Uvod u programiranje

Elementi teorije sa zbirkom riješenih zadataka

Elmedin Selmanović
Sead Delalić

Sadržaj

Osnove programiranja sa zadacima

1. Uvod

Razvoj softvera predstavlja jednu od najvažnijih grana u razvoju modernog društva. Zahvaljujući razvoju informacionih tehnologija, ljudi su u stanju brže i efikasnije komunicirati putem elektronske pošte i servisa za razmjenu poruka, razmjena novca i plaćanje je pojednostavljeno putem online bankarstva, mnogi poslovi se obavljaju udaljeno, transport je modernizovan pomoću autonomnih vozila i asistenata u vožnji, obrazovanje je promijenjeno uvođenjem online kurseva i platformi, te je teško pronaći oblast u kojoj nije došlo do promjena.

Programiranje predstavlja proces pisanja naredbi i uputa koje računar pokušava izvršiti. Skup naredbi i uputa se najčešće naziva kod. Osoba koja piše kod, naziva se programer. Razvoj kompleksnih softvera podrazumijeva mnogo više od programiranja. Kompletan razvoj softvera dijeli se na više faza, od kojih se najčešće ističu: planiranje, implementacija, testiranje i održavanje. Programiranje nije direktno sadržano samo u prvoj fazi. Međutim, značaj planiranja je ključan za isplativ i uspješan razvoj softvera. Kvalitetno kreirane specifikacije značajno olakšavaju proces programiranja. U realnom svijetu, kompletni procesi su rijetko do kraja isplanirani, te se uvijek u toku razvoja softvera specifikacije mijenjaju, što zahtijeva ulaganje dodatnog napora u razvoj. Međutim, proces razvoja softvera traje, te se pri razvoju prikupljaju dodatne informacije o procesima, te se sami procesi često mijenjaju vremenom.

Svaki kompleksniji softver razvija se u timu, te je teško pronaći uspješne projekte koji su autorsko djelo samo jedne osobe. Razvoj softvera je postao kompleksan proces do mjere da je teško pronaći osobu koja može razumjeti svaku komponentu softvera (neophodne hardverske specifikacije, svaki dio koda, strukturu baze podataka, svaki proces u aplikaciji i ostalo). Programeri su često usko specijalizovani za određenu oblast tokom rada na razvoju jednog sistema (npr. kreiraju samo pozadinske elemente, bez dodira za grafičkim interfejsom, ili kreiraju samo grafički interfejs ili samo rade na održavanju i kreiranju baze podataka i slično).

U toku rada, programer značajan dio vremena provodi u procesu planiranja, razmišljanja, istraživanja i učenja. Jedna od najvažnijih vještina koju programer može imati jeste upravo vještina istraživanja i učenja. Proces učenja ne prestaje iz razloga što se programer konstantno susreće sa novim problemima u projektu, sa problemima sa kojima se nije ranije susretao u istom

obliku ili koristeći istu tehnologiju. Istraživanje na internetu uzima značajan dio radnog vremena programera, kao i proces **refaktorizacije** i testiranja koda. Refaktorizacija koda podrazumijeva optimizaciju i modifikaciju određenih dijelova koda, sa ciljem kreiranja jasnijih programskih struktura jednostavnijih za daljnji razvoj.

Proces programiranja odvija se u ciklusima, te se svaki može podijeliti na nekoliko ključnih koraka: dizajn, kodiranje, ispravke, testiranje i ispravke logičkih grešaka. Proces dizajniranja softvera podrazumijeva kreiranje specifikacija za dio koda koji se piše, kao i proces planiranja elemenata koji će biti korišteni. Naredni korak podrazumijeva kodiranje osnovne verzije koda. Ta verzija koda najčešće nije optimalna i ispravna (pogotovo u kompleksnijim dijelovima softvera). Stoga, pristupa se procesu **debagiranja**. Ovaj proces podrazumijeva ispravljanje pogrešnih dijelova koda koji sprječavaju pokretanje i izvršavanje koda. Takve greške se nazivaju **sintaksne greške**. Nakon uspješnog pokretanja koda, slijedi proces testiranja. Proces testiranja treba biti što je moguće detaljniji i treba pokriti što veći broj različitih slučajeva. Neiskusni (a nekada i iskusni) programeri često prave greške u **rubnim slučajevima** (engl. edge case). Zamislimo slučaj gdje korisnik treba unijeti prirodan broj n, a program mu treba iscrtati mnogougao sa n stranica. Većina programera početnika u prvom testiranju nakon implementacije softvera provjerava slučaj trougla, četverougla i eventualno petougla. Međutim, postavlja se pitanje, kako se aplikacija ponaša kada korisnik unese broj nula ili neki negativan broj? Da li aplikacija ispisuje prikladnu poruku i oblik? Šta se dešava kada korisnik unese broj jedan ili broj dva? Upravo ovakvi slučajevi nazivaju se rubnim slučajevima. U procesu testiranja otkrivaju se **logičke greške**. Ovaj tip grešaka teže je ispraviti u odnosu na sintaksne, jer zahtijevaju značajniju promjenu dizajna programa. Program može biti sintaksno ispravan, međutim, isto tako, niko ne garantuje da program radi ono što se traži (možda je programer u potpunosti pogrešno razumio specifikacije problema, odnosno, zadatak). Stoga, greške u kojima program ne ispunjava specifikacije nazivaju se logičkim greškama.

Naredni korak je ponavljanje ciklusa, da li zbog kreiranje novog dizajna pri ispravljanju logičkih problema ili rad na potpuno novom zadatku.

Ključni korak u dizajnu specifikacija za program jeste razbijanje istog na manje i jednostavnije komponente. U slučaju da želimo implementirati *chat* aplikaciju, njen rad razbijamo na više jednostavnih elemenata čijim spajanjem dobijamo kompletnu aplikaciju. Ti jednostavni elementi su prijava (*login*), slanje poruke, čitanje poruke, brisanje poruke, lista poruka, nova poruka ... Slanje poruke može se razdvojiti na dodatne manje procese: uspostavljanje veze sa primaocem, provjera da li je uspostavljena veza sa željenim primaocem, provjera ispravnosti poruke, slanje poruke, potvrda o slanju ... Svaki od ovih podkoraka se može dodatno pojednostaviti, te zaustavljanje procesa razbijanja na jednostavnije podprocese zavisi i razlikuje se od programera do programera.

Iako se početnicima proces dizajna i planiranja može činiti kao nepotrebno gubljenje vremena (i novca na kraju krajeva), iskustvo i praksa su pokazali da je dobro planiran softver u pravilu brže završen i za manje novca od softvera koji nije planiran. Razlog su svakako ispravke i naknadne modifikacije, a jasno je da je jednostavnije modifikovati opis i specifikcije programa od gotovog koda.

Za planiranje programa, često se koriste dijagrami. Jedan od najjednostavnijih dijagrama je dijagram toka. Ovaj dijagram opisuje aplikaciju kroz niz koraka koji se izvršavaju pri pokretanju programa. U dijagramu toka, kompletan proces je osmišljen do detalja. Drugi čest način za opis koda je **pseudokod**. Pseudokod predstavlja tekstualni opis izvršavanja koda. Zamislimo da želimo aplikaciju koja za osobu računa mjesečnu platu. Pseudokod za ovu aplikaciju bi mogao biti napisan kao u primjeru ispod:

- Traži od korisnika unos satnice
- Traži od korisnika unos broja sati

- Pomnoži broj sati i satnicu, te spasi dobijenu vrijednost
- Prikaži dobijenu vrijednost korisniku

Iako je oblast programiranja doživjela ekspanziju u posljednjim desetljećima, za svakog uspješnog programera prvi koraci se nisu značajno mijenjali. Najvažniji korak u učenju programiranja jeste razumijevanje programerske logike i postavljanje kvalitetnih temelja. Osnovni programerski koncepti su slični za veliki broj programskih jezika i nisu se mnogo mijenjali kroz godine. Programer sa ispravnom i razvijenom logikom programiranja može jednostavno i brzo naučiti sintaksu proizvoljnog jezika.

Knjiga daje teorijski uvod i opisuje osnovne koncepte programiranja kroz jedan od najpopularnijih programskih jezika današnjice, a to je programski jezik Python. Osim teorijskog uvoda, knjiga sadrži niz riješenih problema i primjera od kojih su mnogi zasnovani na modeliranju događaja iz realnog svijeta.

Programski jezik Python ima široku primjenu u raznim oblastima nauke i industrije, poput: mašinskog učenja, neuralnih mreža i drugih oblasti vještačke inteligencije, kreiranja web aplikacija, edukacije, bioinformatike, fizike, medicine... Mnoge velike svjetske organizacije i kompanije koriste Python: Google, CERN, NASA, Facebook, Amazon, Instagram, kao i niz drugih. Python je izuzetno moćan i brz programski jezik jednostavan za učenje. Kreiran je 1991. godine sa ciljem da bude jednostavan za rad, učenje i brzu primjenu.

Python koristi **interpreter** za razliku od značajnog broja drugih programskih jezika koji koriste kompajlere (C, C++, Java). U slučaju interpretera, kod se izvršava liniju po liniju, te u slučaju pojave greške u jednom dijelu koda, svi raniji dijelovi su uspješno izvršeni, te program prekida rad na dijelu koda u kojem je došlo do greške.

Standardne i moderne aplikacije sadrže **GUI** (grafički korisnički interfejs, engl. *Graphical User Interface*) koji se sačinjen od prozora, dugmića, polja za unos podataka i raznih drugih grafičkih komponenti. Programski jezik Python podržava kreiranje takvih komponenti. Međutim, za implementaciju zadataka koji će biti obrađeni u ovoj knjizi, korištenje GUI komponenti neće biti neophodno. Izvršavanje programa će se odvijati kroz **konzolu** koja predstavlja osnovni element komunikacije korisnika i programa. Kroz konzolu, korisnik može unositi tražene vrijednosti, ali isto tako, može dobiti odgovarajuće informacije i rezultate izvršavanja programa. Važno je napomenuti da podatke ispisane u konzolu programer ne može mijenjati, te da će konzola od trenutka pokretanja programa biti jedini način komunikacije korisnika i programa. Kompletan proces odvija se u više faza koje se ne preklapaju: programer napiše program, ispravi greške, pokrene ga i ne može ga mijenjati u toku njegovog trajanja, korisnik komunicira sa programom putem konzole i očekuje u konzolu ispisane rezultate izvršavanja programa.

Za pisanje Python koda, može se koristiti bilo koji tekstualni editor uz uslov da je dostupan interpreter za izvršavanje koda. Svaki kod se sastoji od niza naredbi, te se kodovi smještaju u fajlove. Ekstenzija na kraju svakog Python fajla je `.py`.

Kako kompleksni softveri imaju više stotina hiljada ili čak milione linija koda, nepraktično je da kompletan kod bude sadržaj u jednom fajlu. Stoga, kod se razdvaja u više fajlova. Više fajlova se u kompleksnim sistemima uvezuje u projekat.

Python zahtijeva izuzetnu jasnoću i preciznost u kodu. Ta jasnoća se postiže uvlačenjem dijelova koda, ispravnim poravnanjima i kreiranjem blokova. Svaki niz naredbi se razdvaja u blokove. **Blok koda** predstavlja jednu ili više naredbi koje čine jednu logičku cjelinu. Jedan blok naredbi sačinjavaju sve naredbe koje su jednako ili više uvučene. Važno je napomenuti da se svaki dio koda mora precizno definisati, te da od programskog jezika (kao i računara i interpretera) ne možemo očekivati veliku pomoć. Računar izvršava samo i tačno one naredbe koje su navedene u kodu!

Standardni programi sastoje se od tri osnovne komponente: ulaz, izlaz i procesuiranje. **Ulaz** (engl. *input*) predstavlja interakciju korisnika sa programom na način da se od korisnika zahtijeva unos određenih podataka. S druge strane, **izlaz** (engl. *output*) predstavlja prezentaciju određenih informacija korisniku (u našem slučaju kroz konzolu). Procesuiranje podataka čini suštinu svakog programa, to je niz koraka koji od ulaznih podataka kreira odgovarajuće izlazne podatke.

1.1 Ispis u konzolu

Najjednostavniji Python program, a ujedno i program koji je prvi za većinu programera, naveden je ispod:

```
print('Hello world')
```

Navedeni kod u konzolu ispisuje tekst *Hello world* koji je standardni primjer za prvi program u većini programskih jezika. Za ispis se koristi funkcija `print` kojoj prosljeđujemo odgovarajući sadržaj koji treba biti ispisan. Funkcije izvršavaju određen zadatak (u ovom slučaju, `print` ispisuje sadržaj u konzolu), a konkretne vrijednosti koje se proslijede funkciji nazivaju se argumenti funkcije.

U primjer iznad, sadržaj je predstavljen kao `'Hello world'`. Iznad prikazani podaci predstavljaju niz znakova koji se u svijetu programiranja naziva **string**. Kada se string nalazi u sklopu koda, naziva se **string literalom**. String literali se u programskom jeziku Python označavaju apostrofima na početku i kraju. Dakle, ispravan string literal mora sadržavati znak apostrofa (`'`) na početku i na svom kraju. Programski jezik Python dozvoljava da se string literali označavaju i sa dvostrukim navodnicima (`"`) na početku i na kraju. Programerima se ostavlja na izbor koji će način označavanja koristiti, iako se u praksi češće koriste jednostruki navodnici (tj. apostrofi).

U slučaju da je neophodno da string literal koristi dvostruke navodnike ili apostrof u svom sadržaju, moguće je kombinovati navodnike kao u primjerima ispod.

```
print('Rekao je: "Sta ima?"')
print("Rekao je: 'Sta ima?'")
```

U prvom redu će se u konzolu ispisati *Rekao je: "Sta ima?"*, dok će se u drugom slučaju ispisati *Rekao je: 'Sta ima?'*.

U slučaju da želimo prikazati i jednostruke i dvostruke navodnike u jednom ispisu, to je moguće uraditi korištenjem trostrukog apostrofa (`'''`) ili trostrukih dvostrukih navodnika (`"""`) na početku i na kraju ispisa.

```
print('''Rekao je: "Idemo jesti u McDonald's?"''')
print("""Rekao je: "Idemo jesti u McDonald's?""")
```

Ovaj tip označavanja string literala dozvoljava ispisivanje sadržaja u više linija, što nije bilo moguće korištenjem jednog para navodnika (sa ranije pomenutim alatima). Moguće je uočiti da su se dvije rečenice ispisale u dvije linije konzole, jedna ispod druge. Dakle, u slučaju da navedemo dvije ili više naredbi `print`, svaka se ispisuje u novom redu ukoliko to nije drugačije navedeno. Iako je moguće kompletan sadržaj u više `print` naredbi spojiti u jedan, te ga pozvati kroz jedan poziv, jasno je da to nije optimalno rješenje i da ne pomaže u jednom od osnovnih ciljeva programskog jezika Python: jasnoća koda. Stoga, u slučaju da želimo spojiti ispise više `print` naredbi u jednu liniju konzole, koristimo opciju `end=' '` na kraju print naredbe, kao u primjeru ispod.

```
print('Prvi dio linije', end=' ')
print('i nastavak')
```

Atribut `end` omogućava da postavimo način razdvajanja ispisa u odnosu na naredni ispis, tj. da postavimo kursor od kojeg naredni ispis počinje na željenu poziciju. Ispis u prethodnom primjeru bi bio *Prvi dio linije i nastavak*. U slučaju da je umjesto praznog znaka `end` opcija postavljena na `'TEST'`, ispis bi bio *Prvi dio linijeTESTi nastavak*.

Osim spajanja više `print` naredbi u jednu liniju, postoji i postupak ispisivanja teksta u više linija. U tom slučaju, na mjestu na kome želimo napraviti prelom koristimo kombinaciju znakova `\n`, koja označava prelazak u novi red. Ovo je jedan od specijalnih znakova, koji se popularno zovu *escape characters*. Početak specijalnog znaka označava se sa kosom linijom (*backslash*), nakon čega slijedi znak koji dobija posebno značenje. Osim opcije prelaska u novi red, postoji i opcija `\t` (tekst se pomjera za jedan tab horizontalno), `\'` (ispis apostrofa unutar string literala, apostrof se ne smatra završetkom ili otvaranjem novog string literala), `\"` (ispis navodnika unutar string literala, navodnik se ne smatra završetkom ili otvaranjem novog string literala), `\\` (ispis kose linije unutar stringa).

U slučaju da želimo ispisati više string literala unutar jedne `print` naredbe, to možemo uraditi tako što sadržaj navedemo unutar naredbe i razdvojimo zarezom. Primjer takve naredbe naveden je ispod.

```
print('a', 'b', 'c', 'd', 'e')
```

Rezultat ispisa ove naredbe će biti *a b c d e*. U slučaju da elemente želimo razdvojiti na drugi način (ne pomoću razmaka), koristimo argument `sep=''`. U ovom slučaju, ispisi neće biti razdvojeni. Kao separator možemo koristiti proizvoljan string koji će se umetnuti između svaka dva elementa u naredbi `print`.

```
print('a', 'b', 'c', 'd', 'e', sep='')
```

Kod iznad u konzolu ispisuje tekst *abcde*.

1.2 Komentari

Kako je ranije naglašeno, programiranje je timski rad. Česte su situacije gdje jedan kod naslijede kolegice i kolege, te se postavlja razumno pitanje: koliko je teško pratiti kod drugih programera? U tumačenju naslijeđenog koda, od velike su važnosti **komentari**. Komentari su dijelovi u kodu koje interpreter ignorira, te se nalaze tu samo kao napomena programeru (ili osobi koja naslijedi kod). Komentari nisu neophodni za rad programa i ne utiču na njegov rad, međutim, predstavljaju lijepu i korisnu praksu u radu. Označavaju se znakom #. Kompletan sadržaj linije nakon navođenja pomenutog znaka smatra se komentarom i interpreter ga ignorira.

```
# Ovo je prvi komentar
# Ovo je drugi komentar
# A evo i treceg komentara
print('Hello world') #Tu je i komentar broj 4
```

Program iznad, ignorira sadržaj linija nakon znaka #, stoga, ispis koda iznad će biti jednak prvom napisanom Python programu. Komentari se standardno koriste za opis dijelova koda, pogotovo u slučaju kompleksnijih programskih komponenti.

2. Varijable, unos i izrazi

U prvom poglavlju, opisan je jednosmjerni način komunikacije programa i korisnika, gdje program može ispisivati poruke korisniku putem konzole. Međutim, svaki od napisanih programa zasnivao se na ispisu koji nije mogao biti promijenjen u konzolu. Drugačije rečeno, bez obzira na sve ostale elemente, sadržaj koji se ispisuje nije mogao biti drugačiji pri pokretanju programa. U pomenutom primjeru sa računanjem plate, program bi uvijek davao isti ispis, bez obzira koji ga korisnik pokreće. Naravno, jasno je da je ograničenje sa zadacima koji se mogu implementirati ovim putem veliko, te da su neophodni novi teorijski elementi za kreiranje kompleksnijih programa.

2.1 Varijable

Prvi koncept koji će biti uveden je koncept varijabli. Varijable predstavljaju lokacije u memoriji koje mogu sadržavati različite vrijednosti u toku izvršavanja programa. U Python programiranju, varijablama se dodjeljuje vrijednost upotrebom operatora jednako. U primjeru `x = 10`, kreirana je varijabla `x` kojoj je dodijeljena vrijednost 10. Dakle, znak jednako predstavlja *operator dodjele*, te varijabli na lijevoj strani dodjeljuje vrijednost navedenu na desnoj strani. U navedenom primjeru, naziv varijable je označen sa `x`, međutim, taj naziv može biti skoro proizvoljan niz znakova. Postoji svega nekoliko ograničenja pri imenovanju varijabli:

- Naziv varijable može se sastojati od velikih i malih slova engleskog alfabeta, znaka donje crte, te cifara (npr. `nova_varijabla_1`). Ostali znakovi, uključujući i znak praznog mjesta, nisu dozvoljeni.
- Naziv varijable ne može počinjati cifrom.
- Nije dozvoljeno za naziv varijabli koristiti ključne riječi Python programskog jezika (npr. `in`, `for`, `and`).
- Nazivi su osjetljivi na velika i mala slova (engl. *case-sensitive*), što znači da npr. varijable sa nazivom `Test` i `test` nisu jednake.

U slučaju da vrijednost varijable `x` želimo ispisati korisniku u konzolu, koristimo naredbu `print(x)`. Važno je naglasiti da pri ispisu varijable ne smiju stajati navodnici, jer bi se varijabla

u tom slučaju interpretirala kao string kojeg je neophodno ispisati, te bi naziv, a ne vrijednost varijable bio ispisan u konzolu. U programskom jeziku Python, kao i u drugim programskim jezicima, dobra je praksa varijable imenovati na način da opisuju vrijednost koja je sadržana u njima. Ukoliko imamo varijablu koja čuva podatak o godinama neke osobe, tu je varijablu prikladnije imenovati sa `godine`, nego sa `x`. Time kod činimo pristupačnijim drugim programerima, jasnijim i lakšim za **debagiranje**. U Pythonu se za imenovanje varijabli pomoću više riječi koristi standard da se svaka riječ odvaja pomoću donje crte (_), te je svaka riječ napisana malim slovima (npr. `godina_rodjenja`). Primijetite da umjesto slova *đ* koristimo kombinaciju slova *dj*, jer je pri imenovanju varijabli dozvoljeno koristiti samo slova iz engleskog alfabeta.

Vrijednost sačuvane u varijablama često želimo prikazati korisniku pomoću naredbe print. Postoji više načina da to uradimo, te upravo uvođenjem varijabli, naredba `print`, gdje je svaki dio sadržaja razdvojen zarezom poprima puni smisao. Dva načina navedena su u primjeru ispod.

```
sprat = 4
# prvi nacin
print('Stan se nalazi na spratu broj', sprat)

# drugi nacin
print('Stan se nalazi na spratu broj')
print(sprat)
```

U prvom slučaju, kompletna rečenica će biti ispisana u jednoj liniji. U drugom slučaju, rečenica će biti ispisana u dvije linije, međutim, dodavanjem atributa `end = ' '` u prvu `print` funkciju, dobijamo zapis u jednoj liniji. Prvi način se češće koristi, jer zahtijeva pisanje manje linija koda.

Kao što samo ime označava, varijable mogu mijenjati svoju vrijednost. U programskom jeziku Python, varijable mogu sadržavati proizvoljan **tip** vrijednosti (brojeve, stringove i drugo), te jedna varijabla može u jednom trenutku sadržavati vrijednost jednog, a u drugom trenutku vrijednost drugog tipa. U slučaju da želimo promijeniti vrijednost varijable `godine`, dovoljno je dodijeliti novu vrijednost `godine = 21`. Sa varijablama je moguće koristiti standardne aritmetičke operacije, tako da je u varijablu moguće smjestiti vrijednost $21 + 5$, kao `godine = 21 + 5`. Sada će vrijednost smještena u varijabli `godine` biti jednaka 26. Isto tako, moguće je staru vrijednost varijable promijeniti, kao u primjeru `godine = godine + 5`. Iako je ovo matematički nelogičan izraz, sjetimo se da je u znak jednakosti u programiranju operator dodjele. Stoga, varijabli koja se nalazi na lijevoj strani će biti dodijeljena vrijednost sa desne strane. Vrijednost na desnoj strani se izračuna (26 + 5), te se vrijednost 31 smjesti u varijablu `godine`.

Programski jezici dozvoljavaju rad sa više tipova podataka. U programskom jeziku Python, za cijele brojeve se označava tip `int` (engl. *integer*), dok se za decimalne brojeve koristi naziv `float` (nastalo od načina spašavanja u memoriji, tzv. *floating point* zapis). Decimalni zarez se mijenja tačkom, stoga, broj $2,5$ zapisujemo kao `2.5`. Brojevi se smatraju numeričkim literalima u slučaju da su zapisani u kodu. Programski jezik Python automatski pokušava zaključiti tip određenog literala. U slučaju da je broj zapisan bez decimalne tačke (`4`, `-123`), smatramo ga cijelim brojem (`int`). U slučaju da broj sadrži decimalnu tačku, smatra se tipom `float` (`5.24`, `10.0`).

Osim numeričkih tipova varijabli, Python sadrži tip `str` koji se koristi za čuvanje stringova u memoriji. Tip varijable je prvenstveno važan zbog načina definisanja operacija nad tim tipovima (dijeljenje i množenje cijelih brojeva je dozvoljeno, dok dijeljenje stringova nema smisla). Definicija i navođenje string varijable, jednako je ostalim slučajevima, s tim da varijable koje su ovog tipa moraju biti označene navodnicima (standardno je koristiti jednostruke navodnike): `mjesec =`

`'mart'`.

U programskom jeziku Python, varijabla se najjednostavnije opisuje kao referenca (upućuje) na memorijsku lokaciju. Stoga, promjena vrijednosti varijable je jednostavna bez obzira na tip. Kao što smo vrijednost varijable `godine` promijenili na 21, dozvoljeno je postaviti vrijednost `godine = 21.2` ili `godine='dvadeset'`. Obratite pažnju da je `godine = '20'` varijabla tipa string, a ne cijeli broj.

2.2 Unos podataka

Varijable su ključne za uspostavljanje dvosmjerne komunikacije na relaciji program i korisnik. Međutim, da bi varijable dobile potpuni smisao i čuvale određenu vrijednost, neophodno je unos iste zahtijevati od korisnika. Ovakvi programi su standardni u svakodnevnom životu. U slučaju da kreiramo aplikaciju koja poziva neku osobu, neophodno je da korisnik unese broj ili ime osobe koju pozivamo. S druge strane, pri prijavi ili registraciji na neki sistem, neophodno je unijeti korisničko ime i lozinku. U svakoj igrici, akcije se unose od strane korisnika. Stoga, ključno je uspostaviti komunikaciju u kojoj korisnik može proslijediti odgovarajuće podatke programu. Funkcija koja to omogućava naziva se `input`. Način njene upotrebe je `varijabla = input(tekst)`, gdje je `varijabla` upravo naziv varijable u koju ćemo smjestiti unesenu vrijednost (kako bi je kasnije mogli iskoristiti), dok je `tekst` naziv za tekstualni sadržaj koji se korisniku ispisuje kao poruka pri unosu varijable. Nakon izvršavanja ove funkcije, program pauzira i čeka unos korisnika kako bi nastavio izvršavanje.

```
ime = input('Unesite ime')
print(ime)
```

Dakle, ukoliko napišemo program iznad, u konzoli će se ispisati tekst *Unesite ime*. U tom trenutku program pauzira do završetka korisničkog unosa. Nakon unosa, program spašava unesenu vrijednost u varijablu sa nazivom `ime`, te ispisuje tu vrijednost korisniku u narednoj liniji. Naravno, program može zahtijevati više unosa od strane korisnika, kao u primjeru ispod.

```
ime = input('Unesite ime')
prezime = input('Unesite prezime')
godine = input('Unesite broj godina')

print('Ti si', ime, prezime)
print('Imas', godine, 'godina').
```

U navedenom primjeru, nakon ispisa svake poruke pri unosu vrijednosti, program pauzira dok korisnik ne unese traženu vrijednost. Vrijednosti se spašavaju u varijable i na kraju se ispisuju korisniku u željenom formatu. Svaki unos, samim tim i varijabla u koju je unos spašen inicijalno se smatra tipom `str`. Međutim, u tom slučaju nije moguće ispisati korisnikovu starost za pet godina (kako string povećati za 5?). U slučaju da želimo neku varijablu pretvoriti u tip `int`, koristimo funkciju `int`, a ukoliko je želimo pretvoriti u decimalan broj, koristimo funkciju `float`. Način upotrebe ovih funkcija bio bi `godine = int(godine)` ili `godine = float(godine)`. U ovom slučaju, proizvoljna vrijednost varijable prebacuje se u odgovarajući tip, te je moguće izvršavati operacije sa tim varijablama.

Koncept *funkcija* bit će razmatran u posebnom poglavlju, a za sada ih možemo smatrati operacijama koje izvršavaju neki zadatak. U ovom slučaju, određenu vrijednost, funkcija prebacuje

u tip koji nam odgovara.

Obratite pažnju da se u varijablama čije vrijednosti pretvaramo u cijeli ili realan broj moraju nalaziti vrijednosti koje je moguće pretvoriti u taj tip jer će u suprotnom doći do greške. Na primjer, komandu `int("Hello")` nije moguće izvršiti.

```
ime = input('Unesite ime')
prezime = input('Unesite prezime')
godine = input('Unesite broj godina')
godine = int(godine)
god_kasnije = godine + 5

print('Zoves se', ime, prezime)
print('Za pet godina ces imati', god_kasnije, 'godina').
```

2.3 Operacije sa numeričkim podacima

Programski jezik Python omogućava obavljanje standardnih **matematičkih operacija** sa brojevima. Osnovne podržane operacije su:

1. Sabiranje (+)
2. Oduzimanje (-)
3. Množenje (*)
4. Dijeljenje (/)
5. Cjelobrojno dijeljenje (//)

Operacije sabiranja, oduzimanja i množenja su standardne operacije koje se izvršavaju kao i na standardnom kalkulatoru, te vraćaju odgovarajući rezultat korisniku. Međutim, možemo uočiti da Python podržava dva tipa dijeljenja. Oba tipa, kao što im ime govori, obavljaju dijeljenje brojeva. Razlika je u činjenici da operator `/` vraća rezultat kao realan broj, dok operator `//` vraća rezultat u obliku cijelog broja. Međutim, jasno je da pri dijeljenju brojeva nije nužno dobijanje cjelobrojnog rezultata (npr. 5 podijeljeno sa 2). Operator cjelobrojnog dijeljenja zaokružuje dobijenu vrijednost na prvi manji cijeli broj. Dakle, `5/2` će biti jednako `2.5`, dok će `5//2` biti jednako `2`, jer je to prvi cijeli broj manji od `2.5`. Ovaj operator se često koristi u manipulacijama sa brojevima, npr. za odbijanje posljednje cifre broja (cjelobrojno dijeljenje sa 10, jer `123//10` kao rezultat vraća 12).

```
x = int(input('Unesite broj'))
y = int(input('Unesite broj'))

zbir = x + y
razlika = x - y
proizvod = x * y
kolicnik1 = x / y
kolicnik2 = x // y
```

U primjeru iznad, za unesene vrijednosti 5 i 2, program u varijable `zbir`, `razlika`, `proizvod`, `kolicnik1`, `kolicnik2` smješta vrijednosti 7, 3, 10, 2.5 i 2, redom. U slučaju unesenih vrijednosti 15 i 3, dobijene su vrijednosti 18, 12, 45, 5 i 5.

Osim navedenih operatora, postoje i operatori stepenovanja (**) i ostatka pri dijeljenju (%). U slučaju naredbe `a**b`, operator vraća vrijednost `b`-tog stepena broja `a`. S druge strane, naredba `a%b` računa ostatak pri dijeljenju broja `a` sa `b`.

```
x = int(input('Unesite broj'))

print(x ** 2)
print(x % 10)
```

U primjeru iznad, za uneseni broj `x=12`, program ispisuje vrijednosti 144 i 2, jer je $12^2 = 144$, te je ostatak pri dijeljenju broja 12 sa 10 jednak 2.

Svaki izraz može sadržavati kompleksniji niz operatora, poput `12 + 2.3 - 15//3 + 10/2 + 12 * 2**2`. Redoslijed izvršavanja operacija poredan po opadajućem prioritetu je:

1. Operator stepenovanja (`**`)
2. Operatori množenja, dijeljenja i ostatka (`* / // %`)
3. Operatori sabiranja i oduzimanja (`+ -`)

Kada se dva operatora sa jednakim prioritetom nalaze uz isti operand, operatori se izvršavaju sa lijeva na desno.

U slučaju potrebe ili sigurnosti u redoslijed izvršavanja operatora, programeru je dozvoljeno grupisati naredbe pomoću zagrada. U programskom jeziku Python, uglaste i vitičaste zagrade imaju specijalna značenja, stoga se za grupisanje koriste samo male zagrade. Zagrade se uparuju, kao najbliža otvorena i najbliža zatvorena. Nakon uparivanja tih zagrada, one se ignoriraju pri narednom uparivanju. Npr. izraz

$$\frac{1}{1+\frac{1}{2+\frac{1}{3+\frac{1}{4}}}},$$

pisao bi se kao `1/(1+1/(2+1/(3+1/4)))`.

2.4 Moduli

U praktičnom radu, često je neophodno računati vrijednosti sinusa, kosinusa, apsolutne vrijednosti, korijena broja, neophodno je zaokružiti broj na najbližu cjelobrojnu vrijednost, kao i niz drugih matematičkih operacija. Svaka od navedenih operacija koristi se na više mjesta u programima, te predstavlja standardnu operaciju koja je neophodna za rad velikog broja programa i koju koristi veliki broj programera.

Procese koji se često ponavljaju u programima, moguće je izdvojiti u Python **module**. Moduli predstavljaju skup alata koje programeri mogu uključiti i koristiti u svojim programima. Svaki od tih alata je uradio tim drugih programera (ili pojedinac). Python sadrži veliki broj modula koji se mogu koristiti za rad sa Excel fajlovima, za rad sa PDF dokumentima, za kreiranje GUI aplikacija, za generisanje slučajnosti, kao i niz drugih. Svi ti alati nisu uključeni u samom početku korištenja, upravo iz razloga što nepotrebno opterećuju programe. Definitivno nam neće u svakom programu čitanje podataka iz Excel dokumenta.

Kompletan koncept modula moguće je porediti sa procesom spremanja za put. Svaka osoba posjeduje odjeću za razne prilike i godišnja doba. Međutim, u trenutku spremanja, osoba uzima samo one komade odjeće koje misli da će koristiti na tom putu. U slučaju da krenete na more, svakako da nije potrebno nositi skijašku opremu. Dakle, kao što za spremanje na put birate odjeću koju ćete nositi, tako pri programiranju možete izabrati module koji će vam biti potrebni za taj program.

2.4.1 Modul `math`

Jedan od često korištenih modula je modul `math`. Ovaj modul sadrži niz matematičkih funkcija koje značajno olakšavaju proces razvoja programa.

Prije upotrebe svake od funkcija koje modul nudi, neophodno je naznačiti interpreteru da će određen modul biti korišten. U programerskom svijetu, proces uključivanja određenih modula se naziva **import**. U slučaju da želimo importati modul `math`, koristimo naredbu `import math`. Jedna od popularnijih funkcija ovog modula je funkcija `sqrt`. Ova funkcija računa kvadratni korijen. Naravno, da bi kvadratni korijen bio izračunat, važno je funkciju obavijestiti o broju čiji se korijen računa. Ukoliko želimo koristeći modul `math` izračunati i ispisati korijen broja, koristimo kod ispod.

```
import math

x = float(input())
korijen = math.sqrt(x)

print(korijen)
```

Obratite pažnju da je neophodno importovati modul prije upotrebe funkcija iz tog modula. U kodu iznad, moguće je uočiti da je na samom početku programa importovan kompletan modul za matematiku. Nakon unosa broja, kreirana je varijabla `korijen` u koju je smještena vrijednost korijena unesenog broja. Za korijen unesenog broja korištena je funkcija `sqrt` koja se nalazi unutar modula `math`, stoga, neophodno je kod poziva funkcije naglasiti iz kojeg importovanog modula istu koristimo.

U primjeru iznad, `math` nije dug naziv, te kvalitetno opisuje svoj sadržaj. Međutim, paketi mogu imati imena koja nisu pogodna za opis sadržaja ili programer jednostavno želi koristiti paket pod drugim imenom. U tom slučaju, koriste se alijasi.

```
import math as m

x = float(input())
korijen = m.sqrt(x)

print(korijen)
```

U istom primjeru, ukoliko modul `math` importujemo, te mu damo alijas `m`, svaki poziv funkcije `sqrt` moguće je izvršavati pomoću `m.sqrt`.

U prethodnim primjerima, osim funkcije `sqrt` importovan je i niz drugih funkcija iz modula. Međutim, u nekim situacijama programeri žele importovati samo određenu funkciju iz kompletnog modula. Ukoliko je to slučaj sa funkcijom `sqrt`, koristi se kod naveden ispod.

```
from math import sqrt

x = float(input())
korijen = sqrt(x)

print(korijen)
```

U ovom slučaju, za funkciju nije neophodno navoditi inicijalni modul pri pozivanju. Dakle, za import možemo koristiti bilo koju od tri opcije navedene ispod.

1. Import kompletnog modula

2. Import kompletnog modula pod alijasom
3. Import samo neophodnih funkcija

Rad pomenute funkcije `sqrt(x)` možemo posmatrati kao dio koda koji, najjednostavnije opisano, svako pojavljivanje zamjenjuje vrijednošću $\sqrt{x}$. Dakle, svaki poziv funkcije `sqrt(4)` se zamjenjuje brojem 2, sa kojim je moguće vršiti standardne operacije kao i sa svakim drugim brojem.

Osim ovih funkcija, pomenimo još nekoliko:

- `math.ceil(x)`. Ova funkcija računa gornji cijeli dio nekog broja. Drugačije rečeno, ova funkcija vraća prvi cijeli broj veći ili jednak broju `x`. Primjer: `math.ceil(2.3)` vraća vrijednost 3, ali `math.ceil(-2.3)` vraća -2.
- `math.floor(x)`. Ova funkcija računa donji cijeli dio nekog broja. Drugačije rečeno, ova funkcija vraća prvi cijeli broj manji ili jednak broju `x`. Primjer: `math.ceil(2.3)` vraća vrijednost 2, ali `math.ceil(-2.3)` vraća -3.
- `math.factorial(x)`. Ova funkcija računa faktorijel proslijeđenog broja. Drugačije rečeno, ova funkcija vraća proizvod svih prirodnih brojeva manjih ili jednakih `x`. Ukoliko nije proslijeđen prirodan broj, funkcija javlja grešku.
- `math.gcd(x,y)`. Ova funkcija vraća NZD (najveći zajednički djelilac) proslijeđenih brojeva.
- `math.radians(x)`. Ova funkcija prima ugao u stepenima, a vraća vrijednost ugla u radijanima.

Osim ovih funkcija, često su korištene funkcije `acos`, `asin`, `atan`, `cos`, `sin`, `tan`. Povratnu vrijednost ovih funkcija nije potrebno dodatno opisivati, a svi detalji mogu se pronaći u Python dokumentaciji.

Osim funkcija, modul sadrži i vrijednosti konstanti π (`math.pi`) i e (`math.e`).

2.5 Zadaci

Zadatak 2.1.

Napisati program koji narednim redoslijedom od korisnika uzima četiri realne vrijednosti: α, n, k i h. Vrijednosti su unesene jedna ispod druge. Program ispisuje rezultat narednog izraza:

$$\frac{2k^{\frac{2}{3}}}{\sin(\alpha)\sqrt{h+n}}\cos(\alpha) \tag{2.1}$$

Input:

```
80
56.21
23.6
2.4
```

Output:

```
0.37899857290382083
```

Zadatak 2.2.

U geometriji, upisana kružnica trougla je najveća kružnica koju sadrži trougao. Kružnica dodiruje sve tri stranice trougla. Poluprečnik upisane kružnice r izražava se kao:

$$r = c\frac{\sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)}{\cos\left(\frac{\gamma}{2}\right)} \tag{2.2}$$

Napisati program koji od korisnika traži unos uglova α i β, te dužinu stranice c trougla, a potom računa i ispisuje vrijednost polupročnika r upisane kružnice. Primjer:

Input:

```
45
60
3.5
```

Output:

```
0.8441333421411773
```

Zadatak 2.3.

Elipsoid je zatvorena centralno-simetrična ploha drugoga reda. U središtu elipsoida sijeku se tri međusobno okomite ose (glavne ose) simetrije. Elipsoid se može parametarizirati na više načina, a najčešće korišten je:

$$x = a\sin(\theta)\cos(\varphi) \tag{2.3}$$
$$y = b\cos(\theta)\sin(\varphi) \tag{2.4}$$
$$z = c\sin(\theta) \tag{2.5}$$

Napisati program koji od korisnika traži parametare a, b, c, θ i φ, a program potom računa i ispisuje vrijednosti koordinata x, y, z. Primjer:

Input:

```
2.2
1.568
4.8
47.21
39.1
```

Output:

```
1.2529009763613002
0.6717726301427799
3.522472503385265
```

Zadatak 2.4.

Napisati program koji pretvara temperaturu iskazanu u stepenima Celzijusa u termperaturu iskazanu u stepenima Fahrenheita. Odnos između dvije temperaturne skale je:

$$F = C \cdot \frac{9}{5} + 32 \tag{2.6}$$

gdje je C temperatura iskazanu u stepenima Celzijusa a F termperatura iskazanu u stepenima Fahrenheita.

Input:

```
29
```

Output:

```
84.2
```

Zadatak 2.5.

Napisati program koji kao ulazne podatke prima dužine dvije stranice trougla (b i c) i ugla između njih (α). Program treba ispisati dužinu treće stranice (a). Možete koristiti formulu:

$$a^2 = b^2 + c^2 - 2bc\cos(\alpha) \tag{2.7}$$

Input:	Output:
30 40 60	36.05551275463989

Zadatak 2.6.

Napisati program koji izračunava zapreminu torusa:

$$V = 2\pi^2 R r^2 \tag{2.8}$$

gdje je R udaljenost od centra cijevi do središta torusa, a r je polupreč nik cijevi. Program od korisnika traži unos ove dvije vrijednosti, a ispisuje zapreminu. Za vrijednost π potrebno je koristiti konstantu `pi` iz biblioteke `math`.

Input:	Output:
3 0.5	14.804406601634037

Zadatak 2.7.

Kartezijeve koordinate x i y se mogu pretvoriti u polarne koordinate r i φ koristeći:

$$r = \sqrt{x^2 + y^2} \tag{2.9}$$

$$\varphi = \arctan\frac{y}{x} \tag{2.10}$$

Napisati program koji od korisnika traži unos vrijednosti za x i y, a ispisuje r i φ na zasebnim linijama. Za izračun funkcije arctan možete koristiti Pythonovu funkciju `arctan2(y, x)` koja prima dva parametra (brojnik i nazivnik) umjesto razlomka. Također, ugao φ je potrebno ispisati u stepenima a ne u radijanima za šta možete koristiti funkciju `math.degrees()`.

Input:	Output:
5 6	7.810249675906654 50.19442890773481

Zadatak 2.8.

Pomoću vrijednosti izmjerenih prilikom kosog hica moguće je izračunati ubrzanje Zemljine teže g koristeći formulu:

$$g = \frac{{v_0}^2}{R}\sin(2\alpha) \tag{2.11}$$

gdje je v_0 početno ubrzanje, R raspon, a α ugao izbačaja.

Napisati program koji od korisnika zahtijeva unos vrijednosti u redoslijedu v_0, R i α, a ispisuje ubrzanje Zemljine teže izračunato pomoću formule. Također, potrebno je uzeti u obzir da će korisnik unijeti ugao u stepenima, a formula koristi radijane.

Input:

```
11.1
12.5
45
```

Output:

```
9.8568
```

Zadatak 2.9.

Prilikom simulacije kosog hica potrebno je izračunati početnu brzinu objekta. Ukoliko je data trenutna pozicija objekta (x, y) i ugao izbačaja α, početna brzina se računa koristeći formulu:

$$v_0 = \sqrt{\frac{x^2 g}{x \sin(2\alpha) - 2y\cos^2\alpha}} \tag{2.12}$$

gdje je g ubrzanje Zemljine teže i ima vrijednost 9.81.

Napisati program koji od korisnika zahtijeva unos vrijednosti u redoslijedu x, y i α, a ispisuje početnu brzinu izračunatu pomoću formule. Također, potrebno je uzeti u obzir da će korisnik unijeti ugao u stepenima, a formula koristi radijane.

Input:

```
1.5
2.5
62.1
```

Output:

```
12.30278283909394
```

Zadatak 2.10.

Ultrasonični mjerač protoka mjeri brzinu tečnosti koristeći ultrazvuk da izmjeri volumen toka.

Koristeći jedno vrijeme prolaska (t_u), raspon između izlaznog i ulaznog pretvarača (L), ugla uspona (θ) i prosječne brzine tečnosti (V_a) moguće je izračunati drugo vrijeme prolaska (t_d) koristeći formulu:

$$t_d = \frac{1}{\frac{2V_a\cos\theta}{L} + \frac{1}{t_u}} \tag{2.13}$$

Napisati program koji od korisnika zahtijeva unos vrijednosti u redoslijedu V_a, θ, L i t_u, a ispisuje vrijeme prolaska t_d. Također, potrebno je uzeti u obzir da će korisnik unijeti ugao u stepenima, a formula koristi radijane. Primjer:

Input:

```
2.2
30.5
0.5
1.2
```

Output:

```
0.11882595283672398
```

3. Naredbe grananja

Rezultat izvršavanja kompletnog programa u ranijim poglavljima ovisi o unosu korisnika, međutim, redoslijed izvršavanja naredbi je poznat prije početka programa i programer sa trenutno poznatim alatima nije u mogućnosti vršiti njegovu kontrolu. Svaki program se izvršavao naredbu po naredbu, te naredba koja se izvršava ne ovisi striktno od ranijeg izvršavanja. Međutim, stvarne situacije zahtijevaju kompleksnije strukture. Ljudi svakodnevno donose niz odluka i prilagođavaju svoje postupke situacijama u kojima su se našli, te je te situacije neophodno modelirati u programski kod.

Proces izvršavanja koda možemo zamisliti kao trenutak buđenja osobe. Iako je osoba mogla donijeti odluku o tome šta će obući veče prije, ispravnost te odluke ovisi o vremenu vani tokom jutra. Stoga, ispravna odluka nije mogla biti donesena veče prije. S druge strane, osoba je mogla ranije planirati da će ustati u 8 sati i da će popiti kafu i doručkovati u kući prije izlaska. Sve situacije koje je moguće planirati prije buđenja, možemo smatrati situacijama koje možemo riješiti bez trenutnog donošenja odluka, međutim, niz koraka koji ovisi o situaciji koja se dogodila neposredno prije, ne možemo.

Zamislimo proizvoljnu aplikaciju koja zahtijeva prijavu korisnika. Unosi se korisničko ime i lozinka. Na osnovu ispravnosti podataka, program će izvršiti jednu od dvije moguće opcije: odbiti prijavu korisnika zbog neispravnosti podataka ili dozvoliti korisniku da nastavi koristiti aplikaciju i prikazati mu podatke specifične za njega (npr. listu njegovih poruka). Sličnih primjera je mnogo, te današnji IT svijet nije zamisliv bez kontrole toka.

Kako je već opisano, dosadašnje strukture koda ne dozvoljavaju **kontrolu toka**, te se program izvršava **sekvencijalno**, narednu po naredbu. Na osnovu pomenutog motiva, uvode se **naredbe grananja**. Grananje programa može se izvršiti na četiri načina:

1. Ukoliko je ispunjen uslov, uradi nešto.
2. Ukoliko je ispunjen uslov, uradi nešto, inače uradi nešto drugo.
3. Ukoliko je ispunjen prvi uslov, uradi prvi niz naredbi, ukoliko je ispunjen drugi uslov, uradi drugi niz naredbi . . . , ukoliko je ispunjen n-ti uslov, uradi n-ti niz naredbi.
4. Ukoliko je ispunjen prvi uslov, uradi prvi niz naredbi, ukoliko je ispunjen drugi uslov, uradi nešto drugo . . . , ukoliko je ispunjen n-ti uslov, uradi n-ti niz naredbi. Ukoliko nije ispunjen

nijedan od pomenutih uslova, uradi nešto.

Naredbe grananja spadaju u grupu programerskih alata koji su intuitivnu i jednostavni za razumijevanje, jer ih osoba svakodnevno koristi u stvarnom životu, iako toga ne mora biti svjesna. Stoga, svaku od ranijih mogućnosti ćemo detaljnije pojasniti kroz primjere.

Primjer 3.1 Napisati program koji od korisnika traži da unese cijeli broj. Ispisati poruku samo u slučaju da je uneseni broj pozitivan.

```
x = int(input())
if x > 0:
   print("Broj je pozitivan!")
```

U sklopu `if` naredbe neophodno je smjestiti **uslov** (u ovom slučaju, `x>0`), te na kraju uslova postaviti dvotačku. Blok koji slijedi može sadržavati proizvoljan broj naredbi, te se izvršava samo u slučaju da je uslov ispunjen. ■

Primjer 3.2 Napisati program koji od korisnika traži da unese cijeli broj. Provjeriti da li je uneseni broj pozitivan, te ispisati prikladnu poruku.

```
x = int(input())
if x > 0:
   print("Broj je pozitivan!")
else:
   print("Broj nije pozitivan!")
```

Nakon `if` naredbe, te bloka koji se izvrši u slučaju da je uslov ispunjen, dozvoljeno je dodati opciju `else`, koja se izvršava samo u slučaju da uslov nije ispunjen. Navedena struktura se koristi u slučaju da imamo *binarnu* odluku, odluku sa dva izbora. Nakon naredbe `else`, dozvoljeno je navesti proizvoljan blok naredbi. Kao što je vidljivo, uslov gdje provjeravamo da je `x` pozitivno, pišemo jednostavno kao `x>0`. ■

Primjer 3.3 Napisati program koji od korisnika traži da unese broj. Ukoliko je uneseni broj jednak 1, program ispisuje "Ponedjeljak", ukoliko je uneseni broj jednak 2, program ispisuje "Utorak", . . . , ukoliko je uneseni broj jednak 7, program ispisuje "Nedjelja".

```
x = int(input())
if x == 1:
   print("Ponedjeljak")
elif x == 2:
   print("Utorak")
elif x == 3:
   print("Srijeda")
elif x == 4:
   print("Cetvrtak")
elif x == 5:
   print("Petak")
elif x == 6:
   print("Subota")
```

```
elif x == 7:
   print("Nedjelja")

print("Nastavak")
```

U slučaju da želimo provjeriti više različitih uslova, te na osnovu zadovoljenja nekog od njih provesti određen niz naredbi, koristi se struktura `if-elif`, gdje se `if` koristi za provjeru inicijalnog uslova (prvog), te za svaki naredni koristimo naredbu `elif` (nastala kombinacijom riječi *else* i *if*). Nakon svake `elif` naredbe, neophodno je smjestiti uslov koji se provjerava. Uslovi se provjeravaju redom, te u slučaju ispunjenja bilo kojeg od njih, izvršavanje se nastavlja nakon kompletne if-else strukture (ispisuje se tekst `Nastavak`).

Kao što je moguće uočiti, poređenje u programskom jeziku Python (kao i u velikom broju drugih jezika) navodi se pomoću dvostruke jednakosti. Razlog za ovo jeste činjenica da je znak jednako iskorišten za dodjelu vrijednosti, te u slučaju da je uslov napisan kao `x=5`, ovo bi varijabli `x` dodijelilo vrijednost 5, te tok izvršavanja ne bi bio onaj očekivani. Iako programski jezik Python upozorava korisnike na navedene greške, postoji niz programskih jezika gdje bi ovakve naredbe uspješno prošle, ali tok izvršavanja programa ne bi bio uspješan i očekivan. ■

Primjer 3.4 Napisati program koji od korisnika traži da unese broj. Ukoliko je uneseni broj jednak 1, program ispisuje "Ponedjeljak", ukoliko je uneseni broj jednak 2, program ispisuje "Utorak", . . . , ukoliko je uneseni broj jednak 7, program ispisuje "Nedjelja". Ukoliko korisnik ne unese nijedan od ovih brojeva, program ispisuje prikladnu poruku.

```
x = int(input())
if x == 1:
   print("Ponedjeljak")
elif x == 2:
   print("Utorak")
elif x == 3:
   print("Srijeda")
elif x == 4:
   print("Cetvrtak")
elif x == 5:
   print("Petak")
elif x == 6:
   print("Subota")
elif x == 7:
   print("Nedjelja")
else:
   print("Pogresan unos!")
```

Naredba `else` može se smjestiti na kraju bilo koje `if-elif` strukture, te se blok naredbi nakon `else` izvršava samo u slučaju da nijedan od ranijih uslova nije bio ispunjen. Ova se naredba često koristi u slučaju da želimo ispisati poruku u slučaju neispravnog i neočekivanog unosa korisnika. ■

3.1 Uslovi

Pomenuti uslovi u sklopu `if-elif-else` kontrole toka, mogu biti znatno kompleksniji od ranije pokazanih. Osnovni matematički operatori dozvoljeni u programskom jeziku Python su:

- Jednako (==). Primjer upotrebe: `a == b`.
- Različito (!=). Primjer upotrebe: `a != b`.
- Veće (>). Primjer upotrebe: `a > b`.
- Veće ili jednako (>=). Primjer upotrebe: `a >= b`.
- Manje (<). Primjer upotrebe: `a < b`.
- Manje ili jednako (<=). Primjer upotrebe: `a <= b`.

Svako od ovih poređenja može biti tačno ili netačno. Programerski, svako poređenje vraća jednu od dvije vrijednosti `True` (tačno) ili `False` (netačno). Prva vrijednost označava matematički ispravnu tvrdnju, dok druga predstavlja matematički neispravnu tvrdnju.

Često je uslove neophodno kombinovati, te umjesto provjere jednog uslova, program zahtijeva provjeru više uslova.

Primjer 3.5 Za uneseni prirodan broj, provjeriti da li je on pozitivan i paran.

```
x = int(input())
if x > 0 and x % 2 == 0:
   print("Broj je paran i pozitivan!")
else:
   print("Broj nije paran i pozitivan!")
```

Kao što je moguće uočiti u kodu, neophodno je provjeriti kompleksniji uslov koji se sastoji od dva poduslova (broj je pozitivan, broj je paran). Da bi kompletan uslov bio zadovoljen, svaki od poduslova mora biti ispunjen (tačan, `True`). Stoga, koristimo operator `and` koji predstavlja matematičku konjukciju (logičko *i*, $\wedge$).

Slično, u slučaju da želimo provjeriti da li je bar jedan uslov zadovoljen, koristimo ključnu riječ `or`. Uslov `x%2==0 or x >0` će biti tačan ukoliko je `x` pozitivno ili ukoliko je `x` parno, dovoljno je da bude ispunjen ili jedan ili oba uslova. Ovaj operator predstavlja matematičku disjunkciju (logičko *ili*, $\vee$). Treći logički operator je negacija (`not`). Ovaj operator negira navedenu vrijednost. Dakle, `not x == 5` će biti tačno u slučaju da je `x` jednako 5, a inače će biti netačno. Proizvoljan broj operatora se može kombinovati u kreiranju uslova. ■

3.2 Primjeri

U nastavku su analizirana još tri primjera koji kombinuju prethodno prezentovane tehnike.

Primjer 3.6 Napisati program koji od korisnika traži da unese dva prirodna broja koji predstavljaju broj kolone i reda na šahovskoj ploči. Ukoliko je gornji lijevi ugao bijelo polje, te ima koordinate (1,1), a donje desno polje ima koordinate (8,8), za unesene vrijednosti odrediti da li predstavljaju crno ili bijelo polje na ploči.

```
x = int(input())
y = int(input())

if x < 1 or y < 1 or x > 8 or y > 8:
   print("Unos nije ispravan!")
```

```
else:
   if (x + y) % 2 == 0:
      print("Polje je bijelo!")
   else:
      print("Polje je crno!")
```

Navedeni zadatak koristi dva važna elementa. Prvi važan element jeste provjera u sklopu prve `if` naredbe. Provjerava se da li je bilo koja vrijednost `x` ili `y` izašla iz predviđenog opsega (opseg prirodnih brojeva od 1 do 8). U slučaju da bilo koja vrijednost nije ispravna (dovoljno je da jedna nije ispravna, stoga `or`), program ne treba određivati boju unesenog polja. U slučaju da su obje vrijednosti ispravno unesene, dolazimo do narednog važnog elementa **ugniježdene naredbe grananja**. Kako je ranije spomenuto, svaki blok može sadržavati proizvoljan niz naredbi, pa tako taj niz naredbi može biti sačinjen od novih naredbi grananja. Detaljnim posmatranjem polja i odgovarajućih koordinata, možemo primijetiti da je zbir koordinata bijelih polja paran, dok je za crna polja zbir neparan. Stoga, dovoljno je provjeriti parnost izraza $x+y$ da bi odredili boju polja. ■

Primjer 3.7 Napisati program koji od korisnika traži da unese trocifren prirodan broj. Program treba provjeriti da li je uneseni broj palindrom. Za broj kažemo da je palindrom, ukoliko se jednako zapisuje sa lijeve i desne strane.

```
x = int(input())

if x < 100 or x > 999:
   print("Unos nije ispravan!")
else:
   c = x % 10
   x = x // 10

   b = x % 10
   x = x // 10

   a = x % 10

   if a == c:
      print("Palindrom!")
   else:
      print("Broj nije palindrom!")
```

Napisani kod se sastoji također od dva važna elementa. Prvi element je provjera da li je uneseni broj u traženom opsegu. U slučaju da je broj veći od 999 ili manji od 100, on nije trocifren i program prekida sa izvršavanjem. U slučaju da je broj ispravno unesen, znamo da je `x` trocifren prirodan broj.

Da bi provjerili da li je uneseni broj palindrom, od velikog bi značaja bilo izdvajanje pojedinačnih cifara unesenog broja u varijable. Matematički gledano, posljednja cifra nekog broja je ostatak tog broja pri dijeljenju sa 10. Stoga, da bi dobili posljednju cifru broja, koristimo operator `%` koji vraća upravo ostatak. Time izdvajamo cifru jedinica, te je smjestimo u varijablu `c`.

Da bi dobili narednu cifru (cifru desetica), problem nam stvara posljednja cifra broja (napomena, ostatak pri dijeljenju sa 100 ne daje očekivani rezultat, provjerite zašto!). Stoga, koristi se ranije pomenuti trik sa odbijanjem posljednje cifre. Ukoliko iskoristimo cjelobrojno dijeljenje sa 10, postižemo efekat odbijanja posljednje cifre broja (npr. `123//10` je `12`, jer rezultat 12,3 zaokružimo). Dobijenu vrijednost smjestimo u varijablu `x`, te postupak ponovimo za dvocifreni i kasnije jednocifreni broj (s tim da nema potrebe za odbijanjem cifre jednocifrenog broja). Time u varijable `a`, `b` i `c` dobijamo redom cifre stotica, desetica i jedinica unesenog broja. Da bi broj bio palindrom, vrijednost `b` možemo zanemariti, te je neophodno da vrijednosti `a` i `c` budu jednake. Isti postupak se koristi za brojeve sa proizvoljnim brojem cifara, u što ćemo se uvjeriti kasnije! ■

Primjer 3.8 Za tri unesena realna broja koji predstavljaju stranice trougla `a`, `b` i `c`, provjeriti da li je trougao pravougli. Pretpostaviti da stranice čine trougao i da je `c` najduža stranica.

```
a = float(input())
b = float(input())
c = float(input())

# standardni pokusaj
if c*c==a*a + b*b:
   print("Pravougli")
else:
   print("Nije pravougli")

epsilon = 0.00000000001
x = c*c
y = a*a + b*b
if x-y < epsilon and y-x < epsilon:
   print("Pravougli")
else:
   print("Nije pravougli")
```

U naizgled trivijalnom primjeru gdje se koristi Pitagorina teorema za provjeru da li vrijedi relacija $a^2+b^2=c^2$, programeri početnici mogu susresti neočekivane rezultate. Naime, realni brojevi se u računarima ne mogu predstaviti sa potpunom **preciznošću**, nego dolazi do zaokruživanja nakon određene decimale. Stoga, pri poređenju dva naizgled jednaka realna broja, može doći do neslaganja u zaokruživanju na k-toj decimali, gdje je k neki prirodan broj. U slučaju da u program iznad unesemo vrijednosti 3,4 i 5, program će raditi ispravno. Međutim, ukoliko za brojeve unesemo vrijednosti $0,0000000003$, $0,0000000004$ i $0,0000000005$, može doći do greške u zaokruživanju (ukoliko ne dođe, dodajte još nekoliko nula). Stoga, prvi način poređenja nije ispravan za slučaj realnih brojeva i ne može garantovati sigurnost u poređenju.

Međutim, programeri su pronašli način da riješe ovaj problem. Iako Python nudi više mogućih rješenja, u kodu iznad je navedeno rješenje koje je upotrebljivo u dosta programskih jezika i ne zahtijeva korištenje funkcija iz Python modula. Osnovna ideja jeste provjeriti da li su brojevi koji se porede **dovoljno blizu**. U slučaju da se brojevi razlikuju za određenu vrijednost koja je manja od proizvoljno malog pozitivnog realnog broja, smatramo da su oni jednaki. Tu proizvoljno malu vrijednost najčešće nazivamo `epsilon`. Dakle, ideja je da provjerimo da je

apsolutna vrijednost razlike brojeva manja od `epsilon`. Matematički rečeno, provjeravamo da li je $|a-b| < \varepsilon$, gdje je ε upravo proizvoljno mala pozitivna vrijednost. Kako ne koristimo funkciju za apsolutnu vrijednost, dovoljno je provjeriti da li je

$$-\varepsilon < a-b < \varepsilon.$$

Ovo možemo pisati kao

$$a-b < \varepsilon \wedge b-a < \varepsilon.$$

■

3.3 Zadaci

Zadatak 3.1.

Napisati program koji od korisnika traži da unese ostvarene bodove iz predmeta *Programiranje*. Korisnik unosi broj osvojenih bodova na vježbama, broj osvojenih bodova na parcijalnim ispitima, kao i na završnom. Program korisniku ispisuje konačnu ocjenu. Koristi se zakonska skala za ocjenjivanje: 95-100 je ocjena 10, 85 - 94 je ocjena 9, 75 - 84 je ocjena 8, 65 - 74 je ocjena 7, te 55 - 64 je ocjena 6. U slučaju ostvarenog manjeg broja bodova, student nije položio predmet, te upisuje ocjenu 5.

Input:	Output:
`96`	`10`

Zadatak 3.2.

Funkcija f definiše se po segmentima sa

$$f(x) = \begin{cases} -1 & x \leq -2 \\ \frac{1}{3+\frac{2}{x}} & 0 < x \leq 100 \\ 200 & x \geq 105 \\ 0 & \text{inače.} \end{cases}$$

Napisati program koji će za datu vrijednost promjenjive x izračunati vrijednost funkcije $f(x)$.

Input:	Output:
`20`	`0.3225806451612903`

Zadatak 3.3.

Napisati program koji od korisnika traži da unese trocifren broj. Program provjerava da li je uneseni broj trocifren, te ukoliko jeste, računa i ispisuje proizvod njegovih cifara. Ukoliko broj nije trocifren program ispisuje poruku: `Pogresan unos!`.

Input:	Output:
`123`	`6`

Zadatak 3.4.

Napisati program koji od korisnika traži da unese koordinate `x` i `y` centra kruga, te poluprečnik kruga. Program zatim traži unos dvije koordinate koje predstavljaju tačku A. Program provjerava da li unesena tačka A pripada unutrašnjosti kruga, da li se nalazi na kružnici ili se nalazi van kružnice, te na osnovu toga ispisuje poruke: `Tacka pripada unutrasnjosti kruga`, `Tacka pripada kruznici` ili `Tacka ne pripada krugu`.

Input:

```
0
0
1
0.5
0.5
```

Output:

```
Tacka pripada unutrasnjosti kruga
```

Zadatak 3.5.

Napisati program koji od korisnika traži da unese petocifren broj. Program provjerava da li je uneseni broj petocifren, te ukoliko jeste, računa i ispisuje razliku njegove najveće i najmanje cifre. Ukoliko broj nije petocifren program ispisuje poruku: `Pogresan unos!`.

Input:

```
12321
```

Output:

```
2
```

Zadatak 3.6.

Napisati program koji od korisnika traži da unese x i y koordinate centara za dva kruga u dvodimenzionalnoj ravni. Koordinate se unose jedna ispod druge. Korisnik unosi i dužinu poluprečnika prvog kruga. *Poluprečnik* prvog kruga jednak je *prečniku* drugog kruga. Provjeriti da li se uneseni krugovi sijeku, te shodno ispisati jednu od naredne tri poruke: `Kruznice se sijeku`, `Kruznice se dodiruju` ili `Kruznice nemaju zajednickih tacaka`.

Input:

```
0
0
0
1
1
```

Output:

```
Kruznice se sijeku
```

4. Petlje

Elementi sa kojima smo se upoznali u prethodnim poglavljima omogućavaju rješavanje velikog broja jednostavnih problema. Međutim, rješavanje svakog od narednih nekoliko uobičajenih problema izaziva niz poteškoća.

1. Napisati program koji ispisuje 10 puta poruku `Hello world!`
2. Napisati program koji ispisuje 1000 puta poruku `Hello world!`
3. Napisati program koji ispisuje prvih 1000 prirodnih brojeva, jedan ispod drugog.
4. Napisati program koji ispisuje prvih n prirodnih brojeva (jedan ispod drugog), gdje je n broj kojeg unosi korisnik.

Prvi program moguće je riješiti korištenjem jednostavne naredbe `print`. Istina, naredbu `print` neophodno je napisati 10 puta, ali ovaj problem svakako nije nešto što nije moguće riješiti relativno brzo. Međutim, već kod drugog problema, pri ispisu poruke 1000 puta, može se uočiti mana u ranije opisanom pristupu i ograničenja u gradivu iz prva tri poglavlja. Međutim, i ovaj problem je moguće riješiti koristeći samo naredbu `print` (uz više uloženog truda i vremena). Treći problem je dodatno otežan time što nemamo jednake ispise (pa opcija *copy-paste* nije od pomoći). U trećem problemu, posebno je zanimljiva činjenica da je pravilo ispisa izuzetno jednostavno uočiti (svaki put ispiši broj povećan za jedan ili drugačije rečeno, u i-tom redu ispiši broj i). Bez obzira na sve navedeno, koristeći alat koji je na raspolaganju iz prethodnih poglavlja, i ovaj zadatak je moguće riješiti, iako to zahtijeva dosta više uloženog truda i vremena. Međutim, četvrti problem je nemoguće riješiti bez uvođenja dodatnih alata. Od korisnika se zahtijeva unos broja. Koristeći samo `print` i `if` naredbe, možemo mijenjati broj ispisa prije početka izvršavanja koda. U trenutku kada je tačan broj ispisa određen (što se desi nakon unosa korisnika), ne postoji način da utičemo na broj ispisa. Jedini način jeste upotreba `if` naredbe, gdje bi kreirali pojedinačne ispise ukoliko je uneseni broj 1, ukoliko je uneseni broj 2 ... Postavlja se niz razumnih pitanja: gdje stati, je li moguće pokriti baš sve unose korisnika, koliko bi nam vremena trebalo za pisanje takvog koda? Dakle, javlja se potreba za uvođenjem novih alata koji se zovu **naredbe ponavljanja** (često se koristi i naziv **petlje**).

Postoje dva tipa petlji:

- petlje kontrolisane uslovom (`while` petlja)
- petlje kontrolisane brojačem (`for` petlja)

Prvi tip petlji koristi se u slučaju da tačan broj ponavljanja nije poznat i utvrđen. Primjer upotrebe ovog tipa petlji je naredba *hodaj dok ne dođeš do zida*. Tačan broj koraka koje robot treba izvršiti nije poznat, međutim, poznat je uslov koji je dovoljan da robot nastavi kretanje (*nije došao do zida*). Drugi tip petlji koristi se u slučaju da je broj koraka unaprijed poznat. U slučaju robota, to bi bio uslov *hodaj 100 koraka*. Ovaj tip petlji prikladan je i u primjeru gdje ispisujemo prvih n prirodnih brojeva.

Zamislimo ponovo primjer u kojem smo robotu dali naredbu *hodaj 100 koraka*. Međutim, u toku šetnje robot može naići na prepreku koja bi trebala zaustaviti kretanje. Identična situacija se susreće i u radu sa petljama. Naredba `break` prekida petlju u kojoj se nalazi u trenutku kada program izvrši tu liniju. Naredba `break` koristi se i u radu sa `while` i `for` petljama. Naredni koraci se ne izvršavaju, a program nastavlja rad od prve linije nakon petlje.

4.1 `while` petlja

Kako je ranije navedeno, ovaj tip petlji se izvršava sve dok je uslov tačan. U trenutku kada uslov petlje postaje netačan, petlja prestaje sa radom.

Osnovna struktura `while` petlje navedena je ispod:

```
while uslov:
   naredba_1
   naredba_2
   naredba_3
```

Naredbe se izvršavaju u ciklusima, sve dok je navedeni uslov zadovoljen. Redoslijed izvršavanja varijabli je: `naredba_1`, `naredba_2`, `naredba_3`, `naredba_1`, `naredba_2`, `naredba_3`, itd. Nakon svakog ciklusa, tj. završetka posljednje naredbe u bloku, uslov se ponovo testira i donosi se odluka da li će se izvršiti novi ciklus.

Posljednji primjer iz uvoda, koji traži ispisivanje prvih n prirodnih brojeva, pomoću `while` petlji riješio bi se na način naveden ispod.

```
n = int(input())

brojac = 1
while brojac <= n:
   print(brojac)
   brojac += 1 # kraci zapis za: brojac = brojac + 1
print("NASTAVAK")
```

U rješenje je uveden brojač koraka (varijabla `brojac`), koji se nakon ispisa svakog broja povećava za jedan. Uočimo da je naredba `brojac = brojac + 1` kraće zapisana kao `brojac += 1`. Ove dvije naredbe imaju potpuno jednaku funkcionalnost, ali se u programskom jeziku Python upotreba osnovnih aritmetičkih operatora može pojednostaviti ukoliko se odnose na istu varijablu. Isto vrijedi i za operatore oduzimanja (`-=`), množenja (`*=`), stepenovanja (`**=`), dijeljenja (`/=`), cjelobrojnog dijeljenja (`//=`) i ostatka (`%=`). Ovakvi operatori spadaju u operatore dodjele.

U trenutku kada brojač prestigne vrijednost inicijalnog unosa (vrijednosti `n`), petlja staje sa radom i program se nastavlja od prvog ispisa nakon petlje (*NASTAVAK*).

Primjer 4.1 Napisati program koji od korisnika traži unos brojeva sve dok ne unese nulu. Program ispisuje zbir svih unesenih brojeva.

```
suma = 0
while True:
   n = int(input())
   if n == 0:
      break
   suma = suma + n

print(suma)
```

Kreirana je varijabla `suma` koja čuva sumu svih brojeva koji su uneseni do nekog trenutka. Ta varijabla u početku čuva sumu 0. Pomoću `while True` petlje, kreirana je petlja koja će se naizgled izvršavati beskonačan broj koraka (sve dok je uslov tačan, a on je uvijek tačan). Međutim, nakon unosa svakog broja, radi se provjera je li uneseni broj jednak nuli. U trenutku kada korisnik unese nulu, petlja prekida izvršavanje zahvaljujući naredbi `break` i ispisuje se suma. Ukoliko uneseni broj nije jednak nuli, `suma` se osvježava i nadopunjuje za novi uneseni broj. ■

Iako je standardni način rada sa petljama takav da se kreira uslov koji nije `while True`, ovaj način je prikladan za situacije u kojima bi trebali imati inicijalni unos za ulazak u petlju (kako provjeriti je li unos jednak nuli, ukoliko korisnik nije još unosio brojeve?). Važno je napomenuti da se u rješenjima ovog tipa često javljaju **beskonačne petlje**. Beskonačne petlje su petlje kod kojih uslov zaustavljanja nikada ne bude zadovoljen i nikada se ne izvršava naredba `break`. Primjer takve petlje bi bio kod u kojem smo uklonili `if` naredbu iz prethodnog zadatka. Tada bi korisnik unosio brojeve do beskonačnosti i suma bi se osvježavala do beskonačnosti.

Primjer 4.2 Drugi način za rješenje zadatka ne koristi `break` i provjeru unutar petlje, ali zahtijeva jedan inicijali unos prije ulaska u petlju.

```
n = int(input())
suma = n

while n != 0:
   n = int(input())
   suma = suma + n

print(suma)
```

U primjeru iznad, korisnik unosi broj koji se spašava u varijablu `suma`. Petlja provjerava uslov da li je uneseni broj jednak nuli. U svakom ciklusu petlje, korisnik unosi broj koji se dodaje na raniju sumu. U slučaju da je uneseni broj jednak nuli, uslov nije zadovoljen, petlja se prekida i ispisuje se vrijednost sume. ■

4.2 `for` petlja

Drugi tip petlji koristi se, kako je ranije navedeno, u situacijama u kojima je broj koraka unaprijed određen (prije početka petlje). Za broj koraka kažemo da je određen ukoliko je on jednak broju ili zavisi od ranije poznate vrijednosti varijable (robotu možemo reći da hoda 100 ili n koraka).

Osnovna struktura `for` petlje navedena je ispod:

```
for i in [v1, v2, ...]:
   naredba_1
   naredba_2
   naredba_3
```

Ova petlja će za vrijednost varijable `i` uzimati redom vrijednosti navedene u uglastim zagradama (`v1`, `v2` ...). Za svaku od tih vrijednosti izvršavaju se naredbe `naredba_1`, `naredba_2` i `naredba_3`, redom.

4.2.1 Funkcija `range`

Za prolazak kroz niz cijelih brojeva, najčešće se koristi funkcija `range` u obliku `range(a, b, c)`, gdje `a` označava početnu vrijednost, `b` označava krajnju vrijednost, te `c` označava korak **iteracije** tj. ponavljanja. Sva tri broja, `a`, `b` i `c` predstavljaju cjelobrojne vrijednosti, te `c` ne može biti nula. Krajnja vrijednost se ne dostiže.

Neka je pozvana funkcija `range(1, 11, 2)`. Ova funkcija vraća brojeve 1, 3, 5, 7, 9. Ukoliko treća vrijednost nije definisana, uzima se vrijednost 1. Dakle, `range(1, 5)` vraća vrijednosti 1, 2, 3 i 4.

Često je neophodno dobiti vrijednosti poredane od veće prema manjoj. Tada se za korak koristi vrijednost -1 ili neka druga negativna vrijednost. Npr. `range(5, 1, -1)` vraća vrijednosti 5, 4, 3 i 2.

U slučajevima korištenja funkcije `range` u sklopu petlje, brojač petlje poprima u svakom koraku jednu od vraćenih vrijednosti funkcijom `range` (u prvom koraku prvu vraćenu vrijednost, u drugom drugu ...). Primjer sa ispisom prvih n prirodnih brojeva, riješen pomoću `for` petlje, naveden je ispod.

```
n = int(input())

for i in range(1, n+1):
   print(i)
```

Brojač `i` poprima vrijednosti iz raspona od 1 do n, te u svakom koraku program ispisuje upravo vrijednost brojača. Obratite pažnju da zbog činjenice da `range` funkcija ne vraća krajnju vrijednost, završni argument funkcije je postavljen na vrijednost za jedan veću od stvarne vrijednosti koja je potrebna.

Napomena. `for` i `while` petlje se mogu simulirati jedna pomoću druge (razmislite kako, inspiraciju možete pronaći upravu u rješenjima primjera sa ispisom prvih n brojeva).

4.3 Ugniježdene petlje

Naredbe unutar bloka svake petlje mogu biti sačinjene od kompleksnog niza naredbi. Stoga, unutar svake petlje, moguće je imati drugu petlju (kao što je unutar `if`-a moguće imati naredbe grananja). Petlje unutar petlji se nazivaju **ugniježdene petlje** (engl. *nested loops*).

Zanimljiv primjer ovakvih petlji može biti sat. Za svaki od 24 sata dnevno imamo 60 minuta. Za svaku od 60 minuta, sat otkuca 60 sekundi. Ukoliko želimo ispisati sve moguće sate, minute

i sekunde unutar jednog dana, koristimo ugniježdene petlje, gdje za svaki sat i svaku minutu ispisujemo svaku sekundu. Najjednostavnija verzija ovog koda bi izgledala kao u primjeru ispod.

```
for h in range(0, 24):
   for m in range(0, 60):
      for s in range(0, 60):
         print(h, ':', m, ':', s)
```

Kao što se može uočiti, brojači mogu poprimiti naziv proizvoljne varijable, te je važno da u ugniježdenim petljama ne dolazi do duplog imenovanja varijabli (npr. dva puta brojač imenovati sa `i`). Osim toga, brojač za sate ide od 0 do 24 (gdje 24 nije uključeno), što predstavlja sate u danu. Analogno vrijedi i za ostale brojače.

Kombinacija petlji unutar petlji često se koristi pri raznim geometrijskim ispisima (popularna piramida, romb i drugi).

Osim zanimljive naredbe `break` koja prekida kompletnu petlju, može se desiti da u bloku naredbi nekada nije potrebno izvršiti sve naredbe do kraja. U takvim situacijama, koristi se naredba `continue`. Ova naredba ne prekida kompletnu petlju, nego prekida izvršavanje bloka naredbi unutar petlje (prekida jedan ciklus), te se petlja nastavlja od narednog koraka (ponaša se kao da je završen kompletan niz naredbi). Naredba `continue` može se koristiti u svakom dijelu bloka, najčešće unutar `if-elif-else` struktura.

4.4 Zadaci

Zadatak 4.1.

Korisnik prvo unosi cijeli pozitivan broj n, a zatim n riječi jednu ispod druge. Program treba ispisati `jeste` ukoliko je korisnik upisao riječ `programiranje` tačno dva puta, a u svakom drugom slučaju program ispisuje `nije`. U primjeru ispod korisnik prvo unosi vrijednost 6, a nakon toga 6 riječi. Pošto se riječ `programiranje` ponavlja 3 puta program ispisuje `nije`.

Input:

```
6
programiranje
matematika
programiranje
programiranje
pmf
tekst
```

Output:

```
nije
```

Zadatak 4.2.

Korisnik unosi pozitivne cijele brojeve jedan ispod drugog. Kraj unosa korisnik označava sa `-1`. Program ispisuje sumu koju dobije tako što sabira posljednje cifre za jednocifrene brojeve i predzadnje cifre za brojeve koji imaju više od jedne cifre. U narednom primjeru sabiru se vrijednosti 8, 1, 5 i 4, pa je rezultat 18.

Input:

```
8
12
357
4
-1
```

Output:

```
18
```

Zadatak 4.3.

Korisnik unosi cijeli pozitivan broj `n`, a zatim `n` cijelih brojeva. Potrebno je kreirati trakasti grafikon tako što će se za svaku unesenu vrijednost iscrtati toliko zvjezdica ili crtica u tom redu. Prva traka se iscrtava crticama, druga zvjezdicama, treća opet crticama i tako naizmjenično do samog kraja radi preglednosti.

Ispis za n = 2:

```
5
2
7
5
4
8
```

Ispis za n = 5:

```
--
*******
-----
****
--------
```

Zadatak 4.4.

Korisnik unosi pozitivne cijele brojeve jedan ispod drugog. Kraj unosa korisnik označava sa `-1`. Program ispisuje sumu posljednjih cifara unesenih brojeva. U primjeru ispod posljednje cifre unesenih brojeva su: 2, 4, 3, 5 i 0, pa je njihov zbir 14.

Input:

```
32
184
3
25
10
-1
```

Output:

```
14
```

Zadatak 4.5.

Korisnik unosi realne brojeve jedan ispod drugog. Kraj unosa označava praznim redom. Brojeve naizmjence uzimaju Harun i Sead. Prvi uneseni broj je Harunov, drugi Seadov, treći Harunov, četvrti Seadov itd.

Program ispisuje razliku sume Harunovih i sume Seadovih brojeva. U primjeru ispod suma Harunovih brojeva je 19.14 a Seadovih je 11.48 pa je njihova razlika 7.66.

Input:

```
3.14
2.81
4.5
8.67
11.5
```

Output:

```
7.66
```

Zadatak 4.6.

Napisati program u koji korisnik unosi pozitivne brojeve, svaki u zasebnoj liniji. Korisnik označava kraj unosa brojem -1. Ukoliko je uneseni broj paran program ispisuje njegova negirana vrijednost, a ako nije, program ispisuje broj u originalnom obliku. Primjer:

Input:

```
2
341
5
18
-1
```

Output:

```
-2
341
5
-18
```

Zadatak 4.7.

Napisati program u koji korisnik unosi riječi, svaku u zasebnoj liniji. Korisnik označava kraj unosa praznim redom. Program treba ispisati unesene riječi odvojene praznim mjestom.

Input:

```
kruska
jabuka
banana
```

Output:

```
kruska jabuka banana
```

Zadatak 4.8.

Napisati program koji od korisnika zahtijeva unos cijelih brojeva, svaki u zasebnom redu. Korisnik označava kraj unosa praznim redom. Program ispisuje negirane vrijednosti. Program ispisuje vrijednosti u istom redu razdvojene praznim mjestom.

Input:

```
5
0
-11
2
```

Output:

```
-5 0 11 -2
```

Zadatak 4.9.

Napisati program koji od korisnika zahtijeva unos cijelih brojeva, svaki u zasebnom redu. Korisnik označava kraj unosa praznim redom. Program ispisuje `Nalazi se!` ukoliko se broj 2 nalazi među unesenim vrijednostima. Program ispisuje `Ne nalazi se!` ukoliko se broj 2 ne nalazi među unesenim vrijednostima.

Input:

```
8
0
-5
```

Output:

```
Ne nalazi se!
```

Zadatak 4.10.

Korisnik unosi višecifren prirodan broj. Program ispisuje razliku proizvoda i količnika najveće i najmanje cifre broja. Može se pretpostaviti da niti jedna cifra nema vrijednost 0.

Input:

```
27385
```

Output:

```
12.0
```

Zadatak 4.11.

Korisnik unosi pozitivan cijeli broj n. Program ispisuje družinu najdužeg raspona u kojem su sve uzastopne cifre veće od 4. U narednom primjeru imamo četiri raspona sa ciframa većim od 4 i to: 9, 86, 978, 58. Najduži raspon je 978, pa se njegova dužina ispisuje.

Input:

```
928632978158
```

Output:

```
3
```

Zadatak 4.12.

Korisnik dva pozitivna cijela broja `a` i `b`. Program ispisuje sve brojeve u rasponu od `a` do `b` (uključujući obje vrijednosti) koji su djeljivi sa 3 ili koji u sebi sadrže barem jednu cifru djeljivu sa 3. Brojevi se ispisuju od najmanjeg ka najvećem u istom redu razdvojeni jednim praznim mjestom. [Napomena: 0 je djeljiva sa 3.]

Input:

```
8
19
```

Output:

```
9 10 12 13 15 16 18 19
```

Zadatak 4.13.

Za prirodan broj kažemo da je fin ako sadrži dvije uzastopne cifre čija je suma 4. Korisnik unosi prirodan broj n. Program ispisuje n-ti fin prirodan broj.

Input:

```
3
```

Output:

```
31
```

Zadatak 4.14.

Korisnik unosi prirodne brojeve m i n jedan ispod drugog. Program ispisuje matricu sa m redova i n kolona koristeći uzorak ilustrovan u dva naredna primjera. Kolone u matrici su odvojene jednim praznim mjestom.

Input:

```
4
7
```

Output:

```
1 1 1 1 1 1 1
1 2 1 2 1 2 1
1 2 3 1 2 3 1
1 2 3 4 1 2 3
```

Input:

```
6
4
```

Output:

```
1 1 1 1
1 2 1 2
1 2 3 1
1 2 3 4
1 2 3 4
1 2 3 4
```

Zadatak 4.15.

Napisati program koji ispisuje sve **trocifrene** brojeve koji ispunjavaju uslov:

$$abc = a^3 + b^3 + c^3 \tag{4.1}$$

gdje a, b i c predstavljaju cifre trocifrenog broja. Program ispisuje brojeve od najmanjeg ka najvećem u istom redu, ali razdvojene praznim mjestom.

Zadatak 4.16.

Napisati program koji od korisnika uzima broj n. Program iscrtava pravougaonik dimenzija n redova i $2n+1$ kolona, u formatu prikazanom u narednim primjerima.

Ispis za n = 2:

```
xxxxx
x0x0x
```

Ispis za n = 5:

```
xxxxxxxxxxx
x0x0x0x0x0x
xxxxxxxxxxx
x0x0x0x0x0x
xxxxxxxxxxx
```

Zadatak 4.17.

Napisati program koji od korisnika zahtijeva unos cijelog broja. Program provjerava da li su cifre unesenog broja sa lijeva na desno u monotono opadajućem redoslijedu. Drugim riječima, program provjerava da li vrijedi da je $a_n \geq a_{n+1}$, gdje a predstavlja jednu cifru broja, a n njegovu poziciju.

Ukoliko su cifre unesenog broja u monotono opadajućem redoslijedu program ispisuje `jesu`, a u suprotnom program ispisuje `nisu`. Primjeri:

Input:

```
5442110
```

Output:

```
jesu
```

Input:

```
56
```

Output:

```
nisu
```

[Napomena: obratiti pažnju na granični slučaj kada korisnik unese jednocifreni broj.]

Zadatak 4.18.

Napisati program koji od korisnika zahtijeva unos cijelog broja n. Program ispisuje proizvod *neparnih* cifara tog broja koje su *manje od 6*.

Input:

```
1234567854321
```

Output:

```
225
```

Zadatak 4.19.

Za bilo koji cijeli broj $0 \leq n \leq 10000$ koji nije djeljiv sa 2 ili 5, moguće je pronaći proizvod sa nekim drugim brojem u kojem je svaka cifra 1. Program treba pronaći koliko ima cifara u najmanjem takvom proizvodu n. Program od korisnika uzima broj n, a ispisuje broj cifara u najmanjem proizvodu n u kojem je svaka cifra 1. U primjeru ispod, kada se broj 3 pomnoži sa 37 rezultat je 111, i sastoji se od tri jedinice, pa program ispisuje 3.

Input:

```
3
```

Output:

```
3
```

Zadatak 4.20.

Prilikom pisanja velikih brojeva, moguće je poboljšati njihovu čitkost, tako što će se svake tri cifre odvojiti tačkom od prethodnih. Napisati program koji od korisnika zahtijeva unos više vrijednosti, svake u novom redu, a program ispisuje te vrijednosti, ali tako da cifre odvaja tačkom kao što je opisano. Može se pretpostaviti da će korisnik uvijek unositi pozitivne cjelobrojne vrijednosti, a kraj unosa označit će brojem -1.

Input:

```
42
12345
1000
987654321
-1
```

Output:

```
42
12.345
1.000
987.654.321
```

Zadatak 4.21.

Za neki broj n kažemo da je savršen ukoliko je jednak sumi svih svojih pozitivnih djelilaca manjih od n. Na primjer, 28 je savršen broj: njegovi djelioci su 1, 2, 4, 7 i 14, a $1+2+4+7+14=28$. Napisati program koji traži unos cijelih brojeva `a` i `b`, a koji zatim ispisuje sve savršene brojeve u opsegu od `a` do `b`, jedan ispod drugog.

Input:

```
1 28
```

Output:

```
6
28
```

Zadatak 4.22.

Napisati program koji pronalazi dvocifireni broj čiji trocifreni kvadrat ima zadnje dvije cifre iste kao originalni dvocifreni broj. Tj. uzme li se da `A`, `B` i `C` predstavljaju cifre broja, onda je za broj u formatu `AB`, potrebno izračunati `AB * AB` tako da je `AB * AB = CAB`.

Program treba ispisati broj koji zadovoljava ovaj uslov, a u narednom redu kvadrat tog broja.

Zadatak 4.23.

Program od korisnika traži unos brojeva `n` i `m`. Program nakon toga ispisuje sve brojeve od 1 do `n`, ali tako da ispisuje `m` brojeva u jednom redu. Brojevi su razdvojeni jednim praznim mjestom, ali prilikom ispisa jednocifrenih brojeva program ispisuje dodatno prazno mjesto ispred broja zbog adekvatnog poravnanja.

Input:

```
20
6
```

Output:

```
 1  2  3  4  5  6
 7  8  9 10 11 12
13 14 15 16 17 18
19 20
```

Zadatak 4.24.

Za prirodan broj kažemo da je čudan ako je u potpunosti sastavljen od cifara 2 i 3. Korisnik unosi prirodan broj n. Program ispisuje n-ti čudan prirodan broj.

Input:

```
6
```

Output:

```
33
```

Zadatak 4.25.

Korisnik unosi prirodne brojeve jedan ispod drugog. Kraj unosa korisnik označava praznim redom. Program ispisuje proizvod drugog i pretposljednjeg unesenog broja. Možete pretpostaviti da će korisnik unijeti barem dva broja.

Input:

```
1
2
3
4
```

Output:

```
6
```

Zadatak 4.26.

Korisnik unosi prirodan broj n, a zatim n prirodnih brojeva. Prvi broj dobija Sead, sljedeća dva Harun, zatim sljedeća 3 Sead, sljedeća 4 Harun i tako dalje dok se svih n brojeva ne podijeli. Ispisati u prvoj liniji sumu svih brojeva koje je dobio Sead, a u drugoj liniji sumu svih brojeva koje je dobio Harun.

Input:

```
5
1
2
3
4
5
```

Output:

```
10
5
```

Zadatak 4.27.

Na tri horizontalne trake sačinjene od polja postavljena su tri skakavca. Svaki skakavac može biti postavljen na bilo koje polje svoje trake. Korisnik unosi 3 para brojeva. Prvi broj para predstavlja početnu poziciju tog skakavca, a drugi broj para je broj polja koje taj skakavac preskače prilikom skoka. Također, na kraju korisnik označava polje koje predstavlja cilj za sve skakavce.

Skakavci skaču jedan po jedan, počevši sa prvim i pokušavaju stići do cilja. Igra se završava čim jedan od skakavaca dostigne cilj (stane na ciljno polje ili ga preskoči).

U primjeru ispod prvi skakavac je postavljen na polje 3 i preskače po 2 polja, drugi je postavljen na polje 1 i preskače po četiri polja, a treći na polje 6 i preskače jedno po jedno polje.

Program treba iscrtati situaciju na završetku utrke. Polja se označavaju znakom `'-'`, skakavci znakom `'*'`, a cilj znakom `'|'`. Ukoliko skakavac stoji na cilju iscrtava se `'*'` umjesto `'|'`. Sve trake se iscrtavaju u dužini koju je dostigao skakavac pobjednik. Polja se broje počevši sa brojem 1, a ne 0. Možete pretpostaviti da su svi skakavci na samom početku utrke postavljeni prije cilja.

Input:

```
3
2
1
4
6
1
10
```

Output:

```
--------*|---
---------|--*
-------*-|---
```

Zadatak 4.28.

Korisnik unosi pet pozitivnih cijelih brojeva jedan ispod drugog: `r`, `k`, `x1`, `y1` i `d`. Vrijednost `r` predstavlja broj redova, a `k` broj kolona matrice u kojoj se vrši iscrtavanje pravouglog jednakokrakog trougla. Vrijednosti `x1` i `y1` predstavljaju koordinate gornje lijeve tačke trougla, a vrijednost `d` dužinu stranica trougla.

Trougao se iscrtava tako što se sve tačke u matrici koje mu pripadaju označe znakom `'X'`, a sve ostale tačke znakom `'-'`. Između znakova u istom redu se ispisuje prazno mjesto.

Na primjeru ispod, matrica se sastoji od 6 redova i 10 kolona. Gornja lijeva tačka trougla ima koordinate: 3 po x osi i 1 po y osi, i dužinu stranice 4.

Input:

```
6
10
3
1
4
```

Output:

```
- - - - - - - - - -
- - - X X X X - - -
- - - - X X X - - -
- - - - - X X - - -
- - - - - - X - - -
- - - - - - - - - -
```

Zadatak 4.29.

Korisnik unosi dva broja, jedan ispod drugog, od kojih je prvi duži (ima više cifara). Program ispisuje poziciju drugog broja unutar prvog. Za brojeve 123456 i 234 odgovor je `1` jer se uzastopne cifre broja 234 pojavljuju na drugoj poziciji broja 123456 (pozicije se broje počevši od 0, pa druga pozicija ima vrijednost 1). Ukoliko se drugi broj ne nalazi unutar prvog ispisuje se `-1`. Tako za brojeve 987654 i 9887 program ispisuje `-1`.

[*Važno!* Tokom izrade zadatka potrebno je tretirati vrijednosti kao integere, a nikada kao stringove.]

Input:

```
6484316946
946
```

Output:

```
7
```

Zadatak 4.30.

Program iscrtava stepenice na osnovu 2 prirodna broja koja unosi korisnik, jedan ispod drugog. Prvi broj predstavlja visinu na kojoj stepenice počinju, a drugi visinu na kojoj završavaju. Stepenice se iscrtavaju upotrebom znaka `'*'`. Kolone su razdvojene jednim praznim mjestom.

Input:

```
7
3
```

Output:

```
*
* *
* * *
* * * *
* * * * *
* * * * *
* * * * *
```

Zadatak 4.31.

Profesor je napisao na tabli prirodan broj. Studenti jedan po jedan izlaze na tablu i umjesto broja koji je trenutno napisan pišu broj koji je jednak zbiru kvadrata cifara prethodno napisanog broja, sve dok se na tabli ne pojavi broj 1.

Ako je na tabli na početku napisan broj 654, sljedeći napisani broj će biti 77, a nakon njega 98, itd.

Korisnik unosi prirodan broj n. Program ispisuje koliko brojeva će biti napisano na tabli. Ukoliko se nikad neće pojaviti broj 1 na tabli, program ispisuje -1 (kao što je slučaj za broj 654).

Input:

```
7
```

Output:

```
5
```

Zadatak 4.32.

Korisnik unosi matricu u kojoj se nalaze znakovi `'*'` i `'-'`. Kolone su razdvojene praznim mjestom, a redovi se ispisuju jedan ispod drugog. Kraj unosa se označava praznim redom.

Program ispisuje površinu najvećeg pravougaonika koji je u potpunosti ispunjen zvjezdicama. U narednom redu program ispisuje koordinate gornjeg lijevog ugla pravougaonika, te u narednom redu koordinate donjeg desnog ugla pravougaonika. Koordinate su odvojene praznim mjestom.

Input:

```
- - - * - - * - -
- - * * * * * * -
- * * * * * * - -
* * * * * * * - -
- * * * * - * * -
- * * - - * * * -
```

Output:

```
15
1 2
3 6
```

Zadatak 4.33.

Napisati program koji od korisnika uzima broj n (ne veći od 10), a iscrtava tablicu množenja za $n \times n$ brojeva. Obratiti pažnju na poravnavanje kolona.

Input:

```
10
```

Output:

```
    1   2   3   4   5   6   7   8   9  10
 1  1   2   3   4   5   6   7   8   9  10
 2  2   4   6   8  10  12  14  16  18  20
 3  3   6   9  12  15  18  21  24  27  30
 4  4   8  12  16  20  24  28  32  36  40
 5  5  10  15  20  25  30  35  40  45  50
 6  6  12  18  24  30  36  42  48  54  60
 7  7  14  21  28  35  42  49  56  63  70
 8  8  16  24  32  40  48  56  64  72  80
 9  9  18  27  36  45  54  63  72  81  90
10 10  20  30  40  50  60  70  80  90 100
```

Zadatak 4.34.

Upotrebom naredna dva pravila moguće je generisati sekvencu brojeva:

$n = n/2$ (kada je n parno)

$n = 3n + 1$ (kada je n neparno)

Koristeći ova pravila i počevši sa brojem 13 generiše se naredna sekvenca:

```
13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
```

Ova sekvenca sadrži 10 elemenata. Svaka sekvenca završava kada se dođe do broja 1.

Program treba pronaći i ispisati broj u rasponu od 0 od 10000 koji, kada se koristi kao početak sekvence, generiše sekvencu sa najvećim brojem elemenata.

5. Nasumični brojevi

U poglavlju će biti opisan način generisanja slučajnih brojeva neophodnih za kreiranje simulacija i igrica u programiranju. Osim toga, u poglavlju je naveden kratki uvod iz teorije vjerovatnoće neophodan za kreiranje simulacija eksperimenata sa bacanjem kockica, novčića i drugih primjera.

5.1 Generisanje slučajnih brojeva

U procesu programiranja situacija iz stvarnog svijeta, često je neophodno modelirati slučajnosti. Teško je zamisliti bilo koju računarsku igru koja ne uključuje slučajnosti: karte bi se uvijek generisale istim redoslijedom, mine bi se uvijek nalazile na istim mjestima, loptica bi se uvijek kretala jednako, protivnici bi se uvijek jednako ponašali... Osnova za kompletan koncept slučajnosti u programiranju leži u procesu generisanja slučajnih brojeva, jer u slučaju da možemo generisati slučajne brojeve, možemo generisati i slučajne akcije (npr. uz dodatnu pomoć kontrola toka).

Ranije smo se upoznali sa modulom `math` koji sadrži niz matematičkih funkcija. Slučajni brojevi u programskom jeziku `Python` generišu se pomoću modula `random`. Da bi koristili modul, neophodno je uraditi import pomoću naredbe `import random`. Kako bi korištenje modula `random` bilo jednostavnije, često se import radi sa alijasom (npr. `import random as r`). Da bi se funkcija koja pripada ovom modulu pozvala u kodu, neophodno je koristiti naziv modula (ili alijas) kao osnov za pozivanje. Stoga, ukoliko želimo pozvati funkciju `randrange(10)`, poziv se zapisuje kao `r.randrange(10)`. U slučaju da se ne koristi alijas, poziv se izvršava kao `random.randrange(10)`. Isto vrijedi za sve funkcije iz modula.

5.1.1 Generisanje slučajnih cijelih brojeva

Postoji više funkcija za generisanje slučajnih brojeva koje se koriste u zavisnosti od potreba programa. Prva funkcija koja će biti detaljnije opisana je već pomenuta funkcija `randrange(a, b, s)`. Ova funkcija generiše slučajan cijeli broj veći ili jednak `a`, te manji od `b`. Treći parametar funkcije predstavlja korak (engl. *step*) pri generisanju brojeva (analogno ranije opisanoj `range` funkciji). Npr. funkcija `randrange(1, 5, 2)` će generisati ili broj 1 ili broj 3, jer generiše cijele

brojeve veće ili jednake 1, manje od 5 i uzima svaki drugi broj. U slučaju izostavljanja trećeg parametra, uzima se pretpostavljena vrijednost za korak, broj 1. Druga verzija iste funkcije uzima samo jedan parametar `randrange(b)`, što pretpostavlja da korisnik generiše slučajan cijeli broj veći ili jednak 0 i manji od `b`, sa korakom 1.

Druga funkcija je `randint(a,b,s)`, koja generiše slučajan cijeli broj veći ili jednak `a` i manji ili jednak `b`. Treći parametar `s` predstavlja korak (engl. *step*) promjene, tj. vrijednost za koju se mijenjaju vrijednosti od `a` do `b` pri generisanju. Npr. `randint(1, 5, 2)` može generisati brojeve 1, 3 i 5, dakle, krenuvši od `1`, možemo generisati brojeve koristeći korak 2. Pretpostavljena vrijednost za `s` je 1, te u slučaju da se funkcija pozove sa dva parametra, vrijednost trećeg će biti jednaka 1. Kao što je moguće uočiti, funkcije `randrange` i `randint` su sliče, osnovna razlika u tome što druga funkcija uključuje najveću vrijednost, dok je prva ignorira.

5.1.2 Generisanje slučajnih realnih brojeva

Iznad navedene funkcije koriste se za generisanje slučajnih cijelih brojeva. Često je neophodno generisati slučajan realan broj, stoga, modul `random` nudi više funkcija koje to omogućavaju.

Najjednostavnija funkcija koja se koristi pri generisanju realnih brojeva je funkcija `random()`. Ova funkcija generiše slučajan realan broj od 0 do 1. Preciznije rečeno, generiše se broj x takav da $x \in [0,1)$. Dakle, funkcija `random` ne generiše broj 1.

Često je neophodno generisati vrijednost iz većeg raspona od 1, npr. želimo broj iz raspona $[0,b)$, gdje je $b \in \mathbb{R}$ i $b > 1$. U tom slučaju, dovoljno je generisani broj od 0 do 1 pomnožiti sa dužinom raspona, b. Ukoliko je raspon pomjeren od nule, tj. želimo broj iz raspona $[a,b)$, na početnu vrijednost je neophodno dodati minimalnu vrijednost koju želimo generisati. Konačan broj dobijamo množenjem početnog broja sa rasponom $(b-a)$, te dodavanjem broja a na početnu vrijednost. Npr. ukoliko želimo realan broj od 2 do 10, vrijednost generišemo kao `random() * (10-2) + 2`. Ovo će generisati broj od 0 do 1, pomnožiti ga sa 8 (te dobiti broj između 0 i 8). Na dobijenu vrijednost dodajemo 2, je generisan slučajan broj od 2 do 10. Važna napomena jeste da broj 10 neće biti nikada generisan.

Druga korištena funkcija je `uniform(a,b)`. Na osnovu dokumentacije modula, ova funkcija vraća slučajan realan broj iz segmenta `[a,b]` ukoliko je broj `b` veći ili jednak `a`, te iz raspona `[b,a]` inače.

Postoji niz drugih funkcija koje vraćaju slučajne brojeve sa različitim distribucijama (rasporedima unutar segmenata), te se iste mogu koristiti po potrebi. Osim korištene uniformne distribucije, jedna od najčešće korištenih u praksi je normalna (Gaussova) distribucija. Za ovu distribuciju se koristi funkcija `gauss`.

5.2 Generatori i sjeme

Važno je naglasiti da su računari uređaji koji imaju precizno definisano ponašanje i kod kojih je kompletan princip rada takav da *slučajnosti* u pravom smislu te riječi ne postoje. Međutim, kako bi pomenuti svijet igara i niz drugih aplikacija bio krajnje dosadan bez slučajnosti, razvijeni su algoritmi koji naizgled daju slučajne vrijednosti. Vrijednosti koje su samo naizgled slučajne nazivaju se **pseudoslučajnim brojevima**. U pozadini postoji matematički algoritam koji generiše brojeve na način da osoba ne može pretpostaviti koji je naredni broj koji će biti generisan.

Algoritmi koji generišu ovakve brojeve poprilično su kompleksni, te prihvataju određen ulaz pri generisanju brojeva (na osnovu ulaza određen je i rezultat algoritma). Za isti ulaz, algoritmi uvijek generišu jednake vrijednosti. Stoga, neophodno je omogućiti da se taj ulaz razlikuje pri svakom pokretanju programa, kako dobijeni niz generisanih brojeva ne bilo moguće predvidjeti. Često se u

praksi koristi vrijeme u milisekundama proteklo od 1. januara 1970. godine, mada postoji i niz drugih načina za generisanje ulazne vrijednosti. Ulazna vrijednost se naziva **sjeme** (engl. *seed*). Za različitu vrijednost, generišu se različiti nizovi slučajnih brojeva, te za jednake vrijednosti dobijamo generisanje jednakih nizova.

Programeri često nailaze na pogreške (tzv. *bugove*) u kodu. Međutim, u zadacima koji uključuju generisanje slučajnih brojeva, kao i u ostalim zadacima, greške se mogu javiti samo za neke vrijednosti. Ako se brojevi generišu slučajno, onda možemo doći u situaciju da je teško ponoviti scenarij u kojem je došlo do greške, te otkrivanje greške može potrajati (zamislite da se greška događa za vrijednost 99, a mi generišemo brojeve od 1 do 10 000). Stoga, programerima je omogućeno da mogu program sa generisanjem slučajnih brojeva pokrenuti više puta i uvijek dobiti jednake vrijednosti (bez da značajno mijenjaju kod i ručno postavljaju vrijednosti). To se radi pomoću funkcije `seed(a)`, koja postavlja sjeme generatoru pseudoslučajnih brojeva na vrijednost `a`. Sjeme se postavlja na samom početku programa, prije upotrebe funkcija za generisanje slučajnih brojeva (tj. prije vrijednosti koje želimo da ostanu iste pri višestrukom pokretanju programa). Tada možemo uklanjanjem/dodavanjem te linije dobiti kompletan program koji se izvršava svaki put jednako ili svaki put različito.

5.3 Vjerovatnoća

Ukoliko korisnik baca novčić, smatra se da je vjerovatnoća svakog ishoda (pismo ili glava) jednaka 50%. Matematički, vjerovatnoća se često izražava realnim brojem p, takvim da vrijedi $0 \leq p \leq 1$. U pomenutom primjeru, vjerovatnoća pisma je 0.5, te je vjerovatnoća da će pasti glava također 0.5. Svaku od opcija neke simulacije nazivamo događajem. Događaj u kome je vjerovatnoća jednaka 1, nazivamo sigurnim događajem (to je vjerovatnoća da je pri bacanju novčića palo pismo ili glava). Događaj u kome je vjerovatnoća jednaka 0 nazivamo nemogućim događajem (događaj da je pri bacanju standardne igraće kockice sa šest strana pao broj 7). Opciju da je pri bacanju kockice pao broj 5 nazivamo ishodom događaja. Bacanje kockice u ovom slučaju nazivamo eksperimentom.

Najjednostavniji oblik računanja vjerovatnoće zasniva se na pronalasku broja svih mogućih ishoda i na pronalasku broja pozitivnih ishoda. Za računanje vjerovatnoće, često se izvode simulacije. Simulacije se zasnivaju na većem broju ponavljanja određenog eksperimenta (npr. više puta bacimo kockicu). Nakon brojanja uspješnih ishoda i ukupnog broja ponavljanja, vjerovatnoću računamo kao količnik te dvije vrijednosti. U slučaju da smo kockicu bacali 100 puta, te u slučaju da je tačno 20 puta pao broj 2, kažemo da je vjerovatnoća da će pasti broj 2 jednaka $20/100 = 0.2$. Većim brojem simulacija dobijamo veću preciznost u računanju vjerovatnoće uspješnog ishoda. U slučaju da kockicu bacimo 1000000 puta, vjerovatnoća da je pao broj 2 će najvjerovatnije biti bliža $1/6$ nego u slučaju pomenutih 100 bacanja.

Primjer 5.1 Napisati simulaciju bacanja igraće kockice, te na 10000 simulacija izračunati vjerovatnoću da ishod bude broj 3 ili 5.

```
import random as r

broj_pokusaja = 10000
brojac_uspjeha = 0

for i in range(broj_pokusaja):
   ishod = r.randint(1,6)
```

```
    if ishod == 3 or ishod==5:
        brojac_uspjeha = brojac_uspjeha + 1

print(brojac_uspjeha / broj_pokusaja)
```

Nakon uključivanja modula `random` sa alijasom `r`, definišemo varijable koje čuvaju ukupan broj pokušaja i ukupan broj uspješnih ishoda. Broj pokušaja je jednak broju simulacija, dok je broj uspješnih ishoda u početku jednak nuli.

Simuliramo izvršavanje određen broj puta (`broj_pokusaja`), te u svakom koraku generišemo slučajan cijeli broj u rasponu od 1 do 6. Ukoliko je taj ishod jednak broju 3 ili 5, povećavamo brojač uspješnih simulacija.

Nakon prolaska kroz sve simulacije, konačan rezultat dobijamo dijeljenjem broja uspješnih i ukupnog broja simulacija. ■

Višestrukim pokretanjem programa, moguće je uočiti da rezultat varira (iako je uvijek blizu $1/3$). Da bi lakše uočili potencijalnu grešku, u testnoj fazi kodiranja se često postavlja sjeme na određenu vrijednost. Ukoliko bi dodali naredbu `r.seed(42)` prije petlje i nakon importa modula, gdje je 42 proizvoljno odabran pa fiksiran prirodan broj, svakim pokretanjem programa bi dobili jednak rezultat.

5.4 Zadaci

Zadatak 5.1.

U teoriji muzike koriste se četiri osnovne note: C, D, E, F, G, A i H. Računar komponuje kompoziciju tako što nasumično bira brojeve koje predstavljaju note. Vrijednost 1 predstavlja notu C, vrijednost 2 notu D i tako redom do vrijednosti 7 koja predstavlja H. Program treba ispisati vjerovatnoću da će računar u prvih deset izbora odsvirati note C, E i G uzastopno.

Do odgovora treba doći simulirajući generisanje 10.000 kompozicija. Vjerovatnoća se ispisuje kao realan broj koji je u rasponu od 0 do 1, a ne kao procenat.

Zadatak 5.2.

Napisati program koji dozvoljava unos *dužine stranice kvadrata*, koja je tipa `float`. Program potom računa očekivanu (tj. prosječnu) udaljenost između dvije nasumično odabrane tačke unutar kvadrata.

Nasumične koordinate tačke trebaju biti brojevi tipa `float`. Za generisanje nasumičnih `float` brojeva, Python nudi funkciju `random.uniform(a, b)` gdje varijable `a` i `b` predstavljaju raspon mogućih brojeva i obje vrijednosti su uključene.

Zadatak 5.3.

Robot počinje da se kreće sa ishodišta koordinatnog sistema. Robot može da se kreće u četiri smjera. Broj 1 predstavlja smjer gore, broj 2 smjer dole, broj 3 smjer lijevo i broj 4 smjer desno. Robot se kreće u koracima dužine 1. Prije svakog koraka robot bira jedan od četiri moguća smjera nasumično i potom pravi korak u tom smjeru. Potrebno je napisati program koji računa vjerovatnoću da će se robot vratiti na početnu tačku nakon što napravi 10 koraka.

Zadatak 5.4.

Napisati program koji simulira bacanje **dvije kockice**. Kockice imaju šest strana i na njima se nalaze brojevi od 1 do 6. Program treba izračunati, koliko je prosječno potrebno bacanja, kako bi

se *tri puta za redom* dobio *broj veći od 6*.

Zadatak 5.5.

Nedim ima standardnu kockicu sa 6 strana numerisanu brojevima od 1 do 6, a Amina ima kockicu koja ima 9 strana numerisanih brojevima od 1 do 9. Obje kockice su fer, što znači da vjerovatnoća dobivanja jedne od stranica je jednaka za sve stranice. Nedim baca svoju kockicu, a potom i Amina. Koja je vjerovatnoća da će Nedim dobiti broj koji je veći od broja na Amininoj kockici. Da biste došli do odgovora potrebno je simulirati 10000 partija bacanja kockica.

Zadatak 5.6.

Program od korisnika traži unos broja `n`. Nakon toga program predviđa vjerovatnoću dobivanja `n` **uzastopnih** šestica ukoliko bacimo kockicu 30 puta. Za što tačnije predviđanje potrebno je izvršiti tačno 10000 simulacija.

Zadatak 5.7.

Dva robota su postavljena na ravnoj površini. Drugi robot je udaljen dva koraka desno od prvog. Svaki robot se kreće koracima dužine 1 i može da se kreće u četiri smjera. Broj 1 predstavlja smjer gore, broj 2 smjer dole, broj 3 smjer lijevo i broj 4 smjer desno. Prije svakog koraka robot bira jedan od četiri moguća smjera nasumično i potom pravi korak u tom smjeru. Roboti korake prave naizmjenično, a kretanje uvijek počinje prvi robot.

Potrebno je napisati program koji računa vjerovatnoću da će se roboti sudariti prije nego što naprave svaki po 10 koraka (uključujući i desete korake).

Do odgovora treba doći simulirajući nasumično kretanje robota 10.000 puta. Vjerovatnoća se ispisuje kao realan broj koji je u rasponu od 0 do 1, a ne kao procenat.

Zadatak 5.8.

Korisnik u prvom redu unosi niz uzastopnih znakova jedan iza drugog. Niz u sebi sadrži naredne znakove: `'*'`, `'_'` i `'!'`. Niz uvijek sadrži isključivo jedan znak `'*'` koji predstavlja polje na kojem se nalazi skakavac. Niz sadrži više znakova `'_'` koji predstavljaju prazna polja. Niz uvijek sadrži još dva posebna prazna polja označena sa `'!'` koji označavaju početak i kraj opasne zone koju skakavac želi izbjeći (dva polja `'!'` nisu dio opasne zone). Skakavac se uvijek nalazi lijevo od opasne zone.

U narednom redu korisnik unosi dva pozitivna cijela broja razdvojena praznim mjestom. Prvi broj predstavlja minimalan broj polja koje će skakavac preskočiti jednim skokom, a drugi maksimalan.

Skakavac počinje skakati sa lijeve na desnu stranu, i prilikom svakog skoka preskače nasumičan broj polja ograničen dvijema prethodno spomenutim vrijednostima (ali ih uključuje). Program treba ispisati vjerovatnoću da će skakavac preskočiti opasnu zonu tj. da prilikom skakanja neće stati na polja unutar opasne zone. Čim skakavac preskoči opasnu zonu ili stane u nju dalji njegovi skokovi se ne simuliraju.

U narednom primjeru skakavac se nalazi na drugom polju i prilikom svakog skoka može nasumično preskočiti 2, 3 ili 4 polja. Polja 11 i 12 predstavljaju opasnu zonu koju skakavac pokušava izbjeći.

Input:

```
_*_______!__!__
2 4
```

Output:

```
0.3365
```

Zadatak 5.9.

Šahovska ploča sastavljena je od 8 redova i 8 kolona. Figura topa može se kretati horizontalno i vertikalno po ploči. Figura koja se nalazi na nekoj od ovih putanja smatra se napadnutom od strane topa.

Program treba izračunati vjerovatnoću da se 3 topa neće međusobno napadati ukoliko ih nasumično postavimo na ploču, tj. da niti jedan top neće napadati bilo kojeg drugog topa. Obratite pažnju da ne postavite dva topa na isto polje.

Zadatak 5.10.

Korisnik unosi sumu novca sa kojom dolazi u kasino, iznos opklade i koliko novca planira osvojiti. Igrač igra igru gdje u svakoj partiji ima šansu od 49% da udvostruči ulog, ili ga izgubi. Program izračunava šanse za osvajanje željenog iznosa.

Npr. igrač može doći sa 100 KM, i postaviti iznos opklade na 10 KM i pokušati osvojiti 1000 KM. Program treba izračunati njegove šanse za osvajanje iznosa. Za približno predviđanje potrebno je izvršiti tačno 10000 simulacija. Vjerovatnoću je potrebno ispisati u rasponu od 0 do 1 bez znaka % (npr. za vjerovatnoću od 50.2% potrebno je ispisati 0.502).

Input:

```
100
10
1000
```

Output:

```
0.0089
```

Zadatak 5.11.

Pretpostavite da su `a`, `b` i `c`, nasumični brojevi u rasponu od 1 do 1000. Koja je vjerovatnoća da `a`, `b` i `c` mogu formirati stranice oštrouglog trougla. Napomena: tri dužine mogu formirati oštrougli trougao akko je (i) zbir svake dvije vrijednosti veći od treće i (ii) akko je zbir kvadrata svake dvije vrijednosti veći od kvadrata treće.

Za što tačnije predviđanje potrebno je izvršiti tačno 10000 simulacija.

Zadatak 5.12.

15 kuglica se stavlja u bubanj za izvlačenje. 5 kuglica je crvene, 5 zelene, a 5 plave boje. Koji je očekivani broj **različitih boja** koje ćemo imati nakon što nasumično izvučemo **četiri kuglice iz bubnja**, jednu po jednu. Nakon izvlačenja kuglica se ne vraća u bubanj.

Do odgovora treba doći simulirajući 100.000 pokušaja nasumičnog izvlačenja kuglica.

Zadatak 5.13.

Igrač baca n standardnih šestostranih igračih kockica označenih brojevima od 1 do 6 jednu za drugom. Koja je vjerovatnoća da će igrač u n bacanja dobiti sekvencu brojeva 3, 1, 4, 1, gdje se brojevi ne moraju pojavljivati uzastopno, tj. moguće je da se između nekih od brojeva pojavljuje i neka druga vrijednost, ali će se ova četiri broja pojaviti u navedenom redoslijedu. Npr. ukoliko igrač

baca 6 kockica i kao rezultat bacanja dobije vrijednosti 3, 1, 4, 4, 6, 1 smatra se da je dobio traženu sekvencu, dok ukoliko rezultat bude 3, 3, 1, 1, 5, 4 smatra se da nije dobio traženu sekvencu.

Program od korisnika uzima broj kockica koje se bacaju n, a ispisuje vjervotanoću pojavljivanja tražene sekvence. Do odgovora treba doći simulirajući 10.000 pokušaja bacanja šest kockica. Prilikom pisanja programa potrebno je pozvati funkciju `random.seed()` i proslijediti joj vrijednost 42. Vjerovatnoća se ispisuje kao realan broj koji je u rasponu od 0 do 1, a ne kao procenat.

Input:	Output:
`6`	`0.0091`

6. Funkcije

Zamislimo scenarij u kome na desetine mjesta u aplikaciji ispisujemo datum. Datum se uvijek ispisuje u formatu DD.MM.YYYY. (DD je oznaka za dan, MM za mjesec i YYYY za godinu). Na osnovu ranije naučenih alata, ovo je moguće riješiti tako što na svakoj poziciji ručno ispišemo odgovarajući format datuma.

Međutim, pretpostavimo da nakon određenog vremena aplikacija dobije promijenjene specifikacije. Umjesto formata `DD.MM.YYYY.`, odlučeno je da se koristi format `DD. MM. YYYY.` (sa razmakom). To bi značilo da na svakom mjestu korištenja moramo promijeniti kod i dodati neophodne razmake. Iako je naveden jednostavan primjer, u stvarnosti se ovakve situacije često susreću i mogu izazvati veliki problem i mukotrpan posao za osobu koja održava kod. Mnogo bi jednostavnija opcija bila da imamo jedan dio koda koji je centralni za svaki ispis datuma, te da pri modifikaciji tog ispisa modifikujemo kod na samo jednom mjestu. U tom trenutku bi bilo koja promjena formata datuma umjesto desetina promjena koda izazvala samo jednu, na tom centralnom mjestu.

U kompleksnijim softverskim rješenjima, neminovno je da postoje procesi koji se koriste na više mjesta. Kako se radi o istom procesu, procedura za njegovo rješavanje je jednaka i eventualni rezultat zavisi od korištenih varijabli. Koristeći ranije znanje, da bi implementirali takav scenario, neophodno je pisati jednak kod na više mjesta. Međutim, programeri su našli način da riješe takve situacije na najbolji način, te da izbjegnu ponavljanje koda. Jedno od osnovnih pravila kvalitetnog kodiranja jeste izbjegavanje ponavljanja dijelova koda!

Funkcije definišemo kao grupu naredbi koje izvršavaju konkretan zadatak. Funkcijama kompletnu aplikaciju razdvajamo na niz jednostavnijih procesa koje je jednostavnije održavati. Primjer često korištenih funkcija su `sqrt`, `sin`, `min`, `max` ... Situacija u kojoj bi bilo neophodno implementirati čitav proces računanja kvadratnog korijena ili sinusa na svakom mjestu u kodu u kojem koristimo tu vrijednosti teško je izvodiva. Stoga, kreirani su moduli koji sadrže veliki broj funkcija koje obavljaju zamorni posao umjesto nas. U ovom poglavlju, opisan je način kreiranja vlastitih funkcija koje mogu obavljati zadatke u zavisnosti od potreba programera.

Korist upotrebe funkcija je višestruka. Upotrebom funkcija kod postaje čitljiviji i lakši za

modifikacije i održavanje, testiranje koda je jednostavnije, te je mogućnost korištenja gotovih elemenata od velikog značaja za razvoj aplikacija. Kako je razvoj većih sistema najčešće posao kojeg radi tim ljudi, upotrebom funkcija postiže se lakša saradnja između članova tima (npr. svaki član tima dobije zadatak da razvije određene funkcionalnosti koje će drugi članovi koristiti kao gotove komponente, kao što smo mi u ranijim programima koristili funkcije iz modula `random` i `math`).

6.1 Nazivanje funkcija

Kako je ranije navedeno, funkcije predstavljaju niz naredbi koji obavlja određen zadatak. Funkcije se pozivaju (pokreću) po potrebi i u dijelu programa u kojem je to potrebno. Svakoj funkciji neophodno je dodijeliti naziv. Naziv funkcije se u praksi određuje tako da opisuje zadatak kojeg funkcija izvršava (npr. funkciju koja ispisuje datum nazivamo `ispisi_datum`).

Za naziv funkcije može se izabrati skup znakova kao i za nazive varijabli. Za naziv funkcije nije dozvoljeno koristiti Python ključne riječi, naziv funkcija ne može sadržavati prazna mjesta, prvi znak u nazivu mora biti veliko ili malo slovo engleskog alfabeta ili znak donja crta (_). Nakon definisanja prvog znaka, svaki naredni znak može biti bilo koje veliko ili malo slovo alfabeta, bilo koja cifra ili donja crta. U nazivima funkcija, velika i mala slova se razlikuju (funkcije `test` i `TeSt` nisu iste).

Ranije je navedeno da se pri imenovanju varijabli koje sadrže više riječi, svaka nova riječ razdvaja donjom crtom. U svijetu programiranja, pri imenovanju varijabli i funkcija koristi se nekoliko standardnih notacija:

- Kamila notacija (*CamelCase*), gdje je svako slovo u nazivu malo, osim prvog slova svake riječi. Nazivi ne sadrže donje crte. Postoje dva tipa kamila notacije. U prvom tipu, svaka riječ počinje velikim slovom, uključujući i prvu (npr. `NekiNazivFunkcije`). U drugom standardu, prva riječ počinje malim, a svaka naredna velikim slovom (npr. `nekiNazivFunkcije`).
- Zmija notacija (*snake_case*), gdje je svaka riječ napisana malim slovom, te su riječi razdvojene donjom crtom (`neki_naziv_funkcije`). Ova notacija se uzima kao standard za programski jezik Python.

Osim programskog jezika Python, notacija *snake_case* koristi se kao standard u programskim jezicima C, C++, R, Ruby, Rust, ABAP . . . Kamila notacija koristi se kao standard u programskim jezicima Java, Pascal, .NET i drugi.

6.2 Kreiranje funkcija

Za uspješnu upotrebu funkcija, neophodno je ispoštovati dva koraka:

1. Definisanje funkcije
2. Poziv funkcije

Definicijom funkcije naziva se niz naredbi koje opisuju funkciju. Generalni oblik definisanja funkcije naveden je ispod:

```
def naziv_funkcije():
    naredba
    naredba
    naredba
    itd
```

Prva linija koda iznad predstavlja zaglavlje funkcije. Zaglavlje se sastoji od ključne riječi `def`, od naziva funkcije, te u jednom slučaju od otvorene i zatvorene zagrade, te dvotačke. Nakon toga, slijedi blok naredbi koji može sadržavati proizvoljan niz koraka koje se izvršavaju kada korisnik odluči pozvati funkciju. Funkcija se ne izvršava do trenutka kada korisnik pozove istu (kao što se funkcija `sqrt` u ranijim primjerima izvršavala samo u slučajevima kada smo istu mi pozivali).

Sam proces **pozivanja** iznad opisane funkcije sastoji se od naredbe `naziv_funkcije()` koja se po potrebi poziva iz bilo kojeg dijela programa koji slijedi nakon definicije funkcije.

6.3 Podjela funkcija

Postoji više podjela funkcija. Osnovna podjela je na funkcije koje ne vraćaju vrijednost (tzv. `void` funkcije) i funkcije koje vraćaju vrijednost (tzv. `value-returning` funkcije).

Više puta pomenuta funkcija `sqrt` je funkcija koja vraća vrijednost, funkcija od koje očekujemo da izračuna vrijednost korijena nekog broja i vrati nam taj broj na raspolaganje kako bi ga mogli koristiti u izrazima i nastavku programa. Jednaka je situacija sa `input`, `int` ili `float` funkcijama. Dakle, ovaj tip funkcija izvrši neophodan niz naredbi (npr. izračuna korijen broja) i korisniku vrati određen rezultat.

Drugi tip funkcija, `void` funkcije, prođu zadati niz naredbi i tu njihovo izvršavanje stane. Programer ne dobija povratnu informaciju ili rezultat funkcije koji je upotrebljiv. Ovakve funkcije mogu obavljati pozadinski posla (npr. kreirati neki Excel dokument i zapisati tekst u njega), ali se u osnovnim primjerima najčešće koriste za ispis teksta u konzolu. Primjer ovakve funkcije je funkcija `print` koja ne vraća vrijednost, ali ima svrhu u kreiranju obavijesti korisniku kroz konzolu.

Funkcije koje vraćaju vrijednost uvijek sadrže ključnu riječ `return` u sklopu naredbi koje se izvršavaju pozivom funkcije (tj. unutar definicije). Na mjestu pojavljivanja ove riječi, funkcija **prekida izvršavanje** i vraća korisniku vrijednost koja slijedi iza `return` (npr. `return 2` prekida izvršavanje funkcije i vraća korisniku vrijednost 2). Funkcije koje ne vraćaju vrijednost ne sadrže ključnu riječ `return`.

Druga podjela funkcija je na funkcije sa parametrima i bez parametara. Funkcija koja vraća/ispisuje najmanji prirodan broj uvijek ima tačno određeno ponašanje (uvijek vraća/ispisuje broj 1). Njeno izvršavanje ne ovisi od bilo kojih *vanjskih faktora*. S druge strane, funkcija koja računa korijen nekog broja, mora dobiti određeni ulaz u vidu broja čiji korijen računamo. Njen konačni rezultat zavisi od proslijeđenog broja.

Pri definiciji funkcija, unutar zagrada se navode **parametri** koje funkcija prihvata. Parametri se navode kao varijable, te se razdvajaju zarezima. Unutar tijela funkcije, koriste se kao standardne varijable.

```
def ispisi_zbir_brojeva(x, y, z):
    print(x + y + z)
```

Vrijednosti koje se proslijede funkcijama pri pozivu, nazivaju se **argumentima** funkcije.

Pri pozivu funkcije iznad, koristimo naredbu `ispisi_zbir_brojeva(10, 2, 3)` čiji će rezultat izvršavanja biti ispis broja 15 u konzoli.

Kako je ranije navedeno, u ranijim poglavljima knjige korišten je veliki broj funkcija. Neke od tih funkcija, poput `range(a,b,c)`, dozvoljavale su izostavljanje određenih parametara. U tom slučaju, parametri su poprimali pretpostavljenu (engl. *default*) vrijednost. U slučaju da želimo omogućiti poziv iznad kreirane funkcije `ispisi_zbir_brojeva` sa dva argumenta, možemo u definiciji funkcije dodati pretpostavljene vrijednosti. Proizvoljan broj parametara može imati pretpostavljene vrijednosti, međutim, u trenutku kada za neki parametar definišemo takvu vri-

jednost, svi parametri sa desne strane moraju imati pretpostavljenu vrijednost. Razlog za to je svakako nemogućnost izostavljanja predzadnjeg ili nekog ranije parametra u slučaju da je posljednji neophodan. Primjer sa kreiranjem pretpostavljenih parametara u funkciji `ispisi_zbir_brojeva` naveden je ispod.

```
def ispisi_zbir_brojeva(x, y=1, z=0):
    print(x + y + z)
```

Funkciju možemo pozvati na tri načina:

- `ispisi_zbir_brojeva(10, 3, 2)`. U konzolu je ispisan zbir $10+3+2=15$.
- `ispisi_zbir_brojeva(10, 3)`. U konzolu je ispisan zbir $10+3+0=13$.
- `ispisi_zbir_brojeva(10)`. U konzolu je ispisan zbir $10+1+0=11$.

Jasno je da se ne može desiti slučaj izostavljanja parametra `b`, a prosljeđivanje vrijednosti za `c`.

6.4 Primjeri

U nastavku je analizirano nekoliko primjera upotrebe funkcija.

Primjer 6.1 — Funkcija bez parametara i bez povratnog tipa. Napisati funkciju koja ispisuje kvadrat 3×3 sačinjen od zvjezdica u konzolu.

```
def ispisi_kvadrat_3():
    print("* * *")
    print("* * *")
    print("* * *")

ispisi_kvadrat_3()
```

Ovaj primjer jednostavne funkcije definiše funkciju kroz tri jednostavne `print` naredbe. Svaka naredba ispisuje jedan red kvadrata. Funkcija nema parametara (jer se unaprijed zna šta treba biti rezultat njenog izvršavanja), te kako je u specifikacijama problema navedeno da funkcija ispisuje kvadrat u konzolu, funkcija nema povratni tip. Samim tim, kako funkcija nema povratni tip i nema parametara, njen poziv je `ispisi_kvadrat_3()`. ■

Primjer 6.2 — Funkcija bez parametara sa povratnim tipom. Napisati funkciju koja vraća zbir prvih 100 prirodnih brojeva.

```
def vrati_zbir_prvih_100():
    suma = 0
    for i in range(1, 101):
        suma = suma + i

    return suma

x = vrati_zbir_prvih_100()
print(x)
```

Zaglavlje funkcije počinje definisanjem imena funkcije. Funkcija nema parametara, jer suma prvih 100 brojeva ne ovisi od *vanjskih faktora* i volje korisnika. Tijelo funkcije je implementirano

koristeći standardni način za računanje zbira u programerskim zadacima sa pamćenjem trenutne vrijednosti sume, iako postoji jednostavniji način (koristeći Gaussovu formulu za zbir prvih n prirodnih brojeva) ili jednostavnim ispisom sume (zna se kolika je suma). Varijabla `suma` čuva vrijednost trenutnog zbira, koji je u početku nula. Prolaskom kroz sve brojeve od 1 do 100, svaki se dodaje na prethodnu sumu. U konačnici, varijabla `suma` čuva vrijednost u zadatku tražene sume. Izračunatu vrijednost vraćamo korisniku koristeći `return`. Svaka eventualna naredba nakon `return` se ne bi izvršila, jer po definiciji, nailaskom na `return` prekida se izvršavanje funkcije!

U glavnom dijelu programa, vraćena vrijednost se spašava u varijablu `x`, te se dobijena vrijednost ispisuje u konzolu. Bez spašavanja dobijene vrijednosti u varijablu, poziv funkcije bi bio uzaludan. ■

Primjer 6.3 — Funkcija sa parametrima i bez povratnog tipa. Napisati funkciju koja ispisuje kvadrat $n \times n$ sačinjen od zvjezdica u konzolu, gdje je n broj proslijeđen kao parametar funkcije.

```
def ispisi_kvadrat(n):
    for i in range(n):
        for j in range(n):
            print('* ', end='')
        print()

x = int(input())
ispisi_kvadrat(x)
```

Funkcija ispisuje kvadrat dimenzija $n \times n$. Kako je broj n nepoznat prije poziva funkcije, a ista ovisi o toj vrijednosti, vrijednost n je neophodno proslijediti kao parametar funkcije. Stoga, zaglavlje funkcije sadrži jedan parametar, broj n. Funkcija dvostrukom petljom prolazi kroz brojeve od 1 do n za svaki red (brojač i), te za svaki red prolazi kroz petlju koja ispisuje pojedinačne zvjezdice unutar tog reda. Na kraju ispisa svakog reda, prelazimo u novi red i ponavljamo postupak.

U glavnom dijelu programa, dimenzija se učitava kao unos korisnika, te se ista vrijednost proslijedi kao parametar funkciji. Funkcija nema povratni tip, stoga njen rezultat nema potrebe spasiti u varijablu. ■

Primjer 6.4 — Funkcija sa parametrima i sa povratnim tipom. Napisati funkciju koja prihvata prirodan broj n kao parametar, te računa zbir prvih n prirodnih brojeva.

```
def vrati_zbir(n):
    suma = 0
    for i in range(1, n+1):
        suma = suma + i
    return suma

x = int(input())
rez = vrati_zbir(x)
print(rez)
```

Funkcija koja računa i vraća zbir prvih n prirodnih brojeva, ne može se uspješno izvršiti bez prosljeđivanja vrijednosti za koju se zbir računa. Stoga, funkcija prihvata jedan parametar, broj n. Funkcija slično kao u ranijem primjeru računa zbir brojeva, te isti vraća kao rezultat.

Glavni dio programa učitava vrijednost kroz konzolu, pretvara je u cijeli broj, poziva funkciju koja računa zbir, te dobijenu vrijednost spašava u varijablu `rez`. Tu vrijednost koristi u nastavku programa. ■

Primjer 6.5 Napisati funkciju koja za proslijeđeni prirodni broj n provjerava da li je proslijeđeni broj prost ili složen. **Napomena.** Za neki prost broj kažemo da je prost ukoliko nema drugih djelilaca u skupu prirodnih brojeva osim broja 1 i samog sebe. Broj je inače složen. Za broj 1, ne definišemo da li je prost ili složen.

```
def je_li_prost(n):
  if n == 1:
    return 0

  brojac_djelilaca = 0
  for i in range(1, n+1):
    if n%i == 0:
      brojac_djelilaca += 1

  if brojac_djelilaca == 2:
    return 1
  else:
    return -1

n = int(input("Unesi"))
print(je_li_prost(n))
```

Iznad je naveden kod koji provjerava da li je određeni broj prost ili složen. Program radi strogo po definiciji navedenoj u tekstu zadatka. Program prolazi kroz sve brojeve od 1 do n, te provjerava broj djelilaca broja n. Ukoliko je taj broj jednak 2, broj je prost i funkcija vraća vrijednost 1. Inače, broj je složen i funkcija vraća vrijednost -1. Prije početka provjere, ukoliko je proslijeđen broj 1, funkcija vraća vrijednost 0. U kodu iznad, funkcija je definisana tako da treba vratiti tri vrijednosti (jednu za proste, jednu za složene i jednu za broj jedan). Česte su funkcije koje trebaju vratiti dvije vrijednosti i provjeriti da li je nešto tačno ili ne. U tom slučaju, funkcije se najčešće definišu da vraćaju vrijednosti `True` ili `False`. ■

Primjer 6.6 U nastavku je naveden kod koji pristupa rješavanju zadatka iz prethodnog primjera na drugi način.

```
def je_li_prost(n):
  if n == 1:
    return 0

  for i in range(2, n):
    if n%i == 0:
```

```
        return -1

    return 1

n = int(input("Unesi"))
print(je_li_prost(n))
```

Ukoliko drugačije tumačimo definiciju prostog broja, kažemo da je broj n složen ukoliko ima djelioca osim broja 1 i samog sebe. Jasno je da je svaki prirodan broj djeljiv sa 1 i sa samim sobom. Dakle, korisno bi bilo provjeriti sve brojeve od 2 do $n-1$, te ukoliko je n djeljivo sa bilo kojim od tih brojeva, odmah smo sigurni da broj n nije prost. U tom trenutku, možemo vratiti vrijednost -1 kao rezultat funkcije i nema potrebe za nastavkom provjere. Ukoliko kroz kompletnu petlju ne dobijemo vrijednost koja dijeli broj n, broj je prost. Ukoliko smo došli do kraja petlje (nigdje nismo prekinuli funkciju sa `return -1`), možemo vratiti vrijednost 1, jer smo sigurni da je broj prost. ■

Iako je razlika u rješenjima prethodna dva primjera neznatna, drugo rješenje je značajno brže u odnosu na prvo. Posmatrajmo šta se desi u slučaju da korisnik proslijedi vrijednost jedne milijarde. Ovaj broj je očigledno paran, dakle, djeljiv je sa 2 i veći je od 2, pa je složen. U prvom rješenju, naša petlja se u potpunosti izvršava (prođemo kroz milijardu koraka). U drugom rješenju, već na prvom koraku petlje shvatimo da je broj n djeljiv sa 2, pa prekinemo izvršavanje funkcije. Dakle, umjesto milijardu koraka, funkcija završi izvršavanje u 2 koraka. Da bi pojasnili koliko je značajna ova razlika, ukoliko je za jedan korak petlje potrebna jedna sekunda, druga funkcija će završiti svoj rad za jednu sekundu, a prva za 31 godinu (čak malo više). Dobro je programirati rješenja koja su brža i optimalnija.

Iako se u uvodnim programerskim zadacima ne daje veliki značaj broju koraka koje program obavlja (tzv. kompleksnost programa), u kasnijem razvoju svakog programera, ovakva pitanja postaju od izuzetnog značaja. Svaki programer bi trebao razmišljati o broju koraka koje njegov program izvršava, te o načinima da se taj broj koraka smanji. Za svaki problem, korisno je razmisliti kako će se napisano rješenje ponašati u slučaju velikih instanci?

Za razmišljanje. Koji je broj koraka koje funkcije izvrše u slučaju prostog broja 1 000 000 007? Kako bi drugu funkciju mogli dodatno optimizovati? *Uputa.* Da li je zaista neophodno provjeravati sve brojeve do $n-1$? Jedna manja promjena u kodu drugog rješenja može vrijeme izvršavanja za prost broj iznad smanjiti sa 31 godine na nešto manje od 9 sati.

6.5 Zadaci

Zadatak 6.1.

Napisati funkciju koja kao parametar uzima višecifreni broj `n` a potom sabira sve cifre tog broja. Ukoliko je i sam zbir višecifren, proces se ponavlja sve dok se ne dođe do rezultata koji sadrži samo jednu cifru.

Npr. za vrijednost `n = 987987987987` funkcija vraća vrijednost 6, jer se prilikom prvog sabiranja cifri dobije vrijednost 96 koja je višecifrena pa se i njene cifre sabiraju te se dobija vrijednost 15 čije se cifre opet sabiraju i na kraju se dobija vrijednost 6.

Također napisati i dio programa koji testira funkciju tako što od korisnika uzima vrijednost koju potom prosljeđuje funkciji, a nakon proračuna ispisuje rezultat.

Input:	Output:
`987987987987`	`6`

Zadatak 6.2.

Napisati funkciju koja kao parametar prima dva broja od kojih je prvi duži (ima više cifara). Funkcija određuje ukoliko se drugi broj nalazi u prvom. Za brojeve 123456 i 234 odgovor je `True` jer se uzastopne cifre broja 234 pojavljuju uzastopno u broju 123456, dok za brojeve 987654 i 9887 odgovor je `False`.

Također napisati i dio programa koji testira funkciju tako što od korisnika uzima vrijednosti ova dva broja koje potom prosljeđuje funkciji. Program ispisuje poruku `"Nalazi se"` ukoliko se drugi broj nalazi u prvom ili `"Ne nalazi se"` ukoliko se drugi broj ne nalazi u prvom.

[Napomena. Tokom izrade zadatka potrebno je tretirati vrijednosti kao integere, a nikada kao stringove.]

Input:	Output:
`123456` `234`	`Nalazi se`

Zadatak 6.3.

Napisati funkciju koja kao parametar prima vrijednost `n`, a potom iscrtava figuru prikazanu ispod. Također napisati i dio programa koji testira funkciju tako što od korisnika uzima vrijednost koju potom prosljeđuje funkciji.

Input:	Output:

`7`	`1 2 3 4 5 6 7`
	`2 3 4 5 6 7 1`
	`3 4 5 6 7 1 2`
	`4 5 6 7 1 2 3`
	`5 6 7 1 2 3 4`
	`6 7 1 2 3 4 5`
	`7 1 2 3 4 5 6`

Zadatak 6.4.

Napisati funkciju koja za proslijeđeni prirodan broj n (smatrati da je $n < 10$) ispisuje spiralu sačinjenu od brojeva kao u primjeru na slici ispod. Spirala se ispisuje od gornjeg lijevog ugla prema sredini.

Input:	Output:

`6`	` 1  2  3  4  5  6`
	`20 21 22 23 24  7`
	`19 32 33 34 25  8`
	`18 31 36 35 26  9`
	`17 30 29 28 27 10`
	`16 15 14 13 12 11`

Zadatak 6.5.

Počevši od broja 1 i pomjerajući se u smjeru kazaljke na satu, moguće je dobiti 5×5 spiralu formiranu kao u primjeru ispod:

```
21 22 23 24 25
20  7  8  9 10
19  6  1  2 11
18  5  4  3 12
17 16 15 14 13
```

Može se provjeriti da je suma elemenata na glavnim dijagonalama jednaka 101. Napisati funkciju koja vraća vrijednost sume elemenata na glavnim dijagonalama za matricu veličine $n \times n$ gdje je n neparan broj.

Input:

```
7
```

Output:

```
261
```

7. Liste

Koristeći ulaz, izlaz, naredbe granjanja, petlje i funkcije, stečen je snažan alat za programiranje kojim možemo riješiti niz različitih problema iz mnogih oblasti ljudskog djelovanja. Posmatrajmo svaki od narednih problema:

1. Napisati program koji od korisnika traži unos 10 prirodnih brojeva. Program ispisuje najveći među njima.
2. Napisati program koji od korisnika traži unos cijelih brojeva sve dok ne unese nulu ili negativan broj. Program ispisuje najveći uneseni broj.
3. Napisati program koji od korisnika traži unos brojeva sve dok ne unese nulu. Program ispisuje srednji broj po veličini od unesenih (npr. ukoliko je uneseno pet brojeva, program ispisuje treći po veličini).

Prvi problem je moguće riješiti uvođenjem deset varijabli, te nakon unosa svake od vrijednosti pronalazimo najveću. Prolazak najveće vrijednosti možemo riješiti na više načina, od kojih je logički najjednostavnije poređenje svake vrijednosti sa devet preostalih. Ona koja bude veća od svih ostalih je upravo najveća vrijednost koju tražimo. Drugi način za rješavanje jeste uvođenje dodatne varijable u koju spašavamo najveći broj do nekog trenutka. Prolaskom kroz svaki uneseni broj, provjeravamo da li je on veći od tog ranije najvećeg broja, te ukoliko jeste, mijenjamo vrijednost. Za početnu vrijednost pomoćne varijable postavljamo prvu unesenu vrijednost.

Posljednji opisani način može biti iskorišten i za implementaciju rješenja drugog problema, gdje koristeći `while` petlju za svaki uneseni broj provjeravamo je li negativan ili jednak nuli. Ukoliko jeste, prekidamo petlju. Ukoliko nije, provjeravamo da li je broj veći od ranije najvećeg ispitanog broja. Ukoliko je novi broj veći od ranije najvećeg, ta vrijednost postaje trenutno najveći broj. Kod je naveden ispod.

```
n = int(input())
tmax = n
```

```
while n != 0:
   n = int(input())
   if n > tmax:
      tmax = n

print(tmax)
```

Ukoliko detaljnije posmatramo pomenuta rješenja, možemo uvidjeti da je standardni način ljudskog razmišljanja u skladu sa implementacijom ranijeg programa. Ukoliko nam neko govori brojeve, sve dok ne kaže nulu ili negativan broj, te od nas zahtijeva da odredimo najveći, nije prirodno pamtiti sve brojeve. Ljudski um pamti samo najveći broj kojeg je čuo do tog trenutka. Kada čujemo veći broj, njega pamtimo. Upravo je ovo ideja rješenja iznad. Sličan postupak možemo primijeniti i za pronalazak najmanjeg rješenja, kao i za pronalazak razlike najvećeg i najmanjeg, za pronalazak drugog broja po veličini i slično.

Međutim, da bi pronašli srednji broj po veličini, ne možemo unaprijed odrediti koliko mjesta moramo rezervisati u memoriji. Da bi odredili srednji po veličini od pet brojeva, trebamo pamtiti bar tri vrijednosti. Međutim, za srednji po veličini od 100 brojeva, trebamo pamtiti bar 50 vrijednosti (50 najmanjih). Upravo nas ovo navodi na zaključak da je koristeći trenutne alate nemoguće riješiti navedeni zadatak. Dakle, iako su raniji alati izuzetno efikasni i snažni, nisu dovoljni za rješavanje svih problema sa kojima se programer susreće čak i u skupu uvodnih programerskih zadataka.

Na osnovu svega navedenog, javila se potreba za uvođenjem listi. **Listom** nazivamo objekat koji čuva više **elemenata** u tačno određenom redoslijedu. Liste je moguće mijenjati tokom izvršavanja programa. Pod mijenjanjem listi smatramo operacije dodavanja i uklanjanja elemenata, mijenjanja elemenata, sortiranja listi, kao i niz drugih operacija. Za pristupanje elementima najčešće se koristi **indeksiranje**.

Primjer liste u koju je smješteno nekoliko parnih prirodnih brojeva naveden je u nastavku.

```
parni_brojevi = [2, 4, 6, 8, 10, 12]
```

7.1 Indeksiranje elemenata

Svaki podatak u listi naziva se elementom. U primjeru iznad, elementi liste su brojevi 2, 4, 6, 8, 10 i 12. Elementi liste se navode u uglastim zagradama i razdvajaju se zarezom. Liste mogu sadržavati proizvoljan tip elemenata: cijele i realne brojeve, tekst, znakove, razne objekte, pa čak i druge liste. Naziv `parni_brojevi` predstavlja upravo naziv liste, dok se svakom elementu liste pristupa, kao što je ranije navedeno, putem indeksa. **Indeks** predstavlja redni broj elementa u listi. Zanimljiva je činjenica da brojanje (indeksiranje) počinje od nule. Dakle, prvom elementu liste pristupamo kao `parni_brojevi[0]`. Vrijednost koju ćemo dobiti je broj 2.

Liste možemo zamisliti kao ulice sastavljene od kuća. Svaka ulica ima naziv, te nakon lociranja ulice, svakoj pojedinačnoj kući pristupamo pomoću njenog broja. Svaka kuća je određena upravo nazivom ulice i brojem, kao što je slučaj da je svaki element liste određen nazivom liste i indeksom elementa. Svaka adresa može sadržavati različit tip kuća, kao što svaka Python lista može sadržavati različit tip podataka.

7.2 Prolazak kroz elemente liste

Ispis kompletne liste radi se pomoću naredbe `print(naziv_liste)`. U ranijem primjeru, naredba `print(parni_brojevi)` ispisuje `[2, 4, 6, 8, 10, 12]` u konzolu. Pojedinačnim elemen-

tima liste, kao što je već rečeno pristupamo pomoću indeksa, stoga, ukoliko želimo ispisati samo treći element liste iz primjera, koristimo naredbu `print(parni_brojevi[2])`. Ovo će ispisati vrijednost 6 u konzolu.

Ukoliko želimo ispisati svaki element liste, jedan ispod drugog, neophodno je kroz listu proći petljom. Moguće nesvjesno, u toku rada sa petljama, liste su bile konstantno korištene. Naredba `range(a, b)` vraća upravo listu brojeva, te je brojač u `for` petlji prolazio kroz listu. Umjesto funkcije `range` koja vraća listu (tj. umjesto liste koju vrati funkcija `range`), petlju možemo pozvati nad bilo kojom listom, kao u primjeru ispod. Ova petlja će za varijablu `el` postaviti svaku vrijednost iz liste parnih brojeva koju smo ranije definisali.

```
for el in parni_brojevi:
   print(el)
```

Varijabla `el` će poprimiti sve vrijednosti iz liste. Međutim, ukoliko želimo svaki element liste povećati za 1 ili uraditi bilo koju modifikaciju pojedinačnih elemenata i spasiti promjenu u listu, ovakav način prolaska kroz listu neće biti od pomoći. Stoga, umjesto prolaska kroz elemente liste, često je od velike koristi prolazak kroz svaki indeks u listi. Da bi znali tačan broj indeksa, neophodno je izračunati dužinu liste, što se jednostavno radi funkcijom `len`. U primjeru sa parnim brojevima, `len(parni_brojevi)` će vratiti vrijednost 6, jer lista sadrži 6 elemenata. Stoga, da bi prošli kroz sve indekse, trebamo dobiti vrijednosti 0, 1, 2, 3, 4, 5 i 6, što možemo postići korištenjem naredbe `range(len(parni_brojevi))`. Dakle, kod u kome svaku vrijednost u postojećoj listi parnih brojeva listi povećavamo za jedan, naveden je ispod.

```
for i in range(len(parni_brojevi)):
   parni_brojevi[i] = parni_brojevi[i] + 1
```

Brojač `i` prolazi kroz sve indekse liste, te svaki element mijenja starom vrijednošću uvećanom za 1. U slučaju negativnog indeksa `-i`, programski jezik Python uzima `i`-ti element sa kraja liste.

7.3 Ostale operacije sa listama

Zbog njihove široke primjene, Python nudi niz ugrađenih funkcija i operatora koje olakšavaju rad sa listama.

7.3.1 Spajanje listi

Neka su date dvije liste `l1 = [1, 2, 3]` i `l2 = [4, 5]`. Sabiranjem listi dobijamo novu listu koja sadrži sve elemente iz druge liste dodane na elemente prve liste. Rezultat sabiranja `l1+l2` bila bi lista `[1, 2, 3, 4, 5]`.

7.3.2 Izdvajanje elemenata

Osim pristupa elementima liste pomoću indeksa, moguće je izdvojiti dio liste kao novu listu. Neka je data lista `lista`. Naredbom `lista[od:do]` možemo izdvojiti određenu podlistu. Ukoliko je `l= [1, 2, 3, 4, 5, 6]`, naredba `l[2:4]` vraća listu `[3,4]`. Kao što se može uočiti, posljednji indeks (4) nije uključen u izdvajanje. Ukoliko je `od` izostavljeno, uzimamo elemente sa početka liste. Stoga, `l[:3]` izdvaja listu `[1, 2, 3]`. Ukoliko je `do` izostavljeno, vraća se lista od početne pozicije do kraja. U primjeru `l[2:]` dobijamo listu `[3, 4, 5, 6]`. Ukoliko su obje vrijednosti izostavljene, `l[:]` vraća kompletnu listu.

7.3.3 Pretraga liste

Traženje elementa u listi moguće je izvesti putem operatora `in`. Ovaj operator provjerava da li se element nalazi u listi, te vraća `True` ili `False` kao odgovor. Generalni oblik za upotrebu operatora je: `element in lista`. Ukoliko je data lista `li = [1, 2, 3, 4]`, provjera `4 in li` vraća `True`, dok `100 in li` vraća `False`.

7.3.4 Dodavanje elemenata

Jedna od najčešćih operacija sa listama jeste dodavanje elemenata. Dodavanje elemenata u listu radi se pomoću metode `append`. Ukoliko postoji lista `l` sa proizvoljnim elementima i proizvoljnim brojem elemenata (može biti i prazna lista), dodavanje elementa se izvršava pomoću metode `append`. Ukoliko na listu dodajemo broj 5, koristimo `l.append(5)`. Nakon izvršavanja ove metode, element 5 je dodan na kraj liste.

7.3.5 Pretraga indeksa

Jedna od korisnih metoda za rad sa listama je metoda `lista.index(el)`. Ova metoda vraća indeks prvog pojavljivanja elementa `el` u listi `lista`. U slučaju da se element ne nalazi u listi, program javlja grešku.

7.3.6 Dodavanje na određenu poziciju

Osim metode `append`, za dodavanje elementa u listu `lista` koristi se metoda `lista.insert (indeks, el)`. Ova metoda na poziciju `indeks` umeće element `el`. Element koji se ranije nalazio na poziciji umetnutog elementa, kao i svi ostali elementi se jednostavno pomjeraju za jednu poziciju u listi. Ukoliko je zadani indeks negativan, element se dodaje na početak liste. Ukoliko je zadani indeks veći od dužine liste, element se dodaje na kraj.

7.3.7 Uklanjanje elemenata

Uklanjanje elementa se izvodi pomoću metode `remove`, čija je upotreba u listi `lista` zadata kao `lista.remove(el)`. Ova metoda uklanja prvo pojavljivanje specificiranog elementa `el` u listi.

U slučaju da želimo ukloniti element sa određene pozicije (bez obzira koji je element), koristimo naredbu `del`. Ukoliko je data `lista`, uklanjanje elementa na indeksu `i` radi se pomoću naredbe `del lista[i]`.

7.3.8 Kopiranje liste

Neka je data lista `l1 = [1, 2, 3, 4, 5]`. Neka je data naredba `l2 = l1`. Iako je za primitivni tip varijabli zamišljeno da se kreira kopija kompletne vrijednosti, u slučaju listi se to ne dešava. U pozadini, varijabla `l1` referencira na mjesto u memoriji u kojem je smještena lista brojeva. Nakon naredbe `l2=l1`, varijabla `l2` se također referencira na to isto mjesto u memoriji. Dakle, ne radi se kopiranje liste. Posljedica ovoga je jedna pozitivna činjenica da u slučaju liste sa milion elemenata, nećemo raditi kopiranje liste, nego će obje varijable pokazivati na istu listu. Ovo značajno štedi memoriju, međutim, istovremeno stvara problem, jer u slučaju modifikacije bilo kojeg od ovih milion elemenata, obje varijable se referenciraju na modifikovanu listu. Ovo može izazvati probleme u rješavanju zadataka, jer često programer ne želi mijenjati *obje liste*, tj. ne želi da se promjena očituje i na listi na koju referencira `l1` i `l2`.

Postoji više načina za kopiranje liste, jedan od najjednostavnijih je upotreba sabiranja listi.

```
l1 = [1, 2, 3, 4]
l2 = [] + l1
```

Ovim je kreirana kopija liste. Još jedan od načina je upotreba naredbe `l2 = l1[:]`. Također, jedan od načina koji se često koristi je kreiranje prazne liste, te pojedinačni prolazak kroz originalnu listu (petljom) i dodavanje svakog elementa u novu listu. Novije verzije Python-a (3.3 i više) uvode metodu `copy`. Primjer upotrebe ove metode je `l2 = l1.copy()`.

7.3.9 Okretanje liste

Jedna od korisnih metoda je i `reverse`. Ova metoda mijenja originalni raspored elemenata u listi, tako da prvi element postaje posljednji, drugi postaje pretposljednji ... U slučaju liste `lista = [1, 20, 3]`, nakon naredbe `lista.reverse()`, originalna lista postaje `[3, 20, 1]`.

7.3.10 Sortiranje listi, `min` i `max`

Osim metode `reverse`, jedna od najvažnijih metoda u radu sa listama je metoda `sort`. Ova metoda sortira elemente u liste u rastućem poretku. Njen poziv je `lista.sort()`. Nakon ovog poziva, originalna lista sa elementima `[3, 20, 1]` postaje lista sa elementima `[1, 3, 20]`. U slučaju da listu želimo sortirati opadajuće, koristimo `lista.sort(reverse = True)`.

Sortiranje liste je jedna od najznačajnijih operacija sa listama. Ono može značajno pomoći u rješavanju niza zadataka. Osim toga, neke od najvažnijih funkcija sa listama su i `min(lista)` i `max(lista)`. Ove funkcije redom vraćaju najmanji i najveći element liste. Iako svaka od navedenih metoda i funkcija može biti od značaja i ubrzati implementaciju rješenja, od velike je važnosti znati detalje implementacije svake od ovih metoda. Poznavanje implementacije funkcije sortiranja, traženja minimuma i maksimuma, okretanja liste ili niza drugih funkcija i metoda, povećava mogućnost manipulacije programera i daje veću slobodu u razmišljanju. Na osnovu ranije navedenog, nismo sigurni kako bi sortiranje brojeva uradili u slučaju da želimo sortirati brojeve po broju djelilaca, a ne po veličini. U tom slučaju broj 4 bi bio iza broja 5, jer broj 4 ima 3 različita djelioca, a broj 5 samo 2. Razmislite!

Primjer 7.1 Rješenje zadatka sa pronalaskom srednjeg elementa unesenih brojeva je navedeno ispod.

```
def vrati_srednji_element(l):
    l.sort()
    return l[len(l) // 2]
lista = []
while True:
    n = int(input())
    if n <= 0:
        break
    lista.append(n)
element = vrati_srednji_element(lista)
print(element)
```

Svaki uneseni element smješta se u listu, lista se zatim sortira, te se na kraju uzima element iz sredine sortirane liste. ■

Kako je moguće uočiti u kodu, liste se mogu proslijediti kao argumenti funkcijama, kao i svaki drugi podatak. Također, funkcije mogu vratiti listu kao rezultat.

7.4 Dvodimenzionalne liste

U uvodu poglavlja navedeno je da liste mogu sadržavati elemente proizvoljnog tipa, pa čak i ako je taj tip druga lista.

Liste koje sadrže druge liste nazivaju se dvodimenzionalnim listama. Najčešće se u radu sa dvodimenzionalnim listama podrazumijeva da je svaka lista unutar osnovne liste jednakih dimenzija, iako to ne mora biti slučaj.

```
mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Komentari 1 (predstavljanje u 2D)
# 1 2 3
# 4 5 6
# 7 8 9

# Komentari 2 (indeksi)
# 00 01 02
# 10 11 12
# 20 21 22

print(mat[2][0])
```

U primjeru iznad, definisana je lista čiji je svaki element nova lista. Te elemente možemo predstaviti u dvodimenzionalnom obliku (kao u prvim komentarima). Objekat u kome je dimenzija svake liste unutar osnovne liste jednaka, naziva se matrica. Pristupanje elementima se rješava upravo preko pristupanja elementima liste. Naredba `mat[2]` uzima treći element iz osnovne liste. U ovom slučaju, treći element je također lista `[7, 8, 9]`. Samim tim, `mat[2][0]` uzima prvi element iz treće liste, a to je broj 7. Indeksi svakog elementa u matrici iz primjera navedeni su u drugom dijelu komentara u kodu iznad.

Prolazak kroz svaki element matrice (ili proizvoljne dvodimenzionalne liste) rješava se preko dvostruke petlje. Kod za ispis pojedinačnih elemenata matrice iz primjera naveden je ispod.

```
for i in range(len(mat)):
   for j in range(len(mat[i])):
      print(mat[i][j]))
```

Ovaj kod će ispisati svaki element početne dvodimenzionalne liste jedan ispod drugog. Vanjska petlja prolazi kroz osnovnu listu. Za svaki element osnovne liste, unutrašnja petlja prolazi kroz elemente tog elementa. Drugačije rečeno, vanjska petlja prolazi kroz redove dvodimenzionalne liste, dok unutrašnja petlja za svaki red prolazi kroz sve kolone.

7.5 Zadaci

Zadatak 7.1.

Korisnik unosi dvocifrene cijele brojeve jedan ispod drugog, a kraj unosa označava praznim redom. Potrebno je provjeriti da li se posljednji uneseni broj obrnut (zamjenjena prva i druga cifra) pojavljuje među prethodnim brojevima. Ukoliko se nalazi program ispisuje `Da`, a u suprotnom ispisuje `Ne`.

U narednom primjeru uneseno je pet vrijednosti, a posljednja vrijednost je 61. Kada se obrnu cifre tog broja dobijemo vrijednost 16 koja se nalazi u listi, te stoga program ispisuje `Da`. [Napomena. Da je umjesto 61 posljednji uneseni broj bio 11, program bi trebao ispisati `Ne`].

Input:

```
32
16
92
25
61
```

Output:

```
Da
```

Zadatak 7.2.

Korisnik unosi riječi jednu ispod druge, a kraj unosa označava praznim redom. Program ispisuje riječi koje su duže od prosječne dužine svih unesenih riječi. Riječi se ispisuju jedna ispod druge u redoslijedu u kojem su unesene.

Input:

```
programiranje
skup
lista
internet
ABG
tastatura
rjecnik
```

Output:

```
programiranje
internet
tastatura
```

Zadatak 7.3.

Korisnik unosi broj paran broj m. Nakon toga korisnik unosi m cijelih brojeva. Program oduzima zbir $m/2$ najmanjih brojeva od zbira $m/2$ najvećih brojeva i ispisuje rezultat. U primjeru ispod, korisnik unosi 6 vrijednosti, a program oduzima zbir 3 najmanje vrijednosti (2, 1, i 4) od zbira 3 najveće vrijednosti (10, 5 i 4):

Input:

```
6
10
2
1
4
5
4
```

Output:

```
12
```

Zadatak 7.4.

Korisnik unosi nazive gradova. Kraj unosa označava praznim redom. Ukoliko je više gradova koja počinju sa slovom S od onih koji počinju sa slovom B program ispisuje sve gradove koji počinju sa slovom S, jedan nakon drugog, u istom redoslijedu u kojem su uneseni, odvojeni praznim mjestom. Ukoliko je više onih sa početnim slovom B, ispisuju se ti gradovi. A ukoliko ih je jednak broj,

ispisuju se i gradovi koji počinju sa slovom S i sa slovom B u istom redoslijedu u kojem su uneseni.

Input:

```
Sarajevo
Bihac
Tuzla
Sanski Most
Banja Luka
Mostar
```

Output:

```
Sarajevo Bihac Sanski Most Banja Luka
```

Zadatak 7.5.

Korisnik unosi paran broj cijelih brojeva. Kraj unosa označava praznim redom. Vrijednosti se smještaju u listu. Potom, program pronalazi najmanji broj u prvoj polovini liste x i najveći broj u drugoj polovini liste y. Program ispisuje svaki drugi broj u rasponu od x do y. Primjer:

Input:

```
4
2
15
8
12
1
```

Output:

```
2 4 6 8 10 12
```

Zadatak 7.6.

Napisati program koji od korisnika zahtjeva unos 5 brojeva jedan ispod drugog. Program ispisuje treći najveći broj.

Input:

```
5
-1
4
4
0
```

Output:

```
4
```

Zadatak 7.7.

Program od korisnika prvo traži broj `n`. Nakon toga korisnik unosi `n` cijelih brojeva koje program smješta u listu. Nakon unosa ovih brojeva korisnik unosi još i vrijednost `m`. Program treba ukloniti `m` najmanjih i najvećih vrijednosti iz liste. Preostale vrijednosti trebaju biti sortirane od najmanje ka najvećoj i ispisane u istom redu razdvojene praznim mjestom.

U primjeru ispod `n` je 4 i govori nam da korisnik planira unijeti četiri broja, konkretno: 2 6 8 4. Posljednji broj je 1 i to je `m`. On nam govori da želimo ukloniti jednu najmanju i jednu najveću vrijednost. U našem slučaju najmanja vrijednost je 2, a najveća 8 tako da program ispisuje 4 i 6, tj. preostale brojeve u listi sortirane.

Input:

```
4
2
6
8
4
1
```

Output:

```
4 6
```

Zadatak 7.8.

Napisati program u koji korisnik unosi 10 cijelih brojeva, svaki u novom redu. Nakon unosa program prvo ispisuje sve negativne brojeve, zatim sve nule i na kraju sve pozitivne brojeve. Unutar svake od grupa brojevi trebaju biti u originalnom redoslijedu u kojem su uneseni. Prilikom ispisa brojevi se ispisuju u istom redu, a odvojeni su jednim praznim mjestom.

Na primjer, za unesene vrijednosti 3, -4, 8, 0, -1, 0, -6, 1, -1, 0 program treba ispisati:

Ispis 7.1: Output:

```
-4 -1 -6 -1 0 0 0 3 8 1
```

Zadatak 7.9.

Korisnik unosi cijele pozitivne brojeva jedan ispod drugog, a kraj unosa označava vrijednošću -1. Rotiranje broja podrazumijeva proces u kojem prva cifra postaje posljednja, druga pretposljednja itd. Program ispisuje razliku najvećeg rotiranog i najmanjeg nerotiranog broja.

U narednom primjeru najveći rotirani broj je 7822, a najmanji nerotirani broj je 23, pa je njihova razlika 7799

Input:

```
23
511
90
2287
8921
-1
```

Output:

```
7799
```

Zadatak 7.10.

Korisnik unosi neparan broj m, a nakon toga m cijelih brojeva. Brojevi se smještaju u listu. Broj koji se nalazi u sredini liste nazivamo pivot. Potrebno je ispisati brojeve tako da će se sa lijeve strane pivota ispisati sve vrijednosti koje su manje ili jednake pivotu u istom redoslijedu u kojem su bile i u originalnoj listi, a sa desne sve vrijednosti koje su veće. Brojevi se ispisuju jedan nakon drugog razdvojeni praznim mjestom. U narednom primjeru korisnik za m unosi vrijednost 7, nakon čega unosi sedam vrijednosti. Na sredini liste je broj 4 koji postaje pivot. Pošto su vrijednosti 3 i 2 manje ispisuju se sa lijeve strane pivota, a ostale vrijednosti su veće pa se ispisuju sa desne strane.

Input:

```
7
7
3
9
4
6
2
8
```

Output:

```
3 2 4 7 9 6 8
```

Zadatak 7.11.

Napisati program u koji korisnik unosi broj m, a nakon toga unosi m cijelih brojeva koji se smještaju u listu. U nastavku korisnik unosi parove indeksa, a kraj unosa označava vrijednošću -1. Sve vrijednosti se unose jedna ispod druge. Program vrši zamjenu vrijednosti u listi na osnovu parova indeksa. Kao rezultat se ispisuju vrijednosti liste nakon svih izvršenih zamjena. Vrijednosti se ispisuju jedna poslije druge odvojene praznim mjestom.

U primjeru ispod korisnik je prvo unio vrijednost 3, pa nakon toga tri vrijednosti koje se smještaju u listu. Naredne vrijednosti su parovi indeksa koji govore da je potrebno zamijeniti nulti (vrijednost 7) sa drugim (vrijednost 8) elementom liste, a potom prvi (vrijednost 2) sa nultim elementom (sada vrijednost 8).

Input:

```
3
7
2
8
0
2
1
0
-1
```

Output:

```
2 8 7
```

Zadatak 7.12.

Korisnik unosi prirodne brojeve m i n jedan ispod drugog. Zatim korisnik nastavlja unoseći $m+n$ cijelih brojeva jedan ispod drugog.

Prvih m brojeva je dobio Harun, a ostalih n Sead. Ukoliko je moguće da Sead izbaci jedan od svojih brojeva tako da sume Harunovih i Seadovih brojeva budu jednake program ispisuje `jeste`, a u suprotnom ispisuje `nije`.

Input:

```
3
3
1
2
3
3
4
3
```

Output:

```
jeste
```

Zadatak 7.13.

Korisnik u program unosi cijele brojeve, a kraj unosa označava praznim redom. Program treba ispisati unesene brojeve u jednom redu razdvojene praznim mjestom, ali da su parne vrijednosti sortirane, a neparne ostaju na poziciji na kojoj su unesene. Primjer:

Input:

```
9
8
3
5
2
4
7
```

Output:

```
9 2 3 5 4 8 7
```

Zadatak 7.14.

Razmotrimo prvo i posljednje pojavljivanje neke vrijednosti u listi. Definišimo **raspon** kao broj elemenata liste koji se nalazi između ove dvije vrijednosti (uključujući i njih). Ukoliko se neka vrijednost pojavljuje samo jednom u listi, onda ona ima raspon 1. Napisati program koji od korisnika traži unos pozitivnih cijelih brojeva koje smješta u listu. Korisnik unosi brojeve u jednom redu razdvojene jednim praznim mjestom. Program ispisuje najveći raspon u listi.

Na primjeru ispod, najveći raspon je između dvije dvojke (one na poziciji 2 i na poziciji 7). Između njih se nalaze četiri vrijednosti, pa je zajedno sa njima raspon 6.

Input:

```
1 4 2 1 2 4 35 2 7
```

Output:

```
6
```

Zadatak 7.15.

Niz od dva ili više jednakih uzastopnih elemenata liste nazvat ćemo skupinom. Napisati program koji od korisnika traži unos pozitivnih cijelih brojeva koje smješta u listu. Korisnik unosi brojeve u jednom redu razdvojene jednim praznim mjestom. Program ispisuje broj skupina u listi. U narednom primjeru nalaze se četiri skupine i to: tri jedinice, dvije četvorke, dvije petice i ponovo dvije petice.

Input:	Output:
`1 1 1 4 4 3 5 5 2 5 5 3`	`4`

Zadatak 7.16.

Napisati program koji od korisnika zahtijeva unos liste vrijednosti. Vrijednosti se unose u jednom redu razdvojene praznim mjestom. Nakon što unese listu korisnik unosi vrijednost n u narednom redu. Program treba ispisati razliku između najvećeg i n-tog najvećeg broja u listi. Možete pretpostaviti da korisnik nikada neće unijeti n veće od dužine liste. U narednom primjeru najveća vrijednost u listi je 98, a treća najveća 83, pa je njihova razlika 15.

Input:	Output:
`39 98 16 83 78 66 95 13 58` `3`	`15`

Zadatak 7.17.

Napisati program koji od korisnika uzima neki pozitivan cijeli broj a_1. Na osnovu a_1 može se kreirati sekvenca vrijednosti tako što se svaka naredna vrijednost u sekvenci a_k dobije sabiranjem svih cifara prethodne vrijednosti a_{k-1}, te se ovaj zbir spoji sa a_{k-1} sa desne strane. Npr. ukoliko je $a_1 = 98$ onda je $a_2 = 9817$, jer zbir cifara a_1 je 17, pa kada se ovaj rezultat doda sa desne strane a_1 dobije se vrijednost 9817.

Program treba ispisati prvi broj u sekvenci koji u sebi sadrži sve jednocifrene brojeve barem jedanput.

Input:	Output:
`42`	`426121521243033395157698 4`

Zadatak 7.18.

Data je lista cijelih brojeva. Brojevi se unose jedan poslije drugog i razdvojeni su praznim mjestom. Za element kažemo da je lokalni maksimum ukoliko je striktno veći od prethodnog i narednog elementa. Program treba pronaći i ispisati broj elemenata koji se nalazi između prvog i posljednjeg maksimuma u listi, uključujući i njih. Prvi i posljednji *element liste* se nikada ne tretiraju kao lokalni maksimumi. U narednom primjeru se vidi da je prvi lokalni maksimum broj 5 koji se nalazi na četvrtoj poziciji, a posljednji lokalni maksimum je broj 10 koji se nalazi na pretposljednjoj poziciji. Program ispisuje 8 zato što se između ova dva maksimuma uključujući i njih nalazi 8 elemenata.

Input:	Output:
`4 2 -1 5 3 8 75 42 3 7 10 6`	`8`

Zadatak 7.19.

Napisati program koji od korisnika uzima neparan broj `n`. Nakon toga program od korisnika uzima `n` cijelih brojeva i smješta ih u listu. Program izbacuje medijanu iz liste i ispisuje listu sa `n - 1` elemenata u originalnom redoslijedu. Elementi se ispisuju u istom redu razdvojeni praznim

mjestom. [Napomena. Medijana je broj koji razdvaja gornju polovinu liste od donje, tj. vrijednost koja se nalazi u sredini sortirane liste.]

Input:

```
5
1
8
6
4
2
```

Output:

```
1 8 6 2
```

Zadatak 7.20.

Napisati program koji od korisnika prvo uzima cijeli broj n. Nakon toga korisnik unosi $n \times n$ vrijednosti koje formiraju matricu (prvo se popunjava prvi red, zatim drugi itd.) Program ispisuje sumu svake kolone matrice, svaku u zasebnoj liniji.

Input:

```
2
1
2
3
4
```

Output:

```
4
6
```

Zadatak 7.21.

Napisati program koji od korisnika prvo uzima cijeli broj n. Nakon toga korisnik unosi $n \times n$ vrijednosti koje formiraju matricu (prvo se popunjava prvi red, zatim drugi itd.) Program treba pronaći sume dviju dijagonala matrice, i ispisati njihov proizvod.

Input:

```
2
1
2
3
4
```

Output:

```
25
```

Zadatak 7.22.

Korisnik unosi cijele brojeve u istom redu razdvojene praznim mjestom. Ispisati za koliko brojeva vrijedi da su veći od prethodno unesenog broja. Zatim ispisati za koliko brojeva vrijedi da su veći od zbira svih pozitivnih brojeva koji su uneseni nakon njega.

Input:

```
1 3 5 7 5 3 -7 1
```

Output:

```
4
2
```

Zadatak 7.23.

Anagram je vrsta zagonetke u kojoj se premetanjem slova neke riječi ili izraza dobija nova riječ ili izraz. Npr. premetanjem slova u riječi "ANAGRAMI" može se dobiti izraz "NAMA IGRA". Jedan od načina na koji se može predstaviti tačno rješenje anagrama je specificirajući poziciju koju će svako originalno slovo zauzeti u novom izrazu. Tako je rješenje za prethodni anagram `[1, 0, 3, 6, 7, 8, 2, 5]` zato što prvo slovo A originalne riječi ide na prvu poziciju rješenja, slovo N ide na nultu poziciju rješenja, naredno slovo A ide na treću poziciju rješenja itd. Na listi se ne nalazi broj 4 jer se na tom indeksu rješenja nalazi prazno mjesto pa se ni jedno slovo ne premješta na tu poziciju.

Stvari se dodatno komplikuju upotrebom Fibonaccijevih brojeva za predstavljanje pozicija. Tako ukoliko neko slovo treba premjestiti na nultu poziciju, to će se označiti prvim Fibonaccijevim brojem (1), ukoliko slovo treba premjestiti na prvu poziciju označit će se drugim Fibonaccijevim brojem (2), ukoliko slovo treba premjestiti na četvrtu poziciju označit će se petim Fibonaccijevim brojem (8) i sl. Tako da je rješenje za prethodni primjer koje koristi Fibonaccijeve brojeve: [2, 1, 5, 21, 34, 55, 3, 13]

U ovom zadatku koristi se sljedeća Fibonaccijeva sekvenca: 1, 2, 3, 5, 8, 13, 21...

Korisnik unosi riječ ili izraz u jednom redu, a u narednom redu unosi niz koji se sastoji od Fibonaccijevih brojeva koji predstavljaju tačno rješenje anagrama, tj. pozicije na koje treba postaviti slova. Program razmatra samo znake alfabeta i ne premješta prazna mjesta ili znake interpunkcije. Program ispisuje rješenje anagrama velikim slovima. Anagram nikada neće biti duži od 50 slova.

Input:

```
Krasan je odmor!
34 5 2 21 8 13 1 610 55 3 144 233 377
```

Output:

```
JADRANSKO MORE
```

Zadatak 7.24.

Naziv "sekvenca u ogledalu" dat ćemo dijelu liste, u kojem se uzastopni elementi tog dijela pojavljuju još negdje u listi, u obrnutom redoslijedu. Napisati program koji od korisnika traži unos pozitivnih cijelih brojeva koje smješta u listu. Korisnik unosi brojeve u jednom redu razdvojene jednim praznim mjestom. Program ispisuje dužinu najduže "sekvence u ogledalu" u listi. Npr. ukoliko su elementi liste: `[4, 8, 2, 1, 9, 2, 8, 4]` program treba ispisati 3 jer se sekvenca `[4, 8, 2]` pojavljuje u obrnutom redoslijedu u listi `[2, 8, 4]`. Također, program ispisuje 3 za listu `[5, 6, 5, 2]` jer se sekvenca `[5, 6, 5]` pojavljuje u suprotnom smjeru u listi.

Input:

```
7 10 5 3 4 5 10 1 5
```

Output:

```
2
```

Zadatak 7.25.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. Svaki red u fajlu sadrži naredne podatke: naziv grada, država u kojoj se grad nalazi i broj stanovnika tog grada. Svaki red je formatiran na način da se prvo ispisuje naziv grada, zatim zarez i prazno mjesto, potom naziv države, zarez i prazno mjesto, i na kraju broj stanovnika. Program treba da ispiše nazive najvećih gradova (po broju stanovnika) za svaku od država. Najveći gradovi se ispisuju sortirani abecedno, svaki u zasebnom redu.

test01.in

```
Banja Luka, BiH, 150000
Sarajevo, BiH, 350000
London, UK, 8200000
Birmingham, UK, 1000000
New York, USA, 18200000
Washington, USA, 5500000
```

Output:

```
London
New York
Sarajevo
```

Zadatak 7.26.

Napisati program koji od korisnika prvo uzima cijeli broj n. Nakon toga korisnik unosi $n \times n$ vrijednosti koje formiraju matricu (prvo se popunjava prvi red, zatim drugi itd.) Program ispisuje matricu u kojoj su sortirane sve kolone.

Matrica se ispisuje tako što se prvi red ispisuje u prvoj liniji, i elementi su odvojeni praznim mjestom, zatim se ispisuje drugi red u narednoj liniji itd.

Input:

```
3
5
9
8
1
4
7
1
2
0
```

Output:

```
1 2 0
1 4 7
5 9 8
```

Zadatak 7.27.

Sudoku je igra u kom je cilj kvadrat sa mrežom 9×9 ispuniti brojevima od 1 do 9 i to tako da se u svakom redu, koloni, kao i kvadratu 3×3, pojedini broj nalazi samo jednom.

U ovom zadatku će se razmatrati samo jedan 3×3 kvadrat. Kvadrat je ispravno popunjen ukoliko se u njemu nalaze brojevi od 1 do 9, a svaki broj se ponavlja samo jednom.

Napisati program u koji korisnik unosi tri reda sa po tri cijela broja koji su razdvojeni praznim mjestom. Program **ponovo ispisuje kvadrat** i ispod riječ `ispravan` ukoliko je uneseni kvadrat ispravan. Program **ponovo ispisuje kvadrat** i riječ `neispravan` ukoliko kvadrat nije validno popunjen (tj. ukoliko se neki broj ponavlja više puta ili ukoliko je unesen neki broj koji nije u rasponu od 1 do 9).

Input:

```
3 8 9
4 1 7
2 6 5
```

Output:

```
3 8 9
4 1 7
2 6 5
ispravan
```

Input:

```
3 18 9
4 5 7
2 6 5
```

Output:

```
3 18 9
4 5 7
2 6 5
neispravan
```

Zadatak 7.28.

Korisnik unosi cijele brojeve u istom redu razdvojene praznim mjestom. Program ispisuje najveću sumu uzastopnih brojeva datog niza. Ukoliko je niz sačinjen on negativnih brojeva, rezultat je 0 (jer se može uzeti 0 brojeva i tada je zbir 0).

Input:

```
1 2 3 -7 4 -1 2 3 4
```

Output:

```
12
```

8. Stringovi

Kroz implementaciju ranijih programa, susretali smo se sa stringovima koje su korisnici unosili kroz konzolu, kao i sa string literalima koje smo ispisivali kao poruke korisniku. Osim standardnog unosa i ispisa stringova u nepromijenjenom obliku, stringove je često neophodno modifikovati za upotrebu ili detaljnije analizirati. U poglavlju ćemo string literale često kraće nazivati stringovima.

Jedan od najzastupljenijih primjera je provjera lozinke i korisničkog imena pri pristupu sistemu. Lozinke često sadrže ograničenja koja je neophodno provjeriti prije spašavanja i prijave. Ta ograničenja mogu biti u dužini (lozinke mora biti bar 8 znakova), u tipu sadržanih znakova (lozinka mora sadržavati bar jedno veliko i bar jedno malo slovo, te jedan specijalni znak i jednu cifru). Za svaku od ovih provjera, neophodno je detaljno analizirati string.

8.1 Prolazak kroz znakove stringa

Prvi neophodni element u analizi stringa je način pristupanja pojedinačnim **znakovima**. Prolaskom petljom kroz string u stanju smo pristupiti svakom pojedinačnom znaku sadržanom u stringu na dva načina.

Prvi način koristi petlju u obliku `for c in s`, gdje dobijamo svaki znak stringa kroz varijablu `c`, a string koji se analizira je `s`. Sa ovim znakovima možemo raditi mnoge modifikacije i analize. Drugi način prolaska kroz string podrazumijeva pristupanje svakom elementu putem indeksa, slično kao u procesu rada sa listama.

```
s = 'Hello world'
for c in s:
    print(c)
```

Kod iznad ispisuje svako slovo u narednom redu. Drugi opisani način za prolazak kroz string, naveden je ispod. Za potrebe određivanja indeksa prolaska, neophodno je koristiti `len` funkciju koja prihvata string i vraća njegovu dužinu.

```
s = 'Hello world'
for i in range(len(s)):
   print(s[i])
```

Kod iznad također ispisuje svako slovo u narednom redu. Negativni indeksi se ponašaju kao u slučaju listi. Neophodno je paziti da indeks kojem pristupamo ne izlazi iz raspona dužine stringa, u tom slučaju se javlja greška.

8.2 Modifikacija stringa, ASCII kodovi

Stringovi nisu promjenjivi za razliku od nizova. Dakle, nije dozvoljeno koristiti naredbu `s[0] = 'a'`. U slučaju da želimo u neku varijablu smjestiti promijenjen string, najjednostavniji način je kreiranje novog stringa, te dodavanje znakova na taj string.

Svaki znak se predstavlja koristeći ASCII šifru. **ASCII šifre** se koriste za slova engleskog alfabeta, standardne znakove sa tastature, kao i za cifre. Važno je napomenuti da ne postoje ASCII šifre za dijakritičke znakove (slova sa kvačicama i crticama). Svaki slovo ima svoj jedinstveni kod.

Znak `A` ima šifru 65, znak `B` ima šifru 66, znak `C` šifru 67 ... S druge strane, znak `a` ima kod 97, znak `b` ima kod 98 ... Važno je primijetiti da je razlika u kodovima svakog velikog i malog slova jednaka, te da se kodovi povećaju za 1 svakim narednim slovom alfabeta. Slično vrijedi za cifre, pa cifra 0 ima kod 48, cifra 1 ima kod 49...

Kodove svakog znaka nije neophodno pamtiti, jer u svakom trenutku možemo dobiti tačan kod nekog znaka `zn`. U tom slučaju, koristimo funkciju `ord(zn)`. S druge strane, u slučaju da želimo broj `br` pretvoriti u odgovarajući znak, koristimo naredbu `chr(br)`. Ove naredbe se često koriste u manipulaciji sa stringovima ili ispisima sačinjenim od slova. Npr. ukoliko želimo se želimo sa znaka `A` premjestiti na znak `B`, koristimo naredbu `chr(ord('A')+1)`. Postupak se svodi na uzimanje koda znaka `A`, dodavanje broja 1 na taj kod, te pretvaranja dobijenog koda u odgovarajući znak (što kao rezultat daje `B`).

U slučaju da želimo svaki znak u stringu pomjeriti za jednu poziciju, možemo koristiti kod naveden ispod.

```
s = 'ABCD'
r = ''

for c in s:
   r = r + chr(ord(c)+1)

print(r)
```

U kodu iznad, za svaki znak početnog stringa `s`, radimo pomjeranje za jednu poziciju i nadovezujemo na rezultujući string `r`.

U slučaju da je neophodno vršiti replikaciju stringa, dovoljno je isti pomnožiti sa željenim brojem. Stoga, ukoliko je dat string `s = 'a'`, tada će `s * 5` rezultirati sa `'aaaaa'`. Navedeni operator često se može iskoristiti u zadacima sa ispisima raznih oblika, gdje umjesto petlje za ispis n pojavljivanja nekog znaka, koristimo ovaj jednostavni operator.

8.3 Operatori, metode i funkcije u radu sa stringovima

Python nudi niz ugrađenih funkcije i operatora koji olakšavaju rad sa stringovima.

8.3.1 Izdvajanje stringa

Slično kao u radu sa listama, naredbom `string[od:do]` možemo uraditi izdvajanje određenog dijela stringa. U slučaju da nedostaje vrijednost `od` ili `do`, ponašanje operatora je jednako kao u radu sa listama.

8.3.2 Pretraga stringa

Python dozvoljava korištenje operatora `in` i `not in` za provjeru da li prvi string pripada drugom. Operatori se koriste kao `s1 in s2` gdje dobijamo podatak da li se string `s1` nalazi unutar stringa `s2`.

```
s = 'Hello world'
if 'world' in s:
   print('Da')
else:
   print('Ne')
```

Operator `not in` provjerava da li se string `s1` **ne** nalazi u stringu `s2`, te se operator piše kao `s1 not in s2`.

8.3.3 Metode testiranja

Postoji niz metoda za testiranje stringa. Prva metoda je metoda `isdigit()` koja provjerava da li se string sastoji u potpunosti od cifara. U slučaju da imamo string `s = '123'`, naredba `s.isdigit()` vraća vrijednost `True`. S druge strane, u slučaju da je string `s = '12a'`, povratna vrijednost je `False`.

Osim ovih metoda, postoji i niz drugih.

- `isalnum()`. Ova metoda vraća vrijednost `True` ukoliko je sačinjen od alfanumeričkih znakova (slova i brojevi). Ukoliko je string dužine 0 ili ne sadrži samo alfanumeričke znakove, vraća `False`.
- `isalpha()`. Vraća vrijednost `True` u slučaju da je string dužine veće od 0 i sadrži samo slova.
- `islower()` Vraća vrijednost `True` u slučaju da je string dužine veće od 0 i sadrži samo mala slova.
- `isspace()` Vraća vrijednost `True` u slučaju da je string dužine veće od 0 i sadrži samo prazna mjesta (znak `' '`).
- `isupper()` Vraća vrijednost `True` u slučaju da je string dužine veće od 0 i sadrži samo velika slova.

Na osnovu iznad navedenog, jednostavno možemo provjeriti da li je unesena lozinka ispravna u slučaju ograničenja navedenih u uvodu poglavlja.

8.3.4 Metode modifikacije

Postoji niz metoda za modifikaciju stringa. Svaka metoda vraća kopiju originalnog stringa sa izvršenom prikladnom modifikacijom. Metode modifikacije navedene su ispod:

- `lower()`. Ova metoda vraća kopiju stringa gdje je svako veliko slovo zamijenjeno verzijom istog malog slova.
- `upper()`. Ova metoda vraća kopiju stringa gdje je svako malo slovo zamijenjeno verzijom istog velikog slova.

- `lstrinp(zn)`. Ova metoda vraća kopiju stringa gdje su sve instance znaka `zn` koje se nalaze na početku stringa uklonjene.
- `rstrinp(zn)`. Ova metoda vraća kopiju stringa gdje su sve instance znaka `zn` koje se nalaze na kraju stringa uklonjene.
- `strinp(zn)`. Ova metoda vraća kopiju stringa gdje su sve instance znaka `zn` koje se nalaze na početku i na kraju stringa uklonjene.
- `lstrinp()`. Ova metoda vraća kopiju stringa gdje su sva prazna mjesta koja se nalaze na početku stringa uklonjena.
- `rstrinp()`. Ova metoda vraća kopiju stringa gdje su sva prazna mjesta koja se nalaze na kraju stringa uklonjena.
- `strinp()`. Ova metoda vraća kopiju stringa gdje su sva prazna mjesta koja se nalaze na početku i na kraju stringa uklonjena.

Ove metode se često koriste u zadacima za analizu rečenica unesenih od strane korisnika, čime omogućavamo jednostavnu pretvorbu unosa u format koji je pogodan za analizu i manipulaciju.

8.3.5 Razdvajanje stringa

Naredba `split` koristi se za razbijanje stringa na više elemenata liste. Pretpostavljena vrijednost za razbijanje stringa je znak praznog mjesta. Stoga, ukoliko string `s = 'Danas je lijep dan'` iskoristimo sa metodom `split` (`s.split()`), rezultat je lista koja sadrži elemente `['Danas', 'je', 'lijep', 'dan']`. Ova metoda je od velikog značaja pri rješavanju zadataka u kojima se od korisnika traži unos, te je neophodno raditi analizu *riječ po riječ*.

Moguće je proslijediti parametar metodi `split` koji razdvaja string na isti način. U slučaju da imamo string: `s = 'Danas.je.lijep.dan'`, koristimo naredbu `s.split('.')`, koja će vratiti listu `['Danas', 'je', 'lijep', 'dan']`.

8.4 Zadaci

Zadatak 8.1.

Korisnik unosi tekst u jednom redu. Program pronalazi i ispisuje riječ koja u sebi sadrži najviše slova. Riječ se ispisuje malim slovima. Ukoliko je više riječi ima jednak, a opet najveći broj slova, ispisuju se jedna ispod druge u redoslijedu u kojem su se pojavile u rečenici, također malim slovima.

Input:

```
Popocatepetl je vulkan u sredisnjem Meksiku, a njegov manji parnjak je
   ↪ vulkan Iztaccihuatl. Drugi je vulkan Sjeverne Amerike po visini,
   ↪ poslije Orizabe. To je jedan od aktivnijih vulkana u Meksiku.
   ↪ Zabiljezeno je vise od 20 velikih erupcija ovoga vulkana od
   ↪ dolaska Spanjolaca u Meksiko.
```

Output:

```
popocatepetl
iztaccihuatl
```

Zadatak 8.2.

Korisnik unosi tekst. Program ispisuje samoglasnik (a, e, i, o, u) koji se najčešće ponavlja u toj rečenici. Možete pretpostaviti da će uvijek biti samo jedan pobjednik. Program ne pravi razliku

između velikih i malih slova.

Input:

```
Popocatepetl je vulkan u sredisnjem Meksiku. Njegov manji parnjak je
   ↪ vulkan Iztaccihuatl. Drugi je vulkan Sjeverne Amerike po visini,
   ↪ poslije Orizabe. To je jedan od aktivnijih vulkana u Meksiku.
   ↪ Zabiljezeno je vise od 20 velikih erupcija ovoga vulkana od
   ↪ dolaska Spanjolaca u Meksiko.
```

Output:

```
e
```

Zadatak 8.3.

Korisnik unosi prirodan broj n, a zatim n stringova jedan ispod drugog. Korisnik treba ispisati sve stringove koji su duži od svih prethodnih zajedno.

Input:

```
5
i
da
ali
zavrsni
programiranje
```

Output:

```
i
da
zavrsni
```

Zadatak 8.4.

Korisnik unosi riječi jednu ispod druge. Kraj unosa označava praznim redom. Program ispisuje, jednu ispod druge, riječi koje počinju i završavaju istim slovom, ali se to slovo ne pojavljuje drugdje u riječi. Program ne pravi razliku između velikih i malih slova, i riječi ispisuje malim slovima.

Input:

```
Olovo
test
Sarajevo
naspavan
nenaspavan
Oslo
```

Output:

```
test
naspavan
oslo
```

Zadatak 8.5.

Korisnik unosi prirodne brojeve jedan ispod drugog. Kraj unosa se označava praznim redom. Brojevi se sastoje od neparnog broja cifara. Program ispisuje sve brojeve kojima je srednja cifra veća od 5. Brojevi se ispisuju jedan do drugog razdvojeni praznim mjestom.

Input:

```
183
153
9
28964
886626688
```

Output:

```
183 9 28964
```

Zadatak 8.6.

Napisati program koji korisniku dozvoljava unos stringova, jedan ispod drugog. Svaki string je dužine 1 i može sadržavati slovo ili broj. Kraj unosa se označava **strinogm koji nije dužine 1**. Program ispisuje **zbir svih parnih brojeva**.

Input:

```
A
2
5
T
4
Kraj
```

Output:

```
6
```

Zadatak 8.7.

Napisati program koji korisniku dozvoljava unos rečenica. Svaka rečenica se unosi u novom redu. Kraj unosa korisnik označava praznim redom. Program ispisuje rečenice koje završavaju tačkom i koje su kraće od 20 znakova.

Input:

```
Da dvadeset znakova.
Broj je na granici.
Zasto nema tacke?
Zato, i tacka.
```

Output:

```
Broj je na granici.
Zato, i tacka.
```

Zadatak 8.8.

Napisati program koji od korisnika zahtijeva unos dva cijela broja n i m. Program dodaje decimalnu tačku broju n, i to na poziciju m gledano sa desna na lijevo. Možete pretpostaviti da je m veće od 0, a manje od broja cifara n. U primjeru ispod m je 2 pa se tačka stavlja prije predzadnje cifre n.

Input:

```
593213
2
```

Output:

```
5932.13
```

Zadatak 8.9.

Napisati program koji izbacuje samoglasnike iz riječi. Korisnik unosi riječi jednu ispod druge, a kraj unosa označava praznim redom. Program ispisuje riječi bez samoglasnika jednu ispod druge.

Input:

```
Oblak
ProGramiranje
Koordinata
Prst
```

Output:

```
blk
PrGrmrnj
Krdnt
Prst
```

Zadatak 8.10.

Napisati program koji od korisnika zahtjeva unos neke rečenice. Ukoliko se u rečenici češće pojavljuje slovo A od slova E program ispisuje `A`. Ukoliko se češće pojavljuje slovo E od slova A program ispisuje `E`. Ukoliko se oba navedena slova pojavljuju jednak broj puta program ispisuje `AE`. Program ne treba praviti razliku između velikih i malih slova (tj. broje se i velika i mala slova).

Input:

```
Ovo jE recenica koja se testira.
```

Output:

```
E
```

Zadatak 8.11.

Napisati program u koji korisnik unosi riječi, svaku u zasebnoj liniji. Korisnik označava kraj unosa praznim redom. Program treba ispisati unesene riječi **koje ne počinju slovom 'b'**, odvojene praznim mjestom. Program *pravi razliku* između velikih i malih slova, tako da će ignorisati samo riječi koje počinju malim slovom `b`.

Input:

```
kruska
jabuka
banana
lubenica
```

Output:

```
kruska jabuka lubenica
```

Zadatak 8.12.

Korisnik unosi prirodan broj n, a zatim n stringova jedan ispod drugog. Korisnik treba ispisati svaki string koji je neparne dužine i duži je od svakog prethodno unesenog stringa ili svaki string koji je parne dužine i kraći je od svakog prethodno unesenog stringa. Možete pretpostaviti da nijedan string neće biti dužine veće od 100.

Input:

```
5
aaa
ab
abcdefg
xyz
aaaababcdefgxyza
```

Output:

```
aaa
ab
abcdefg
```

Zadatak 8.13.

Napisati program koji od korisnika uzima riječ. Program zatim provjerava koje je prvo slovo riječi i uklanja svako pojavljivanje tog slova u riječi. Zatim ponovo provjerava prvo slovo u riječi i uklanja svako njegovo ponavljanje. Proces se ponavlja dok riječ nije sastavljena od jednog slova (koje se može ponavljati više puta), što se i ispisuje na ekran. Program ne pravi razliku između velikih i malih slova, a rezultat ispisuje malim slovima.

U narednom primjeru iz riječi `tiRurirU`, se prvo uklanjaju sva slova `t` (u ovom slučaju jedno), zatim sva slova `i`, i sva slova `r`. Ono što preostaje je slovo `u` koje se ponavljalo dva puta, pa se to i ispisuje.

Ispis za n = 2:

```
tiRurirU
```

Ispis za n = 5:

```
uu
```

Zadatak 8.14.

Korisnik unosi rečenicu. Napisati program koji vraća broj riječi u rečenici koje počinju i završavaju sa istim slovom. Program ne pravi razliku između velikih i malih slova. Potrebno je i obratiti pažnju na znakove interpunkcije. U narednom primjeru 6 riječi koje počinju i završavaju istim slovom su: korisnik, ispravi, a, odlicno, i, racunar. Primjeri:

Input:

```
Korisnik greske koje uoci treba da ispravi, a odlicno je da podatke
   ↪ provjeri i racunar.
```

Output:

```
6
```

Zadatak 8.15.

Napisati program koji omogućava korisniku unos dva stringa, jedan ispod drugog. Program provjerava da li je drugi string sadržan u prvom. Znakovi drugog stringa **ne moraju biti uzastopno sadržani u prvom**, ali se moraju pojavljivati svi znakovi i u originalnom redoslijedu. Program ne pravi razliku između velikih i malih slova. Program ispisuje `Nalazi se!` ukoliko je drugi string sadržan u prvom ili `Ne nalazi se!` ukoliko nije. Npr. string `Sarajevo` ne sadrži string `raja` jer se iza znaka `j` u stringu `Sarajevo` ne pojavljuje više znak `a`. U primjeru ispod string `Programiranje I` sadrži string `porani` zato što se svi znakovi drugog stringa nalaze u ispravnom redoslijedu u prvom. [Napomena: u primjeru je rimsko 1 označeno velikim slovom i.]

Input:

```
Programiranje I
porani
```

Output:

```
Nalazi se!
```

Zadatak 8.16.

Prilikom pisanja Word dokumenta, ukoliko je spellchecker postavljen na engleski jezik, svako zasebno malo slovo `"i"` će biti zamijenjeno velikim slovom `"I"`. Napisati program u koji korisnik unosi tekst, a program ispisuje tekst u kojem je svako zasebno slovo `"i"` veliko. Program u narednom redu treba označiti lokacije na kojima je napravio izmjene koristeći povlaku.

Ispis 8.1: Input:

```
When I study, i get a good grade. i studied this time.
```

Ispis 8.2: Output:

```
When I study, I get a good grade. I studied this time.
              -                   -
```

Zadatak 8.17.

Napisati program koji od korisnika zahtijeva unos dvije riječi. Program treba ispisati koliko znakova se nalazi u prvoj, ali ne i u drugoj riječi. Program ne pravi razliku između malih i velikih slova.

Input:

```
Programiranje
Analiza
```

Output:

```
7
```

Zadatak 8.18.

Napisati program koji pronalazi i ispisuje tri broja koja ispunjavaju naredne uslove:

- Sva tri broja su palindromi (brojevi koji se čitaju jednako sa lijeva na desno i sa desna na lijevo, poput 8118).
- Prvi broj je dvocifren.
- Drugi broj je trocifren.
- Treći broj je četverocifren i zbir je prva dva broja.

Zadatak 8.19.

Program od korisnika zahtjeva unos stringa. Program treba vratiti dužinu najdužeg bloka u stringu. U ovom slučaju blok definišemo kao sekvencu istih **uzastopnih** ASCII znakova. Npr. za string `"koordinata"` program treba ispisati broj 2 zbog toga što je najduži blok u ovom slučaju `"oo"`. Za prazan string `""` program treba ispisati 0. Program `pravi` razliku između velikih i malih slova.

Input:

```
koordinata
```

Output:

```
2
```

Zadatak 8.20.

Šatrovački jezik je žargon u bosanskom, hrvatskom i srpskom jeziku. Nove riječi u šatrovačkom se stvaraju permutovanjem slogova. Na primjer, riječ **ne**moj postaje moj**ne**, **sa**raj'vo postaje rajvo**sa**, **zdra**vo postaje vo**zdra** itd.

Napisati program koji od korisnika traži unos riječi, a ispisuje tu riječ na šatrovačkom jeziku. Program pronalazi prvi samoglasnik koji *nije prvo slovo riječi* i prebacuje dio riječi od početka do tog samoglasnika (uključujući i njega) na kraj riječi. Program ne pravi razliku između velikih i malih slovo i ispisuje novu riječ malim slovima. [Napomena. Samoglasnici su: a, e, i, o, u.]

Input:

```
Sarajvo
```

Output:

```
rajvosa
```

Zadatak 8.21.

Da bi povećali sigurnost kupca, web stranice koje vrše naplatu proizvoda putem kartice, često sakrivaju broj kartice. Ipak da bi kupac mogao prepoznati kojom karticom plaća, 4 posljednje cifre broja kartice se prikazuju.

Napisati program koji od korisnika traži unos broja kartice. Program ispisuje broj ali tako da su sve cifre osim posljednje četiri zamijenjene $*$ znakom. Također, ispisane znakove (zvjezdice i brojeve) je potrebno grupisati, tako da se nakon svaka četiri ispisana znaka ispisuje prazno mjesto. Možete pretpostaviti da će broj cifara kartice uvijek biti djeljiv sa četiri.

Input:

```
123456789012
```

Output:

```
**** **** 9012
```

Zadatak 8.22.

Akrostih u književnosti je pjesma u kojoj početna slova stihova čitana odozgo prema dolje daju neku riječ. Npr. prva strofa pjesme "Voda je moja mati" autora Enesa Kiševića glasi:

Voda - kažem - i već žeđ bivam i žega.
Ova voda bistra oku tako godi.
Dovoljna je voda, pa da imaš svega.
A što god se rađa, s vodom se i rodi.

Upotrebom prethodne ideje moguće je sakriti tajnu poruku u nekom tekstu. Jedna od varijacija je da se ne koristi uvijek početno slovo, već drugo, treće ili slova po nekom unaprijed definisanom uzorku.

U ovom zadatku potrebno je napisati program koji će od korisnika zahtijevati unos stringova, svaki u novom redu. **Kraj unosa bit će označen praznim redom**. Tajna poruka će se dobiti tako što se uzme prvi znak iz prvog reda, drugi znak iz drugog reda, treći znak iz trećeg reda itd. Program podjednako tretira znakove bilo da su slova, prazna mjesta ili interpunkcijski znaci. Također, program ne pravi razliku između velikih i malih slova i prilikom ispisa, **rezultat se ispisuje malim slovima**. Na primjer, ukoliko bi u program unijeli iznad navedenu strofu pjesme ispis bi bio `vvvt`.

Input:

```
Tu se krije jedna poruka.
Naci je nije previse tesko.
Najbitnije je posmatrati odredjene znakove.
Dvanaesti znak u dvanaestom redu je vazan.
Druga slova u dvanaestom redu nisu vazna.
```

Output:

```
tajna
```

Zadatak 8.23.

Cenzura iako često ima negativne konotacije, pronalazi i veoma korisne primjene. Npr. cenzura se može koristiti u zaštiti nečijeg identiteta u dokumentu, ili sakrivanja nepoželjnih riječi u tekstu koji čitaju djeca.

Potrebno je napisati program koji će primiti **dva stringa**. Prvi string će sadržavati listu riječi koje treba cenzurisati. Riječi su odvojene zarezom i praznim mjestom. Nakon toga slijedi duži string, u kojem se nalazi tekst koji treba cenzurisati.

Program ispisuje drugi string tako da sva slova riječi koje treba cenzurisati zamjeni zvjezdicama. Program pravi razliku između velikih i malih slova, tj. cenzuriše samo riječi koje se u potpunosti

poklapaju. Također program cenzuriše i riječi koje su dio neke duže riječi. Npr. ukoliko je riječ koja se treba cenzurisati `sam` onda će se riječ `samostalan` ispisati kao `***ostalan`. Primjer:

Input:

```
program, racunar
Programiranje je pisanje uputstva racunaru sta i kako da ucini, a izvodi
   ↪  se u nekom od programskih jezika. Programiranje je umjetnost i
   ↪ umijece u stvaranju programa za racunare. Stvaranje programa
   ↪ sadrzi u sebi elemente dizajna, umjetnosti, nauke, matematike kao
   ↪  i inzinjeringa. Osoba koja stvara program zove se programer.
```

Output:

```
Programiranje je pisanje uputstva *******u sta i kako da ucini, a izvodi
   ↪  se u nekom od *******skih jezika. Programiranje je umjetnost i
   ↪ umijece u stvaranju *******a za *******e. Stvaranje *******a
   ↪ sadrzi u sebi elemente dizajna, umjetnosti, nauke, matematike kao
   ↪  i inzinjeringa. Osoba koja stvara ******* zove se *******er.
```

Zadatak 8.24.

Korisnik unosi tekst u jednom redu. Program pronalazi riječi koje su duže od 5 slova i podvlači ih. Riječi se podvlače tako što se u narednom redu tačno ispod te riječi ispisuju znakovi `-`. Kao rezultat se ispisuje originalni tekst u jednom redu, a u narednom redu se ispisuju prazna mjesta i znakovi `-` koji odgovaraju tekstu iznad.

Input:

```
Popocatepetl je vulkan u sredisnjem Meksiku, i jedan je od aktivnijih.
```

Output:

```
Popocatepetl je vulkan u sredisnjem Meksiku,
------------    ------   ---------- -------
i jedan je od aktivnijih.
             ----------
```

Zadatak 8.25.

Korisnik unosi rečenicu. Napisati program koji vraća riječi koje se ponavljaju kao i njihov broj. Program ne pravi razliku između velikih i malih slova. Potrebno je i obratiti pažnju na znakove interpunkcije koji se ignorišu i iza kojih uvijek slijedi prazno mjesto. Riječi se ispisuju redoslijedom u kojem su se prvi put pojavile u rečenice i ispisuju se malim slovima. Nakon svake riječi koja se ponavljala ispisuje se dvotačka, prazno mjesto i broj ponavljanja.

Input:

```
Zavrsni Ispit iz programiranja
   ↪ ima cetiri zadataka, a
   ↪ ispit iz programiranja
   ↪ tokom semestra ima tri
   ↪ zadatka, i znaci da ima
   ↪ jedan zadatak vise, ali
   ↪ zato ispit ima i vise
   ↪ vremena za izradu sva
   ↪ cetiri zadatka.
```

Output:

```
ispit: 3
iz: 2
programiranja: 2
ima: 4
cetiri: 2
zadatka: 2
i: 2
vise: 2
```

Zadatak 8.26.

Napisati program od korisnika traži unos dva stringa i provjerava da li su oni anagrami. Dva stringa su anagrami ukoliko se slova jednog mogu ispremještati tako da formiraju drugi string. Na primjer, string "Krasan je odmor!" je anagram od stringa "Jadransko more". Program ispisuje `Anagrami` ukoliko su dva stringa anagrami, a u suprotnom ispisuje `Nisu anagrami`

Program treba ignorisati interpunkcijske znake i ne treba praviti razliku između velikih i malih slova. [Napomena. Ukoliko se u jednom stringu pojavljuju 2 slova `a`, onda se i u drugom moraju pojavljivati tačno 2 slova `a` da bi bili anagrami.]

Input:

```
Krasan je odmor!
Jadransko more
```

Output:

```
Anagrami
```

Zadatak 8.27.

Interesantan fenomen je da većina ljudi može pročitati riječ čija su slova izmiješana, osim prvog i zadnjeg slova. Npr: `Mugoce je catiti oakvav teskt`.

U ovom zadatku program neće miješati slova koja se nalaze između prvog i zadnjeg, već ih treba sortirati.

Program od korisnika zahtjeva unos riječi. Program generiše i ispisuje novi string tako što ostavlja prvo i posljednje slovo riječi na originalnim pozicijama, ali sortira slova koja se nalaze između njih. Program ne pravi razliku između velikih i malih slova i prilikom ispisa **rezultat se ispisuje malim slovima**.

Input:

```
CitAti
```

Output:

```
caitti
```

Zadatak 8.28.

Kvadrat riječi se sastoji od riječi koje su ispisane u kvadratnoj matrici, tako da se **ista riječ može čitati i horizontalno i vertikalno**. Broj riječi, koji je jednak broju slova u svakoj riječi, naziva se *red* kvadrata. Npr. ovo je kvadrat riječi, reda pet:

```
H E A R T
E M B E R
A B U S E
```

```
R E S I N
T R E N D
```

Potrebno je napisati program u koji korisnik unosi kvadrat riječi, kao stringove, svaku riječ u novom redu. Program sam prepoznaje kraj unosa, na osnovu broja slova u prvoj riječi. Unesene riječi mogu biti ispisane velikim i malim slovima. Može se pretpostaviti da će broj riječi i broj slova uvijek biti ispravan, tj. broj slova u svakoj riječi bit će jednak broju unesenih riječi.

Program treba ponovo ispisati kvadrat riječi, ali tako da je svako slovo u riječi veliko i razdvojeno praznim mjestom. Također, program ispod kvadrata ispisuje riječ `ispravan` ukoliko je uneseni kvadrat ispravan (tj. ukoliko se ista riječ može čitati i horizontalno i vertikalno.). Program ispod kvadrata ispisuje riječ `neispravan` ukoliko kvadrat nije ispravan. Primjeri:

Input:

```
Card
area
REAR
dART
```

Output:

```
C A R D
A R E A
R E A R
D A R T
ispravan
```

Input:

```
Bit
ICE
baN
```

Output:

```
B I T
I C E
B A N
neispravan
```

9. Fajlovi

Varijable možemo posmatrati kao memorijske lokacije u kojima čuvamo vrijednosti tokom izvršavanja programa. Vrijednostima sačuvanim u varijablama imamo pristup u najboljem slučaju od momenta početka do momenta kraja izvršavanja programa. Nakon toga, sve vrijednosti sačuvane u varijablama se gube (jer se inicijalno nalaze u radnoj memoriji).

U praksi, postoji velika potreba da se vrijednosti sačuvaju i nakon završetka programa, te da se te vrijednosti naknadno koriste. Takvih primjera je mnogo, od pisanja tekstualnih fajlova, čuvanja fotografija, do spašavanja najboljih rezultata i trenutnog stanja igre, preko samog spašavanja programskog koda (svaki kod bi nestao gašenjem okruženja u kojem isti kucamo). Način za trajnije čuvanje podataka jesu upravo **fajlovi**. Podaci se čuvaju u fajlovima na disku računara, te je pristup istima moguć u bilo kojem trenutku.

Standardne operacije sa fajlovima su čitanje i spašavanje podataka. Proces spašavanja vrijednosti često se naziva zapisivanjem. U svakom radu sa fajlovima, neophodno je poštovati niz koraka:

1. otvoriti ili kreirati željeni fajl,
2. pročitati ili zapisati podatke,
3. zatvoriti fajl.

Identičan niz koraka izvodi se pri radu sa fajlovima kroz programski jezik Python. Python omogućava nekoliko jednostavnih naredbi/funkcija za rad sa fajlovima.

9.1 Kreiranje i otvaranje fajla

Prva je svakako opcija kreiranja ili otvaranja fajla. Fajl se otvara u slučaju da postoji od ranije, te se kreira novi fajl u slučaju da istoimeni nije postojao. Funkcija koja se koristi za otvaranje fajla je `open`. Funkcija se poziva sa dva parametra, te se koristi u kombinaciji sa spašavanjem učitanog fajla u varijablu. Format korištenja je `file = open(filename, mode)`. U standardom formatu, `file` je varijabla koja se odnosi na učitani fajl, `filename` je naziv fajla koji se učitava, dok je `mode` string koji specificira način pristupanja fajlu. Iako postoji nekoliko kompleksnijih načina

pristupanju fajlovima, za uvodni rad sa fajlovima dovoljno je koristiti opcije `'r'`, `'w'`, `'a'`.

- Opcija `'r'` koristi se za čitanje iz fajla. Fajl je nemoguće promijeniti ukoliko je otvoren sa ovim načinom rada. Neophodno je napomenuti da fajl kojem se pristupa mora postojati (inače čitanje nema smisla, zar ne?).
- Druga opcija, opcija `'w'`, koristi se za zapisivanje u fajl. Kompletan prethodni sadržaj (ukoliko je postojao) se briše, te ukoliko fajl nije postojao, kreira se novi.
- Opcija `'a'` se koristi također za zapisivanje u fajl, međutim, raniji sadržaj se ne briše, te korisnik nastavi zapisivanje u fajl. Ukoliko fajl nije postojao, isti će biti kreiran.

9.1.1 Putanja fajla

Pretpostavljena lokacija fajla određena je radnim folderom (folderom u kojem se nalazi kod iz kojeg se pristupa fajlu). Ukoliko se fajl nalazi (ili se novokreirani fajl treba nalaziti) u tom folderu, dovoljno je za `filename` postaviti naziv fajla. Ukoliko se radi sa fajlom koji se nalazi na drugoj lokaciji, dovoljno je navesti tačnu putanju do fajla.

Prije stringa koji označava lokaciju fajla, neophodno je dodati slovo `r`. Ukoliko želimo fajl pročitati iz foldera (ili smjestiti u folder) `C:\Users\test\`, dovoljno je navesti `file = open(r'C:\Users\test\fajl.txt', 'w')`. Prefiks *r* (*raw string*) signalizira Python interpreteru da `\` interpretira kao obični znak, a ne kao specijalni znak (*escape character*).

9.2 Metode `file` objekta

Varijabla `file` predstavlja objekat. Ovaj objekat ima niz funkcija koje se odnose na njega, te se takve funkcije nazivaju **metode**. Nakon otvaranja fajla, svaka operacija sa istim će uključivati pozivanje metoda tog objekta.

9.2.1 Metoda `close()`

Prva metoda koju ćemo spomenuti je metoda `close()`. Ova metoda zatvara korišteni fajl i navodi se na kraju svakog korištenja i rada sa fajlovima. Pozivanje ove metode je jednostavno. Za otvoreni fajl `file`, metoda se poziva sa `file.close()`. Kao što je ranije navedeno, metoda je funkcija koja se odnosi na jedan objekat, u ovom slučaju nazvan `file`. Navođenje ove naredbe je obavezno, jer u slučaju nekorištenja iste može doći do gubljenja podataka u fajlovima. Razlog za to je činjenica da se podaci često čuvaju u privremenim spremnicima (tzv. *buffer*), te nakon punjenja istog, spašavaju se u fajl. U slučaju naglog prekida rada sa fajlom (gašenje programa), podaci koji se nalaze u *bufferu* mogu biti biti izgubljeni. Korištenje ovih privremenih spremnika neophodno je zbog efikasnijeg rada sa fajlovima (zapisivanje u trenutnu memoriju je značajno brže od spašavanja u fajl).

9.2.2 Čitanje iz fajla

U procesu čitanja podataka iz fajla, razlikujemo dvije najčešće korištene metode:

- `read()`
- `readline()`

Prva metoda učitava kompletan sadržaj fajla kao string. Druga metoda učitava jednu (narednu) liniju fajla. Pozivom naredbe `readline`, pozicija čitanja se pomjera na prvu poziciju nakon učitane linije fajla. Uzastopnim pozivanjem naredbe `readline`, učitava se naredna linija fajla. Dakle, prvi poziv funkcije `readline` učitava prvu liniju, drugi poziv drugu liniju i proces se nastavlja do kraja fajla. Prelazak u narednu liniju svakog fajla označen je kombinacijom `\n`.

Primjer 9.1 Napisati program koji otvara fajl sa nazivom `test.in` za čitanje i njegov sadržaj ispisuje u konzolu.

```
file = open('test.in', 'r')
text = fajl.read()
print(text)
file.close()
```

Ekstenzija fajla može biti proizvoljna, te će se u primjerima koristiti ekstenzija `in` za čitanje i `out` za upis u fajl. Koristeći funkciju `read()`, kompletan sadržaj fajla učitan je u varijablu `text`, te je fajl zatvoren. U sklopu varijable `text`, svaki naredni red u fajlu je moguće pronaći pomoću kombinacije znakova `\n`. Ovo je standardni način rada u situaciji da želimo pročitati komentar sadržaj fajla kroz jedan poziv funkcije. ■

Primjer 9.2 Napisati program koji otvara fajl sa nazivom `test.in` za čitanje i ispisuje prve tri linije koje se nalaze u fajlu, s tim da se treća linija u fajlu ispisuje prva, a prva linija u fajlu se ispisuje treća.

```
file = open('test.in', 'r')
l1 = file.readline()
l2 = file.readline()
l3 = file.readline()

print(l3)
print(l2)
print(l1)

file.close()
```

Prve tri linije otvorenog fajla učitavaju se u varijable `l1`, `l2` i `l3`, redom. Program zatim ispisuje te linije u obrnutom poretku, te zatvara fajl. Proces učitavanja teksta iz fajla ovisi najviše od sadržaja fajla. Često se učitavanje vrši upotrebom `while` petlje, gdje je kraj fajla označen određenim nizom znakova (ili praznom linijom). Tada se učita linija po linija i petlja se zaustavlja nakon što je kompletan sadržaj uspješno učitan. ■

9.2.3 Zapisivanje u fajl

Za zapisivanje u fajl, koristimo `write` metodu sa sintaksom, `file.write(tekst)`. Rezultat izvršavanja je da je u fajl na kojeg se referencira varijabla `file` zapisan tekst smješten u varijabli `text`. Da bi zapisivanje uspješno završilo, fajl mora biti otvoren sa svrhom zapisivanja (`'w'` ili `'a'`). Podaci koji se zapisuju u fajl mogu biti proizvoljni.

Primjer 9.3 Napisati program koji zahtijeva dva unosa podataka koje zatim spašava u fajl pod nazivom `test.out`.

```
text1 = input()
text2 = input()

file = open('test.out', 'w')
```

```
file.write(text1)
file.write(text2)
file.close()
```

Iznad navedeni kod u fajl `test.out` zapisuje dva unesena stringa jedan nakon drugog. U slučaju da želimo razdvojiti stringove u nove linije, neophodno je na kraj prvog stringa dodati `\n`. ■

9.3 Zadaci

Zadatak 9.1.

Napisati program koji otvara fajl pod nazivom "test01.in". U fajlu se nalazi mapa prostora u kojem se nalazi robot. Prazno mjesto prostora je obilježeno znakom `'-'`, a zid znakom `'#'`. Mjesto na kojem se nalazi robot obilježeno je slovom `'R'`. Znakovi u istom redu odvojeni su jednim praznim mjestom.

Robot će pokušati izaći iz pomenutog prostora tako što će ići desno dok ne izađe ili ne udari u zid. Ako udari u zid ići će dolje dok ne izađe iz prostora. Ukoliko i tada udari u zid, prestat će sa kretanjem.

U prvoj liniji program ispisuje koliko koraka će se robot kretati. U narednoj liniji program ispisuje `'da'` ukoliko će robot izaći iz datog prostora, a u suprotnom ispisuje `'ne'`.

test01.in

```
- # - - - #
# R - - # #
# # - - # -
- - # - # -
```

Output:

```
5
da
```

Zadatak 9.2.

Napisati program koji otvara fajl pod nazivom "test01.in". U fajlu se nalaze dvije matrice istih dimenzija jedna ispod druge. U prvoj matrici se nalaze prirodni brojevi, a u drugoj znakovi `'*'` i `'-'`. Obje matrice su formirane na način da su kolone razdvojene jednim praznim mjestom, a redovi prelaskom u novi red.

Znak `'*'` se koristi za označavanje brojeva iz prve matrice, na način da njegova pozicija odgovara poziciji broja kojeg označava. Program treba ispisati zbir svih parnih označenih brojeva.

test01.in

```
8 9 1 6
4 281 22 1
3 71 18 9
* * - -
- * * -
- - * -
```

Output:

```
48
```

Zadatak 9.3.

Napisati program koji otvara fajl pod nazivom "test01.in". U fajlu su zapisani izbori jednog od tri moguća ispita koje su studenti mogli prijaviti za neki ispitni rok. Svaki red u fajlu predstavlja izbor jednog studenta tako da su ime, prezime i izbor studenta odvojeni jednim praznim mjestom. Ispiti su označenih brojevima 1, 2 i 3.

Program treba ispisati liste studenata za svaku od tri grupe. Liste se ispisuju tako što se ime i prezime studenta koji pripada toj grupi ispiše u jednom redu razdvojeno praznim mjestom. Prvo se ispisuju studenti koji su izabrali prvi ispit, zatim studenti koji su odabrali drugi, a zatim studenti koji su odabrali treći. Liste su odvojene ispisom dodatnog praznog reda. Studenti se u svakoj listi ispisuju u redoslijedu u kojem su se pojavljivali i u originalnoj listi.

test01.in

```
Sead Delalic 3
Meliha Kurtagic 2
Ognjen Bostjancic 1
Malek Chahin 3
Elmedin Selmanovic 1
Harun Hindija 2
```

Output:

```
Ognjen Bostjancic
Elmedin Selmanovic

Meliha Kurtagic
Harun Hindija

Sead Delalic
Malek Chahin
```

Zadatak 9.4.

Napisati program koji otvara fajl pod nazivom "test01.in". Fajl se može sastojati od 0 ili više linija. U svakoj liniji se nalaze korisničko ime i lozinka nekog korisnika, a razdvojeni su praznim mjestom. Program treba identifikovati nesigurne šifre. U ovom zadatku nesigurnu šifru ćemo definisati kao onu koja je u potpunosti jednaka korisničkom imenu, ili u sebi sadrži uzastopnu sekvencu “1234” ili je kraća od 5 znakova. Program ispisuje korisnička imena i šifre koje identifikuje kao nesigurne, u originalnom redoslijedu.

test01.in

```
smart_user W?4M7Luj
admin admin
root pass1234word
safe_user fa!70gEVa1
user abc
```

Output:

```
admin admin
root pass1234word
user abc
```

Zadatak 9.5.

Napisati program koji otvara fajl `test01.in`. U fajlu se nalazi matrica realnih brojeva veličine 3×3. Program ispisuje sumu svih brojeva u matrici.

test01.in

```
2.3 8.16 3.141
8 23.5 914.5
18.42 6 987.1
```

Output:

```
1971.121
```

Zadatak 9.6.

Napisati program koji otvara fajl pod nazivom `test01.in`. U fajlu je nacrtan zatvoreni geometrijski oblik. Program ispisuje njegovu površinu (broj polja koje obuhvata, uključujući i granice). Prazna mjesta su označena znakom `'-'`, a konture oblika znakom `'#'`. Svi znakovi su razdvojeni praznim mjestom koje se ne trebaju ubrajati u površinu. Obratite pažnju da svi znakovi koji su unutar oblika, sa sve četiri strane nailaze na znak `'#'`.

`test01.in`

```
- - - # # - - # - - -
- - # - - # # - # # -
- # - - - - - - - - #
- # - - - - - - - # -
- - # # - - - - # - -
- - - # - - - - # - -
- - # - - - - # - - -
- - - # # # # - - - -
- - - - - - - - - - -
```

Output:

```
53
```

Zadatak 9.7.

Napisati program koji otvara fajl pod nazivom "test01.in". U fajlu se nalazi matrica sa m redova i $2 \times m - 1$ kolona. U svakom redu nalazi se m znakova, razdvojenih praznim mjestom zbog preglednosti. Tri znaka se mogu pojavljivati u matrici: crtica (`-`), broj `8` i broj `0`. Matrica predstavlja svijet nastanjen bubama i mravima. Broj `8` predstavlja mrava, broj `0` bubu a crtica prazan prostor. (Znakovi *space* `'␣'` koji razdvajaju ostale znakove ne predstavlja ništa, već su tu samo radi preglednosti.)

Buba će pojesti jednog mrava koji se nalazi uz nju horizontalno ili vertikalno i preći na njegovo polje. Buba ne može pojesti mrava koji se nalazi uz nju dijagonalno. Može se desiti da se više mrava nalazi oko jedne bube, ali će i tada pojesti samo jednog mrava prema sljedećim preferencama: iznad, desno, lijevo, te ispod. [Dakle, ukoliko se više mrava nalazi oko bube, a jedan od mrava je iznad nje ona će ga pojesti; a ukoliko je više mrava, ali ni jedan nije iznad nje, i jedan mrav se nalazi desno, pojest će njega, itd.] Ukoliko se ni jedan mrav ne nalazi pored bube ona umire i u njeno polje se upisuje crtica.

Pošto u svijetu najčešće živi više buba, redoslijed razmatranja njihovih aktivnosti treba biti sa lijeva na desno, i odozgo prema dole, kao prilikom čitanja. Program ispisuje izgled svijeta nakon što se razmotri svaka buba (tj. nakon što svaka buba napravi potez). Možete pretpostaviti da se ni jedna buba u početnoj matrici neće nalaziti na ivici svijeta (u prvom ili zadnjem redu ili u prvoj ili zadnjoj koloni).

test01.in

```
- 8 - - - - -
8 0 8 0 - - -
- - - - 8 0 8
- 0 - - - 8 -
8 - - - - - -
- 0 8 0 - 0 8
- - - - - 8 -
```

Output:

```
- 0 - - - - -
8 - 0 - - - -
- - - - 8 - 0
- - - - - 8 -
8 - - - - - -
- - 0 - - - 0
- - - - - 8 -
```

Zadatak 9.8.

Prilikom odabira lozinke mnoge internet stranice postavljaju jako puno zahtijeva u cilju poboljšanja sigurnosti, pa zahtijevaju da lozinka bude duža od 8 znakova, sadrži broj i veliko slovo, i bar neki specijalni znak. Korisnicima je jako teško zapamtiti ovakve šifre. Postoji alternativni, a ništa manje siguran pristup, kojim se kreiranju duže lozinke sastavljene od nekoliko nasumičnih riječi.

Napisati program koji otvara fajl pod nazivom "test01.in". U prvom redu se nalazi jednocifreni broj n. Nakon toga slijedi lista riječi, gdje je svaka u novom redu. Program izabire n nasumičnih riječi i spaja ih, te na taj način generiše i ispisuje lozinku za korisnika. Riječi u fajlu mogu sadržavati velika i mala slova, ali predložena lozinka treba biti ispisana malim slovima. U zadatku je potrebno inicijalizirati generator nasumičnih brojeva, brojem 42, a koristeći komandu `seed(42)`. Može se desiti da se jedna riječ ponovi više puta u generisanoj lozinci.

test01.in

```
3
rijec
stablo
Programiranje
kompjuterske
Nauka
matematicar
Racunar
Ispit
lozinka
```

Output:

```
stablorijecnauka
```

Zadatak 9.9.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. U fajlu se nalazi genetska sekvenca. Sekvenca je predstavljena kao dugački string u kojem se ponavljaju slova slova A, C, G, T. Sekvenca u više navrata sadrži niz slova `"AGGT"`. Program treba ispisati najdužu podsekvencu koja je ograničena slovima `"AGGT"` sa lijeve i desne strane.

test01.in

```
TAGGTAAATAGCTCTAGGTGAACGCAGGTG
```

Output:

```
AAATAGCTCT
```

Zadatak 9.10.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. Fajl je formatiran na način da se u prvom redu nalaze dva cijela broja razdvojena praznim mjestom. Prvi broj predstavlja broj redova, a drugi broj kolona u matrici.

U narednom redu se nalazi uređeni par cijelih brojeva u zagradi razdvojen zarezom nakon čega slijedi prazno mjesto pa neki ASCII znak. Par brojeva predstavlja koordinate u matrici na koje se postavlja navedeni znak. Nakon ovog reda slijedi još redova istog formata i svaki sadrži koordinate i znak. Program treba da iscrta ovako definisanu matricu. Ukoliko na određenoj koordinati nije postavljen znak ispisuje se prazno mjesto. [Napomena. Koordinate mogu biti višecifrene.]

test01.in

```
2 2
(0, 1) =
(0, 0) !
(1, 1) A
(0, 1) *
```

Output:

```
!*
 A
```

Zadatak 9.11.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. U fajlu se nalaze mjesečne temperature izmjerene u nekom gradu u posljednjih 5 godina. U svakom redu se nalazi 12 temperatura (po jedna za svaki mjesec) odvojenih praznim mjestom. Prva temperatura u redu je temperatura za mjesec januar. Fajl ima 5 ovakvih redova (po jedan za svaku godinu). Napisati program koji ispisuje prosječne mjesečne temperature odvojene praznim mjestom. U narednom redu program ispisuje naziv najtoplijeg mjeseca, a u redu ispod najhladnijeg.

Ispis 9.1: test01 . in

```
-7.66 23.35 -1.23 -8.06 26.16 0.35 26.71 15.26 6.82 20.17 -2.9 2.8
-8.87 -4.65 12.21 7.26 20.03 19.29 -8.52 27.63 26.31 10.34 -2.62 27.14
29.06 -9.73 25.65 21.17 18.3 2.26 1.31 19.83 -1.4 -3.01 0.11 -4.07
28.39 6.44 29.97 10.47 5.57 11.18 16.97 4.64 16.53 -0.48 3.16 23.72
25.21 20.53 9.67 10.43 -2.63 29.92 23.63 2.15 -8.88 3.7 24.97 17.58
```

Ispis 9.2: Output:

```
13.26 7.19 15.25 8.25 13.48 12.60 12.02 13.90 7.87 6.14 4.54 13.43
Mart
Novembar
```

Zadatak 9.12.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. U fajlu se nalaze temperature iz više gradova, za svaki dan u sedmice. Fajl je formatiran na sljedeći način. U prvoj liniji se nalazi veliko slovo `P` i ono označava ponedjeljak. U narednom redu se nalazi naziv grada prazno mjesto pa izmjerena temperatura za taj grad. U redu ispod je drugi grad sa svojom temperaturom. Linje sa gradovima i temperaturama se ponavljaju dok se ne pojavi slovo `U` u zasebnom redu koje označava početak liste temperatura za utorak. Nakon slova `U` opet slijedi lista istih gradova sa temperaturama za utorak. Ovaj uzorak se ponavlja za svih sedam dana u sedmici koji su označeni slovima: P, U, S, C, P, S, N.

Program ispisuje dva reda. U prvom se nalazi grad sa najnižom izmjerenom temperaturom, kao i nazivom dana kada je izmjerena ta temperatura. U narednom se nalazi grad sa najvišom izmjerenom temperaturom i danom.

test01.in

```
P
Banja Luka -3.2
Mostar 5.8
Sarajevo -7.1
U
Banja Luka -1.3
Mostar 7.2
...
```

Output:

```
Sarajevo -7.1 ponedjeljak
Mostar 8.2 nedjelja
```

Zadatak 9.13.

Digitalna fotografija se može predstaviti kao matrica piksela i sačuvati u fajl. Kod crno-bijelih fotografija svaki piksel može imati jednu od 256 vrijednosti. Vrijednost 0 predstavlja piksel crne boje, a vrijednost 255 predstavlja piksel bijele boje. Ostale vrijednosti od 1 do 254 predstavljaju nijanse sive od tamnije ka svjetlijoj.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. U fajlu je pohranjena crno-bijela fotografija. U prvom redu nalaze se dva cijela broja odvojena praznim mjestom, koji prestavljaju broj redova n i broj kolona m matrice piksela. Nakon toga slijedi n redova, gdje se u svakom redu nalaz m vrijednosti u rasponu od 0 do 255, odvojenih praznim mjestom. Svaka vrijednost predstavlja intenzitet tog piksela.

Program treba da ispiše tri boje (vrijednosti) koje se najčešće pojavljuju na slici. Svaka od boja se ispisuje u zasebnom redu zajedno sa praznim mjestom, te brojem pojavljivanja te boje na slici. Boje se ispisuju opadajućim redoslijedom od one koja se pojavljuje najčešće, zatim one koja je druga po učestalosti, a na kraju se ispisuje boja koja je treća po učestalosti.

test01.in

```
4 6
0 255 0 255 15 127
132 14 127 255 255 15
201 200 199 198 197 196
0 127 255 0 0 255
```

Output:

```
255 6
0 5
127 3
```

Zadatak 9.14.

Napisati program koji otvara fajl `test01.in`. U fajlu se nalazi matrica znakova. Za matricu kažemo da je savršena ako su svaka kolona i svaki red palindromi (kolone imaju jednake vrijednosti kada se čitaju prema dolje i prema gore, a redovi imaju jednake vrijednosti kada se čitaju slijeva i zdesna).

Ukoliko su sve kolone palindromi, ali svi redovi nisu palindromi ili obrnuto, matrica je polusavršena. Ako ni svi redovi nisu palindromi, a ni sve kolone nisu palindromi, matrica je nesavršena. Na osnovu analize matrice program ispisuje jednu riječ: `savrsena`, `polusavrsena` ili `nesavrsena`.

test01.in

```
X - X
! 0 1
X - X
```

Output:

```
polusavrsena
```

Zadatak 9.15.

Napisati program koji otvara fajl pod nazivom `test01.in`. U fajlu se nalazi karta sa blagom predstavljena pomoću znakova: `'#'`, `'S'`, `'X'`, `'-'` i `'␣'` (prazno mjesto). Znak `'#'` se koristi za iscrtavanje okvira karte. Znak `'S'` označava početak potrage za blagom, a znak `'X'` njegovu lokaciju. Znakovi `'-'` i `'|'` se koriste za iscrtavanje puta koji je potrebno preći da bi se stiglo do blaga, gdje `'-'` označava horizontalno kretanje, a znak `'|'` vertikalno kretanje.

Program treba ispisati smjerove i broj koraka koje je potrebno preći da bi se stiglo do blaga. Rezultat se ispisuje tako što se u svakom redu ispisuje smjer kretanja, prazno mjesto, i potom broj koraka koje je potrebno preći u tom smjeru. R predstavlja smjer desno (right), L predstavlja smjer lijevo, U predstavlja smjer gore (up), a D smjer dole. Možete pretpostaviti da se putanja neće granati ili ukrštati sa samom sobom i da postoji samo jedan izlaz iz polja S.

test01.in

```
###########
#   -----|#
# X |    |#
# | | S---#
# | |     #
# --|     #
###########
```

Output:

```
R 3
U 2
L 5
D 4
L 2
U 3
```

Zadatak 9.16.

U igri Tetris bodovi se sakupljaju kada igrač blokove koji padaju vertikalno poreda na način da oni u potpunosti popune jedan cijeli red.

U ovom zadatku dat je fajl pod nazivom "test01.in" koji predstavlja tablu iz igre Tetris, u određenom trenutku. Prazna mjesta su označena znakom `'-'`, a popunjena znakom `'X'`. Na vrhu se uvijek nalazi centrirani T blok koji je potrebno rotacijom i horizontalnim pomjeranjem pozicionirati tako da prilikom pada na dno table formira popunjenu liniju. Rotacija bloka se vrši u smjeru kazaljke na satu što rezultira položajima koji su prikazani ispod, a blok je na početku uvijek postavljen u položaj sa slike 1.

Položaj 1:

```
- X -
X X X
- - -
```

Položaj 2:

```
- X -
- X X
- X -
```

Položaj 3:

```
- - -
X X X
- X -
```

Položaj 4:

```
- X -
X X -
- X -
```

Program ispisuje dvije vrijednosti jednu nakon druge odvojene praznim mjestom. Prva je broj rotacija koje je potrebno izvršiti, a druga je pozitivan ili negativan broj koji označava horizontalno pomjeranje. Pozitivna vrijednost označava pomjeranje desno, a negativna lijevo. Možete pret-

postaviti da se blok treba uvijek smjestiti na samo dno ploče i da ploča uvijek ima dimenzije 8×7. Ukoliko ima više mogućih rješenja, potrebno je preferirati ono sa manjim brojem rotacija.

test01.in:

```
- - - X - - -
- - X X X - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - X
- - - X - X X
X - X X X X X
```

Output:

```
1 -2
```

Zadatak 9.17.

Napisati program koji otvara fajl pod nazivom `test01.in`. U fajlu se nalazi slika koja predstavlja snijeg koji pada na objekat koji je na zemlji. Silueta objekta je označena znakom `'#'`, a pahuljice znakom `'*'`. Ostatak slike sadrži znakove `'␣'` tj. prazna mjesta. Također radi preglednosti svi znakovi su odvojeni dodatnim praznim mjestom.

Program iscrtava sliku koja se dobije kada sve pahuljice padnu na objekte ili zemlju.

test01.in

```
  *   *   *
*   *   *     *
    *     * *
      *
  * *   *   *
  *     # # #
    # # #     #
  #           #
```

Output:

```
    *   * * *
  * * * * * *
  * * * # # # *
  * # # #     #
* #           #
```

Zadatak 9.18.

Napisati program koji otvara fajl pod nazivom "test01.in". U fajlu se nalazi matrica znakova koja predstavlja labirint. Zidovi su označeni zvjezdicama ($*$), a prolazi praznim povlakama (`-`). Ulaz u labirint je jedan i uvijek se nalazi na samom vrhu, a izlaza iz labirinta može biti više i uvijek se nalaze na samom dnu. Svaki labirint je formiran tako da nakon ulaska **uvijek** slijedi prazan prostor cijelom širinom labirinta, a zatim zid cijelom dužinom labirinta sa jednim ili više prolaza. Potom opet slijedi prazan prostor cijelom širinom, pa zid sa jednim ili više prolaza i tako sve do izlaska iz labirinta.

U labirint se pušta robot koji može da se kreće lijevo, desno i dole. Ukoliko može robot se uvijek kreće prema dole, ali kada naiđe na zid, onda ide lijevo ili desno do najbližeg prolaza. U slučaju da je ima dva prolaza koja su jednako udaljena sa lijeve i desne strane, robot bira lijevu stranu. Program treba ispisati broj koraka potreban robotu da od ulaza dođe do izlaza.

U narednom primjeru robot polazi sa ulaza i pravi jedan korak dole, te nailazi na zid. Pošto su otvori lijevo i desno jednako udaljeni robot bira lijevi i pravi dva koraka lijevo, a zatim dva koraka dole, te opet nailazi na zid. Naredni najbliži otvor robotu je četiri koraka desno, nakon kojih robot

nastavlja da se krece prema dole. Nakon 3 koraka robot se nalazi na izlazu iz labirinta, pa program kao rezultat ispisuje broj 12.

test01.in

```
*******-*********
*---------------*
**-**-***-*******
*---------------*
*********-****-**
*---------------*
********--***-***
```

Output:

```
12
```

Zadatak 9.19.

Napisati program koji otvara fajl pod nazivom "test01.in". U fajlu se nalazi matrica sa m redova i $2m-1$ kolona. U svakom redu nalazi se m znakova razdvojenih praznim mjestom zbog preglednosti. Znak je ili crtica - ili neko slovo engleskog alfabeta. Potrebno je napisati program koji će ispisati matricu što će svaku crticu zamijeniti slovom koje joj je najbliže. Za mjeru daljine koristi se Euklidska udaljenost. [Napomena: Znakovi *space* `' '` koji razdvajaju ostale znakove ne ulaze u kalkulaciju udaljenosti, već su tu samo radi preglednosti, tj. crtica koje se nalazi desno od trenutnog slova, je podjednako udaljeno od crtice koje se nalazi ispod tog slova.]

Ukoliko je određena crtica podjednako udaljena od dva ili više slova onda se treba zamijeniti slovom koje se pojavljuje ranije u alfabetu. Program ne pravi razliku između velikih i malih slova, a prilikom ispisa uvijek koristi velika slova. Također, možete pretpostaviti da se u ulaznoj matrici slovo nikada neće ponavljati više od jednog puta.

test01.in

```
- - - - - - - -
- - - - - - - -
- D - - - X - -
- - - - - - - -
- - - H - - - -
- - - - - - - -
- - - - - - - -
- - - - - A - -
```

Output:

```
D D D D X X X X
D D D D X X X X
D D D D X X X X
D D D H H X X X
D D H H H H X X
D H H H H A A A
H H H H A A A A
H H A A A A A A
```

Zadatak 9.20.

Postoje mnogi načini za šifriranje teksta kako se ne bi mogao pročitati ukoliko dospije u pogrešne ruke. Ovaj zadatak zahtjeva pisanje programa koji će dešifrovati poruke koje su šifrirane kako je opisano u narednom primjeru. Dat je šifrirani tekst: `saoreecessinntfi`. Potrebno je primijetiti da se sastoji od šesnaest slova. Tekst je moguće postaviti u matricu veličine 4×4 čime se dobije:

```
s a o r
e e c e
s s i n
n t f i
```

Tekst će se dešifrovati ukoliko se znakovi poredaju na osnovu kolona odozgo prema dole, tj. prvo znakovi iz prve kolone, potom znakovi iz druge kolone itd. Tako da je tražena riječ: sesnaestocifreni.

Program otvara fajl pod nazivom "test01.in". U fajlu se nalaze šifrirani tekstovi jedan ispod drugog. Program treba ispisati dešifrovane tekstove svaki u zasebnoj liniji. Ukoliko broj znakova ulaznog teksta nije kvadrat nekog broja program ispisuje riječ `GRESKA`.

`test01.in`

```
saoreecessinntfi
pmrrj ie2
dobar tekst
borj
Bjsr aoOm
```

Output:

```
sesnaestocifreni
primjer 2
GRESKA
broj
Broj Osam
```

Zadatak 9.21.

Profesor koji nije volio ocjenjivati testove studenata i koji je uvijek kasnio sa objavom rezultata odlučio je automatizovati proces ocjenjivanja. Opredijelio se da isključivo daje pitanja sa pet ponuđenih odgovora (A, B, C, D i E). Prilikom testiranja, svaki student će dobiti list na kojem je za svako pitanje ponuđeno pet kružića, a student treba ispuniti kružić za koji smatra da pripada tačnom odgovoru.

Profesor je napisao softver koji identifikuje kružiće na skeniranom listu. Softver potom svakom kružiću dodijeli vrijednost između 0 i 255 u zavisnosti od toga koliko je ispunjen. Ukoliko je u potpunosti ispunjen dobije vrijednost 0, a ukoliko ga je student ostavio praznim dobije vrijednost 255. U idealnom slučaju, ukoliko su kružićima na nekom pitanju dodijeljene vrijednosti: 255, 0, 255, 255, 255 znamo da je student odabrao odgovor B.

U brzini, studenti rijetko u potpunosti ispune kružić pa softver često dodijeli i druge vrijednosti. Profesor je odlučio da će smatrati ispunjenim sve kružiće koji imaju vrijednost manju ili jednaku broju 127, a praznim one kružiće koji imaju vrijednost veću od 127.

Također može se pretpostaviti da svi kružići neće uvijek biti ispravno označeni, tj. student greškom može označiti više od jednog kružića za isto pitanja, ili može ostaviti sve kružiće praznim. U ovakvim slučajevima odgovor na pitanje se treba smatrati netačnim.

Nakon što je napisao softver koji radi kako je navedeno iznad, profesoru su iskrsle dodatne obaveze, pa je zamolio vas da napišete program koji će računati broj tačnih odgovora. Profesor će vam dostaviti fajl pod nazivom `test01.in`. U prvom redu će se nalaziti broj studenata koji je testiran i broj pitanja koji je bio na testu, odvojeni praznim mjestom. Nakon toga u svakom redu slijede vrijednosti koje je softver prepoznao za svako pitanje svakog studenta, odvojeni praznim mjestom. U zadnjem redu fajla su dati tačni odgovori na pitanja, razdvojeni prazni mjestom. Vaš softver treba napisati broj tačnih odgovora za svakog studenta, jedan ispod drugog.

Na primjeru ispod vidimo da su testu pristupila četiri studenta, a na testu su data dva pitanja. Vrijednosti kružića prvog studenta se nalaze u naredna dva reda i vidimo da je student odabrao odgovor za prvo pitanje A, a za drugo E. Drugi student je odabrao odgovore B i E, itd. U zadnjem redu je navedeno da je tačan odgovor za prvo pitanje B, a za drugo E.

test01.in

```
4 2
0 255 255 255 255
255 255 255 255 0
255 127 255 255 255
255 128 215 130 20
200 200 200 0 200
200 1 200 200 1
255 5 200 130 205
1 2 3 4 5
B E
```

Output:

```
1
2
0
1
```

Zadatak 9.22.

Veliki problem u modernoj komunikaciji predstavlja spam tj. nepoželjna elektronska pošta. Osobe koje šalju spam najčešće ciljaju veliki broj osoba, pa tako šalju mnogo identičnih poruka na različite adrese u jako kratkom periodu. Potrebno je napisati program koji će identifikovati spam poštu.

Dat je tekstualni fajl pod nazivom "test01.in", u kojem se nalazi lista pristigle pošte na neki mail server u periodu od jedne minute. Svaki red u fajlu predstavlja jednu poruku. U redu se nalazi adresa pošiljaoca, adresa primaoca, kao i naslov poruke. Navedeni podaci odvojeni su praznim mjestom.

Za određenu e-mail adresu pretpostavit će se da šalje spam ukoliko su sa te adrese poslane poruke sa istim naslovom na barem 5 različitih e-mail adresa.

Program treba ispisati sve e-mail adrese koje je identifikovao da šalju spam jednu ispod druge, u redoslijedu u kojem se i originalno pojavljuju u fajlu.

U narednom primjeru vidimo da su kao spam pošiljaoci identifikovane mail adrese: `spam_1 @spam.ba` i `spam_2@spam.ba` jer su poslale poruke sa istim naslovom na 5 ili više različitih adresa. Prvo se ispisuje adresa `spam_1@spam.ba` jer se ona prva pojavljuje na listi, iako je adresa `spam_2@spam.ba` prva ispunila uslov. Adresa `nespam@spam.ba` nije ispunila uslov zato što je poslala pet poruka sa istim naslovom, ali je istu poruku poslala dva puta na `mail_3@pmf.unsa.ba`. Na petu adresu je poslala poruku, ali sa promijenjenim naslovom, pa i zbog toga nije ispunila uslov. Također, adresa `spam_1@spam.ba` se ispisuje samo jednom iako je dva puta zadovoljila uslov.

test01.in

```
spam_1@spam.ba mail_1@pmf.unsa.ba Povoljna ponuda
nespam@spam.ba mail_1@pmf.unsa.ba Povoljna ponuda
spam_2@spam.ba mail_1@pmf.unsa.ba Povoljna ponuda
spam_2@spam.ba mail_2@pmf.unsa.ba Povoljna ponuda
spam_2@spam.ba mail_3@pmf.unsa.ba Povoljna ponuda
spam_2@spam.ba mail_4@pmf.unsa.ba Povoljna ponuda
spam_2@spam.ba mail_5@pmf.unsa.ba Povoljna ponuda
spam_2@spam.ba mail_6@pmf.unsa.ba Povoljna ponuda
spam_1@spam.ba mail_2@pmf.unsa.ba Povoljna ponuda
spam_1@spam.ba mail_3@pmf.unsa.ba Povoljna ponuda
spam_1@spam.ba mail_4@pmf.unsa.ba Povoljna ponuda
spam_1@spam.ba mail_5@pmf.unsa.ba Povoljna ponuda
```

```
nespam@spam.ba mail_2@pmf.unsa.ba Povoljna ponuda
nespam@spam.ba mail_3@pmf.unsa.ba Povoljna ponuda
nespam@spam.ba mail_4@pmf.unsa.ba Povoljna ponuda
nespam@spam.ba mail_3@pmf.unsa.ba Povoljna ponuda
nespam@spam.ba mail_5@pmf.unsa.ba Povoljna akcija
spam_1@spam.ba mail_1@pmf.unsa.ba Povoljna akcija
spam_1@spam.ba mail_2@pmf.unsa.ba Povoljna akcija
spam_1@spam.ba mail_3@pmf.unsa.ba Povoljna akcija
spam_1@spam.ba mail_4@pmf.unsa.ba Povoljna akcija
spam_1@spam.ba mail_5@pmf.unsa.ba Povoljna akcija
```

Output:

```
spam_1@spam.ba
spam_2@spam.ba
```

Zadatak 9.23.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. Fajl čuva ocjene učenika jednog razreda iz predmeta: fizika, hemija i biologija. U prvoj liniji fajla se nalazi ime i prezime prvog učenika. U narednom redu se nalazi naziv predmeta prazno mjesto pa ocjena učenika iz tog predmeta. U redu ispod je naredni predmet sa svojom ocjenom, a u narednom treći predmet i ocjena. Nakon ovoga, u fajlu se nalazi novi red sa imenom i prezimenom narednog učenika, te u naredna 3 reda njegovi predmeti i ocjene. Redoslijed predmeta nije uvijek isti za svakog učenika. Također, broj učenika u fajlu nije poznat unaprijed i program treba pročitati podatke za sve navedene učenike u fajlu.

Program treba ispisati ime i prezime svakog učenika i njegov uspjeh (prosjek ocjena) u jednom redu odvojene praznim mjestom. Ukoliko učenik ima nezadovoljavajući uspjeh (ocjenu 1 iz nekog predmeta) program ispisuje njegovo ime i prezime, te listu predmeta koje učenik nije položio odvojene praznim mjestom.

test01.in

```
Edin Dzeko
Fizika 5
Hemija 4
Biologija 5
Semir Ceric
Bilogija 1
Hemija 2
Fizika 1
Mustafa Nadarevic
Hemija 4
Biologija 5
Fizika 3
```

Output:

```
Edin Dzeko 4.666666666666667
Semir Ceric Biologija Fizika
Mustafa Nadarevic 4.0
```

Zadatak 9.24.

Roboti se kreću u ravni, a startna pozicija im je $(0,0)$. Svaki robot se može pomjerati određeni broj koraka u smjerovima: gore (`'G'`), dole (`'DO'`), lijevo (`'L'`) i desno (`'DE'`). Dat je fajl pod nazivom `"test01.in"`. U fajlu je lista komandi za više robota, svaka zasebnom redu.

Svaka komada započinje oznakom robota kojem je upućena ta komanda. Oznake su u formatu R# gdje # predstavlja jednocifreni broj robota. Nakon toga slijedi prazno mjesto pa jedna od četiri oznake smjera kretanja. Zatim ide prazno mjesto pa broj koraka koje robot pravi u tom smjeru.

Potrebno je napisati program koji će pročitati ovakav fajl i ispisati udaljenost od ishodišta, tj. tačke $(0,0)$, do tačke na kojoj se robot zaustavio, za svakog robota. Prilikom ispisa roboti se sortiraju i ispisuju u zasebnim redovima sa pripadajućom udaljenošću.

test01.in

```
R2 G 2
R1 L 4
R2 DE 1
R3 DO 5
R1 G 3
R3 DE 2
```

Output:

```
R1 5.0
R2 2.23606797749979
R3 5.385164807134504
```

Zadatak 9.25.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. U fajlu se nalaze podaci o prodanim automobilima. U svakom redu se nalazi kôd prodavača koji je prodao automobil (sastavljen od brojeva i slova), marka automobila i cijena. Ova tri podatka su odvojena praznim mjestom.

Program prvo ispisuje kodove tri najuspješnija prodavača na osnovu ukupne sume prodaje. Kodovi se ispisuju jedan ispod drugog poredani po uspješnosti. Uz svaki kod se ispisuje i ukupna suma koju je prodavač ostvario, odvojena praznim mjestom. Program zatim ispisuje tri najprodavanije marke automobila na osnovu broja prodanih automobila. Marke se ispisuju jedna ispod druge. Uz svaku se ispisuje i broj prodanih primjeraka.

test01.in

```
P10 VW 40000
P2 Audi 45000
P10 BMW 50000
P32 Audi 50000
P32 VW 30000
P2 BMW 28000
P4 Ferarri 4000000
P1 VW 42000
P1 VW 10000
P14 Audi 30000
```

Output:

```
P4 4000000
P10 90000
P32 80000
VW 4
Audi 3
BMW 2
```

Zadatak 9.26.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. Svaki red u fajlu predstavlja rezultat fudbalske utakmice. Svaki red počinje nazivom kluba domaćina zatim ide prazno mjesto, zatim

broj postignutih golova domaćina, prazno mjesto, povlaka, prazno mjesto, broj postignutih golova gostiju, prazno mjesto i na kraju naziv gostujućeg kluba.

Ekipa koja pobjedi dobiva 3 boda, ekipa koja izgubi dobiva 0 bodova, a ukoliko je rezultat neriješen obje ekipe dobivaju po 1 bod.

Program treba da na osnovu rezultata sortira i ispiše klubove prema broju osvojenih bodova. Program ispisuje naziv najuspješnijeg kluba zatim prazno mjesto i osvojen bodove, prelazi u novi red i ispisuje naziv i bodove narednog kluba i nastavlja tako sve dok ne ispiše sve klubove.

Možete pretpostaviti da dvije ekipe neće imati isti broj bodova.

test01.in

```
Sarajevo 3 - 0 Zeljeznicar
Sloboda 2 - 1 Mladost
Zeljeznicar 2 - 2 Sloboda
Mladost 0 - 2 Sarajevo
```

Output:

```
Sarajevo 6
Sloboda 4
Zeljeznicar 1
Mladost 0
```

Zadatak 9.27.

Napisati program koji otvara fajl pod nazivom `"test01.in"`. Svaki red u fajlu predstavlja vrijeme. Vrijeme je dato u narednom formatu: prvo su navedeni **sati** kao cijeli broj, zatim slijedi **znak dvotačka**, potom su navedene **minute** kao cijeli broj, nakon čega se ispisuje **prazno mjesto**, a na kraju se ispisuje jedna od dvije oznake `AM` ili `PM`. **Program treba ispisati vremena sortirana od najranijeg do najkasnijeg, svako u zasebnom redu.**

Pravila:

- Oznaka `AM` označava prijepodnevno vrijeme, a oznaka `PM` označava poslijepodnevno vrijeme. U ulaznom fajlu oznake ne moraju biti napisane isključivo velikim slovima već se mogu očekivati sve kombinacije: `AM`, `Am`, `aM` i `am`. **Prilikom ispisa potrebno ih je ispisati velikim slovima.**
- U 12 časovnom prikazu vremena, sati ne mogu imati vrijednost 0, ali mogu imati vrijednost 12 (koja mijenja 0). Stoga je ponoć predstavljena kao `12:0 AM`, a podne kao `12:0 PM`. Pola sata iza ponoći predstavlja se kao `12:30 AM`.
- Jednocifrene vrijednosti sati i minuta su uvijek date bez prethodnog znaka `'0'`, što znači da će jutarnjih 5 sati i 2 minute, biti predstavljeno kao `5:9 AM`. **Prilikom ispisa, potrebno je dodati znak `0` kod jednocifrenih vrijednosti**, pa će `5:9 AM` postati `05:09 AM`.
- Dato vrijeme uvijek će biti u ispravnom formatu i imat će validne vrijednosti (tj. neće se pojavljivati vrijednosti tipa: `-8:62 PM` ili `13:30 PM`).

test01.in

```
1:45 Pm
11:59 AM
1:35 am
12:4 pm
11:20 am
12:30 PM
11:28 pM
12:1 Am
5:8 am
```

Output:

```
12:01 AM
01:35 AM
05:08 AM
11:20 AM
11:59 AM
12:04 PM
12:30 PM
01:45 PM
11:28 PM
```

Zadatak 9.28.

Napisati program koji otvara fajl `test01.in`. U fajlu se nalazi matrica gdje su polja označena crticama, koje su razdvojene praznim mjestom. Jedno polje je označeno znakom $*$ koji predstavlja loptu. Ispod matrice se nalazi oznaka jednog od četiri dijagonalna smjera u kojem se lopta počinje kretati. Oznake koje se koriste su: `SI` za sjeveroistok, `SZ` za sjeverozapad, `JI` za jugoistok i `JZ` za jugozapad. Kada dođe do ivice matrice, lopta se odbija pod pravim uglom u odnosu na dotadašnji smjer kretanja. Ukoliko dođe u ćošak matrice lopta se počinje kretati u suprotnom smjeru.

Program treba iscrtati izgled matrice, nakon što lopta pređe 100 polja, gdje će sva polja koja je lopta prešla biti označena znakom $*$, a ostala ostaju nepromijenjena tj. označena znakom `-`. Možete pretpostaviti da matrica neće biti dimenzija manjih od 2×2.

`test01.in`

```
- - - - - -
- - - - - -
- - - - - -
- * - - - -
- - - - - -
SI
```

Output:

```
* - * - * -
- * - * - *
* - * - * -
- * - * - *
* - * - * -
```

10. Rekurzija

U procesu rješavanja problema, ljudski um često pokušava problem svesti na ranije poznate probleme. Rekurzija koristi upravo tu osobinu, sa modifikacijom da pri svođenju problema na jednostavniji, problem svodimo na **isti** jednostavniji problem. Nakon ispravnog shvatanja rekurzije, postoji značajna klasa problema koje je mnogo lakše riješiti rekurzivno nego bez korištenja rekurzije.

Opišimo rekurziju na primjeru gdje je neophodno napisati funkciju koja računa **faktorijel** nekog broja. Pri računanje faktorijela broja, matematički zapisano, faktorijel se označava sa

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-2) \cdot (n-1) \cdot n.$$

Grupisanjem prvih $(n-1)$ elemenata u proizvodu na desnoj strani, izraz možemo pisati kao

$$n! = (n-1)! \cdot n.$$

Ukoliko sa $f(n)$ označimo vrijednost proizvoda prvih n brojeva, vidimo da relaciju možemo pisati kao

$$f(n) = f(n-1) \cdot n.$$

Detaljnijim posmatranjem datog izraza, moguće je uočiti da smo upravo vrijednost faktorijela broja n definisali preko vrijedosti faktorijela broja $n-1$, uz *manju* modifikaciju dobijenu množenjem te vrijednosti sa n. Sada možemo računati

$$f(n) = f(n-1) \cdot n = f(n-2) \cdot (n-1) \cdot n = \ldots$$

Postavlja se pitanje, gdje stati? Nastavljanjem sličnog postupka, možemo dobiti

$$f(n) = f(-3) \cdot (-2) \cdot (-1) \cdot 0 \cdot \ldots \cdot n.$$

Navedeni izraz očigledno nije ispravan, jer se u proizvodu nalazi broj 0, a istovremeno je neophodno izračunati faktorijel negativnog broja (što se ne definiše). Stoga, neophodno je odrediti baznu vrijednost (vrijednost kod koje stajemo sa računanjem). U slučaju faktorijela, to je $f(0) = 1$.

Sada se računanje faktorijela svodi na definiciju baznog slučaja, $f(0) = 1$, te se svaki slučaj koji nije bazni računa kao $f(n) = f(n-1) \cdot n$. Time računanje faktorijela za n svodimo na računanje faktorijela za $n-1$ i postupak nastavljamo dok ne dođemo do baznog slučaja.

10.1 Definicija rekurzivnih funkcija

Iz programerskog aspekta, susretali smo se sa funkcijama koje pozivaju druge funkcije (npr. značajan broj naših funkcija poziva funkciju `print`). Rekurzivna funkcija je funkcija koja poziva **samu sebe**. Osnovna ideja jeste da se definiše bazni slučaj, te da funkcija pozove samu sebe sa parametrima kojima se postepeno približava poziv funkcije u baznom slučaju. Kod za faktorijel funkciju naveden je ispod.

```
def f(n):
    if n == 0:
        return 1
    return n * f(n-1)
```

Kodovi za rekurziju standardno su kraći od nerekurzivnih. Svaka rekurzivna funkcija mora ispunjavati tri uslova navedena ispod.

- Definisati bazne slučajeve. Bazni slučajevi definišu se na samom početku funkcije i neophodno je da garantujemo da će se funkcija svesti na njih nakon konačnog broja koraka.
- Unutar funkcije koju kodiramo, poziva se ista ta funkcija (rekurzivni poziv). Poziv funkcije treba biti zasnovan na manjim instancama.
- Izbjegavati beskonačnu petlju!

Dakle, pri rješavanju određenog problema koristeći rekurziju, neophodno je definisati bazne slučajeve, odrediti način svođenja rješenja problema za veću instancu na uzorak manje instance, pri čemu smatramo da problem za manju instancu znamo riješiti. I tu teorija prestaje. U slučaju faktorijela, definisali smo bazne slučajeve $f(0) = 1$, definisali smo računanje veće instance $f(n)$ preko manjih instanci koje pretpostavljamo da znamo izračunati ($f(n) = f(n-1) \cdot n$) i garantujemo da ćemo procesom smanjivanja n za jedan u svakom rekurzivnom pozivu svesti slučaj na bazni.

Kako je navedeno, u primjeru iznad, provjeravamo da li je unesena vrijednost jednaka 0. U slučaju da jeste, vraćamo vrijednost 0!, jer je to vrijednost poznata pri definiciji zadatka. Za svaku drugu vrijednost, faktorijel računamo kao $n \cdot f(n-1)$, gdje ne ulazimo u proces računanja $f(n-1)$, što i čini jednostavnost rekurzivnih kodova. Možemo smatrati da pri računanju faktorijela za vrijednost n znamo izračunati sve faktorijele za vrijednosti manje od n.

> ! **Oprez.** U slučaju nenavođenja ispravnih baznih slučajeva, funkcija može upasti u beskonačnu petlju što je česta greška u radu sa rekurzivnim funkcijama.

10.2 Primjeri

Naredni dio poglavlja sadrži riješena i detaljno objašnjena dva popularna primjera za upotrebu rekurzije: računanje Fibonačijevih brojeva i Hanoi kule.

Primjer 10.1 — Fibonačijevi brojevi. Napisati funkciju koja za proslijeđenu vrijednost računa vrijednost n-tog Fibonačijevog broja.

```
def fib(n):
    if n == 1 or n == 2:
        return 1

    return fib(n-2) + fib(n-1)
```

Fibonačijevi brojevi definišu se kao niz brojeva čija su prva dva elementa jedinice, a svaki naredni element definiše se rekurzivno kao suma prethodna dva. Matematički zapisano, $f(1) = 1$, $f(2) = 1$ i

$$f(n) = f(n-1) + f(n-2), \forall n > 2.$$

Prvi dio funkcije sačinjavaju uslovi zaustavljanja (bazni slučajevi). Kako je rečeno, prvi i drugi Fibonačijevi brojevi su jedinice. Svaki naredni element, računa se kao suma prethodna dva, pa je rekurzivni poziv `fib(n-2)+fib(n-1)`. Važno je uočiti da u slučaju funkcije `fib` vršimo poziv funkcije za dvije manje instance. Stoga, neophodno je navesti bar dva bazna slučaja. Ista situacija vrijedi kod više rekurzivnih poziva (ako imamo poziv za instance $n-1$, $n-2$ i $n-3$, neophodno je navesti bar tri bazna slučaja). Obratite pažnju da bazni slučajevi ne moraju biti konkretne instance (`n==1` i slično), to mogu biti i drugačiji matematički izrazi, npr. `n<=2`. U primjeru za Fibonačijeve brojeve, takav uslov bi bio `if n <= 2` umjesto `if n==1 or n==2`. U oba slučaja, pokrivamo bar dvije vrijednosti, s tim da slučaj sa operatorom `<=` definiše dešavanja u slučaju unosa negativnih brojeva. ■

Osim navedenih primjera, postoji značajan broj problema koje je jednostavnije riješiti rekurzivno, nego bez rekurzije. Važno je napomenuti da se svako rekurzivno rješenje može pretvoriti u rješenje koje ne uključuje rekurziju. Ovaj proces se uobičajno naziva oslobađanje rekurzije.

Primjer 10.2 — Hanoi kule. Napisati funkciju `Hanoi(n, a, b, c)` koja simulira rješavanje Hanoi kula, pri čemu se premješta `n` diskova sa kule `a` na kulu `c` uz pomoć kule `b`.

Hanoi kule predstavljaju zanimljivu igru koja se sastoji od tri kule i n diskova poredanih na jednoj kuli od najvećeg do najmanjeg (najmanji na vrhu). Konačan cilj igre je premještanje svih n diskova sa jedne kule na drugu. U svakom koraku prebacuje se tačno jedan disk i nikada se ne smije desiti slučaj kada je neki veći disk stavljen iznad manjeg na kuli.

```
def Hanoi(n, a, b, c):
   if n == 1:
      print("Prebacujem disk sa " + a + " na " + c)
      return

   Hanoi(n-1, a, c, b)
   Hanoi(1, a, b, c)
   Hanoi(n-1, b, a, c)
```

Postupak za rješavanje Hanoi kula izrazito se komplikuje povećanjem broja diskova. Međutim, proces se jednostavno opisuje pomoću rekurzije, gdje za rješavanje Hanoi kule sa n diskova pretpostavljamo da isti problem možemo riješiti za bilo koji slučaj sa manje od n diskova.

Neka imamo n diskova na kuli a, te iste želimo premjestiti na kulu c uz korištenje b. Postupak se svodi na prebacivanje $n-1$ diskova sa a na b. Sada je na kuli a ostao jedan (najveći) disk, kojeg prebacujemo sa kule a na kulu c. Trenutno na kuli a nemamo diskova, na kuli c imamo najveći disk, a na kuli b imamo $n-1$ disk poredan po veličini. Stoga, kako je pretpostavka da znamo riješiti Hanoi kule za $n-1$ diskova, koristimo to znanje da prebacimo diskove sa b na c uz pomoć a. ■

Razmišljati o rekurzivnom rješenju može biti korisno u svakoj situaciji gdje rješavanje nekog problema možemo svesti na rješavanje istog problema sa manjim instancama (manjim brojevima, manjim brojevima elemenata, manjim dužinama stringova i slično).

10.3 Zadaci

Zadatak 10.1.

Napisati funkciju koja računa sumu kvadrata prvih n prirodnih brojeva. Program od korisnika uzima vrijednost n a ispisuje sumu koju računa rekurzivno.

Input:

```
4
```

Output:

```
30
```

Zadatak 10.2.

Napisati program koji od korisnika uzima prirodan broj n, te potom rekurzivno računa i ispisuje vrijednost izraza:

$$S(n) = \frac{1}{1^3+1} + \frac{1}{2^3+2} + \ldots + \frac{1}{n^3+n}$$

Input:

```
5
```

Output:

```
0.6557315233785821
```

Zadatak 10.3.

Korisnik unosi matricu cijelih brojeva veličine 4×4. Kolone su razdvojene praznim mjestom, a redovi se ispisuju jedan ispod drugog.

Počevši u gornjem lijevom uglu i krećući se samo ili desno ili dole, potrebno je pronaći minimalnu putanju do donjeg desnog ugla. Minimalna putanja je ona čiji je zbir posjećenih polja najmanji. Program ispisuje sumu vrijednosti minimalne putanje.

U primjeru ispod putanju čine vrijednosti 1, 10, 6, 5, 1, 2, 1.

Input:

```
1 20 30 1
10 6 5 1
2 11 4 2
3 29 2 1
```

Output:

```
26
```

Zadatak 10.4.

Napisati program koji od korisnika traži unos liste prirodnih brojeva. Brojevi se unose jedan ispod drugog. Kraj unosa se označava unosom vrijednosti -1. Program rekurzivno pronalazi najveći broj u listi i ispisuje ga.

Input:

```
12
19999
1233
4
-1
```

Output:

```
19999
```

Zadatak 10.5.

Napisati program koji od korisnika traži unos prirodnog broja n. Program rekurzivno računa sumu cifara broja n i ispisuje je.

Input:	Output:
19999	37

II Rješenja

11. Varijable, unos i izrazi

Zadatak 2.1.

```
1  import math
2
3  alfa = float(input())
4  n = float(input())
5  k = float(input())
6  h = float(input())
7
8  alfa = math.radians(alfa)
9
10 rezultat = (2 * (k ** (2/3))) / \
11     (math.sin(alfa) * math.sqrt(h + n)) \
12     * math.cos(alfa)
13
14 print(rezultat)
```

Zadatak 2.2.

```
1  import math
2
3  alfa = float(input())
4  beta = float(input())
5  c = float(input())
6
7  gama = 180 - alfa - beta
8  ar = math.radians(alfa)
9  br = math.radians(beta)
```

```
gr = math.radians(gama)

print(c * (math.sin(ar/2) * math.sin(br/2))/(math.cos(gr/2)))
```

Zadatak 2.3.

```
import math

a = float(input())
b = float(input())
c = float(input())
theta = float(input())
phi = float(input())

tr = math.radians(theta)
pr = math.radians(phi)

x = a * math.sin(tr) * math.cos(pr)
y = b * math.cos(tr) * math.sin(pr)
z = c * math.sin(tr)

print(x)
print(y)
print(z)
```

Zadatak 2.4.

```
C = float(input())
F = (9 / 5) * C + 32
print(F)
```

Zadatak 2.5.

```
import math

b = float(input())
c = float(input())
alfa = float(input())

a = math.sqrt(b ** 2 + c ** 2 - 2 * b * c *
              math.cos(math.radians(alfa)))

print(a)
```

Zadatak 2.6.

```
import math
```

```
2
3 R = float(input())
4 r = float(input())
5
6 V = 2 * (math.pi ** 2) * R * r ** 2
7
8 print(V)
```

Zadatak 2.7.

```
1  import math
2
3  x = float(input())
4  y = float(input())
5
6  r = math.sqrt(x**2 + y**2)
7  phi = math.degrees(math.atan2(y, x))
8
9  print(r)
10 print(phi)
```

Zadatak 2.8.

```
1 import math
2
3 v0 = float(input())
4 R = float(input())
5 alfa = float(input())
6 alfar = math.radians(alfa)
7
8 g = (v0 ** 2) / R * math.sin(2*alfar)
9 print(g)
```

Zadatak 2.9.

```
1  import math
2  g = 9.81
3  x = float(input())
4  y = float(input())
5  alfa = float(input())
6  alfar = math.radians(alfa)
7  brojnik = x * x * g
8  nazivnik = x * math.sin(2 * alfar) -
9             2 * y * (math.cos(alfar) ** 2)
10 v0 = math.sqrt(brojnik/nazivnik)
11 print(v0)
```

Zadatak 2.10.

```
from math import *

Va = float(input())
theta = float(input())
L = float(input())
tu = float(input())

td = 1 / ((2*Va*cos(radians(theta))/L) + (1/tu))

print(td)
```

12. Naredbe grananja

Zadatak 3.1.

```
1  vjezbe = int(input("Vjezbe: "))
2  parcijalni = int(input("Parcijalni ispit: "))
3  zavrsni = int(input("Zavrsni ispit: "))
4
5  ukupno = vjezbe + parcijalni + zavrsni
6
7  if ukupno > 100:
8     print("GRESKA!")
9  elif ukupno >= 95:
10    print("Deset")
11 elif ukupno >= 85:
12    print("Devet")
13 elif ukupno >= 75:
14    print("Osam")
15 elif ukupno >= 65:
16    print("Sedam")
17 elif ukupno >= 55:
18    print("Sest")
19 else:
20    print("Pet")
```

Zadatak 3.2.

```
1  x = float(input())
2
3  if x <= -2:
```

```
    print(-1)
elif 0 < x and x <= 100:
    print(1/(3+2/x))
elif 105 >= x:
    print(200)
else:
    print(0)
```

Zadatak 3.3.

```
n = int(input())

if n > 99 and n < 1000:
    c = n % 10
    n = n // 10
    b = n % 10
    n = n // 10
    a = n

    print(a*b*c, a+b+c)
else:
    print("Pogresan unos!")
```

Zadatak 3.4.

```
from math import sqrt

epsilon = 0.000000000000001
x = float(input())
y = float(input())
r = float(input())

Ax = float(input())
Ay = float(input())

d = sqrt((Ax-x)**2 + (Ay-y)**2)

if d < r:
    print("Tacka pripada unutrasnjosti kruga")
elif d - r < epsilon and r - d < epsilon :
    print("Tacka pripada kruznici")
else:
    print("Tacka ne pripada krugu")
```

Zadatak 3.5.

```
x = int(input())
```

```
2
3  if x > 9999 and x < 100000:
4     e = x % 10
5     x = x // 10
6     d = x % 10
7     x = x // 10
8     c = x % 10
9     x = x // 10
10    b = x % 10
11    x = x // 10
12    a = x
13
14    tmax = a
15    if b > tmax:
16       tmax = b
17
18    if c > tmax:
19       tmax = c
20
21    if d > tmax:
22       tmax = d
23
24    if e > tmax:
25       tmax = e
26
27    tmin = a
28    if b < tmin:
29       tmin = b
30
31    if c < tmin:
32       tmin = c
33
34    if d < tmin:
35       tmin = d
36
37    if e < tmin:
38       tmin = e
39
40    print(tmax - tmin)
41 else:
42    print("Pogresan unos!")
```

Zadatak 3.6.

```
1  from math import sqrt
2  epsilon = 0.000001
3
```

```
4  x1 = float(input())
5  y1 = float(input())
6
7  x2 = float(input())
8  y2 = float(input())
9
10 r = float(input())
11 d = sqrt((x2-x1)**2 + (y2-y1)**2)
12
13 # jer je poluprecnik jednak precniku
14 dr = r * 1.5 # udaljenost centara
15
16 if d < dr:
17     print("Kruznice se sijeku")
18 elif d - dr < epsilon and dr - d < epsilon :
19     print("Kruznice se dodiruju")
20 else:
21     print("Kruznice nemaju zajednickih tacaka")
```

13. Petlje

Zadatak 4.1.

```
1  n = int(input())
2
3  suma = 0
4  for i in range(n):
5      t = input()
6      if t == 'programiranje':
7          suma += 1
8
9  if suma == 2:
10     print('jeste')
11 else:
12     print('nije')
```

Zadatak 4.2.

```
1  n = int(input())
2
3  suma = 0
4  while n != -1:
5      if n < 10:
6          suma += n % 10
7      else:
8          suma += (n // 10) % 10
9      n = int(input())
10
11 print(suma)
```

Zadatak 4.3.

```
n = int(input())

for i in range(n):
    a = int(input())
    for j in range(a):
        if i % 2 == 0:
            print('-', end='')
        else:
            print('*', end='')
    print()
```

Zadatak 4.4.

```
n = int(input())

suma = 0
while n != -1:
    suma += n % 10
    n = int(input())

print(suma)
```

Zadatak 4.5.

```
a = input()
sead = 0
harun = 0
brojac = 0
while a != '':
    a = float(a)
    if brojac%2 == 0:
        harun += a
    else:
        sead += a
    brojac += 1
    a = input()

print(harun - sead)
```

Zadatak 4.6.

```
a = int(input())
while a != -1:
    if a % 2 == 0:
        print(-a)
    else:
```

```
6           print(a)
7       a = int(input())
```

Zadatak 4.7.

```
1 n = input()
2 while n != '':
3     print(n, end='␣')
4     n = input()
```

Zadatak 4.8.

```
1 n = input()
2 while n != '':
3     n = int(n)
4     print(-n, end='␣')
5     n = input()
```

Zadatak 4.9.

```
1  n = input()
2  nalazi_se = False
3  while n != '':
4      n = int(n)
5      if n == 2:
6          nalazi_se = True
7      n = input()
8  if nalazi_se:
9      print('Nalazi␣se!')
10 else:
11     print('Ne␣nalazi␣se!')
```

Zadatak 4.10.

```
1  n = int(input())
2
3  najveca = n % 10
4  najmanja = n % 10
5
6  while n * 10 // 10 > 0:
7      cifra = n % 10
8      if cifra > najveca:
9          najveca = cifra
10     if cifra < najmanja:
11         najmanja = cifra
12     n //= 10
13
14 print(najveca * najmanja - najveca / najmanja)
```

Zadatak 4.11.

```
a = int(input())

max_raspon = 0
t_raspon = 0

while a > 0:
    cifra = a % 10
    if cifra < 5:
        if t_raspon > max_raspon:
            max_raspon = t_raspon
        t_raspon = 0
    else:
        t_raspon += 1
    a //= 10

if t_raspon > max_raspon:
            max_raspon = t_raspon
print(max_raspon)
```

Zadatak 4.12.

```
a = int(input())
b = int(input())

for i in range(a, b + 1):
    if i % 3 == 0:
        print(i, end='␣')
    else:
        djeljiv = False
        n = i
        while n != 0:
            if n % 10 % 3 == 0:
                djeljiv = True
            n //= 10
        if djeljiv:
            print(i, end='␣')
```

Zadatak 4.13.

```
n = int(input())

broj_finih = 0
broj = 9
while broj_finih != n:
    broj += 1
    t = broj
```

```
8       c1 = t%10
9       t //= 10
10      while t > 0:
11          c2 = t%10
12          if c1 + c2 == 4:
13              broj_finih += 1
14              break
15          c1 = c2
16          t //= 10
17
18  print(broj)
```

Zadatak 4.14.

```
1   m = int(input())
2   n = int(input())
3
4   for i in range(m):
5       trenutni = 1
6       for j in range(n):
7           print(trenutni, end="␣")
8           if trenutni == i+1:
9               trenutni = 1
10          else:
11              trenutni += 1
12      print()
```

Zadatak 4.15.

```
1   for a in range(1, 10):
2       for b in range(0, 10):
3           for c in range(0, 10):
4               broj = a * 100 + b * 10 + c
5               if broj == (a ** 3 + b ** 3 + c ** 3):
6                   print(broj, end='␣')
```

Zadatak 4.16.

```
1   n = int(input())
2
3   for i in range(n):
4       for j in range(n * 2 + 1):
5           if i % 2 == 0 or j % 2 == 0:
6               print('x', end='')
7           else:
8               print('0', end='')
9       print()
```

Zadatak 4.17.

```
n = int(input())

jesu = True

while n // 10 != 0:
    cifra1 = n % 10
    n = n // 10
    cifra2 = n % 10
    if cifra1 > cifra2:
        jesu = False

if jesu:
    print('jesu')
else:
    print('nisu')
```

Zadatak 4.18.

```
n = int(input())

kraj = False
ispunjen_uslov = False
proizvod = 1

while not kraj:
    cifra = n % 10
    if cifra % 2 == 1 and cifra < 6:
        ispunjen_uslov = True
        proizvod *= cifra
    n //= 10
    if n == 0:
        kraj = True

if ispunjen_uslov:
    print(proizvod)
else:
    print(0)
```

Zadatak 4.19.

```
n = int(input())

s = 1
while s % n != 0:
    s = s * 10 + 1

```

```
print(len(str(s)))
```

Zadatak 4.20.

```
n = int(input())

while n != -1:
    hiljade = 0
    rezultat = ''
    while n > 0:
        rezultat = str(n%10) + rezultat
        n = n // 10
        hiljade += 1
        if n > 0 and hiljade == 3:
            rezultat = '.' + rezultat
            hiljade = 0
    print(rezultat)
    n = int(input())
```

Zadatak 4.21.

```
a = int(input())
b = int(input())

for i in range(a, b+1):
    suma = 0
    for j in range(1, i//2 + 1):
        if i % j == 0:
            suma += j
    if suma == i:
        print(i)
```

Zadatak 4.22.

```
for i in range(10, 100):
    if i**2 < 1000 and i == i**2 % 100:
        print(i, i**2, sep='\n')
```

Zadatak 4.23.

```
n = int(input())
m = int(input())

for i in range(1, n + 1):
    if i < 10:
        print('␣', end='')
        print(i, end='␣')
```

```
8       else:
9           print(i, end='␣')
10      if i % m == 0:
11          print()
```

Zadatak 4.24.

```
1  n = int(input())
2
3  brojac = 0
4
5  trenutni = 1
6
7  while brojac < n:
8      trenutni += 1
9      cudan = True
10     kopija = trenutni
11     while kopija > 0:
12         if kopija % 10 != 2 and kopija % 10 != 3:
13             cudan = False
14         kopija //= 10
15
16     if cudan:
17         brojac += 1
18
19 print(trenutni)
```

Zadatak 4.25.

```
1  n = input()
2  zadnji = 0
3  predzadnji = 0
4  drugi = 0
5  i = 1
6
7  while n != '':
8      n = int(n)
9      predzadnji = zadnji
10     zadnji = n
11     if i == 2:
12         drugi = n
13     i += 1
14     n = input()
15
16 print(drugi * predzadnji)
```

Zadatak 4.26.

```
n = int(input())

trenutni = 0
koliko = 1
seadu = True
harun = 0
sead = 0
trenutno = 0

while trenutni < n:

    s = int(input())

    if seadu:
        sead += s
        trenutno += 1
        if trenutno == koliko:
            trenutno = 0
            koliko += 1
            seadu = False
    else:
        harun += s
        trenutno += 1
        if trenutno == koliko:
            trenutno = 0
            koliko += 1
            seadu = True

    trenutni += 1

print(sead)
print(harun)
```

Zadatak 4.27.

```
poz1 = int(input())
skok1 = int(input())
poz2 = int(input())
skok2 = int(input())
poz3 = int(input())
skok3 = int(input())
cilj = int(input())
max = 0

while True:
    poz1 += skok1
    if poz1 >= cilj:
```

```
13          max = poz1
14          break;
15      poz2 += skok2
16      if poz2 >= cilj:
17          max = poz2
18          break;
19      poz3 += skok3
20      if poz3 >= cilj:
21          max = poz3
22          break
23
24  for i in range(1, max+1):
25      if i != cilj and i != poz1:
26          print('-', end='')
27      elif i == poz1:
28          print('*', end='')
29      else:
30          print('|', end='')
31  print()
32
33  for i in range(1, max+1):
34      if i != cilj and i != poz2:
35          print('-', end='')
36      elif i == poz2:
37          print('*', end='')
38      else:
39          print('|', end='')
40  print()
41
42  for i in range(1, max+1):
43      if i != cilj and i != poz3:
44          print('-', end='')
45      elif i == poz3:
46          print('*', end='')
47      else:
48          print('|', end='')
49  print()
```

Zadatak 4.28.

```
1  r = int(input())
2  k = int(input())
3  x1 = int(input())
4  y1 = int(input())
5  d = int(input())
6
7  for i in range(r):
```

```
    for j in range(k):
        if i < y1 or i > y1 + d - 1 or j < x1 or
           j > x1 + d -1 or j < x1 + (i - y1):
            print('-', end='␣')
        else:
            print('X', end='␣')
    print()
```

Zadatak 4.29.

```
a = int(input())
b = int(input())

temp = b
desetine = 1
bc2 = 0
while temp != 0:
    desetine *= 10
    temp //= 10
    bc2 += 1

bc1 = 0
index = -1
while a != 0:
    temp = a % desetine
    if temp == b:
        index = bc1
    a //= 10
    bc1 += 1

if index == -1:
    print(index)
else:
    print(bc1 - index - bc2)
```

Zadatak 4.30.

```
a = int(input())
b = int(input())

r = max(a, b)
k = abs(a - b) + 1

for i in range(r):
    for j in range(k):
        if (b <= a and j < i+1) or \
           (a < b and j+a > r-i-1):
```

```
            print('*', end='␣')
        else:
            print('␣', end='␣')
    print()
```

Zadatak 4.31.

```
n = int(input())

brojKoraka = 0

listaPrethodnih = [n]

pojavilaSeJedinica = True
while n != 1:
    sumaKvadrataCifara = 0
    t = n
    while t > 0:
        sumaKvadrataCifara += (t % 10) * (t % 10)
        t = t // 10
    n = sumaKvadrataCifara
    if n in listaPrethodnih:
        print(-1)
        pojavilaSeJedinica = False
        break
    listaPrethodnih.append(n)
    brojKoraka += 1

if pojavilaSeJedinica:
    print(brojKoraka)
```

Zadatak 4.32.

```
matrica = []

red = input()
while red != '':
    matrica.append(red.split())
    red = input()
visina = len(matrica)
sirina = len(matrica[0])

najpovrsina = 0

for gly in range(visina):
    for glx in range(sirina):
        for ddy in range(gly, visina+1):
```

```
            for ddx in range(glx, sirina+1):
                popunjen = True
                povrsina = 0
                for i in range(gly, ddy):
                    for j in range(glx, ddx):
                        if matrica[i][j] == '-':
                            popunjen = False
                        povrsina += 1
                if popunjen and povrsina > najpovrsina:
                    najpovrsina = povrsina
                    najgly = gly
                    najglx = glx
                    najddy = ddy-1
                    najddx = ddx-1

print(najpovrsina)
print(najgly, najglx)
print(najddy, najddx)
```

Zadatak 4.33.

```
n = int(input())
print('␣␣␣', end='')
for i in range(1, n+1):
    if i < 10:
        print('␣', end = '')
    print(str(i), end = '␣␣')
print()
for i in range(1, n+1):
    if i < 10:
        print('␣', end = '')
    print(i, end = '')
    for j in range(1, n+1):
        if i * j < 10:
            print('␣', end ='')
        if(i * j < 100):
            print('␣', end = '')
        print(str(i * j), end='␣')
    print()
```

Zadatak 4.34.

```
maxb = 0
maxn = 0
for i in range(1, 10000):
    brojac = 1
    n = i
```

```
6       while n != 1:
7           if n % 2 == 0:
8               n //= 2
9           else:
10              n = 3 * n + 1
11          brojac += 1
12          #print(n)
13      if brojac > maxb:
14          maxb = brojac
15          maxn = i
16
17  print(maxn)
```

14. Nasumični brojevi

Zadatak 5.1.

```
1  import random
2
3  BROJ_SIM = 10000
4  broj_uspjeha = 0
5
6  for i in range(BROJ_SIM):
7      ppn = 0 # prije prethodne note
8      pn = 0 # prethodna nota
9      for j in range(10):
10         nota = random.randint(1, 7)
11         if ppn == 1 and pn == 3 and nota == 5:
12             broj_uspjeha += 1
13             break
14         ppn = pn
15         pn = nota
16 print(broj_uspjeha / BROJ_SIM)
```

Zadatak 5.2.

```
1  import random
2  import math
3
4  n = float(input())
5  sims = 100000
6  ukupno = 0
7
```

```
for i in range(0, sims):
    x1 = random.uniform(0, n)
    y1 = random.uniform(0, n)
    x2 = random.uniform(0, n)
    y2 = random.uniform(0, n)

    d = math.sqrt((x2 - x1) ** 2 +
                  (y2 - y1) ** 2)
    ukupno += d

print(ukupno/sims)
```

Zadatak 5.3.

```
import random

sims = 100000
cilj = 0

for j in range(sims):
    x = 0
    y = 0
    for i in range(10):
        d = random.randint(1, 4)
        if d == 1:
            y += 1
        elif d == 2:
            y -= 1
        elif d == 3:
            x -= 1
        elif d == 4:
            x += 1
    if x == 0 and y == 0:
        cilj += 1

print(cilj / sims)
```

Zadatak 5.4.

```
import random

sims = 10000

broj_bacanja = 0

for i in range(sims):
    b1 = False
```

```
    b2 = False
    b3 = False
    broj = 0
    while b1 == False or b2 == False or b3 == False:
        k1 = random.randint(1, 6)
        k2 = random.randint(1, 6)
        k = k1 + k2
        broj += 1
        b1 = b2
        b2 = b3
        if k > 6:
            b3 = True
        else:
            b3 = False
    broj_bacanja += broj

print(broj_bacanja / sims)
```

Zadatak 5.5.

```
import random

broj_sim = 10000
broj_pobjeda = 0
for i in range(broj_sim):
    if random.randint(1, 6) > \
            random.randint(1, 9):
        broj_pobjeda += 1

print(broj_pobjeda / broj_sim)
```

Zadatak 5.6.

```
import random

n = int(input())

BROJ_SIM = 10000
broj_uspjeha = 0

for j in range(0, BROJ_SIM):
    broj_sestica = 0
    for i in range(0, 30):
        kockica = random.randint(1, 6)
        if kockica == 6:
            broj_sestica += 1
        else:
```

```
            broj_sestica = 0
        if broj_sestica >= n:
            broj_uspjeha += 1
            break

print(100 * broj_uspjeha / BROJ_SIM)
```

Zadatak 5.7.

```
import random

BROJ_SIM = 10000
broj_sudara = 0

for i in range(BROJ_SIM):
    r1x = 0
    r1y = 0
    r2x = 2
    r2y = 0
    sudar = False
    for j in range(10):
        smjer = random.randint(1, 4)
        if smjer == 1:
            r1y += 1
        elif smjer == 2:
            r1y -= 1
        elif smjer == 3:
            r1x -= 1
        elif smjer == 4:
            r1x += 1
        if r1x == r2x and r1y == r2y:
            sudar = True
        smjer = random.randint(1, 4)
        if smjer == 1:
            r2y += 1
        elif smjer == 2:
            r2y -= 1
        elif smjer == 3:
            r2x -= 1
        elif smjer == 4:
            r2x += 1
        if r1x == r2x and r1y == r2y:
            sudar = True
    if sudar:
        broj_sudara += 1

print(broj_sudara / BROJ_SIM)
```

Zadatak 5.8.

```
import random

BROJ_SIMULACIJA = 10000
preskocio = 0

mapa = input()
minmax = [int(x) for x in input().split()]

z1 = mapa.index('!')
z2 = mapa.index('!', z1+1)
kraj = len(mapa)

for i in range(BROJ_SIMULACIJA):
    poz = mapa.index('*')
    while poz <= z1:
        poz += random.randint(minmax[0], minmax[1])
    if poz >= z2:
        preskocio += 1

print(preskocio/BROJ_SIMULACIJA)
```

Zadatak 5.9.

```
import random

def kraljica_napada(y, x):
        for i in range(8):
            if i != x and p[y][i] != '-':
                return True
            if i != y and p[i][x] != '-':
                return True
        # komentarisuci ovaj dio, ne provjeravaju se dijagonale
        # na taj nacin se provjerava kretanje topa
        # for i in range(1, 8):
        #     if x-i >= 0 and y-i >= 0 and p[y-i][x-i] != '-':
        #         return True
        #     if x+i < 8 and y-i >= 0 and p[y-i][x+i] != '-':
        #         return True
        #     if x-i >= 0 and y+i < 8 and p[y+i][x-i] != '-':
        #         return True
        #     if x+i < 8 and y+i < 8 and p[y+i][x+i] != '-':
        #         return True
        return False

BROJ_SIMULACIJA = 10000
broj_napadanja = 0
```

```
for i in range(BROJ_SIMULACIJA):
    p = []              # ploca
    for i in range(8):
        p.append(['-'] * 8)
        broj_kraljica = 0
    while broj_kraljica < 3:
        x = random.randrange(8)
        y = random.randrange(8)
        if p[y][x] == '-':
            p[y][x] = '*'
            broj_kraljica += 1

    napadaju_se = False
    for i in range(8):
        for j in range(8):
            if p[i][j] == '*' and kraljica_napada(i, j):
                napadaju_se = True
    if napadaju_se:
        broj_napadanja += 1

print(broj_napadanja/BROJ_SIMULACIJA)
```

Zadatak 5.10.

```
import random

BROJ_SIM = 10000

ulog = int(input())
opklada = int(input())
cilj = int(input())

broj_pobjeda = 0
for s in range(BROJ_SIM):
    novac = ulog
    while novac > 0 and novac < cilj:
        if random.randrange(0, 100) < 49:
            novac += opklada
        else:
            novac -= opklada
    if novac >= cilj:
        broj_pobjeda += 1

print(broj_pobjeda/BROJ_SIM)
```

Zadatak 5.11.

```
import random
```

```
 2
 3 BROJ_SIM = 10000000
 4 broj_ostrougli = 0
 5
 6 for i in range(0, BROJ_SIM):
 7     a = random.randint(1, 1000)
 8     b = random.randint(1, 1000)
 9     c = random.randint(1, 1000)
10
11     ostrougli = True
12     if(a + b <= c or a + c <= b or b + c <= a):
13         ostrougli = False
14     if(a**2 + b**2 <= c**2 or a**2 + c**2 <=
15        b**2 or b**2 + c**2 <= a**2):
16         ostrougli = False
17     if ostrougli:
18         broj_ostrougli += 1
19
20 print(100.0 * broj_ostrougli / BROJ_SIM)
```

Zadatak 5.12.

```
 1 import random
 2
 3 BROJ_SIMULACIJA = 100000
 4 broj_boja = 0
 5
 6 for i in range(BROJ_SIMULACIJA):
 7     crvena = False
 8     zelena = False
 9     plava = False
10
11     granica_crvene = 5
12     granica_zelene = 10
13
14     kuglica1 = random.randrange(15)
15     if kuglica1 < granica_crvene:
16         crvena = True
17         granica_crvene -= 1
18         granica_zelene -= 1
19     elif kuglica1 < granica_zelene:
20         zelena = True
21         granica_zelene -= 1
22     else:
23         plava = True
24
25     kuglica2 = random.randrange(14)
```

```
26      if kuglica2 < granica_crvene:
27          crvena = True
28          granica_crvene -= 1
29          granica_zelene -= 1
30      elif kuglica2 < granica_zelene:
31          zelena = True
32          granica_zelene -= 1
33      else:
34          plava = True
35
36      kuglica3 = random.randrange(13)
37      if kuglica3 < granica_crvene:
38          crvena = True
39          granica_crvene -= 1
40          granica_zelene -= 1
41      elif kuglica3 < granica_zelene:
42          zelena = True
43          granica_zelene -= 1
44      else:
45          plava = True
46
47      kuglica4 = random.randrange(12)
48      if kuglica4 < granica_crvene:
49          crvena = True
50          granica_crvene -= 1
51          granica_zelene -= 1
52      elif kuglica4 < granica_zelene:
53          zelena = True
54          granica_zelene -= 1
55      else:
56          plava = True
57
58      if plava == True:
59          broj_boja += 1
60      if zelena == True:
61          broj_boja += 1
62      if crvena == True:
63          broj_boja += 1
64
65  print(broj_boja/BROJ_SIMULACIJA)
```

Zadatak 5.13.

```
1  import random
2
3  n = int(input())
4
```

```
5  sekvenca = [3, 1, 4, 1]
6  random.seed(42)
7  BROJ_SIM = 10000
8
9  pobjede = 0
10
11 for i in range(BROJ_SIM):
12     lista = []
13     nalaze_se = True
14     for j in range(n):
15         lista.append(random.randint(1, 6))
16     for broj in sekvenca:
17         if broj in lista:
18             lista = lista[lista.index(broj)+1:]
19         else:
20             nalaze_se = False
21     if nalaze_se:
22         pobjede += 1
23
24 print(pobjede/BROJ_SIM)
```

15. Funkcije

Zadatak 6.1.

```
n = int(input())

while n > 9:
    result = 0
    while n > 0:
        result += n % 10
        n //= 10
    n = result

print(result)
```

Zadatak 6.2.

```
a = int(input())
b = int(input())

temp = b
desetine = 1
while temp != 0:
    desetine *= 10
    temp //= 10

seNalazi = False

while a != 0:
    temp = a % desetine
```

```
14      if temp == b:
15          seNalazi = True
16          break
17      a //= 10
18
19  if seNalazi:
20      print("Nalazi␣se")
21  else:
22      print("Ne␣nalazi␣se")
```

Zadatak 6.3.

```
1  n = int(input())
2
3  for i in range(0, n):
4      for j in range(0, n):
5          print((i + j) % n + 1, end='␣')
6      print()
```

Zadatak 6.4.

```
1  def spirala(n):
2    for i in range (1, n+1):
3      for j in range (1, n+1):
4          # okvir u kojem crtamo
5          N = min(i, j, (n - i + 1), (n - j + 1)) - 1
6
7
8          if N == i-1:
9              poz = j - N
10         elif N == n - j:
11             poz = (j - N) + (i - N) - 1
12         elif N == n - i:
13             poz = (2 * (n - 2 * N) - 1) + ((n - j + 1) - N) - 1
14         elif N == j - 1:
15             poz = (3 * (n - 2 * N) - 2) + ((n - i + 1) - N) - 1
16
17         print("%3s" % int((2 * n * N) + (2 * N * (n - 2 * N)) +
18               poz), end='␣')
19     print()
20
21 n=int(input())
22 spirala(n)
```

Zadatak 6.5.

```
1  def suma_na_dijagonalama(n):
```

```
2       if n == 1:
3         return 1
4       suma = 1
5
6       for i in range(2, int((n+1)/2)+1):
7           suma += (2 * i - 1) ** 2
8           suma += (2 * i - 1) ** 2 - (2 * i - 2)
9           suma += (2 * i - 1) ** 2 - 3 * (2 * i - 2)
10          suma += (2 * i - 1) ** 2 - 2 * (2 * i - 2)
11
12      return suma
13
14  n=int(input())
15  print(suma_na_dijagonalama(n))
```

16. Liste

Zadatak 7.1.

```
1  n = input()
2
3  lista = []
4  while n != '':
5      lista.append(int(n))
6      n = input()
7
8  obrnut = str(lista[-1])
9  obrnut = int(obrnut[1] + obrnut[0])
10 del lista[-1]
11
12 if obrnut in lista:
13     print('Da')
14 else:
15     print('Ne')
```

Zadatak 7.2.

```
1  rijeci = []
2
3  suma = 0
4  rijec = input()
5  while rijec != '':
6      rijeci.append(rijec)
7      suma += len(rijec)
8      rijec = input()
```

```
9
10 prosjek = suma / len(rijeci)
11
12 for r in rijeci:
13     if len(r) > prosjek:
14         print(r)
```

Zadatak 7.3.

```
1 m = int(input())
2 lista = []
3
4 for i in range(m):
5     lista.append(int(input()))
6
7 lista.sort()
8 print(sum(lista[m//2:]) -
9       sum(lista[:m//2]))
```

Zadatak 7.4.

```
1  gradovi_s = []
2  gradovi_b = []
3  gradovi_bs = []
4
5  grad = input()
6  while grad != '':
7      if grad[0].upper() == 'S':
8          gradovi_s.append(grad)
9          gradovi_bs.append(grad)
10     if grad[0].upper() == 'B':
11         gradovi_b.append(grad)
12         gradovi_bs.append(grad)
13     grad = input()
14
15 if len(gradovi_s) > len(gradovi_b):
16     gradovi = gradovi_s
17 elif len(gradovi_b) > len(gradovi_s):
18     gradovi = gradovi_b
19 else:
20     gradovi = gradovi_bs
21
22 for grad in gradovi:
23     print(grad, end='␣')
```

Zadatak 7.5.

```
lista = []

x = input()
while x != '':
    lista.append(int(x))
    x = input()

x = min(lista[:len(lista)//2])
y = max(lista[len(lista)//2:])

for k in range(x, y+1, 2):
    print(k, end="␣")
```

Zadatak 7.6.

```
lista = []
for i in range(0, 10):
    lista.append(int(input()))
lista.sort()
print(lista[-3])
```

Zadatak 7.7.

```
n = int(input())
lista = []
for i in range(0, n):
    lista.append(int(input()))
m = int(input())
lista.sort()
lista2 = lista[m:len(lista)-m]
for e in lista2:
    print(e, end="␣")
```

Zadatak 7.8.

```
l1 = []
l2 = []
l3 = []
for i in range(0, 10):
    n = int(input())
    if n < 0:
        l1.append(n)
    elif n == 0:
        l2.append(n)
    else:
        l3.append(n)
for e in l1:
```

```
13      print(e, end='␣')
14  for e in l2:
15      print(e, end='␣')
16  for e in l3:
17      print(e, end='␣')
```

Zadatak 7.9.

```
1   def rotiraj(a):
2       r = str(a)
3       return int(r[::-1])
4
5   lb = [] # lista brojeva
6   lr = [] # lista rotiranih
7
8   n = int(input())
9   while n != -1:
10      lb.append(n)
11      lr.append(rotiraj(n))
12      n = int(input())
13
14  print(max(lr) - min(lb))
```

Zadatak 7.10.

```
1   m = int(input())
2
3   lista = []
4   for _ in range(m):
5       lista.append(int(input()))
6
7   pivot = lista[m//2]
8   del lista[m//2]
9   manji = []
10  veci = []
11
12  for e in lista:
13      if e <= pivot:
14          manji.append(e)
15      else:
16          veci.append(e)
17
18  for e in manji:
19      print(e, end='␣')
20  print(pivot, end='␣')
21  for e in veci:
22      print(e, end='␣')
```

Zadatak 7.11.

```
m = int(input())

brojevi = []
for _ in range(m):
    brojevi.append(int(input()))

m = int(input())
parovi = []
while m != -1:
    p1 = m
    p2 = int(input())
    parovi.append([p1, p2])
    m = int(input())

for par in parovi:
    temp = brojevi[par[0]]
    brojevi[par[0]] = brojevi[par[1]]
    brojevi[par[1]] = temp

for e in brojevi:
    print(e, end=' ')
```

Zadatak 7.12.

```
m = int(input())
n = int(input())

harun = []
sead = []

sumaHarunovih = 0
sumaSeadovih = 0

for i in range(m):
    s = int(input())
    harun.append(s)
    sumaHarunovih += s

for i in range(n):
    s = int(input())
    sead.append(s)
    sumaSeadovih += s

brojKojiSeadTrebaIzbaciti = sumaSeadovih - sumaHarunovih

if brojKojiSeadTrebaIzbaciti in sead:
```

```
23      print('jeste')
24  else:
25      print('nije')
```

Zadatak 7.13.

```
 1  lista = []
 2
 3  x = input()
 4  while x != '':
 5      lista.append(int(x))
 6      x = input()
 7
 8  for i in range(len(lista)):
 9      for j in range(len(lista)):
10          if lista[i] < lista[j] and \
11                  lista[i]%2 == 0 and \
12                  lista[j]%2 == 0:
13              y = lista[i]
14              lista[i] = lista[j]
15              lista[j] = y
16
17  for i in lista:
18      print(i, end=' ')
```

Zadatak 7.14.

```
 1  max_raspon = 0
 2
 3  s = input()
 4  lista = s.split()
 5
 6  brojevi = [int(i) for i in lista]
 7  ogledalo = brojevi[::-1]
 8  skup = set(brojevi)
 9
10  for i in skup:
11      start = brojevi.index(i)
12      kraj = len(ogledalo) - ogledalo.index(i) - 1
13      raspon = kraj - start + 1
14      if raspon > max_raspon:
15          max_raspon = raspon
16
17  print(max_raspon)
```

Zadatak 7.15.

```
l = input()
lista = l.split()
lista = [int(x) for x in lista]

skupine = 0
uzastopni = False
for i in range(1, len(lista)):
    if lista[i-1] == lista[i] and not uzastopni:
        skupine += 1
        uzastopni = True
    if lista[i-1] != lista[i]:
        uzastopni = False

print(skupine)
```

Zadatak 7.16.

```
lista = input().split()
n = int(input())

int_lista = []
for i in lista:
    int_lista.append(int(i))

int_lista.sort(reverse=True)

print(int_lista[0] - int_lista[n-1])
```

Zadatak 7.17.

```
def postavi_cifre(n, c):
    for i in range(10):
        if str(i) in str(n):
            c[i] = True

n1 = int(input())

cifre = [False] * 10
postavi_cifre(n1, cifre)

while False in cifre:
    n2 = n1
    zbir = 0
    while n1 > 0:
        zbir += n1 % 10
        n1 //= 10
    n2 = int(str(n2) + str(zbir))
```

```
18      postavi_cifre(n2, cifre)
19      n1 = n2
20
21  print(n1)
```

Zadatak 7.18.

```
 1  l = input()
 2  l = l.split()
 3  for i in range(0, len(l)):
 4      l[i] = int(l[i])
 5  prvi = 0
 6  zadnji = 0
 7  for i in range(1, len(l)-1):
 8      if ((l[i-1] < l[i]) and (l[i+1] < l[i])):
 9          if prvi == 0:
10              prvi = i
11          zadnji = i
12  if(prvi == 0):
13      print(0)
14  else:
15      print(zadnji - prvi + 1)
```

Zadatak 7.19.

```
 1  n = int(input())
 2  lista = list()
 3  for i in range(0, n):
 4      lista.append(int(input()))
 5
 6  lista2 = list(lista)
 7  lista2.sort()
 8  lista.remove(lista2[n//2])
 9
10  for element in lista:
11      print(element, end='␣')
```

Zadatak 7.20.

```
1  n = int(input())
2  lista = [0]*n
3  for i in range(0, n*n):
4      m = int(input())
5      lista[i%n] += m
6  for i in range(0, n):
7      print(lista[i])
```

Zadatak 7.21.

```
n = int(input())
matrica = []
for i in range(0, n):
    red = []
    for j in range(0, n):
        red.append(int(input()))
    matrica.append(red)
suma1 = 0
suma2 = 0
for i in range(0, n):
    suma1 += matrica[i][i]
    suma2 += matrica[i][n - 1 - i]
print(suma1 * suma2)
```

Zadatak 7.22.

```
lista = input()
lista = lista.split()
lista = [int(x) for x in lista]
n = len(lista)

brojVecih = 0

for i in range(1,n):
    if lista[i] > lista[i-1]:
        brojVecih += 1

print(brojVecih)

brojVecihOdSume = 0
zbirPozitivnih = 0
if lista[-1] > 0:
    zbirPozitivnih = lista[-1]

for i in range(n-2,-1,-1):
    if lista[i] > zbirPozitivnih:
        brojVecihOdSume += 1
    if lista[i] > 0:
        zbirPozitivnih += lista[i]

print(brojVecihOdSume)
```

Zadatak 7.23.

```
def fib_lista():
    lista = []
```

```
    lista.append(1)
    lista.append(2)
    for i in range(2,50):
        lista.append(lista[i - 1] + lista[i - 2])
    return lista

fl = fib_lista()
tekst = input()
brojevi = input().split()
brojevi = [int(i) for i in brojevi]

broj_znakova = fl.index(max(brojevi)) + 1

rjesenje = ['␣'] * broj_znakova
i = 0
for znak in tekst:
    if znak.isalpha():
        rjesenje[fl.index(brojevi[i])] = znak
        i += 1

print(''.join(rjesenje).upper())
```

Zadatak 7.24.

```
brojevi = input().split()
brojevi = [int(i) for i in brojevi]

d = len(brojevi)
najvece_preklapanje = 0

for i in range (d):
    for j in range(d-1, -1, -1):
        pocetak = i
        kraj = j

        preklapanje = 0

        while (pocetak < d and kraj >= 0 and
               brojevi[pocetak] == brojevi[kraj]):
            pocetak += 1
            kraj -= 1
            preklapanje += 1

        najvece_preklapanje = max(najvece_preklapanje,
                                  preklapanje)

print(najvece_preklapanje)
```

Zadatak 7.25.

```
fajl = open('test01.in', 'r')
drzave = dict()
for red in fajl:
    podaci = red.strip().split(',␣')
    if podaci[1] not in drzave or \
            drzave[podaci[1]][1] < int(podaci[2]):
        drzave[podaci[1]] = [podaci[0], int(podaci[2])]
gradovi = []
for kljuc, lista in drzave.items():
    gradovi.append(lista[0])
gradovi.sort()
for grad in gradovi:
    print(grad)
```

Zadatak 7.26.

```
n = int(input())
matrica = []
for i in range(0, n):
    red = []
    for j in range(0, n):
        red.append(int(input()))
    matrica.append(red)
for i in range(0, n):
    for j in range(0, n):
        for k in range(0, n-1):
            if matrica[k][i] > matrica[k+1][i]:
                temp = matrica[k][i]
                matrica[k][i] = matrica[k+1][i]
                matrica[k+1][i] = temp

for i in range(0, n):
    for j in range(0, n):
        print(matrica[i][j], end='␣')
    print()
```

Zadatak 7.27.

```
matrica = []
for i in range(0, 3):
    red = input()
    matrica.append(red.split())

jeIspravna = True
for i in range(1, 10):
    ponavljanja = sum(x.count(str(i)) for x in matrica)
```

```
9       if ponavljanja != 1:
10          jeIspravna = False
11          break
12
13  for i in range(0, 3):
14      print('␣'.join(map(str, matrica[i])))
15  print('ispravan' if jeIspravna else 'neispravan')
```

Zadatak 7.28.

```
1   brojevi = input()
2   brojevi = brojevi.split()
3   brojevi = [int(x) for x in brojevi]
4
5   najvecaSuma = 0
6
7   for i in range(len(brojevi)):
8       suma = 0
9       for j in range(i,len(brojevi)):
10          suma += brojevi[j]
11          if najvecaSuma < suma:
12              najvecaSuma = suma
13
14  print(najvecaSuma)
```

17. Stringovi

Zadatak 8.1.

```
1  tekst = input()
2
3  rijeci = tekst.lower().split()
4
5  ciste_rijeci = []
6  max = 0
7  for r in rijeci:
8      rijec = r.strip('.,!?-␣')
9      ciste_rijeci.append(rijec)
10     if len(rijec) > max:
11         max = len(rijec)
12
13 for r in ciste_rijeci:
14     if len(r) == max:
15         print(r)
```

Zadatak 8.2.

```
1  s = 'aeiou'
2  b = [0] * 5
3
4  tekst = input()
5  tekst = tekst.lower()
6  for slovo in tekst:
7      if slovo == 'a':
8          b[0] += 1
```

```
    elif slovo == 'e':
        b[1] += 1
    elif slovo == 'i':
        b[2] += 1
    elif slovo == 'o':
        b[3] += 1
    elif slovo == 'u':
        b[4] += 1

index = b.index(max(b))
print(s[index])
```

Zadatak 8.3.

```
n = int(input())
duzina = 0

for i in range(n):
    s = input()
    if duzina < len(s):
        print(s)
    duzina += len(s)
```

Zadatak 8.4.

```
r = input().lower()

while r != '':
    if r[0] == r[-1] and r[0] not in r[1:-1]:
        print(r)
    r = input().lower()
```

Zadatak 8.5.

```
n = input()

while n != '':
    if int(n[len(n)//2]) > 5:
        print(n, end='␣')
    n = input()
```

Zadatak 8.6.

```
s = input()

zbir = 0

```

```
while len(s) == 1:
    if s.isnumeric():
        i = int(s)
        if i % 2 == 0:
            zbir += i
    s = input()

print(zbir)
```

Zadatak 8.7.

```
tekst = input()

while tekst != '':
    d = len(tekst)
    if tekst[d - 1] == '.' \
            and d < 20:
        print(tekst)
    tekst = input()
```

Zadatak 8.8.

```
n = input()
m = int(input())

poz_tacke = len(n) - m
print(n[:poz_tacke] + '.' + n[poz_tacke:])
```

Zadatak 8.9.

```
samoglasnici = 'aeiouAEIOU'
rijec = input()
while rijec != '':
    for samoglasnik in samoglasnici:
        rijec = rijec.replace(samoglasnik, '')
    print(rijec)
    rijec = input()
```

Zadatak 8.10.

```
recenica = input()
recenica = recenica.upper()
if recenica.count('A') > recenica.count('E'):
    print('A')
elif recenica.count('A') < recenica.count('E'):
    print('E')
else:
    print('AE')
```

Zadatak 8.11.

```
n = input()
while n != '':
    if n[0] != 'b':
        print(n, end='␣')
    n = input()
```

Zadatak 8.12.

```
n = int(input())

najduzi = 0

najkraci = 101

for i in range(n):
    s = input()

    if len(s) % 2 == 1 and len(s) > najduzi:
        print(s)
    elif len(s) % 2 == 0 and len(s) < najkraci:
        print(s)

    if len(s) > najduzi:
        najduzi = len(s)

    if len(s) < najkraci:
        najkraci = len(s)
```

Zadatak 8.13.

```
word = input().lower()

while word.count(word[0]) != len(word):
    word = word.replace(word[0], '')

print(word)
```

Zadatak 8.14.

```
x = input().lower()
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

brojac = 0
lista = x.split()
for element in lista:
    e = element.strip(punctuations)
```

```
    if e[0] == e[-1]:
        brojac += 1

print(brojac)
```

Zadatak 8.15.

```
s1 = input().lower()
s2 = input().lower()

nalazi_se = True

i = 0
for s in s2:
    nasao = False
    while i < len(s1) and not nasao:
        if s1[i] == s:
            nasao = True
        i += 1
    if not nasao:
        nalazi_se = False

if nalazi_se:
    print('Nalazi se!')
else:
    print('Ne nalazi se!')
```

Zadatak 8.16.

```
tekst1 = input()
tekst2 = tekst1.replace(' i ', ' I ')
print(tekst2)
for i in range(0, len(tekst2)):
    if tekst1[i] != tekst2[i]:
        print('-', end='')
    else:
        print(' ', end='')
```

Zadatak 8.17.

```
r1 = input().lower()
r2 = input().lower()
s1 = set()
s2 = set()
for slovo in r1:
    s1.add(slovo)
for slovo in r2:
```

```
    s2.add(slovo)
razlika = s1-s2
print(len(razlika))
```

Zadatak 8.18.

```
for i in range(11, 100):
    stri = str(i)
    if stri[0] == stri[1]:
        for j in range(101, 1000):
            strj = str(j)
            if strj[0] == strj[2]:
                r = int(stri) + int(strj)
                strr = str(r)
                if len(strr) == 4 and strr[0] == strr[3] \
                        and strr[1] == strr[2]:
                    print(i, j, r)
```

Zadatak 8.19.

```
tekst = input()

najduzi = 0
i = 0
while i < len(tekst):
    duzina = 0
    j = i
    while(j < len(tekst) and tekst[j] == tekst[i]):
        duzina += 1
        j += 1
    if duzina > najduzi:
        najduzi = duzina
    i += duzina

print(najduzi)
```

Zadatak 8.20.

```
rijec = input().lower()
samoglasnici = "aeiou"
pozicija = 0
for i in range(1, len(rijec)):
    if rijec[i] in samoglasnici:
        pozicija = i
        break
print(rijec[pozicija+1:] + rijec[:pozicija+1])
```

Zadatak 8.21.

```
broj = input()
grupe = len(broj) // 4
for i in range(0, grupe - 1):
    print('****', end='␣')
print(broj[-4:])
```

Zadatak 8.22.

```
red = input()
brojac = 0
rijec = ''
while red != '':
    rijec = rijec + red[brojac]
    brojac += 1
    red = input()

print(rijec.lower())
```

Zadatak 8.23.

```
rijeci = input().split(',␣')
tekst = input()
for rijec in rijeci:
    tekst = tekst.replace(rijec, '*' * len(rijec))
print(tekst)
```

Zadatak 8.24.

```
tekst = input()

podvlacenje = ''
trenutno = ''

for slovo in tekst:
    if slovo.isalpha():
        trenutno += slovo
    else:
        if len(trenutno) > 5:
            podvlacenje += '-' * len(trenutno)
        else:
            podvlacenje += '␣' * len(trenutno)
        podvlacenje += '␣'
        trenutno = ''

if len(trenutno) > 5:
        podvlacenje += '-' * len(trenutno)
```

```
else:
    podvlacenje += '␣' * len(trenutno)
trenutno = ''

print(tekst)
print(podvlacenje)
```

Zadatak 8.25.

```
rijeci = input().lower().split()

for i in range(len(rijeci)):
    rijeci[i] = rijeci[i].strip('␣.,!?:;')

provjerene = set()
ponavljanja = []
for rijec in rijeci:
    broj_ponavljanja = 0
    if rijec not in provjerene :
        broj_ponavljanja = rijeci.count(rijec)
        provjerene.add(rijec)
    if broj_ponavljanja > 1:
        ponavljanja.append([rijec,
                            broj_ponavljanja])

for element in ponavljanja:
    print(element[0], ':␣', element[1], sep='')
```

Zadatak 8.26.

```
s1 = input().lower()
s2 = input().lower()

su_anagrami = True
for ch in s1:
    if ch.isalpha() and s1.count(ch) != s2.count(ch):
        su_anagrami = False

if su_anagrami:
    print('Anagrami')
else:
    print('Nisu␣anagrami')
```

Zadatak 8.27.

```
tekst = input().lower()
lista = list(tekst[1:-1])
```

```
lista.sort()
sortirani = ''
for slovo in lista:
    sortirani += slovo
novi = tekst[0] + sortirani + tekst[-1]
print(novi)
```

Zadatak 8.28.

```
rijeci = [input().upper()]
red = len(rijeci[0])
for i in range (1, red):
    rijeci.append(input().upper())

jeIspravan = True

for i in range(0, red):
    for j in range(0, red):
        print(rijeci[i][j], end=' ')
        if(rijeci[i][j] != rijeci[j][i]):
            jeIspravan = False
    print()

print('ispravan' if jeIspravan else 'neispravan')
```

18. Fajlovi

Zadatak 9.1.

```
f = open('test01.in', 'r')
matrica = []
while True:
    s = f.readline()
    if s == '':
        break
    matrica.append(s.split())

brojKoraka = 0

x = 0
y = 0
for i in range(len(matrica)):
    for j in range(len(matrica[i])):
        if matrica[i][j] == 'R':
            x = i
            y = j

while y < len(matrica[x]) - 1 and matrica[x][y + 1] == '-':
    y += 1
    brojKoraka += 1

if y == len(matrica[x])-1:
    brojKoraka += 1
    print(brojKoraka)
    print('da')
```

```
else:
    while x < len(matrica)-1 and matrica[x + 1][y] == '-':
        x += 1
        brojKoraka += 1

    if x == len(matrica)-1:
        brojKoraka += 1
        print(brojKoraka)
        print('da')
    else:
        print(brojKoraka)
        print('ne')
```

Zadatak 9.2.

```
matrica = []
suma = 0

with open('test01.in', 'r') as f:
    for red in f:
        red = red.strip().split()
        matrica.append(red)

m1 = matrica[:(len(matrica)//2)]
m2 = matrica[len(matrica)//2:]

for i in range(len(m1)):
    for j in range(len(m1[0])):
        broj = int(m1[i][j])
        if m2[i][j] == '*' and broj%2 == 0:
            suma += broj

print(suma)
```

Zadatak 9.3.

```
lista1 = []
lista2 = []
lista3 = []

with open('test01.in', 'r') as f:
    for red in f:
        red = red.strip()
        if red[-1] == '1':
            lista1.append(red[:-2])
        elif red[-1] == '2':
            lista2.append(red[:-2])
```

```
12          elif red[-1] == '3':
13              lista3.append(red[:-2])
14
15  for red in lista1:
16      print(red)
17  print()
18
19  for red in lista2:
20      print(red)
21  print()
22
23  for red in lista3:
24      print(red)
25  print()
```

Zadatak 9.4.

```
1   fajl = open('test01.in', 'r')
2
3   korisnici = []
4   shifre = []
5
6   for red in fajl:
7       k, s = red.strip().split()
8       if k == s or '1234' in s or len(s) < 5:
9           korisnici.append(k)
10          shifre.append(s)
11
12  for i in range(0, len(korisnici)):
13      print(korisnici[i], shifre[i])
```

Zadatak 9.5.

```
1   suma = 0
2   with open('test01.in', 'r') as f:
3       for red in f:
4           suma += sum([float(x)
5               for x in red.strip().split()])
6
7   print(suma)
```

Zadatak 9.6.

```
1   matrica = []
2
3   with open('test01.in', 'r') as f:
4       for red in f:
```

```
        red = red.strip().split()
        matrica.append(red)

sirina = len(matrica[0])
visina = len(matrica)
nova = []
for i in range(visina):
    red = ['-'] * sirina
    nova.append(red)

for j in range(sirina):
    kolona = []
    for i in range(visina):
        kolona.append(matrica[i][j])
    for i in range(visina):
        if matrica[i][j] == '#':
            nova[i][j] = '#'
        elif '#' in matrica[i][:j] and '#' in matrica[i][j:]
             and '#' in kolona[:i] and '#' in kolona[i:]:
            nova[i][j] = '#'

brojac = 0
for red in nova:
    for kolona in red:
        if kolona == '#':
            brojac += 1
print(brojac)
```

Zadatak 9.7.

```
file = open('test01.in')

matrix = []
for row in file:
    matrix.append(row.rstrip().split())

m = len(matrix)

oldMatrix = [row[:] for row in matrix]
for i in range(m):
    for j in range(m):
        if oldMatrix[i][j] == '0':
            matrix[i][j] = "-"
            if matrix[i-1][j] == '8':
                matrix[i-1][j] = '0'
            elif matrix[i][j+1] == '8':
                matrix[i][j+1] = '0'
```

```
            elif matrix[i+1][j] == '8':
                matrix[i+1][j] = '0'
            elif matrix[i][j-1] == '8':
                matrix[i][j-1] = '0'

for l in matrix:
    for e in l:
        print(e, end= "␣")
    print()
```

Zadatak 9.8.

```
import random

fajl = open('test01.in', 'r')

broj = int(fajl.readline().strip())

rijeci = []
rijec = fajl.readline().strip().lower()
while rijec != '':
    rijeci.append(rijec)
    rijec = fajl.readline().strip().lower()

lozinka = ''

for i in range(broj):
    lozinka += rijeci[random.randrange(
                        len(rijeci))]

print(lozinka)
```

Zadatak 9.9.

```
fajl = open('test01.in', 'r')
sekvenca = fajl.read()
najduza = ""
duzina = 0
indeks1 = sekvenca.find('AGGT')
while indeks1 != -1:
    sekvenca = sekvenca[indeks1+4:]
    indeks2 = sekvenca.find('AGGT')
    if indeks2 != -1:
        podsekvenca = sekvenca[:indeks2]
        if len(podsekvenca) > duzina:
            duzina = len(podsekvenca)
            najduza = podsekvenca
```

```
14      indeks1 = indeks2
15  print(najduza)
```

Zadatak 9.10.

```
 1  fajl = open('test01.in', 'r')
 2  linija = fajl.readline()
 3  lista = linija.split()
 4  r = int(lista[0])
 5  c = int(lista[1])
 6  matrica = []
 7  for i in range(0, r):
 8      red = []
 9      for j in range(0, c):
10          red.append('␣')
11      matrica.append(red)
12
13  linija = fajl.readline()
14  while linija != '':
15      (red, kolona, znak) = linija.split()
16      red = int(red.strip('('))
17      kolona = int(kolona.strip(')'))
18      matrica[red][kolona] = znak
19      linija = fajl.readline()
20
21  for i in range(0, r):
22      for j in range(0, c):
23          print(matrica[i][j], end='')
24      print()
```

Zadatak 9.11.

```
 1  fajl = open('test01.in', 'r')
 2  mr = {0:'Januar', 1:'Februar', 2:'Mart', 3:'April',
 3        4:'Maj', 5:'Juni', 6:'Juli', 7:'August', 8:'Septembar',
 4        9:'Oktobar', 10:'Novembar', 11:'Decembar'}
 5
 6  godine = []
 7  for line in fajl:
 8      lista = line.split()
 9      mjeseci = []
10      for mjesec in lista:
11          mjeseci.append(float(mjesec))
12      godine.append(mjeseci)
13
14  prosjecne = []
15  for i in range(0, 12):
```

```
    suma = 0
    for j in range(0, 5):
        suma += godine[j][i]
    prosjecne.append(suma/5)
    print(suma/5, end=' ')

print()
print(mr[prosjecne.index(max(prosjecne))])
print(mr[prosjecne.index(min(prosjecne))])
```

Zadatak 9.12.

```
fajl = open('test01.in', 'r')
dan = -1
najveca_temp = -1000
najveci_dan = ''
najveci_grad = ''
najmanja_temp = 1000
najmanji_dan = ''
najmanji_grad =''

dani = ['ponedjeljak', 'utorak', 'srijeda',
        'cetvrtak', 'petak', 'subota', 'nedjelja']
red = fajl.readline()
while red != '':
    if len(red) == 2:
        dan += 1
        red = fajl.readline()
    else:
        red_lista = red.split()
        temp = float(red_lista[-1])
        grad = red_lista[0]
        for i in range(1, len(red_lista) - 1):
            grad += ' ' + red_lista[i]
        if temp > najveca_temp:
            najveca_temp = temp
            najveci_grad = grad
            najveci_dan = dani[dan]
        elif temp < najmanja_temp:
            najmanja_temp = temp
            najmanji_grad = grad
            najmanji_dan = dani[dan]
        red = fajl.readline()

print(najmanji_grad, najmanja_temp, najmanji_dan)
print(najveci_grad, najveca_temp, najveci_dan)
```

Zadatak 9.13.

```
fajl = open('test01.in', 'r')
zaglavlje = fajl.readline()
n, m = zaglavlje.split()

counts = dict()
for red in fajl:
    red = red.split()
    for element in red:
        counts[element] = counts.get(element, 0) + 1

s = [(k, counts[k]) for k in
     sorted(counts, key=counts.get, reverse=True)]
for i in range(0, 3):
    print(s[i][0], s[i][1])
```

Zadatak 9.14.

```
f = open('test01.in', 'r')
matrica = []
while True:
    s = f.readline()
    if s == "":
        break
    matrica.append(s.split())

redoviPalindromi = True

for i in range(len(matrica)):
    for j in range(len(matrica[i]) // 2):
        if matrica[i][j] != matrica[i][len(matrica[i]) - 1 - j]:
            redoviPalindromi = False

kolonePalindromi = True

for i in range(len(matrica[0])):
    for j in range(len(matrica) // 2):
        if matrica[j][i] != matrica[len(matrica) - 1 - j][i]:
            kolonePalindromi = False

if redoviPalindromi and kolonePalindromi:
    print("savrsena")
elif redoviPalindromi or kolonePalindromi:
    print("polusavrsena")
else:
    print("nesavrsena")
```

Zadatak 9.15.

```
matrica = []

smjer = ''

with open('test01.in', 'r') as f:
    for red in f:
        red = red.strip()
        matrica.append(list(red))

for i in range(len(matrica)):
    for j in range(len(matrica[i])):
        if matrica[i][j] == 'S':
            y = i
            x = j

while matrica[y][x] != 'X':
    brojac = 0
    while smjer != 'U' and \
            (matrica[y+1][x] == '|' or \
             matrica[y+1][x] == 'X'):
        brojac += 1
        y += 1
        smjer = 'D'
    if brojac > 0:
        print('D', brojac)

    brojac = 0
    while smjer != 'D' and \
            (matrica[y-1][x] == '|' or \
             matrica[y-1][x] == 'X'):
        brojac += 1
        y -= 1
        smjer = 'U'
    if brojac > 0:
        print('U', brojac)

    brojac = 0
    while smjer != 'L' and \
            (matrica[y][x+1] == '-' or \
             matrica[y][x+1] == 'X'):
        brojac += 1
        x += 1
        smjer = 'R'
    if brojac > 0:
        print('R', brojac)
```

```
46
47      brojac = 0
48      while smjer != 'R' and \
49              (matrica[y][x-1] == '-' or \
50               matrica[y][x-1] == 'X'):
51          brojac += 1
52          x -= 1
53          smjer = 'L'
54      if brojac > 0:
55          print('L', brojac)
```

Zadatak 9.16.

```
1  matrica = []
2
3  with open('test01.in', 'r') as f:
4      for red in f:
5          red = red.strip().split()
6          matrica.append(red)
7
8  index = matrica[-1].index('-')
9
10 if index != 6 and matrica[-1][index+1] == '-':
11     print(0, index-2)
12 elif index != 6 and matrica[-2][index+1] == '-':
13     print(1, index-3)
14 # naredni slucaj u ovakvoj postavci zadatka se ne moze desiti
15 # elif index != 6 and index != 0 and matrica[-2][index+1] == '-'
16 # and matrica[-2][index-1] == '-':
17 #   print(2, index-2)
18 else:
19     print(3, index-3)
```

Zadatak 9.17.

```
1  matrica = []
2  with open('test01.in', 'r') as f:
3      for red in f:
4          red = red.rstrip('\n')
5          r = []
6          for i in range(0, len(red), 2):
7              r.append(red[i])
8          matrica.append(r)
9
10 visina = len(matrica)
11 sirina = len(matrica[0])
12
```

```
for j in range(sirina):
    pahuljice = 0
    red = 0
    while red < visina and matrica[red][j] != '#':
        if matrica[red][j] == '*':
            pahuljice += 1
            matrica[red][j] = '␣'
        red += 1
    for i in range(pahuljice):
        matrica[red - 1 - i][j] = '*'

for red in matrica:
    for znak in red:
        print(znak, end='␣')
    print()
```

Zadatak 9.18.

```
file = open('test01.in', 'r')

labirint = []
for red in file:
    labirint.append(red.rstrip())

koraci = 0
red = 0
kol = labirint[0].index('-')

while red + 1 < len(labirint):
    red += 1
    ld = 1      # lijevo - desno
    while labirint[red][kol] != '-':
        if kol - ld > 0 and \
                labirint[red][kol - ld] == '-':
            kol -= ld
        elif kol + ld < len(labirint[red]) and \
                labirint[red][kol + ld] == '-':
            kol += ld
        ld += 1
        koraci += 1
    koraci += 1

print(koraci)
```

Zadatak 9.19.

```
import math
```

```
2
3  def distance (a, b, c, d):
4      return math.sqrt((a-c)**2 + (b-d)**2)
5
6  file = open('test01.in')
7
8  matrix = []
9  for row in file:
10     matrix.append(row.rstrip().upper().split())
11
12 m = len(matrix)
13
14 new_matrix = []
15 for i in range(m):
16     new_matrix.append([])
17     for j in range(m):
18         if matrix[i][j] != '-':
19             new_matrix[i].append(matrix[i][j])
20         else:
21             min_distance = distance(0, 0, len(matrix),
22                                     len(matrix[i]))
23             char = 'Z'
24             for k in range(m):
25                 for l in range(m):
26                     if matrix[k][l] != '-':
27                         curr_distance = distance(i, j, k, l)
28                         if (curr_distance < min_distance) or \
29                                 (curr_distance == min_distance and
30                                  matrix[k][l] < char):
31                             char = matrix[k][l]
32                             min_distance = curr_distance
33
34             new_matrix[i].append(char)
35
36 for line in new_matrix:
37     for element in line:
38         print(element, end='␣')
39     print()
```

Zadatak 9.20.

```
1  import math
2
3  def provjeri_kvadrat(broj):
4      if broj < 1:
5          return -1
6      int_korjen = int(math.sqrt(broj) + 0.5)
```

```
    if int_korjen ** 2 == broj:
        return int_korjen
    else:
        return -1

def desifruj(rijec):
    n = provjeri_kvadrat(len(rijec))
    if n == -1:
        return 'GRESKA'

    brojac = 0
    matrica = []
    for i in range(n):
        red = []
        for j in range(n):
            red.append(rijec[brojac])
            brojac += 1
        matrica.append(red)

    rjesenje = ''
    for i in range(n):
        for j in range(n):
            rjesenje += matrica[j][i]

    return rjesenje

fajl = open('test01.in', 'r')
for red in fajl:
    red = red.strip()
    print(desifruj(red))
```

Zadatak 9.21.

```
mapa = {'A' : 0, 'B' : 1, 'C' : 2, 'D' : 3, 'E' : 4}
svi_odgovori = []

fajl = open('test01.in', 'r')
sb, pb = fajl.readline().split()
sb = int(sb)    # broj studenata
pb = int(pb)    # broj pitanja

for i in range(sb * pb):
    odgovori_pitanje = fajl.readline().split()
    odgovori_pitanje = \
        [int(x) <= 127 for x in odgovori_pitanje]
    svi_odgovori.append(odgovori_pitanje)

```

```
15 tacni = fajl.readline().split()
16
17 fajl.close()
18
19
20 rezultati = []
21 for i in range(sb):
22     broj_tacnih = 0
23     for j in range(pb):
24         pitanje = svi_odgovori[i * pb + j]
25         if pitanje.count(True) == 1 and \
26                 pitanje[mapa[tacni[j]]] == True:
27             broj_tacnih += 1
28     rezultati.append(broj_tacnih)
29
30 for element in rezultati:
31     print(element)
```

Zadatak 9.22.

```
1  import collections
2
3  with open('test01.in') as f:
4      redovi = list(f)
5
6  d = collections.OrderedDict()
7  for red in redovi:
8      elementi = red.strip().split(maxsplit=2)
9      kljuc = elementi[0] + '␣' + elementi[2]
10     vrijednost = elementi[1]
11     adrese = d.get(kljuc, set())
12     adrese.add(vrijednost)
13     d[kljuc] = adrese
14
15 ispisane = set()
16 for k, v in d.items():
17     if len(v) > 4:
18         adresa = k.split()[0]
19         if adresa not in ispisane:
20             print(adresa)
21             ispisane.add(adresa)
```

Zadatak 9.23.

```
1 fajl = open('test01.in', 'r')
2 red = fajl.readline()
3 while red != '':
```

```
4       ucenik = red.strip()
5       uspjeh = 0
6       predmeti = []
7       for i in range(0, 3):
8           p, o = fajl.readline().strip().split()
9           o = int(o)
10          if o == 1:
11              predmeti.append(p)
12          uspjeh += o
13      print(ucenik, end='␣')
14      if len(predmeti) == 0:
15          print(uspjeh/3)
16      else:
17          for predmet in predmeti:
18              print(predmet, end='␣')
19          print()
20      red = fajl.readline()
```

Zadatak 9.24.

```
1   import math
2
3   def pozicija(p, s, k):
4       if s == 'G': p[1] += k
5       elif s == 'DO': p[1] -= k
6       elif s == 'DE': p[0] += k
7       elif s == 'L': p[0] -= k
8       return p
9
10  fajl = open('test01.in', 'r')
11  pozicije = dict()
12  for red in fajl:
13      r, s, k = red.strip().split()
14      k = int(k)
15      if r not in pozicije:
16          pozicije[r] = pozicija([0, 0], s, k)
17      else:
18          pozicije[r] = pozicija(pozicije[r], s, k)
19
20  for k in sorted(pozicije.keys()):
21      print(k, end ='␣')
22      print(math.sqrt(pow(pozicije[k][0], 2) +
23            pow(pozicije[k][1], 2)))
```

Zadatak 9.25.

```
1   fajl = open('test01.in', 'r')
```

```
prodavaci = dict()
auta = dict()
for red in fajl:
    red = red.strip().split()
    if red[0] not in prodavaci:
        prodavaci[red[0]] = int(red[2])
    else:
        prodavaci[red[0]] += int(red[2])
    if red[1] not in auta:
        auta[red[1]] = 1
    else:
        auta[red[1]] += 1

v=list(prodavaci.values())
k=list(prodavaci.keys())
for i in range(0, 3):
     maxv = max(v)
     max_index = v.index(maxv)
     maxk = k[max_index]
     print(maxk, maxv)
     del v[max_index]
     del k[max_index]

v=list(auta.values())
k=list(auta.keys())
for i in range(0, 3):
     maxv = max(v)
     max_index = v.index(maxv)
     maxk = k[max_index]
     print(maxk, maxv)
     del v[max_index]
     del k[max_index]
```

Zadatak 9.26.

```
fajl = open('test01.in', 'r')
tabela = dict()
for red in fajl:
    red = red.strip().split()
    if red[0] not in tabela:
        tabela[red[0]] = 0
    if red[4] not in tabela:
        tabela[red[4]] = 0
    if int(red[1]) > int(red[3]):
        tabela[red[0]] += 3
    elif int(red[1]) < int(red[3]):
        tabela[red[4]] += 3
```

```
    else:
        tabela[red[0]] += 1
        tabela[red[4]] += 1

v = list(tabela.values())
k = list(tabela.keys())
for i in range(0, len(tabela)):
    for j in range(0, len(tabela)-1):
        if(v[j] < v[j+1]):
            temp = v[j]
            v[j] = v[j+1]
            v[j+1] = temp
            temp = k[j]
            k[j] = k[j+1]
            k[j+1] = temp

for i in range(0, len(tabela)):
    print(k[i], v[i])
```

Zadatak 9.27.

```
def u_minute(vin):
    vt = list(vin)
    mins_lista = []
    for v in vt:
        if v[0] == '12':
            v[0] = '0'
        mins = int(v[0]) * 60 + int(v[1]) + 60 * 12 * \
               (1 if v[2].lower() == 'pm' else 0)
        mins_lista.append(mins)
    return mins_lista

def u_sate(mins_lista_in):
    mli = list(mins_lista_in)
    vremena = []
    for mins in mli:
        oznaka = 'AM'
        sati = mins // 60
        if sati > 11:
            sati = sati - 12
            oznaka = 'PM'
        if sati == 0:
            sati = 12
        minute = mins % 60
        sati = str(sati)
        minute = str(minute)
        if len(sati) < 2:
```

```
            sati = '0' + sati
        if len(minute) < 2:
            minute = '0' + minute
        vremena.append(sati + ':' + minute + ' ' + oznaka)
    return vremena

vremena_in = []
with open('test01.in', 'r') as f:
    vremena_in = list(f)

vremena_in = [str.strip(x).split(':')
              for x in vremena_in]
vremena_minute = u_minute(vremena_in)
vremena_minute.sort()
vremena_out = u_sate(vremena_minute)

for v in vremena_out:
    print(v)
```

Zadatak 9.28.

```
matrica = []
smjer = ''
l_red = 0
l_kol = 0
redova = 0
kolona = 0

with open('test01.in', 'r') as f:
    br_red = 0
    for red in f:
        red = red.strip().split()
        if red[0] == '*' or red[0] == '-':
            matrica.append(red)
        else:
            smjer = red[0]
        if '*' in red:
            l_red = br_red
            l_kol = red.index('*')
        br_red += 1
    redova = len(matrica)
    kolona = len(matrica[0])

for i in range(100):
    if smjer == 'SI':
        if l_red-1 >= 0 and l_kol+1 < kolona:
            l_red = l_red-1
```

```
27              l_kol = l_kol+1
28          elif l_red-1 < 0 and l_kol+1 >= kolona:
29              l_red = l_red+1
30              l_kol = l_kol-1
31              smjer = 'JZ'
32          elif l_red-1 < 0 and l_kol+1 < kolona:
33              l_red = l_red+1
34              l_kol = l_kol+1
35              smjer = 'JI'
36          elif l_red-1 >= 0 and l_kol+1 >= kolona:
37              l_red = l_red-1
38              l_kol = l_kol-1
39              smjer = 'SZ'
40      elif smjer == 'SZ':
41          if l_red-1 >= 0 and l_kol-1 >= 0:
42              l_red = l_red-1
43              l_kol = l_kol-1
44          elif l_red-1 < 0 and l_kol-1 < 0:
45              l_red = l_red+1
46              l_kol = l_kol+1
47              smjer = 'JI'
48          elif l_red-1 < 0 and l_kol-1 >= 0:
49              l_red = l_red+1
50              l_kol = l_kol-1
51              smjer = 'JZ'
52          elif l_red-1 >= 0 and l_kol-1 < 0:
53              l_red = l_red-1
54              l_kol = l_kol+1
55              smjer = 'SI'
56      elif smjer == 'JI':
57          if l_red+1 < redova and l_kol+1 < kolona:
58              l_red = l_red+1
59              l_kol = l_kol+1
60          elif l_red+1 >= redova and l_kol+1 >= kolona:
61              l_red = l_red-1
62              l_kol = l_kol-1
63              smjer = 'SZ'
64          elif l_red+1 >= redova and l_kol+1 < kolona:
65              l_red = l_red-1
66              l_kol = l_kol+1
67              smjer = 'SI'
68          elif l_red+1 < redova and l_kol+1 >= kolona:
69              l_red = l_red+1
70              l_kol = l_kol-1
71              smjer = 'JZ'
72      elif smjer == 'JZ':
73          if l_red+1 < redova and l_kol-1 >= 0:
```

```
74              l_red = l_red+1
75              l_kol = l_kol-1
76          elif l_red+1 >= redova and l_kol-1 < 0:
77              l_red = l_red-1
78              l_kol = l_kol+1
79              smjer = 'SI'
80          elif l_red+1 < redova and l_kol-1 < 0:
81              l_red = l_red+1
82              l_kol = l_kol+1
83              smjer = 'JI'
84          elif l_red+1 >= redova and l_kol-1 >= 0:
85              l_red = l_red-1
86              l_kol = l_kol-1
87              smjer = 'SZ'
88      matrica[l_red][l_kol] = '*'
89
90  # print(redova, kolona, l_red, l_kol)
91
92  for red in matrica:
93      print(*red, sep='␣')
```

19. Rekurzija

Zadatak 10.1.

```
def s(n):
    if n == 0:
        return 0

    return s(n-1) + n*n

n = int(input())
print(s(n))
```

Zadatak 10.2.

```
def s(n):
   if n == 0:
      return 0

   return s(n-1) + 1 / (n**3 + n)

n = int(input())
print(s(n))
```

Zadatak 10.3.

```
def najkraca(i, j):
    suma = matrica[i][j]
    if i == br_redova-1 and j == br_kolona-1:
        return matrica[i][j]
```

```
    elif i == br_redova-1:
        return matrica[i][j] + najkraca(i, j+1)
    elif j == br_kolona-1:
        return matrica[i][j] + najkraca(i+1, j)
    else:
        return matrica[i][j] + min(najkraca(i+1, j),
                                   najkraca(i, j+1))

matrica = []

for i in range(4):
    red = input()
    red = red.split()
    red = [int(x) for x in red]
    matrica.append(red)

br_redova = len(matrica)
br_kolona = len(matrica[0])
print(najkraca(0, 0))
```

Zadatak 10.4.

```
def nadjiNajveci(lista):
    zadnji = lista.pop()

    if (len(lista) == 0):
        return zadnji

    return max(zadnji, nadjiNajveci(lista))

lista = []

n = int(input())
while n != -1:
    lista.append(n)
    n = int(input())

print(nadjiNajveci(lista))
```

Zadatak 10.5.

```
def sumaCifara(n):
   if n < 10:
      return n

   return n % 10 + sumaCifara(n//10)

```

```
7 n = int(input())
8 print(sumaCifara(n))
```

Literatura

[1] Paul Barry. *Python bez oklevanja*. 2 **edition**. CET, 2017. ISBN: 978-86-7991-397-5.

[2] David Beazley **and** Brian K. Jones. *Python kuvar*. 3 **edition**. Mikro knjiga, 2015. ISBN: 978-86-7555-408-0.

[3] Naomi Ceder. *The quick Python book*. Shelter Island, NY: Manning Publications, 2018. ISBN: 978-1617294037.

[4] *Coderbyte*. URL: `https://coderbyte.com/challenges`.

[5] *Codewars*. URL: `https://www.codewars.com/`.

[6] *CodingBat code practice*. URL: `https://codingbat.com/python`.

[7] Michael Dawson. *Python: uvod u programiranje*. 3 **edition**. Mikro knjiga, 2010. ISBN: 978-86-7555-362-5.

[8] Peter Farrell. *Math Adventures with Python: An Illustrated Guide to Exploring Math with Code*. No Starch Press, 2019. ISBN: 1593278675.

[9] Eric Freeman. *Um caruje: Naučite programiranje*. 1 **edition**. Mikro knjiga, 2018. ISBN: 978-86-7555-430-1.

[10] Tony Gaddis. *Starting Out with Python (4th Edition)*. Pearson, 2017. ISBN: 0134444329.

[11] John Guttag. *Introduction to computation and programming using Python : with application to understanding data*. Cambridge, Massachusetts: The MIT Press, 2016. ISBN: 978-0262529624.

[12] Colin Hughes. *Project Euler*. URL: `https://projecteuler.net/archives`.

[13] David Kopec. *Classic computer science problems in python*. Place of publication not identified: O'REILLY MEDIA, 2019. ISBN: 978-1617295980.

[14] *LeetCode*. URL: `https://leetcode.com/problemset/all/`.

[15] Bill Lubanovic. *Uvod u Python*. 1 **edition**. CET, 2015. ISBN: 978-86-7991-379-1.

[16] Mark Lutz. *Learning Python*. Sebastopol, CA: O'Reilly, 2013. ISBN: 978-1449355739.

[17] Mark Lutz. *Python pocket reference*. Beijing: O'Reilly, 2014. ISBN: 978-1449357016.

[18] Eric Matthes. *Python crash course : a hands-on, project-based introduction to programming*. San Francisco: No Starch Press, 2016. ISBN: 978-1593276034.

[19] Brian Overland. *Python opušteno*. 1 **edition**. CET, 2018. ISBN: 978-86-7991-402-6.

[20] William F. Punch **and** Richard Enbody. *The Practice of Computing Using Python (3rd Edition)*. Pearson, 2016. ISBN: 0134379764.

[21] Luciano Ramalho. *Fluent Python*. Sebastopol, CA: O'Reilly, 2015. ISBN: 978-1491946008.

[22] Zed Shaw. *Learn Python 3 the hard way*. Boston: Addison-Wesley, 2017. ISBN: 978-0134692883.

[23] Brett Slatkin. *Effective Python : 59 specific ways to write better Python*. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN: 978-0134034287.

[24] *Sphere Online Judge (SPOJ)*. URL: `https://www.spoj.com/problems/classical/`.

[25] Ben Stephenson. *The Python workbook : a brief introduction with exercises and solutions*. Cham: Springer, 2014. ISBN: 978-3319142395.

[26] Al Sweigart. *Uvod u Python, automatizovanje dosadnih poslova*. 1 **edition**. Kompjuter biblioteka, 2016. ISBN: 978-86-7310-505-5.

[27] Al Sweigart. *Cracking codes with Python : an introduction to building and breaking ciphers*. San Francisco: No Starch Press, Inc, 2018. ISBN: 978-1593278229.

[28] *Topcoder*. URL: `https://www.topcoder.com/challenges/`.

[29] Michael Urban **and** Joel Murach. *Murach's Python Programming*. Mike Murach & Associates, 2016. ISBN: 1890774979.

[30] *UVa Online Judge*. URL: `https://uva.onlinejudge.org/`.

[31] Lee Vaughan. *Impractical Python projects : playful programming activities to make you smarter*. San Francisco: No Starch Press, Inc, 2019. ISBN: 978-1593278908.

[32] John Zelle. *Python programming : an introduction to computer science*. Portland, Oregon: Franklin, Beedle & Associates Inc, 2017. ISBN: 978-1590282755.

Indeks

I

K

L

M

N

P

R

S

T

U

V