor(protected http: HttpClient) {}

        login(email: string, password: string): Observable<string> {
            return this.http.post<any>("http://localhost:5000/login", {"email":
email, "pass": password});
        }
    }

    /* login.component.ts */
    export class LoginComponent {
        email: string;
        password: string;

        constructor(protected loginService: LoginService){}
```

```
        login(): void {
            this.loginService.login(email, password).subscribe( result => {
                console.log(result);
            });
        }
    }
```

**77. MongoDB struktura baze?**

- Sastoji se od jedne ili više kolekcija, kolekcije se sastoje od dokumenata, a dokument je osnovna jedinica podataka MongoDB baze. Struktura MongoDB dokumenta je kao JSON, koji se čuva u binarnom JSON formatu radi brzine i efikasnosti rada sa podacima.

**78. Kreiranje MongoDB baze - nodeJS?**

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

**79. MongoDB upiti - nodeJS?**

```
const MongoClient = require('mongodb').MongoClient;
const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  const dbo = db.db("mydb");
  const query = {address: "Park Lane 38"};

dbo.collection("customers").find(query).toArray(function(err, result) {
        if(err) throw err;
        console.log(result);
        db.close();
      });
});
```

**80. Kako se kombinuju MongoDB i Express u nodeJS?**

```
/* config.js */
config = {
    port: 5000,
    dbConnection: "mongodb://localhost//:27017"
}

module.exports = config

/* index.js */
```

```
var express = require('express');
var app = express();
var config = require("./config")
var mongoose = require('mongoose');

mongoose.connect(config.dbConnection);
app.listen(config.port, () => {
    console.log(`Running on port: ${config.port}`);
});
```