Spisak pitanja iz CS100

1. Šta predstavlja imperativno programiranje?
    - Imperativno programiranje je programska paradigma koja opisuje računanje kao izraze koji menjaju stanje programa. Kod imperativnog programiranja, program čini niz instrukcija računara.
    - Osnovni alat koji omogućava ovaj način programiranja je uslovni skok. Imperativno programiranje često koristi dijagram toka (en. flow diagram), često nazvan algoritmom.
    - Imperativno programiranje opisuje računarski proces kao stanja programa i naredbi koje menjaju stanje. Stanje računarskog sistema je definisano sadržajem memorije i naredbama mašinskog jezika.

2. Šta predstavlja proceduralno programiranje?
    - Proceduralno programiranje je lista ili skup instrukcija koje definišu računaru šta treba uraditi metodom korak po korak i kako se to obavlja od prvog koda na drugom koda.
    - Procedure se sastoje od sekvence računarskih koraka koji se mogu izvršavati pozivom u bilo kojoj tački izvršavanja celog programa, uključujući slučaj izvršavanja i u drugim procedurama. Sam program time postaje niz poziva procedura, i koda koji povezuje te pozive.
    - Jezici koji podržavaju proceduralnu paradigmu su C/C++, Java, Fortran, Pascal, BASIC, i mnogi drugi.

3. Šta predstavlja deklarativno programiranje?
    - Deklarativno programiranje (en. declarative programming) je paradigma programiranja, stil izgradnje struktura i elemenata računarskih programa, koji izražavaju logiku računanja bez opisivanja njenog kontrolnog toka.
    - Mnogi jezici primenjuju ovu paradigmu tako što opisuju šta program treba da postigne (u smislu domena problema), umesto da opisuje kako ostvaruje niz koraka ka rešavanju problema.
    - Ovo je u suprotnosti sa imperativnim programiranjem, u kojoj se algoritmi sprovode u smislu eksplicitnih koraka.
    - Deklarativno programiranje može u velikoj meri pojednostaviti pisanje paralelnih programa.
    - Deklarativni jezici često uključuju jezika upita sistema kao što su SQL, XQuery, ali i Prolog, LISP, Miranda, Haskell.

4. Šta je sistemski, a šta aplikativni softver?
    - Programi koji kontrolišu i upravljaju osnovne operacije računara se u opštem slučaju nazivaju sistemskim softverom.

- Programi pomoću kojih računar rešava razne svakodnevne zadatke spadaju u aplikacioni softver.

5. Šta predstavljaju šeme kodovanja realnih brojeva?
   - Realni brojevi se koduju notacijom pokretnog zareza ili pokretne tačke (en. floating point notation), kao što je standard IEEE 754.

6. Kako se alfanumerički karakteri mogu predstaviti u memoriji računara?
   - Kada se slovo skjladišti u memoriji, najpre se konvertuje u numerički kod, pa onda u memoriju računara, kao binarni broj.

7. Koje operacije mogu direktno izvršiti elektronska kola u računaru?
   - Elektronska kola računara mogu prepoznati i direktno izvršiti samo ograničen skup jednostavnih instrukcija, pa svaki program, da bi bio izvršen, treba prevesti u takav skup instrukcija. Te osnovne instrukcije retko su složenije od sledećih:
     a) Saberi dva broja.
     b) Proveri da li je zadati broj nula.
     c) Kopiraj skup podataka iz jednog dela memorije računara u drugi.

8. Opisati proces prevođenja programa na mašinski jezik koristeći kompajler.
   - **L1** – Jezik koji razumeju ljudi (Python, C#, C++, Ruby, JavaScript, Asembly…)
   - **L0** – Masinski jezik koji razume kompjuter
   - Jedan način izvršavanja programa pisanih na jeziku L1 jeste da se najpre svaka njegova instrukcija zameni ekvivalentnim nizom instrukcija na jeziku L0. Tako će se rezultujući program sastojati isključivo od instrukcija na jeziku L0.
   - Računar tada izvršava nov program na jeziku L0 umesto starog programa na jeziku L1. Ova tehnika se naziva prevođenje (en. translation), a program koji vrši prevođenje se naziva prevodilac (en. translator), ili češće, ako je u pitanju prevođenje na mašinski jezik, kompajler (en. compiler).

9. Opisati proces prevođenja programa na mašinski jezik koristeći interpreter.
   - **L1** – Jezik koji razumeju ljudi (Python, C#, C++, Ruby, JavaScript, Asembly…)
   - **L0** – Masinski jezik koji razume kompjuter
   - Drugim načinom se program pisan na jeziku L1 učitava instrukciju po instrukciju, svaka se instrukcija redom ispituje i odmah izvršava njoj ekvivalentan skup instrukcija na jeziku L0. Na taj način se izbegava prethodno generisanje čitavog novog programa na jeziku L0. Tehnika

se zove interpretiranje (en. interpretation), a program koji vrši interpretaciju se naziva interpreter (en. interpreter).

- Prevođenje i interpretiranje su slične tehnike. U oba slučaja računar izvršava instrukcije na jeziku L1 tako što izvršava ekvivalentne nizove instrukcija na jeziku L0.
- Pri prevođenju ceo program je pisan na jeziku L1 I on se najpre prevede u program na jeziku L0. Yatim se odbaci prvi program (L1), a novi program (L0) učita se u memoriju računara i izvrši. Tokom izvršavanja, aktivan je samo novi program (L0) i on upravlja resursima računara.
- Pri interpretaciji, svaka instrukcija na jeziku L1 analizira se i prevodi, a zatim odmah izvršava. Ne generiše se nov program. Ovde interpreter upravlja računarom. Program napisan na jeziku L1 za njega predstavlja samo ulazne podatke.

10. Opisati razlike između programskih jezika niskog nivoa i programskih jezika visokog nivoa.
    - Iako pomoću asemblerskih jezika više nije neophodno pisati instrukcije na mašinskom jeziku, i dalje je pisanje programa na ovim jezicima relativno teško. Asemblerski jezici predstavljaju direktnu pod-rutinu (en. subroutine) za mašinske jezike, i zahtevaju znanje konkretnog CPU-u koji će izvršavati program. Takođe, ovi jezici zahtevaju pisanje velikog broja instrukcija čak i za najjednostavnije programe. Upravo iz razloga bliskosti asemblerskih jezika sa mašinskim jezikom, često se ovi jezici nazivaju programski jezici niskog nivoa (en. low-level programming languages). Sredinom prošlog veka, nova generacija programskih jezika se pojavila, nazvana jezicima visokog nivoa (en. high-level programming languages). Pomoću jezika visokog nivoa mogu se napisati kompleksni programi bez potrebe znanja kako radi CPU i bez pisanja velikog broja instrukcija niskog nivoa. Takođe, veliki broj ovakvih programskih jezika jesu laki za razumevanje i učenje.

11. Opisati korake u ciklusu razvoja programa.
    - Proces kreiranja programa koji radi korektno obično zahteva pet faza koje su prikazane na Slici 1 i čine ciklus razvoja programa.

    1) **Dizajniranje programa**: Kada programeri započnu novi projekat, obično pisanje koda nije prvi korak, već najpre počinju kreiranjem dizajna programa.
    2) **Pisanje koda**: Nakon dizajniranja programa, programer počinje da piše kod na jeziku visokog nivoa kao što je Python. Svaki jezik ima svoja pravila, poznata kao sintaksa, koja se moraju poštovati prilikom pisanja programa. Pravila sintakse jezika diktiraju stvari kao što su način na koji se mogu koristiti ključne reči, operatori i znakovi interpunkcije. Sintaksička greška se javlja ako programer prekrši bilo koje od ovih pravila.

3) **Ispravljanje sintaksnih grešaka**: Ako program sadrži sintaksičku grešku, ili čak jednostavnu grešku, kao što je pogrešno napisana ključna reč, kompajler ili interpreter će prikazati poruku koja pokazuje u čemu je greška.
4) **Testiranje programa**: Kada je kod u izvršnom obliku, onda se testira da bi se utvrdilo da li postoje logičke greške. Logička greška je greška koja ne sprečava da program bude pokrenut, ali dovodi do toga da proizvodi netačne rezultate. Matematičke greške su često uzrok logičkih grešaka.
5) **Ispravljanje logičkih grešaka**: Ako program daje netačne rezultate, programer otklanja greške u kodu. Ponekad tokom ovog procesa, desi se da programer otkrije da originalni dizajn programa mora biti promenjen. U tom slučaju počinje ciklus razvoja programa ponovo i nastavlja sve dok se nijedna greška ne može pronaći.

12. Opisati proces dizajniranja programa i navesti korake.
- Proces dizajniranja programa je verovatno najvažniji deo ciklusa razvoja programa. Dizajn programa je suštinski njegova osnova. Ukoliko je program loše dizajniran, na kraju će biti dosta posla oko popravljanja programa. Proces dizajniranja programa može se sažeti u sledeća dva koraka:
    a) Razumeti zadatak koji program treba da izvrši.
    b) Odrediti korake koji se moraju preduzeti da bi se zadatak izvršio.

13. Objasniti potrebu za pseudokodom i dijagramom toka.
- Pseudokodu je neformalni jezik koja nema pravila sintakse i nije predviđen za prevođenje ili izvršavanje. Programeri koriste pseudokod za kreiranje modela ili „maketu“ programa, jer ne moraju da brinu o sintaksičkim greškama dok ga pišu. Samim tim, mogu da se fokusiraju u potpunosti na dizajn programa. Jednom kada je napravljen zadovoljavajući dizajn sa pseudokodom, pseudokod se može prevesti direktno u stvarni kod.
- Dijagram toka je još jedan alat koji programeri koriste za dizajniranje programa. Dijagram toka je dijagram koji grafički prikazuje korake koji se odvijaju u programu. Postoje tri tipa simbola u dijagramu toka: ovali, paralelogrami i pravougaonici. Svaki od ovih simbola predstavlja korak u programu:
    a) Ovali, koji se pojavljuju na vrhu i na dnu dijagrama toka, nazivaju se terminalni simboli. Terminalni simbol Početak označava početnu tačku programa, a terminalni simbol Kraj označava završnu tačku programa.
    b) Paralelogrami se koriste kao ulazni i izlazni simboli. Oni predstavljaju korake po kojima program čita ulaz ili prikazuje izlaz.

c) Pravougaonici se koriste kao simboli za obradu. Oni predstavljaju korake u kojima program vrši neki proces na podacima, kao što je matematički proračun.

- Simboli su povezani strelicama koje predstavljaju "tok" programa. Kako bi se išlo kroz simbole u odgovarajućem redosledu, počinje se na Početak terminalu i prate se strelice dok se ne dođe do terminala Kraj.

14. Šta predstavlja promenljiva i kako se promenljivoj dodeljuje vrednost?
    - Kako bi programi imali mogućnost da čuvaju podatke u memoriji, oni koriste promenljive. Promenljiva je ime koje predstavlja vrednost u memoriji računara. Kada promenljiva predstavlja vrednost u memoriji računara, kažemo da promenljiva upućuje na vrednost. Da bismo kreirali promenljivu koristimo izraz za dodeljivanje vrednosti, koji glasi:

      `ime_promenljive = “vrednost_promenljive”`,

    - i na taj način kreirana promenljiva upućuje na vrednost koju smo joj dodelili. Nakon kreiranja promenljive godine, ona upućuje na vrednost koju smo joj dodelili.
15. Šta predstavlja tip podataka?
    - Pošto se različiti tipovi brojeva čuvaju i manipulišu na različite načine, u programiranju se koriste različiti tipove podataka za kategorizaciju vrednosti u memoriji. Kada se ceo broj čuva u memoriji, on je klasifikovan kao int, a kada je realan broj uskladišten u memoriji, on se klasifikuje kao float. Broj koji je upisan u programski kod naziva se numerički literal. Kada Python interpreter čita numerički literal u programskom kodu, on određuje njegov tip podataka prema sledećim pravilima:
      a) Numerički literal koji je napisan kao ceo broj bez decimalne tačke smatra se da je int. Primeri: 7, 124 i −9.
      b) Numerički literal koji je napisan sa decimalnim zarezom smatra se da je float. Primeri: 1,5, 3,14159 i 5,0.
    - Pored int i float tipova podataka, Python takođe ima tip podataka pod nazivom str, koji se koristi za čuvanje stringova u memoriji.

16. Šta predstavlja, i čemu služi komentar u programu?
    - Komentari su kratke beleške postavljene u različite delove programa, koje objašnjavaju kako ti delovi programa rade. Iako su komentari neizostavan i veoma važan deo svakog programa, oni su ignorisani od strane Python interpretera. Komentari su namenjeni bilo kojoj osobi koja čita programski kod, a ne računaru.
    - U Python-u komentar počinje znakom `#`. Kada Python interpreter vidi # znak, ignoriše sve od tog znaka do kraja reda. Komentari se obično

pišu na kraju linije u kodu i na taj način komentarišu (objašnjavaju) izraz u toj liniji koda.

17. Opisati funkciju **input()**.
    - Većina programa koje pišemo moraće da pročita neki ulaz, a zatim da izvrše neku operaciju nad tim ulazom. Kada program pročita podatke sa tastature, obično te podatke čuva u promenljivoj kako bi ih kasnije mogao koristiti. Za unošenje ulaza najčešće se koristi ugrađena funkcija input koja služi za čitanje ulaza sa tastature. Funkcija input čita podatke koji su uneti putem tastature i vraća ih u vidu stringa nazad programu. Obično se funkcija input poziva u formi:

      ```
      ime_promenljive = input("Tekst koji se ispisuje")
      ```

    - Kad se pokrene naredba input, prvo se ispisuje tekst koji smo napisali u zagradama, nakon toga program staje i ceka nas unos sa tastature. Kad unesemo zeljeni tekst pretisnemo enter i taj tekst koji smo uneli se cuva u promenljivoj kojoj dodeljujemo vrednost. Posto ugradjena funkcija input sve sto unesemo cuva kao string, ako zelimo da sacuva kao broj moramo da koristimo int() ili float().

18. Opisati funkciju **print()**.
    - Print jedna od osnovnih funkcija u pythonu. Ona se uglavnom koriti u sledecoj formi:

      ```
      print("tekst koji se ispisuje")
      ```

    - gde se pozivanjem funkcije print ispisuju vrednosti koje su prosledjene funkciji. Ako se prosledi vise vrednosti print funkciji ona ce ih automacki razdvojiti sa razmakom. Ovo mozemo promeniti koristeci argument sep koji prosledjujemo print funkciji u sledecoj formi:

      ```
      print("tekst koji se ispisuje", sep=" ")
      ```

    - i u stringu koji dodeljujemo sep argumentu ce razdvajati sve vrednosti koje prosledimo. Takodje postoji i end argument koji se koristi za zavrsavanje svakog ispisa, koji je bez promene u print funkciji novi red.

19. Navesti i opisati osnovnih sedam matematičkih operatora u Python programskom jeziku.
    - U pythonu postoje sledeci operatori:
        a) Sabiranje **(+)**: koristi se za sabiranje brojeva ili spajati dve vrednosti
        b) Oduzimanje **(-)**: koristi se za oduzimanje dva broja
        c) Mnozenje **(*)**: koristi se u mnozenju brojeva i stringova

d) Deljenje **(/)**: koristi se za deljenje dva broja
e) Celobrojno deljenje **(//)**: deljenje dva broja u kome je rezultat uvek celobrojan
f) Ostatak pri deljenju **(%)**: ostatak pri deljenju dva broja
g) Stepenovanje/Eksponent **(**)**: koristi se za dizanje broja na odredjeni stepen

20. Šta je sekvencijalna, a šta struktura odlučivanja programa? Kako se struktura odlučivanja opisuje kroz dijatram toka, a kako u programskom jeziku Python?
    - Sekvencijalna struktura je skup naredbi koje se izvršavaju redosledom kojim se pojavljuju, sto bi znacilo da redom kojim smo napisali sve naredbe da se tim redom izvrsavaju. Iako se sekvencijalna struktura u velikoj meri koristi u programiranju, ona ne može da reši sve vrste zadataka. To je zato što se neki problemi jednostavno ne mogu rešiti izvršavanjem koraka redosledom kojim se pojavljuju. Na primer, recimo da pravimo program za obračun plata koji utvrđuje da li je zaposleni radio prekovremeno. Ako je zaposleni radio više od 40 sati, on ili ona se dodatno plaćaju za sve sate preko 40. U suprotnom, obračun prekovremenog rada treba preskočiti. Ovakva vrsta programa zahteva drugačiju vrstu kontrolne strukture, odnosno strukturu koja će da izvršava niz naredbi samo pod određenim uslovima. Ovo se postiže takozvanom strukturom odlučivanja. Kod strukture odlučivanja u najjednostavnijem obliku, određena radnja se izvodi samo ako postoji određeni uslov. Ako uslov ne postoji, radnja se ne izvodi. Simbol dijamanta predstavlja uslov tačno/ netačno. Ako je uslov tačno, idemo jednim putem, koji vodi do izvršenja radnje. Ako je uslov netačno, sledimo drugi put, koji preskače radnju. U programskom jeziku se predstavlja preko if klauze. **(if klauza je opisana u zadatku 21)**

21. Opisati i dati primer za if klauzulu u programskom jeziku Python.
    - If klauza se koristi kad imamo uslov koji treba da se ispuni kako bi neki blok koda bio izvrsen. Najjednostavniji primer je uslov da li je neki broj manji od drugog broja, ako jeste blok koda u if klauzi se izvrsava, ali ako taj uslov nije ispunjen onda se taj blok koda preskace. Sintaksa if petlje je if uslov: novi red i onda blok koda pod if klauzom pisemo uvuceno sa jos jednim jednim tabom.

22. Šta predstavljaju Bulovi izrazi? Opisati šest relacionih operatora u programskom jeziku Python.
    - Kao što je ranije pomenuto, if naredba testira izraz kako bi utvrdila da li je on tačan ili netačan. Izrazi koji se testiraju if naredbom nazivaju se Bulovi izrazi, nazvani u čast engleskog matematičara Džordža Bula. U 1800-im, Bul je izmislio sistem matematike u kojem se apstraktni pojmovi tačnog i netačnog mogu koristiti u proračunima. Tipično,

Bulov izraz koji se testira if naredbom formira se pomoću relacionog operatora. Relacioni operator određuje da li postoji određena veza između dve vrednosti i ti oparati su vece **(>)**, manje **(<)**, vece jednako **(>=)**, manje jednako **(<=)**, jednako **(==)** i razlicito **(!=)**.

23. Opisati strukturu odlučivanja sa jednom alternativom kroz dijagram toka, i u naredbama programskog jezika Python. **(L04)**
    - Opisati strukturu odlučivanja sa jednom alternativom kroz dijagram toka, i u naredbama programskog jezika Python. Ova struktura pruza samo jedan alternativni put izvrsenja,Ako je uslov u simbolu dijamanta tacan,idemo altrenativnim putem.U suprotnom izlazimo iz strukture.

24. Opisati strukturu odlučivanja sa dve alternative kroz dijagram toka, i u naredbama programskoj jezika Python.
    - Struktura odlucivanja sa dve alternative znaci da postoje dva moguca puta izvrsenja. Jedan put se uzima ako je uslov tacan a drugi put se uzima ako je uslov netacan. U kodu strukturu odlucivanja pisemo kao if-else naredbu. Ako je uslov tacan,izvrsava se blok uvucenih naredbi koji slede kluzulu if,a zatim kontrola programa skace na naredbu koja sledi odmah nakon if-else naredbe. Ako je uslov netacan,blok uvucenih naredbi koje slede iza klauzule else se izvrsavaju a zatim kontrola programa skace na naredbe nakon if-else klauzule.

25. Šta predstavlja ugnježdena struktura odlučivanja i na koja dva načina se može realizovati u Python programskom jeziku?
    - Vecina programa je dizajnirana tako da se sekvencijalna struktura i struktura odlucivanja kombinuju. Sluze za testiranje vise uslova. Ugnjezdena struktura odlucivanja predstavlja ili strukturu odlucivanja unutra sekvencijalne strukture ili obrnuto.

26. Opisati logički operator not, dati tablicu istinitosti i napisati kratak primer.
    - **NOT operator** je unarni operator, što znači da radi sa samo jednim operandom. Operand mora biti Bulov izraz. Operator not preokreće istinosnu vrednost njegovog operanda. Ako se primeni na izraz koji je tačan, operator vraća netačno. Ako se not operator primeni na izraz koji je netačan, operator vraća tačno.

27. Opisati logički operator and, dati tablicu istinitosti i napisati kratak primer.
    - **AND operator** povezuje dva Bulova izraza u jedan složeni izraz. Oba podizraza moraju biti tačna da bi složeni izraz bio istinit.

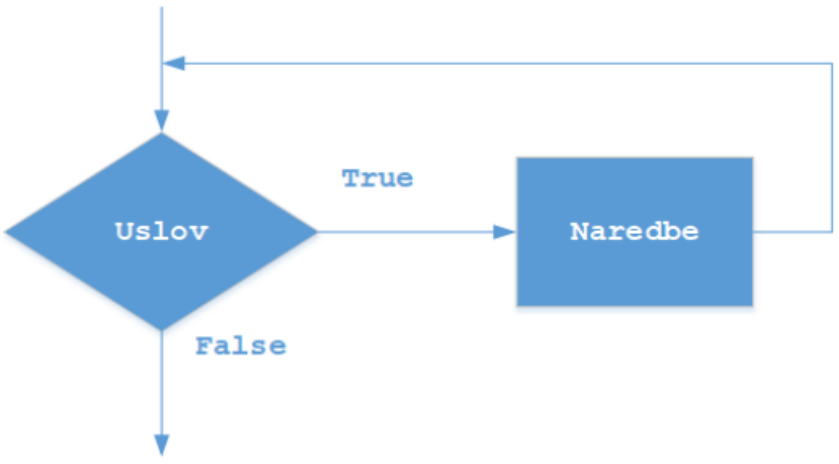
28. Opisati logički operator or, dati tablicu istinitosti i napisati kratak primer.
    - **OR operator** povezuje dva Bulova izraza u jedan složeni izraz. Jedan ili oba podizraza moraju biti tačna da bi složeni izraz bio istinit. Potrebno je samo da jedan od podizraza bude tačan (nije bitno koji) kako bi složeni izraz bio tačan.

29. Opisati tip podataka bool.
    - Pored ostalih tipova podataka, Python takođe ima bool tip podataka. Ovaj tip podataka dozvoljava da kreiramo tzv. Bulove promenljive, koje mogu upućivati na jednu od dve moguće vrednosti: tačno ili netačno.

30. Šta predstavlja petlja? Opisati opštu strukturu petlje.
    - Petlja (en. loop) predstavlja jedan od najosnovnijih koncepata programiranja.
    - Može se reći da petlja postavlja pitanje. Ukoliko odgovor na to pitanje zahteva neku radnju, ta radnja se izvršava. Isto pitanje se ponovo pita sve dok nije potrebno ništa više uraditi. Svaki put kada se postavlja pitanje čija je posledica izvršenje koda, dešava se jedna iteracija (en. iteration).
    - Najveća prednost korišćenja petlji jeste da programer ne mora više puta koristiti isti kod. Svi programski jezici sadrže neki vid petlji.

31. Koje su dve kategorije po kojima delimo petlje? Dati primer petlje u Python jeziku za svaku kategoriju.
    - Petlje se mogu podeliti u dve kategorije: **uslovno kontrolisane petlje** (en. condition controlled loops, primer: **while** petlja) i **brojčano kontrolisane petlje** (en. count controlled loops, primer: **for** petlja).

32. Opisati uslovno-kontrolisanu petlju i nacrtati dijagram toka koji sadrži uslovno kontrolisanu petlju. Dati primer uslovno-kontrolisane petlje u Python jeziku.
    - Uslovno kontrolisana petlja uzrokuje da se naredba ili skup naredbi ponavlja sve dok je ispunjen određeni (unapred dati) uslov. U programskom jeziku Python, kao uslovno kontrolisana petlja koristi se while petlja.

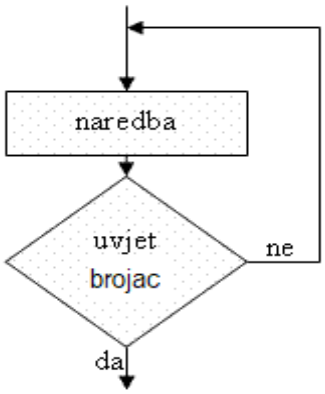


- While petlja je dobila ime po rečju while (sr: dok), jer na taj način i radi: sve dok je ispunjen neki uslov, treba odradti zadatak. Sama petlja sastoji se dva dela:
    a) Uslov koji se testira za True ili False vrednosti,
    b) Naredba ili skup naredbi koji se ponavlja dokle god je uslov ispunjen.

33. Opisati while klauzulu u Python programskom jeziku.
    - Zbog jednostavnosti, prva linija while petlje nazvana je while klauzula. While klauzula počinje sa rečju while, nakon koje sledi condition, koji predstavlja uslov koji će vratiti True ili False (tj. Boolean uslov). Nakon condition-a, sledi dvotačka. U narednim linijama koda (u bloku koda) ispisuju se naredbe unutar petlje.
    - Kada Python interpreter izvršava while petlju, najpre se testira condition. Ukoliko condition vrati True, onda se izvršavaju naredbe u bloku koda koje su ispod while klauzule. Ukoliko condition vrati False, program izlazi iz petlje.
    - While petlja ima sledecu formu:

```
while uslov:
    komanda
    komanda
    ...
```

34. Opisati brojčano-kontrolisanu petlju i nacrtati dijagram toka koji sadrži brojčano kontrolisanu petlju. Dati primer brojčano-kontrolisane petlje u Python jeziku.
    - Brojčano kontrolisana petlja (en. count-controlled loop) predstavlja petlju koja prolazi kroz iteracije tačno određen broj puta. Ovakve petlje se često koriste pri izrazi programa.



    - U Python programskom jeziku (kao i u mnogim drugim programskim jezicima) for petlja se koristi za slucajeve kad nam treba brojcano kontrolisanu petlju. Za razliku od drugih programskih jezika, for

naredba koristi se u tzv. iterabilnim tipovima podataka (en. iterable data types), odnosno, sa tipovima podataka koji su u sekvenci.

35. Opisati for klauzulu u Python programskom jeziku.
    - Za pisanje brojčano kontrolisane petlje u Python programskom jeziku (kao i u mnogim drugim programskim jezicima) koristi se for petlja. Za razliku od drugih programskih jezika, for naredba koristi se u tzv. iterabilnim tipovima podataka (en. iterable data types), odnosno, sa tipovima podataka koji su u sekvenci.

36. Šta predstavlja beskonačna petlja? Da li i kada su one poželjne? Kako možemo izbeći beskonačnu petlju?
    - Ukoliko petlja nema način da se u testiranju while klauzule dobije vrednost False, onda se dobija beskonačna petlja (en. infinite loop). Ovakva petlja nastavlja sa izvršenjem tela petlje sve dok se program ne prekine (ručno, tj. od strane korisnika, ili od strane operativnog sistema).
    - Beskonacne petlje uglavnom nisu pozeljne ali jedan od primera gde one mogu biti izuzetno korisne je u GUI Console menijima gde korisnik moze odbrati da izvrsi neku radnju ponudjenu u meniju beskonacno mnogo puta.
    - Kako bi izbegli beskonacnu while petlju neophodno je iskoristiti break ili neku drugu metodu za prekidanje u telu while petlje.

37. Šta predstavlja konceprt validacije ulaznih podataka?
    - Računari ne mogu razlikovati dobre ulazne podatke od loših učaznih podataka. Ukoliko korisnik prosledi programu loše ulazne podatke, onda će program obratiti te loše podatke, i kao rezultat, dati loše izlazne podatke. Primer, ukoliko se u program za racunanje mesecne plate greškom unese broj 400 umesto 40 nedeljnih radnih sati, on ce ispisati vrednost koja je 10 puta veca nego ona prava vrednost, što i će prijatno iznenaditi radnika, ali program nije „prepoznao“ loše ulazne podatke.

38. Šta predstavlja ugnježdena petlja? Dati primer ugnježdene while petlje unutar for petlje.
    - Ugnježdena petlja (en. nested loop) prestavlja petlju unutar druge petlje. Najbolji primer predstavlja časovnik. Kazaljka za sat napravi jedan pun krug za svakih 12 krugova minutare, a sekundara napravi 60 krugova za vreme potrebno minutari da napravi jedan krug. Ovo znači da za svaki krug kazaljke za sat (12 sati) potrebno je da sekundara obiće ukupno 720 krugova. Da bi simulirali rad sata, možemo napraviti petlju za sekunde (od 0 do 59) unutar petlje za minute (od 0 do 59) unutar petlje za sate (od 0 do 23), sto bi izgledalo ovako:

```
for sat in range(23):
    for minut in range(59):
        for sekunda in range(59):
            naredba
            naredba
            ...
```

- Primer ugnjezdene while petlje unutar for petlje bi izgledalo ovako:

```
while uslov:
    for vrednost in [vred1, vred2, vred3, itd.]:
        naredba
        naredba
        ...
```

39. Šta predstavlja tzv. pretest petlja (en. pretest loop)?
    - While petlja je tzv. petlja pre testiranja (en. pretest loop), zato što proverava uslov pre formiranja prve iteracije. To je razlog zbog koga je poželjno imati naredbe koji osiguravaju da će se petlja izvršiti barem jednom.

40. Šta predstavlja funkcija u programskom jeziku?
    - Funkcija (en. function) predstavlja grupu naredbi koje postoje unutar programa za cilj obavljanja specifičnog zadatka.
    - Umesto pisanja jednog obimnog programa kao veliki niz naredbi, moguće je napisati više manjih funkcija, gde svaka izvršava jedan specifičan deo zadatka. Ove male funkcije se onda mogu izvršiti u redosledu kojem programer odredi, sa ciljem da se odradi ukupni zadatak.
    - Ovakav pristup programiranju u žargonu se naziva zavadi-pa-vladaj (en. divide and conquer) iz razloga zato što je komplikovan zadatak podeljen u nekoliko manjih zadataka koji se lakše izvršavaju.

41. Šta predstavlja modularizovan program i kako se razlikuje od sekvencijalog programa?
    - U modularnom programiranju se uz pomoc funkcija svaki zadatak u programu izoluje u posebnu funkciju, olakšavajući pisanje i ponovno korišćenje koda. Kada se koriste funkcije u programu, u opštem slučaju izoluje se svaki zadatak (ili više zadataka) unutar programa u posebnu funkciju.
    - Kako se razlikuje od sekvencijalnog programa i zasto je bolje:
        a) Jednostavniji kod

b) Ponovno iskorišćenje koda
c) Bolje testiranje
d) Brz razvoj softvera
e) Lakši rad u timu programera

42. Koja dva tipa funkcija postoje, i u čemu se razlikuju?
- Postoje dva tipa funkcija: **void funkcije** (en. void functions) i **funkcije sa povratnim vrednostima** (en. value-returning functions).
- Kada se poziva **void funkcija** , onda takva funkcija jednostavno izvršava naredbe u telu funkcije, i onda funkcija završava sa izvršenjem.
- Kada se poziva **funkcija sa povratnom vrednošću**, funkcija izvršava sve naredbe u telu funkcije, i vrati vrednost nazad naredbi koja je pozvala funkciju.

43. Kako se definiše funkcija, a kako poziva?
- Funkcija se definise sledecom formom:

```
def ime_funkcije(param1, param2, itd…):
    naredba
    naredba
    ...
```

- a poziva se sledecom sintaksom:

```
ime_funkcije(param1, param2, itd…)
```

44. Šta predstavljaju lokalne, a šta globalne promenljive?
- **Opseg** (en. scope) promenljive jeste deo koda gde se ta promenljiva može prepoznati, Parametri i promenljive definisani unutar funkcije nisu vidljivi izvan funkcije. Zbog toga oni imaju **lokalni opseg** (en. local scope). **Životni ciklus promenljive** (en. the lifetime of a variable) jeste vreme za koje ta promenljiva postoji u memoriji. Životni ciklus promenljive unutar funkcije je onoliko koliko se potrebno da se ta funkcija izvrši. Kada se nakon poziva funkcije izvrši povratak u glavni program, lokalne promenljive se brišu dok globalne promenljive ostaju raspolozive.

45. Šta su parametri, a šta argumenti funkcije?
- Pri definisanju funkcije, u opštem slučaju, ne postoji granica o broju parametara koji se prosleđuje funkciji. Svi ulazni parametri se razdvajaju zapetom. Parametar predstavlja promenljivu koja se upisuje unutar zagrada prilikom definicije funkcije. Argumenti su vrednosti

koje prosledjujemo funkciji kad je pozivamo. Funkcija može imati i proizvoljan broj parametara i argumenta.

46. Šta predstavljaju povratne vrednosti funkcije?
    - Povratne vrednosti su vrednosti koje dobijamo prilikom pozivanja promenljive. Uz pomoc return komanda mozemo dobiti povratnu vrednost iz funkcije. Ako funkcija nema return ona vraca void. Kako bi stavili povratnu vrednost u neku promenljivu moramo koristiti istu metodu kao kod definisanja promenljive, ali umesto vrednosti koju zelimo da stavimo u promenljivu mi pozivamo funkciju. To mozemo uraditi preko sledece forme:

      ```
      ime_promenljive = ime_funkcije(param1, param2, itd…)
      ```

47. Šta su ugrađene funkcije u Python jeziku? Dati primer tri ugrađene funkcije.
    - Ugradjene funkcije u pythonu-u su funkcije koje ne moramo da definisemo da bi ih koristili. Primeri su: print(), range(), len(), min(), max(), itd…

48. Šta predstavljaju lambda funkcije?
    - Lambda funkcija može imati bilo koji broj parametra, ali se mora izvršiti u jednom redu koda.
    - Korisničke definisane funkcije u opštem slučaju mogu sadržati proizvoljni broj linije koda. Ukoliko se želi definisati funkcija koja je jednostavna i koja se može izvršiti u jednoj liniji, preporuka je koristiti tzv. lambda funkciju. (en. lambda function).
    - Mozemo definisati lambda funkciju na sledeci nacin:

      ```
      ime_funkcije = lambda p1, p2 …: povratna_vrednost
      ```

49. Koje su funkcije za standardni ulaz i izlaz u Python programskom jeziku? Opisati ove funkcije.
    - Funkcija za standardni ulaz je integrisana funkcija **`input()`** i detaljno je objasnjena u **17.** zadatku, a funkcija za standardni izlaz je integrisana funkcija **`print()`** i definisana je u **18.** zadatku.

50. Šta predstavlja datoteka, šta direktorijum, a šta sistem datoteka?
    - Datoteka (en. file) predstavlja skup podataka sadržanu u jednoj celini, identifikovana po imenu i tipu (ekstenziji, en. extension). Datoteka može biti tekstualni dokument, slika, audio ili video podatak, biblioteka podataka, izvršni program ili bilo koji skup podataka. Datoteke se mogu otvoriti, sačuvati, obrisati i premestiti u različite direktorijume

unutar sistema datoteka.

51. Kako Python deli datotke? Koja je razlika između njih?
    - Direktorijum (en. directory, folder) predstavlja kataloški sistem koji sadrži reference na datoteke.

52. Šta se dešava kada učitamo podatak iz spoljašnje memorije? **(L06)**
    - Taj podatak se smesta i cuva u RAM memoriji racunara sve dok program ne prestane sa radom ili mi obrisemo taj podatak iz memorije.

53. Opisati metode .write(), .seek() i .tell().
    - Metoda .write() predstavlja najjednostavniji način upis podataka u datoteku. Za razliku od čitanja iz datoteke, nije neophodno imati unapred napravljenu datoteku. Ukoliko datoteka ne postoji, ova metoda će napraviti.
    - Metoda .seek() podešava trenutnu poziciju toka podataka koji se upisuje.
    - Metoda .tell() vraća poziciju gde je stao tok podataka koji se upisuje.

54. Koja je razlika između otvaranja datoteke „normalno“ i korišćenja kontekstnog menadžera?
    - Kontekstni menadžer dozvoljava alociranje i oslobađanje resursa u tačno preciziranim trenucima, i automatizuje kontrolu resursa. Upotreba kontekstnog menadžera u Python jeziku postiže se ključnom rečju **with**. Prednost korišćenja kontekstnog menadžera jeste da omogućavaju rad sa datotekama unutar bloka koda, a kada se izađe iz bloka koda, datoteka se automatski zatvara.

55. Ukoliko otvaramo datoteku, da li treba ta datoteka da postoji, i (podrazumevano) gde?
    - Ako otvaramo datoteku sa modom za pisanje onda nije potrebno da ta datoteka postoji jer ako ne postoji python ce kreirati novu datoteku pod imenom ili u direktorijumu koji smo uneli, ali ako postoji python ce je otvoriti.
    - Ako otvaramo datoteku u modu za citanje onda je obavezno da ta datoteka postoji, u suprotnom python ce izbaciti gresku.

56. Ukoliko čuvamo podatke u datoteku, da li treba ta datoteka da postoji i zbog čega?
    - **isti odgovor kao za 55. pitanje**

57. Šta od parametara sadrži funkcija open()?
    - Od obaveznih parametara funkcija open() sadrzi direktorijum do fajla koji zelimo da manipulisemo. Takodje postoji parametar koji odredjuje

tip otvaranja fajla: citanje ('r'), pisanje ('w'), binarno citanje ('rb'), binarno pisanje ('wb'), itd. ako ne uracunamo ovaj parametar bice podrazumevano da zelimo da citamo iz fajla. (*Takodje postoji encoding parametar koji sadrzi tip encodinga koji je koristen u fajlu fajla, ovaj deo u zagradi nije obavezan ali svakako bi bilo lepo da se spomene*)

58. Kako kontrolišemo pristup datoteci? **(L06)**

Postoji nekoliko načina na koje možemo kontrolisati pristup datoteci u Python-u:

1. Koristiti metodu "open" sa odgovarajućim atributom za pristup: "r" za čitanje, "w" za pisanje i "a" za dopisivanje.
2. Koristiti os module: "os.chmod()" funkcija se koristi za postavljanje dozvola pristupa datoteci.
3. Koristiti "try" blok i "except" blok: "try" blok se koristi za pokušaj pristupa datoteci, a "except" blok se koristi za hvatanje izuzetka u slučaju neuspješnog pristupa.
4. Koristiti "with" konstrukciju: "with" konstrukcija se koristi za automatsko zatvaranje datoteke nakon što je korištena.
5. Koristiti modul "pathlib" : modul nudi klase za rad sa putanjama, uključujući i kontrolu pristupa datotekama.

Na koji način ćemo kontrolisati pristup datoteci zavisi od naših potreba i konkretne situacije u kojoj se nalazimo.

59. Šta se dešava ukoliko se datoteka ne zatvori?
    - Datoteka ostaje u radnoj memoriji sve dok se program ne zavrsi. Zato je bolje koristiti kontekstni menadžer jer se tad datoteka automacki zatvara.

60. Koji tipovi grešaka se mogu javiti u programiranju?
    - U opštem slučaju, greške možemo podeliti na:
        a) Sintaksne greške,
        b) Semantičke ili logičke greške,

c) Greške pri pokretanju programa.

61. Šta predstavlja sintaksna greška?
    - Sintaksna greška (en. syntax error) je greška koja se odnosi na struktura programa i na pravila te strukture. Primer: umesto **while** napisemo **wihle**.

62. Šta predstavlja logička greška?
    - Semantička greška (en. semantic error) ili logička greška (en. logic error) javlja se u situacijama kada se program uspešno izvrši, tj. interpreter ne javlja nikakve greške; međutim, program ne izvršava onaj zadatak za koji je namenjen. Uglavnom se desava kad osoba koja pise program zaboravi na neki korak tokom pisanja programa.

63. Šta predstavlja greška pri pokretanju programa?
    - Greška pri pokretanju programa (en. runtime error) javlja se tek kada je program pokrenuo sa izvršenjem. Ove greške se često nazivaju izuzecima (en. exceptions) jer se dešavaju u izuzetno (najčešće lošim) situacijama.

      Primer: Pokusaj da se ucita element iz liste pod indeksom koji ne postoji, i tada dobijamo “index out of range” exception.

64. Šta predstavlja izuzetak?
    - Izuzetak (en. exception) predstavlja događaj koji se desi tokom izvršenja programa, a taj događaj poremeti normalni tok instrukcija.
    - Postoje mnoge vrste grešaka koje mogu prouzrokovati izuzetke: od problema hardverske prirode (prestanak rada spoljašnje memorije ili diska), do jednostavnih grešaka u programiranju, kao što je pokušaj pristupa elementu liste koji ne postoji.
    - Kada se desi izuzetak, Python napravi objekat klase Exception i preda ga interpreteru. Ovaj objekat sadrži informaciju o grešci koja je napravljena.

65. Za šta koristimo ključnu reč try, za šta finally, a za šta Raise?
    - U try bloku koda upisuje se deo koda koji može izazvati grešku, a u except bloku vršimo obradu izuzetka.
    - Kod koji će se izvršiti kada nema izuzetaka biće u bloku koda nakon reči else, dok će se kod nakon reći finally izvršiti bilo da postoji greška ili ne.
    - Raise kljucna rec se koristi za namerno dizanje/izazivanje/kreiranje izuzetka.

66. Šta predstavlja proces debagovanja?
    - Debagovanje (en. debugging) jeste proces koji podrazumeva identifikovanja i ispravljanja grešaka u kodu programa
    - Proces debagovanja se sastoji iz pet koraka:
        a) Identifikacija problema
        b) Opis problema
        c) Hvatanje problema
        d) Analiza uhvaćenog problema
        e) Popravka problema

67. Koja je razlika između debagovanja i testiranja?
    - Debagovanje je resavanje problema i izuzetaka u programu, dok je testiranje potvrdjivanje da je logika programa ispravna i da nema semantickih gresaka.

68. Opisati prihvatno, integraciono i jedinično testiranje.
    - Prihvatno testiranje (en. acceptance testing), koje proverava da li ceo softverski sistem radi na način kako je predviđen.
    - Integraciono testiranje (en. integration testing), koje osigurava da komponente softvera ili funkcije softvera rade zajedno.
    - Jedinično testiranje (en.unit testing), koje validira svaku softversku jedinicu da li radi na predviđen način. Jedinica u ovom smislu je najmanja komponenta koja se može testirati.

69. Opisati funkcionalno, regresiono i stres testiranje.
    - Funkcionalno testiranje (en. functional testing), koje proverava funkcije softvera tako što emulira određene scenarije prema funkcionalnim zahtevima.Funkcionalno testiranje izvršava se tehnikom crne kutije (en. black-box testing).
    - Regresiono testiranje (en. regression testing), koje proverava da li dodavanje novih funkcionalnosti ruši ili degradira postojeće funkcionalnosti softvers.
    - Stres testiranje (en. stress testing), koje proverava koliko opterećenja može softver podneti pre nego što prestane sa radom

70. Šta su liste? **(L08)**
    - Lista je sekvenca.
    - Sekvenca je objekat koji sadrži više stavki podataka, pri čemu se jedna stavka čuva nakon druge. Možemo da izvršimo operacije nad sekvencom koju želimo da ispitamo i da manipulišemo stavkama koje se u njoj čuvaju. U Python-u postoji nekoliko različitih tipova sekvenci.

71. Šta je indeksiranje?

- Indeksiranje je jedan od nacina na koji mozemo da pristupimo pojedinacnim elementima liste. Svaki element na listi ima indeks koji specificira njegovu poziciju na listi. Indeksiranje počinje od 0, tako da je indeks prvog elementa 0, indeks drugog elementa je 1, i tako dalje. Indeks poslednjeg elementa u listi je za 1 manji od broja elemenata u listi.

72. Kako se vrši spajanje lista?
    - Liste se mogu spajati uz pomoc operatora +.

73. Na koje sve načine možemo seći liste?
    - Listu mozemo seci uz pomoc slice izraza. Sintaksa slice izrazaglasi:

      ```
      proizvoljna_lista[pocetak:kraj]
      ```

74. Nabrojati najčešće korišćene metode kod lista.
    - Append, Sort, Reverse, Copy, Remove, Index.

75. Kako možemo kopirati listu?
    - Kopiranje liste je moguce uz for petlju u kojoj se svaki element iz jedne liste appenduje u drugu listu. Takodje je moguce koristiti .copy() da bi prekopirali jednu listu u drugu.

76. Šta su dvodimenzionalne liste?
    - Dvodimenzionalne liste su tipovi listi koje kao svoje elemente sadrze liste.

77. Šta su tuple?
    - Tuple (ili torke) su nepromenljive sekvence, što znači da sadržaj tuple ne može biti promenjen. Tupla je niz, veoma sličan listi. Primarna razlika između torki i lista je da su torke nepromenljive. To znači da kada se torka napravi, ne može se promeniti. Kada kreiramo tuple, zatvaramo njene elemente u skup običnih zagrada.

78. Koja je razlika između lista i tupla?
    - Primarna razlika izmedju lista i tupla (torki) je u tome da liste posle inicijalizacije mogu da se promene dok tuple ne mogu. Takodje tuple ne podržavaju metode kao što su append, remove, insert, reverse i sort.

79. Koje operacije ne možemo vršiti kod tupla, a mogli smo kod lista? Zašto?
    - Tuple podržavaju sve iste operacije kao i liste, osim onih koje menjaju sadržaj liste. Torke ne podržavaju metode kao što su append, remove, insert, reverse i sort.

80. Definisati biblioteku u Python-u.
    - Biblioteka je kolekcija unapred kompajliranih kodova koji se kasnije mogu koristiti u programu za neke specifične, dobro definisane operacije.
    - U Python-u pod bibliotekom podrazumevamo kolekciju povezanih modula. Biblioteke sadrže delove koda koji se mogu više puta koristiti u različitim programima.

81. Šta je Python standardna biblioteka?
    - Svakako najvažnija Python biblioteka je Python standardna biblioteka (engl. Python Standard Library). Python standardna biblioteka sadrži tačnu sintaksu, semantiku i tokene Python-a. Sadrži ugrađene module koji omogućavaju pristup osnovnim sistemskim funkcijama kao što su I/O i nekim drugim osnovnim modulima. Većina Python biblioteka je napisana u programskom jeziku C. Python standardna biblioteka se sastoji od više od 200 osnovnih modula.

82. Navesti najpopularnije i najčešće korišćene biblioteke u Python-u.
    - TensorFlow, Matplotlib, Numpy, SciPy, PyGame, PyTorch, PyBrain

83. Šta je modul u Python-u?
    - Pod terminom modul podrazumevamo datoteku koja sadrži kod za obavljanje određenog zadatka i čuva se sa ekstenzijom .py. Modul može da sadrži promenljive, funkcije, klase itd.

84. Na koje sve načine možemo da uvezemo modul i funkciju iz modula u Python-u?
    - Ponekad želimo da uvezemo samo određenu funkciju ili klasu iz modula, bez da učitavamo ceo modul. Kada je to slučaj, možemo koristiti ključnu reč from sa naredbom import. Takođe, možemo da navedemo nazive više stavki, odvojenih zarezima, kada koristimo ovu formu uvoženja modula.

      `Fromat : from math import sqrt,radians`

    - Za uvoz modula možemo koristiti takozvani wildcart uvoz. Ovaj tip uvoženja takođe učitava celokupan sadržaj modula. Razlika između wildcart uvoza i prvog načina uvoza je u tome što kod ovog načina import izraz ne zahteva da koristimo kvalifikovana imena stavki u modulu.

Format : from math import *

- Da bismo koristili bilo koju od stavki koje se nalaze u modulu, stavljamo za prefiks imena stavke alias pod kojim smo uvezli modul, praćen tačkom.

Format : import math as mt
Format : from math import sqrt as square_root

85. Objasniti razliku između paketa i modula u Python-u.
    - Pod terminom paket podrazumevamo kolekciju jednog ili više povezanih modula. Paketi su u osnovi direktorijumkolekcije modula. Paketi dozvoljavaju hijerarhijsku strukturu imenskog prostora modula.

86. Šta je modularizacija?
    - Modularizacija je tehnika programiranja koja se koristi za organizovanje koda u odvojene module (ili dijelove) koji imaju specifičnu funkciju. Cilj modularizacije je da se kod razdvoji na manje dijelove koji su lakši za razumjeti, testirati i održavati.
    - Modularizacija također omogućava lakše razvoj i održavanje softvera, jer se svaki modul može razvijati i testirati nezavisno od ostalih modula. Ako je potrebno promijeniti neku funkcionalnost, to se može učiniti na modulima bez utjecaja na ostale dijelove koda.
    - Modularizacija također omogućava lakše razmjenjivanje koda između različitih projekata, jer se moduli mogu koristiti u više projekata bez potrebe za kopiranjem koda.
    - Python podržava modularizaciju kroz korištenje modula i paketa. Moduli su datoteke sa ekstenzijom .py koje sadrže kod

87. Ukratko opisati za šta se koristi pyplot modul iz matplotlib biblioteke?
    - Pyplot modul iz matplotlib biblioteke se koristi za kreiranje i prikazivanje grafikona u Python-u. On omogućava jednostavno kreiranje linija, histograma, dijagrama torte, i drugih vrsta grafikona koristeći metode kao što su plot, hist i scatter. Pyplot takođe omogućava konfigurisanje aspekata grafikona, kao što su oznake osa, naslovi i legende.

88. Ukratko opisati za šta se koristi numpy biblioteka.
    - Biblioteka NumPy predstavlja jednu od mnogobrojnih biblioteka u Pajtonu. NumPy je skraćenica od "Numerical Python". NumPy se sastoji od mnogo numeričkih funkcija koje služe za obradu višedimenzionih nizovnih objekata. Koristeći NumPy biblioteku mogu se izvesti sledeće operacije: matematičke i logičke operacije nad nizovima, operacije u vezi sa linearnom algebrom, poput operacija nad

vektorima, matricama, rešavanja sistema jednačina, itd...

89. Navesti neke najvažnije funkcije iz math modula.
    - acos, asin, atan, ceil, cos, degrees, exp, floor, hypot, log, log10, radians, sin, sqrt, tan...

90. Kako se vrši sečenje stringova?
    - Rez (en. slice) je opseg vrednosti uzete iz jedne sekvence. Kada se uzme rez stringa, dobije se opseg karaktera unutar tog stringa. Ovi stringovi se često nazivaju podstringovi (en. substring)

      Format: **`string[start : end]`**

    - U opštem formatu, start predstavlja indeks prvog karaktera reza stringa, dok je end indeks koji označava kraj stringa.Ova naredba vratiće string koji sadrži kopiju karaktera od indeksa start do (ali neuključujući) indeksa end.

91. Kako se vrši konkatenacija stringova?
    - Jedna od najosnovnijih operacija nad stringovima jeste konkatenacija (en. concatenation), tj. spajanje stringova dodavanjem jednog stringa na kraju drugog. Za konkatenaciju stringova koristi se operator + koji konkatenira stringove sa leve i desne strane operatora. Rezultat operatora + daje string koji je kombinacija dve operanda.

      Format: **`string1 + string2`**

    - Konkatenacija stringova se može izvršiti i operatorom += sa razlikom da operand na levoj strani operatora mora već postojati, tj. mora postojati promenljiva koja referencira levi string.

      Format: **`string1 += string2`**

92. Šta je šema kodovanja?
    - Poznato je da računari skladište alfanumeričke karaktere korišćenjem neke od šema kodovanja (en. encoding scheme). Šeme kodovanja mogu biti različite, ali je veoma bitno da prate dati standard.
    - Najpoznatije šeme jesu ASCII i UTF šeme.

93. Šta rade funkcije ord() i chr()?
    - Funkcije ord i chr su ugrađene funkcije pomoću kojih možemo "prelaziti" iz karaktera i njihovih numeričkih vrednosti koje se koriste za predstavljanje karaktera unutar stringa.
    - Funkcija ord() vratiće numerički kod ili ordinal (en. ordinal) za jedan karakter po Unicode šemi kodovanja.

- Funkcija chr() vratiće vrednost karaktera za unetu brojčanu vrednost po Unicode šemi kodovanja.

94. Da li su stringovi nepromenljivi? Objasniti.
    - U programskom jeziku Pythom stingovi su nepromenljivi (en. immutable), što znači da kada se jednom kreiraju, ne mogu se menjati!

95. Šta je operator testiranja?
    - U Python programskom jeziku može se koristiti operator testiranja in da bi se odredilo da se jedan string (ili jedan karakter) nalazi unutar drugog stringa.

      Format: **`string1 in string2`**

    - Rezultat operatora in je Bulov izraz (en. Boolean expression), tako da može vratiti vrednosti True ukoliko se string1 nalali u string2, ili False ukoliko to nije slučaj.

96. Nabrojati neke string metode.
    - Metode za kompletno testiranje stringova:
        a) isalnum(),
        b) isalpha(),
        c) isdigit(),
        d) islower(),
        e) isspace(),
        f) isupper()
    - Metode za modifikaciju stringova:
        a) lower(),
        b) lstrip(),
        c) rstrip(),
        d) strip(),
        e) upper()
    - Metode za pretragu i zamenu stringova:
        a) endswith(substring),
        b) find(substring),
        c) replace(old,new),
        d) startswith(substring)

97. Koja je razlika između rečnika i liste?
    - U Python-u, rečnik je objekat koji čuva kolekciju podataka. Svaki element koji se čuva u rečniku ima dva dela: ključ i vrednost. U stvari, elementi rečnika se obično nazivaju parovi ključ/vrednost (engl. key/value pairs).Dok liste cuvaju samo vrednosti.

98. Da li su rečnici promenljivi objekti?
    - Rečnici su promenljivi objekti. Možemo dodati nove parove ključ/vrednost u rečnik pomoću izraza dodele u opštem formatu:

      `ime_rečnika[ključ] = vrednost`

    - U opštem formatu, ime_rečnika je promenljiva koja upućuje na rečnik, a ključ je ključ. Ako ključ već postoji u rečniku, njegova pridružena vrednost će biti promenjena u vrednost. Ako ključ ne postoji, biće dodat u rečnik, zajedno sa vrednošću kao njegovom pridruženom vrednošću.

99. Koji deo rečnika je promenljiv, a koji nije?
    - **Pitanje 98.**
    - Vrednost je promenljiv dok ključ nije.

100. Kako se pravi prazan rečnik?
    - Možemo koristiti prazan skup sa vitičastim zagradama da napravimo prazan rečnik.

      Format: **`imenik = {}`**

    - Takođe možemo koristiti ugrađenu metodu dict() da kreiramo prazan rečnik.

      Format: **`imenik = dict()`**

101. Šta se dešava ukoliko želimo da prikažemo vrednost za ključ koji ne postoji u rečniku?
    - Ako ključ postoji u rečniku, ovaj izraz vraća vrednost koja je povezana sa ključem. Ako ključ ne postoji, podiže se izuzetak KeyError.

102. Šta vraća metoda .items()?
    - Vraća sve ključeve u rečniku i njihove pridružene vrednosti kao niz torki. Metoda items vraća sve ključeve rečnika i njihove pridružene vrednosti. Oni su vraćeni kao poseban tip sekvence poznat kao prikaz rečnika. Svaki element u prikazu rečnika je torka, a svaka torka sadrži ključ i njegovu pridruženu vrednost.

103. Šta vraća metoda .keys(), a šta .values()?
    - Keys() - Vraća sve ključeve u rečniku kao niz torki.
    - Metoda keys vraća sve ključeve rečnika kao prikaz rečnika, koji je vrsta sekvence. Svaki element u prikazu rečnika je ključ iz rečnika.

- Metoda values vraća sve vrednosti rečnika (bez njihovih ključeva) kao prikaz rečnika, koji je vrsta sekvence. Svaki element u prikazu rečnika je vrednost iz rečnik.

104. Koliko elemenata se zadaje funkciji set()?
    - Možemo zadati jedan argument funkciji set. Argument koji prosleđujemo mora biti objekat koji sadrži elemente koji se mogu ponavljati, kao što su lista, torka ili string.

105. Pomoću kojih metoda možemo dodavati elemente skupu?
    - Pomću metoda add i update.

      Format:

      ```
      mojskup.add(value)
      mojskup.update([value1,value2,value3])
      ```

106. Šta predstavlja proces pickling-a u Python jeziku?
    - U Python-u, proces serijalizacije objekta se naziva kiseljenje (engl. pickling). Python standardna biblioteka obezbeđuje modul po imenu pickle koji ima različite funkcije za serijalizaciju, ili kiseljenje objekata.
    - Jednom kada uvezemo modul za kiseljenje, izvršićemo sledeće korake za kiseljenje objekta:
        a) Otvaramo datoteku za binarno pisanje.
        b) Pozivamo metodu dump modula pickle da bismo kiselili objekat i zapisali ga u navedenu datoteku.
        c) Nakon što odaberemo sve objekte koje želimo da sačuvamo u datoteku, zatvaramo datoteku.

107. Koje su razlike između skupova i rečnika?
    - Skup je objekat koji čuva kolekciju podataka na isti način kao i matematički skupovi. Svi elementi u skupu moraju biti jedinstveni. Nijedna dva elementa ne mogu imati istu vrednost. Skupovi su neuređeni, što znači da elementi u skupu nisu uskladišteni u nekom određenom redosledu. Elementi koji se čuvaju u skupu mogu biti različitih tipova podataka.

108. Pomoću kojih metoda možemo brisati elemente iz skupova?
    - Možemo ukloniti stavku iz skupa ili metodom remove ili metodom discard.

      Format:

      ```
      mojskup.remove(value)
      mojskup.discard(value)
      ```

109. Definisati rekurzivnu funkciju.

- Funkcija koja poziva samu sebe naziva se rekurzivna funkcija (en. recursive function).

110. Šta je dubina rekurzije?
- Broj koji pokazuje koliko puta funkcija sama sebe poziva naziva se dubina rekurzije (en. recursion depth)

111. Da li se svaki problem koji se rešava rekurzijom može rešiti pomoću petlji?
- Da bi se neki problem rešio, nije uvek neophodno koristiti rekurziju. Svaki problem koji se može rešiti rekurzijom, može se rešiti i kroz petlje.

112. Kada je poželjno koristiti rekurziju prilikom rešavanja problema?
- Problem se može rešiti rekurzijom ukoliko se može podeliti na manje problem koji su identični u strukturi većeg problema.

113. Koje su mane korišćenja rekurzije?
- Zapravo, rekurzivni algoritmi često su manje efikasni of iterativnih algoritama. Razlog ovome jeste taj da proces pozivanja funkcije zahteva više resursa of računara. Ovi resursi predstavljaju alokaciju parametara i lokalnih promenljivih u memoriji, vraćanje lokacija gde svaka rekurzija vrati kontrolu kada se konkretni poziv završi. Ove procesi se nekad nazivaju dodatni ili procesi opših troškova (en. overhead), i izvršavaju se prilikom svakog poziva funkcije. Ovi dodatni procesi nisu neophodni ukoliko se problem rešava korišćenjem petlje.
- Međutim, neki problemi koji se ponavljaju se lakše rešavaju sa rekurzijom nega sa petljom. Dok rešavanjem kroz petlju program može brže da se izvrši, rekurzivni algoritam se može lakše projektovati.

114. Šta je bazni slučaj kod rekurzije?
- Smanjenjem problema pozivanjem rekurzivne funkcije doći će se do baznog slučaja i data će rekurzija prestati.
- Najpre treba idendifikovati barem jedan slučaj gde se problem može rešiti bez rekurzije,ovaj slučaj naziva se osnovni slučaj ili bazni slučaj (en. base case)

115. Šta je diretkna, a šta indirektna rekurzija?
- Direktna rekurzija (en. direct recursion):
    - Ovakav slučaj se dešava kada funkcija A poziva funkciju A.
- Indirektna rekurzija (en, indirect recursion):
    - Ovakav slučaj se dešava kada funkcija A poziva funkciju B, koja ponovo poziva funkciju A.
    - Naravno, moguće je napraviti i više funkcija koje bi bile uključene u indirektnu rekurziju.

116. Da li kod rekurzije može postojati više od jednog baznog slučaja?
- Moze

117. Da li je efikasnija rekurzija ili petlje?
- Rekurzivne funkcije su svakako manje efikasnije nego petlje.
    a) Svaki put kada se funkcija pozove, nastaju dodatni troškovi sistema koji nisu uvek prisutni kod petlji.
    b) U mnogim slučajevima, pristup sa petljama je očigledniji nego rekurzivno rešenje.
- Međutim, neki problemi se lakše rešavaju kroz rekurziju nego kroz petlje. Matematička definicija najvećeg zajedničkog delioca je jako prilagođena rekurzivnom rešenju.

118. Šta je prekoračenje steka?
- Najveći problem koji može doći prilikom poziva rekurzivne funkcije je tzv. prekoračenje steka (en. stack overflow). Svaki put kada se program startuje, alociraju deo memorije za njihov stek, i kada se popuni stek memorija program naglo prekida sa izvršenjem (en. program crash).
- Zbog prirode rekurzivnih funkcija, svaki poziv funkcije rezervisaće mesto u stek memoriji za argumente i lokalne promenljive te funkcije. Ukoliko imamo stek memoriju malog kapaciteta a veliki broj rekurzivnih poziva, može doći do prekoračenja steka.

119. Objasniti šta je ternarni uslov.
- Kod ternarnog uslova (ternarnog operatora) moguće je izvršiti ispitivanje uslova u jednom redu.

120. Objasniti prednosti korišćenja kontekstnog menadžera.
- Upotrebom kontekstnih menadžera lakše se upravljaju resursi koji bi se posle korišćenja morali zatvoriti.
- Pri radu sa kontrolom resursa, kao što su datoteke ili niti (en. thread), treba voditi računa o resursima da se resursi zatore nakon korišćenja. Ukoliko se koriste kontekstni menadžeri, onda se unutar bloka kod kontekstnog menadžera izvršavaju operacije nad resursima, a čim se izađe iz tog bloka koda, resurs se zatvara.

121. Za šta se koristi funkcija enumerate()?
- Upotrebom funkcije enumerate moguće je dobiti i indeks i promenljivu unutar petlje.

    Format:

```
for index,item in enumerate(list):
    pass
```

- Korišćenjem funkcije enumerate() može se odmah dobiti indeks neke iterabilne promenljive prilikom prolaska kroz petlju

122. Za šta se koristi funkcija zip()?
   - Funkcija zip napraviće zip objekat sa tuple-ovima koji sadrže elemente listi koje su parametri funkcije. Mapiranje se vrši dok se ne istroše elementi najkraće liste. Zip objekat nije moguće direktno odštampati, već je potrebno pretvoriti u listu.

123. Za šta se koristi funkcija help()?
   - Funkcija help() omogućavaju programeru da nauče kako se radi sa nekim objektom.

     Format: **`help(object)`**

   - Kao izlaz, vratiće se objašnjenje objekta (ukoliko postoji).

124. Za šta se koristi funkcija dir()?
   - Funkcija dir() ispisuje koje su funkcije dostupne u okviru modula ili paketa.

     Format:

     ```
     import module
     dir(module)
     ```

   - Izlaz ove funkcije ispisaće kao listu sve funkcije koje je moguće pozvati kada se modul ili paket učita.

125. Objasniti pojam razumevanja lista.
   - Razumevanje listi (en. list comprehension) jeste metoda pisanja listi i sličnih iterabilnih promenljiva koja je bliža matematičkoj notaciji konstrukciji skupova.

126. Navesti česte propuste prilikom programiranja u Python-u.
   a) Konflikti u imenovanju
   b) Promenljivi podrazumevani argumenti
   c) Uvoz kompletnog modula

127. Objasniti šta je PEP8.
   - PEP 8 preporuka prestavlja uputstvo za stil pisanja Python koda (en. Style Guide for Python Code) Ova preporuka daje konvencije kodiranja Python koda i standardne biblioteke unutar Python instalacije.

128. Šta je objekat u okviru objektno-orijentisanog programiranja?
   - Objekat je softverski entitet koji sadrži i podatke i procedure.

- Objekat je, konceptualno, samostalna jedinica koja se sastoji od atributa podataka i metoda koje izvode operacije nad atributima podataka.

129. Šta je atribut, a šta metoda u okviru objektno-orijentisanog programiranja?
    - Podaci sadržani u objektu poznati su kao atributi podataka objekta. Atributi podataka objekta su jednostavno promenljive koje pozivaju podatke.
    - Procedure koje objekat izvodi poznate su kao metode. Metode objekta su funkcije koje obavljaju operacije nad atributima podataka objekta.

130. Šta je enkaplusacija u okviru objektno-orijentisanog programiranja?
    - Enkapsulacija je jedna od ključnih karakteristika objektno orijentisanog programiranja. Enkapsulacija se odnosi na spajanje atributa i metoda unutar jedne klase. Sprečava spoljašnje klase da pristupe i menjaju atribute i metode klase. Ovo takođe pomaže u postizanju skrivanja podataka.

131. Šta je sakrivanje podataka u okviru objektno-orijentisanog programiranja?
    - Skrivanje podataka se odnosi na sposobnost objekta da sakrije svoje atribute podataka od koda koji je izvan objekta. Samo metode objekta mogu direktno pristupiti i izvršiti promene nad atributima podataka objekta. Objekat obično sakriva svoje podatke, ali dozvoljava spoljnom kodu da pristupi njegovim metodama.

132. Šta su javne metode u okviru objektno-orijentisanog programiranja?
    - Metode kojima mogu pristupiti entiteti izvan objekta su poznati kao javne metode.

133. Šta je klasa u okviru objektno-orijentisanog programiranja?
    - Klasa je kod koji specificira atribute podataka i metode određenog tipa objekta. Možemo zamisliti klasu kao „nacrt“ (engl. blueprint) iz kojeg se mogu kreirati objekti.

      Format :

```
class Ime_Klase:
    pass
```

134. Šta je instanca u okviru objektno-orijentisanog programiranja?
    - Svaki objekat koji je kreiran iz klase naziva se instanca klase.

135. Objasniti init metodu u okviru objektno-orijentisanog programiranja.

- Metoda __init__ je poznata kao metoda inicijalizacije, jer inicijalizuje atribute podataka objekta. Naziv metode počinje sa dva znaka za donju crtu, nakon čega sledi reč init, nakon čega slede još dva znaka za donju crtu. Metoda __init__ je obično prva metoda unutar definicije klase.

  Format:

```
class Ime_Klase:
    def __init__(self):
        #What will happend when class is created
        return #not necessary
    pass #not necessary
```

136. Objasniti parametar self u okviru objektno-orijentisanog programiranja.
   - Parametar self je potreban u svakoj metodi klase. Kao što smo već rekli, kod objektno- orijentisanog programiranja metoda vrši operacije nad atributima podataka određenog objekta. Kada se metoda izvrši, ona mora imati način da zna koji su atributi podataka objekta nad kojima bi trebalo da deluje. Tu se pojavljuje parametar self. Kada se pozove metoda, Python pravi da self parametar upućuje na određeni objekat nad kojim metoda treba da deluje.

137. Kako vršimo sakrivanje atributa u okviru objektno-orijentisanog programiranja?
   - U Python-u možemo sakriti atribut tako što ćemo njegovo ime započeti sa dva znaka za donju crtu.

     Format: `__atribut` #Je sakriven(privatan) atribut

138. Objasniti str metodu u okviru objektno-orijentisanog programiranja.
   - Prikazivanje stanja objekta je uobičajen zadatak. Toliko je uobičajeno da mnogi programeri "opremaju" svoje klase metodom koja vraća string koji sadrži stanje objekta. U Python-u ovoj metodi dato je posebno ime __str__. Gotovo uvek je elegantniji način da prikažemo stanje objekta pomoću __str__ metode prilikom kreiranja klase.

139. Objasniti setere i getere u okviru objektno-orijentisanog programiranja.
   - Metoda koja vraća vrednost iz atributa klase, ali je ne menja, poznata je kao pristupni metod.
   - Metoda koja čuva vrednost u atributu podataka ili menja vrednost atributa podataka na neki drugi način je poznat kao mutatorski metod.
   - Mutatorske metode se veoma često u literaturi nazivaju seteri, a pristupne metode se nazivaju geteri.

140. Šta je UML dijagram?
    - Kada dizajniramo klasu, često je korisno nacrtati UML dijagram. UML je skraćenica za jedinstveni jezik za modelovanje (engl. Unified Modeling Language). UML nam obezbeđuje skup standardnih dijagrama za grafički prikaz objektno-orijentisanih sistema.