Programski paket *Julia*

Nedeljko Stojaković
Marko Pejić

Jun, 2021.

Cilj ovog materijala je upoznavanje sa osnovnim funkcionalnostima i mogućnostima *Julia* programskog paketa prilagođenim za predmete *Modelovanje/Modeliranje i simulacija sistema* na studijskim programima Računarstvo i automatika, Primenjeno softversko inženjerstvo, Biomedicinsko inženjerstvo, Mehatronika, Geodezija i geoinformatika kao i predmet *Upravljanje, modelovanje i simulacija sistema* na studijskom programu Elektronika, energetika i telekomunikacije. U okviru nastave na ovim predmetima cilj je da se upoznamo sa konceptima modelovanja različitih tipova sistema, načinima kako ih možemo matematički predstaviti i rešiti, kao i numeričkim metodama za njihovu simulaciju kroz programsko rešenje *Julia*. Drugim rečima, nije nam akcent na usavršavanju programiranja u *Julia* programskom okruženju, jer *Julia* nije jedini alat koji može da se koristi za ovu namenu, te stoga ove predmete ne treba poistovetiti sa softverskim okruženjem koje se koristi. Ohrabrujemo studente da se samostalno upoznaju sa drugim softverskim rešenjima, ali sa napomenom da će polaganje ovih predmeta biti koncentrisano na *Julia* programsko rešenje.
U materijalu će, kroz jednostavnije i složenije primere, biti predstavljeno *Julia* programsko okruženje, odnosno upoznaćemo se sa tipovima podataka, promenljivim, aritmetičkim i logičkim operacijama, ugrađenim funkcijama, osnovnim paketima, kao i radom sa vektorima i matricama.

1. Uvod

Julia je fleksibilni dinamički jezik[1] opšte namene sa odličnim performansama, pogodan za istraživanje i numeričko računanje. Veoma je popularan u različitim oblastima, od kojih su najznačajnije numerička analiza, inženjerstvo, mašinsko učenje, statistika, *Data Science*, metaprogramiranje, itd. Sličan je programskim jezicima *Matlab*, *Python*, *R*, a po brzini se poredi sa *C/C++*. Autori su sa *Massachusetts Institute of Technology* (MIT) i u januaru 2019. godine su dobili prestižnu nagradu *Wilkinson Prize for Numerical Software*.[2]

Programski paket *Julia* obuhvata i interaktivni komandni prozor **REPL** (*read-eval-print loop*), ugrađen u izvršnu datoteku programskog paketa. **REPL** omogućava brzu i laku evaluaciju izlaza, takođe ima pretraživu istoriju i *help* režim. Opciono tipiziranje, *multiple dispatch* i dobre performanse obezbeđene su zaključivanjem tipova (*type inference*) i kompajliranjem "na licu mesta"(*just-in-time*). *Julia* kombinuje karakteristike imperativnog, funkcionalnog i objektno-orijentisanog programiranja, čime pruža jednostavnost i izražajnost za numeričke proračune na

[1] Dinamički tipizirani jezici vrše proveru tipova u *runtime*-u, za razliku od statički tipiziranih jezika koji proveru tipova vrše pre izvršavanja, tokom kompajliranja.

[2] Nagrada se dodeljuje svake četiri godine za izuzetan doprinos u oblasti numeričkog softvera. Ime je dobila u znak sećanja na izvanredne doprinose naučnika *James H. Wilkinson*-a u istoj oblasti.

visokom nivou, ali takođe podržava i opšte programiranje. Da bi to postigla, *Julia* se nadovezuje na liniju matematičkih programskih jezika, ali takođe pozajmljuje mnogo iz popularnih dinamičkih jezika, uključujući *List*, *Perl*, *Python*, *Lua* i *Ruby*.

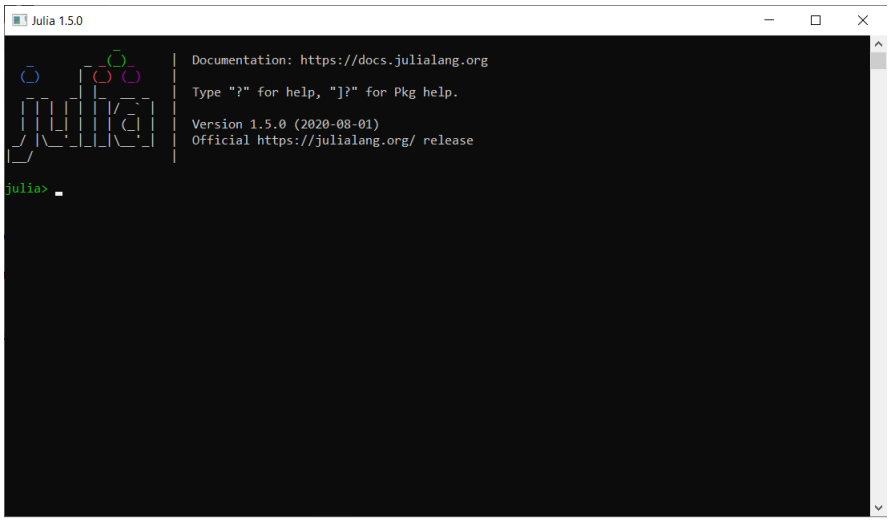
Julia je besplatan i *open-source* softver sa izuzetno jednostavnim postupkom instaliranja.[3]

[3] Instalacioni fajl, kao i uputstvo za instalaciju može se pronaći na link-u: https://julialang.org/downloads.

2. *Julia* okruženje

Najlakši način da se nauči i eksperimentiše sa *Julia* jezikom je korišćenjem interaktivnog okruženja (terminala) poznatim kao **REPL**, prikazanim na slici 1. Naredbe se kucaju u jednoj liniji, a pritiskom na taster *Enter* naredba se izvršava i rezultat se ispisuje u nastavku terminala. Ukoliko se na kraj naredbe doda simbol tačka-zarez (**;**), zaustaviće se ispis rezultata te naredbe na terminalu, ali će naredba svakako biti izvršena.[1]

[1] Na primer: $a = 5+3$ videćemo rezultat naredbe $a = 8$, dok naredbom $a = 5 + 3;$ a isto dobija vrednost **8**, ali nećemo videti ispis tog rezultata. Predlažemo da isprobate ove naredbe, pre nego što nastavite dalje.



Slika 1: *Julia* **REPL**

Naravno, korisniku je data mogućnost rada sa datotekama. Ekstenzija za *Julia* datoteke je ***.jl***. Naredbom *include("ime_datoteke.jl")* u terminalu, izvršava se skup naredbi koji je sačuvan u datoteci pod nazivom *ime_datoteke.jl*. Prethodno, neophodno je u terminalu definisati putanju do datoteke na računaru upotrebom funkcije *cd()*.[2]

[2] Recimo da je datoteka (ili više njih) sačuvana u folderu *MISS* koji se nalazi na *Desktop*-u, naredba bi bila sledeća: ***cd("C:/Users/.../Desktop/MISS")*** Obratiti pažnju na kose linije u putanji.

Postoji nekoliko različitih režima rada koji su podeljeni na osnovu uloge koju imaju. Trenutno aktivni režim vidljiv je u terminalu i to je upravo mesto gde se očekuje kucanje određene naredbe. Na slici 1 vidimo aktivni režim *julia* predstavljen zelenim slovima. Na tom mestu se očekuje kod za *Julia* naredbe. Pored ovog režima, tu je *help* režim, kom se pristupa tako što se u terminal unese komanda *?*. Ovim *help* režim postaje aktivan, što znači da će *Julia* pokušati da pronađe i prikaže

dokumentaciju za bilo kakav unos sa tastature. Osim ova dva režima, tu je i režim za rad sa paketima, ali o tome će biti više reči u posebnom poglavlju.

Pored osnovnog okruženja, moguće je podesiti naprednije okruženje za rad. Alati (editori) poput *Visual Studio Code*-a (VSC), *Atom*-a (*Juno IDE*), *Jupyter*-a i drugih, omogućavaju drugačije okruženje koje korisnik može da podesi prema svojim potrebama, instaliranjem nekoliko ekstenzija.[3]

[3] Za svako okruženje moguće je na internetu pronaći uputstva za podešavanje. Većina naprednih okruženja su besplatna.

3. Promenljive

Kao i u svakom programskom jeziku, promenljive predstavljaju imena koja su povezana sa vrednostima. Veoma su korisne za čuvanje rezultata, pogotovo kada u kodu imamo više naredbi i kada postoji potreba za čuvanjem međurezultata naših naredbi. *Julia* obezbeđuje jako fleksibilan sistem imenovanja promenljivih. Imena su *case-sensitive*[1] i nemaju nikakvo semantičko značenje.

Kada napišemo naredbu $\boldsymbol{a = 5}$, definisali smo promenljivu $\boldsymbol{a}$ u koju smo sačuvali vrednost **5**. Ono što možemo primetiti jeste da nigde nismo definisali tip promenljive $\boldsymbol{a}$, kao i da dodela vrednosti ide sa desne na levu stranu znaka jednakosti. *Julia* vodi računa o tipu promenljive na osnovu dodeljene vrednosti. Treba biti pažljiv prilikom definisanja imena promenljivih. Praksa je da imena promenljivih budu opisna, tj. da se na osnovu imena promenljive može zaključiti kakvu vrednost čuva.[2] Radi lakše čitljivosti koda, imena promenljivih treba da budu optimalne dužine.[3] Takođe, ukoliko nakon prethodne naredbe $\boldsymbol{a = 5}$, napišemo novu naredbu u kojoj koristimo isto ime promenljive $\boldsymbol{a = "tekst"}$, prethodna naredba će biti prepisana, tj. više neće postojati promenljiva $\boldsymbol{a}$ koja je čuvala celobrojnu vrednost (tip *Int64*), već nova promenljiva $\boldsymbol{a}$ koja čuva niz karaktera (tip *string*). *Julia* poseduje predefinisane promenljive: *Inf* za beskonačnost (*Infinity*), *NaN* za vrednost koja nije broj (*Not a Number*), *pi* za broj π, *i* ili *j* za imaginarnu jedinicu.
Konvencija za dodelu imena:

- Imena promenljvih pišu se malim slovima.
- Za razmak između reči koristi se donja crta (_).
- Imena funkcija pišu se malim slovima, ali bez upotrebe donje crte.
- Funkcijama koje svojim izvršavanjem menjaju vrednosti argumenata (ulazne vrednosti funkcije), imena završavaju sa uzvičnikom (!).

[1] Osetljivost na razliku između velikih i malih slova. **A** nije isto što i **a**.

[2] Na primer, definisaćemo promenljivu koja čuva rezultat naredbe za obim kruga: $\boldsymbol{obim\_kruga = 2 * r * pi}$

[3] Loš naziv: **povrsina_jednakokrakog_trapeza**. Praktičnije je napisati **pov_trapeza** ili **p_trapeza**, dok se u komentaru može ostaviti napomena o kakvom trapezu se radi.

4. Operatori

Svi operatori u *Julia* programskom jeziku mogu se podeliti na aritmetičke, relacione i logičke operatore. Zajedničko za ove grupe operatora je to što se mogu primenjivati na skalarne vrednosti, kao i na nizove (vektore, matrice) vrednosti. Za primenu operatora na nizove vrednosti, neophodna je upotreba operatora tačka, o kom će biti više reči na kraju ove sekcije.

4.1 Aritmetički operatori

Osnovni aritmetički operatori su: sabiranje (+), oduzimanje (-), množenje (*), (desno) deljenje (/), levo deljenje (\), stepenovanje (∧). Počevši od najvišeg, prioriteti operacija su sledeći: unarni minus, stepenovanje, množenje i deljenje, sabiranje i oduzimanje. Prioritet primene operatora može se promeniti upotrebom zagrada.

4.2 Relacioni operatori

Relacioni ili operatori poređenja su: == (jednako), != (različito), < (manje), > (veće), <= (manje ili jednako), >= (veće ili jednako). Rezultat izvršavanja relacionih operatora je uvek logička (*bool*) vrednost.

4.3 Logički operatori

Logički operatori su: negacija (!), konjukcija, odnosno operator logičkog "i"(&&) i disjunkcija, odnosno operator logičkog "ili"(||).

4.4 Operator tačka

Kao što je već naglašeno, svi navedeni operatori mogu se primenjivati i na nizove vrednosti, adekvatnom upotrebom tačka-operatora (*dot operator*). *Julia* definiše odgovarajuće tačka-operacije za svaki binarni operator, dizajnirane tako da rade na nivou elementa (*element-wise*) sa kolekcijama vrednosti. Prema tome, operator uz koji stoji tačka primenjuje se na svaki element odgovarajuće kolekcije. U programskom jeziku *Julia*, operator tačka obezbeđuje vektorizaciju, čime se znatno smanjuje potreba za pisanjem petlji, definisanjem privremenih kolekcija i slično.

Sledeći primer ilustruje kvadriranje svih elemenata niza, prvo upotrebom *for* petlje, a potom i upotrebom tačka-operatora.

```
fib = [1, 1, 2, 3, 5]

for i in 1:length(fib)
    fib[i] ^= 2
end
```

Identičan rezultat dobiće se i izvršavanjem vektorizovane verzije, odnosno upotrebom tačka-operatora uz operator za stepenovanje i dodelu vrednosti.

```
fib .^= 2
```

5. Paketi

Paket je biblioteka funkcija posvećena određenoj oblasti ili funkcionalnosti. *Julia* ima ugrađen paket menadžer, koji se koristi za instaliranje dodatnih funkcionalnosti

(paketa). Kako bismo instalirali neki od dodatnih paketa, neophodno je izvršiti sledeće naredbe:[1]

```
using Pkg
Pkg.add("ImePaketa")
```

[1] Napomena: instalaciju paketa nije potrebno izvršavati više puta, odnosno jednom kada je paket dodat kroz *Pkg*, on se može koristiti u različitim *Julia* fajlovima bez ponovnog instaliranja.

Na ovaj način instaliran je dodatni paket, što znači da je moguće koristiti ga u *Julia* komandnom prozoru i skriptama sa *.jl* ekstenzijom. Kako bismo koristili funkcionalnosti nekog ranije instaliranog paketa, neophodno je uključiti taj paket:

```
using ImePaketa
```

Kao što je već spomenuto u sekciji 2, *Julia* sadrži i poseban režim za rad sa paketima, kom se pristupa tako što se u komandnu liniju unese karakter]. Indikator uspešnog prelaska u režim za rad sa paketima je promena teksta u komandnoj liniji: umesto zelenog *julia* ispisa vidimo plavi ispis koji sadrži verziju *Julia*-e i oznaku *pkg* (npr. *(@v1.5) pkg*). Na ovaj način ne moramo da izvršavamo naredbu za uključivanje menadžera paketa (*using Pkg*), već direktno unosimo naredbe za dodavanje (instaliranje) paketa.

Neki od paketa zanimljivih za simulacije, koje ćemo koristiti u daljem radu su:

- *LinearAlgebra* - sadrži funkcije za rad sa vektorima i matricama.
- *Statistics* - sadrži osnovne statističke funkcije.
- *Plots* - sadrži funkcije za iscrtavanje i uređivanje grafika.
- *DifferentialEquations* - sadrži funkcije za rešavanje diferencijalnih jednačina.
- *ControlSystems* - sadrži funkcije za rad sa linearnim modelima i projektovanje upravljanja.

6. Funkcije

Funkcije su objekti koji mapiraju vrednosti ulaznih parametara na povratnu vrednost. U nastavku ovog poglavlja biće predstavljena osnovna sintaksa za definisanje funkcija, rad sa funkcijama, kao i neke naprednije mogućnosti definisanja i korišćenja *Julia* funkcija. Osnovna sintaksa je:

```
function zbir(x, y)
    x + y
end
```

Na ovaj način kreirali smo funkciju sa imenom *zbir* koja ima dva ulazna parametra *(x, y)*, a kao povratna vrednost funkcije vraća se zbir $x + y$.

Postoji i druga, sažeta sintaksa za definisanje funkcija, koja se najčešće koristi kada su u pitanju jednostavne funkcije poput prethodne. Ekvivalentna funkcija prethodnoj je:

```
zbir(x, y) = x + y
```

Pozivanje funkcije se vrši na uobičajen način, navođenjem imena funkcije i njenih argumenata unutar običnih zagrada.

```
zbir(2, 3)
```

Izvršavanjem naredbe, rezultat se ispisuje u terminalu. Ono što treba imati na umu jeste da funkcije definisane na ovaj, sažeti način ne mogu da sadrže više naredbi.

Kao što se može primetiti, definicija funkcije ne mora da sadrži naredbu *return*. U tom slučaju, povratna vrednost funkcije je poslednja naredba u telu funkcije. Naredbom *return* definiše se povratna vrednost iz funkcije, bez obzira na eventualno postojanje nekih naredbi nakon naredbe *return*.

```
function g(a, b)
  return a * b
  a + b
end
```

Pozivom funkcije g kao povratna vrednost dobija se rezultat naredbe $a * b$. Naredba $a + b$ neće biti izvršena, jer je ona definisana nakon naredbe *return*.

Ukoliko nam je potrebna funkcija koja će kao povratnu vrednost da vrati zbir i proizvod ulaznih parametara, to se može uraditi na sledeći način:

```
function f(a, b)
  return a * b, a + b
end
```

Za poziv funkcije $f(2, 3)$, rezultat će biti torka (*tuple*)[1] sa dva elementa $(6, 5)$. Ukoliko želimo da povratna vrednost bude vektor, onda će izmena u funkciji biti:

[1] Bitna razlika između torke i vektora je to što se vrednosti u okviru torke ne mogu menjati, jer *tuple* predstavlja nepromenljivu (*immutable*) kolekciju podataka.

```
function f(a, b)
  return [a * b, a + b]
end
```

Pored prethodno opisanih načina za definisanje povratne vrednosti funkcije, moguće je definisati više povratnih vrednosti bez korišćenja naredbe *return* na sledeći način:

```
function f(x, y)
    a = x + y
    b = x - y
    c = x * y
    d = x / y
    a, b, c, d
end
```

Poziv ovako definisane funkcije $f(10, 5)$ kao rezultat daje torku $(15, 5, 50, 2.0)$.[2] Do sada nije bilo reči o tome na koji način možemo da sačuvamo povratnu vrednost funkcije. Postoji više načina kako se to može uraditi, a u nastavku ćemo navesti nekoliko primera.

[2] Torka kao elemente može da sadrži vrednosti različitog tipa. $15, 5$ i 50 su tipa *Int64*, dok je 2.0 tipa *Float64*.
Izmeniti ovaj primer tako da funkcija kao povratnu vrednost vraća vektor. Komentarisati dobijeni rezultat.

```
p = f(20, 30)
x, y, z, w = f(20, 30)
```

6.1 Opcioni parametri funkcije

U nastavku ćemo reći nešto više o naprednim opcijama u radu sa funkcijama. Za početak, prikazaćemo na koji način možemo da koristimo predefinisane parametre u funkciji. Radi lakšeg razumevanja, proći ćemo kroz primer.

```
function f(x, y, z = 0)
    println(x)
    println(y)
    println(z)
end
```

Funkcija sadrži predefinisani parametar z čija vrednost je postavljena na nula. Ovakve funkcije možemo da pozovemo na više načina, u zavisnosti od toga koliko imamo predefinisanih parametara. U ovom konkretnom primeru funkciju možemo da pozovemo na dva načina.[3] Funkciju možemo pozvati kao $f(5, 10)$. Za ovakav poziv, argumenti x i y će dobiti vrednosti 5 i 10, respektivno, dok će argument z biti nula. Promenom poziva funkcije $f(5, 10, 15)$ definisali smo da će sada parametar z dobiti vrednost 15, tj. više neće biti nula.

[3] U *Julia* terminalu ćemo dobiti ispis *f (generic function with 2 methods)*, koji nam zapravo govori na koliko načina možemo da pozovemo funkciju f.

Važno je napomenuti da se opcioni parametri navode na kraju liste ulaznih parametara. Nakon opcionih parametara ne možemo više navoditi obavezne parametre.

Rad sa opcionim parametrima postaje nepraktičan ukoliko imamo veći broj tih parametara. Nepraktičnost se ogleda u tome što, da bi promenili vrednost, recimo, petog opcionog parametra, moramo definisati vrednosti i prva četiri. Drugim rečima, moramo se pridržavati redosleda opcionih parametara. Da bi se ovo izbeglo, uveden je novi tip parametara, a to su imenovani (*keyword*) parametri.

6.2 Imenovani (*keyword*) parametri funkcije

Imenovani parametri su parametri koji se definišu pomoću ključne reči. U odnosu na opcione parametre, imenovani parametri su praktičniji zbog toga što prilikom poziva funkcije ne moramo da pamtimo redosled parametara u funkciji, već prosto pomoću ključne reči dodelimo vrednost željenom parametru. Prilikom definisanja funkcije obavezni i imenovani parametri se razdvajaju naredbom tačka-zarez. Slično kao kod opcionih parametara, prvo se navode obavezni parametri, pa nakon toga imenovani. Ovakvih parametara može biti konačno mnogo.

```
function f(x, y; a = 0, b = 0, c = 0, d = 0)
    println(x)
    println(y)
    println(a)
    println(b)
    println(c)
    println(d)
end
```

Navedeni primer prikazuje dva obavezna parametra x i y, dok imamo četiri imenovana parametra a, b, c, d, što su ujedno i ključne reči, a inicijalno su postavljeni na vrednost nula.

Sada ako želimo da pozovemo funkciju f tako da od imenovanih parametara promenimo samo vrednost parametra c, možemo to uraditi na sledeći način: $f(5, 10, c = 15)$. Ostali imenovani parametri ne menjaju inicijalnu vrednost. Ovako nešto ne bi bilo moguće izvesti sa opcionim parametrima. Takođe, ono što je prednost funkcija sa imenovanim parametrima je to što u pozivu funkcije ne moramo da vodimo računa o redosledu imenovanih parametara. Moguće je pozvati funkciju kao *f(5, 10, c = 15, a = 3)*, ili kao *f(b = 8, 5, c = 4, 10)*.

Kombinovanje opcionih i imenovanih parametara je dozvoljeno. Primer koji to ilustruje prikazan je u nastavku.

```
function f(x, y, z = 0; a = 0, b = 0, c = 0, d = 0)
    println(x)
    println(y)
    println(z)
    println(a)
    println(b)
    println(c)
    println(d)
end
```

6.3 Anonimne funkcije

Anonimne funkcije, kako im naziv govori, su funkcije kojima ne definišemo ime. Čest primer upotrebe anonimne funkcije je situacija kada jednoj funkciji kao argument prosleđujemo drugu funkciju. Anonimne funkcije ne mogu da se pozivaju kao obične funkcije.

Anonimne funkcije se mogu definisati na više načina. Osnovni primer definisanja anonimne funkcije je:

```
function (x, y)
    x + y
end
```

Iz ovog primera vidimo da jedina razlika u izgledu ove funkcije u odnosu na ostale je ta što ova funkcija nema ime.

Drugi, verovatno češći u upotrebi, način za definisanje anonimnih funkcija je:

```
(x, y) -> x + y
```

Prethodni zapis predstavlja anonimnu funkciju sa kraćim zapisom, gde u običnim zagradama definišemo ulazne parametre funkcije (ukoliko ih ima više), a nakon strelice definišemo telo funkcije, tj. šta ta funkcije računa i vraća kao povratnu vrednost.

Jedan jednostavan primer na kome možemo videti mehanizam korišćenja anonimne funkcije je sledeći:[4]

[4] Pogledati *help* za funkciju *map*.

```
function f(g, x, y)
    g(x, y)
end
```

Primer poziva funkcije:

```
f((x, y) -> x + y, 2, 3)
```

U funkciji f kao ulazni parametri definisani su g koje će uzeti vrednost anonimne funkcije i celobrojne vrednosti x i y. U telu funkcije, praktično je pozvano izvršavanje anonimne funkcije koja će biti prosleđena kao argument. U primeru poziva funkcije, kao prvi argument definisana je anonimna funkcija čija je uloga da sabere ulazne argumente.

7. Kontrola toka programa

Kontrola toka programa odnosi se na uslovno izvršavanje (grananje) i programske petlje, tj. ponavljanje izvršavanja.

7.1 Uslovno grananje

Naredbe koje se mogu koristiti za uslovno grananje su *if-elseif-else* konstrukcija i ternarni operator (*?:*). Kod uslovnog grananja, vrši se provera da li je dati relacioni izraz tačan ili netačan, i u zavisnosti od dobijene logičke vrednosti izvršava se različit blok naredbi. Korišćenje *if-elseif-else* konstrukcije je uobičajeno, odnosno podrazumeva sledeće varijante:

- izvršavanje dela koda ako je uslov ispunjen (samo *if* deo),
- zavisno od vrednosti uslova, izvršava se jedna od dve moguće opcije (*if-else* konstrukcija),
- ulančani uslovi, tj. proveravanje sekvence navedenih uslova i izvršavanje bloka koda prvog tačnog uslova (*if-elseif* konstrukcija). Ukoliko nijedan od uslova iz sekvence nije tačan, može se definisati blok naredbi koji se tada izvršava (*else* deo).

Naredni primer ilustruje upotrebu različitih varijanti *if-elseif-else* konstrukcija.

```
t = 8

# samo if deo
if t > 7
    print("Vrednost je veca od 7")
end

# if-else konstrukcija
if t < 5
    y = 0
else
    y = 1
end

# if-elseif-else konstrukcija
if t < 5
   y = 0
elseif t < 10
   y = 1
elseif t < 15
   y = 2
else
   y = 3
end
```

Za jednostavna uslovna grananja moguća je i upotreba ternarnog operatora, na način standardan za većinu programskih jezika.

```
y = t < 5 ?  0 : 1
```

7.2 *For* petlja

Prebrojive petlje se upotrebljavaju kada se broj prolazaka kroz petlju može unapred odrediti, odnosno njihova uobičajena upotreba je prolaz kroz niz ili opseg vrednosti. Naredni primeri ilustruju sintaksu različitih načina za definisanje prebrojive petlje, upotrebom ključne reči *for*.

```
array = [1, 2, 3, 4, 5]
arr_sum = 0

for i in 1:length(array)
    global arr_sum += array[i]
end

arr_sum = 0

for el in array
    global arr_sum += el
end
```

Prethodni primer prikazuje računanje sume svih elemenata vektora pomoću *for* petlje, na dva nešto drugačija načina. Bitno je primetiti ključnu reč *global*, koja se koristi zato što se u okviru petlje pristupa promenljivoj iz okruženja. Ukoliko bismo isti kod smestili unutar funkcije, promenljiva *arr_sum* bi bila lokalna, odnosno

ključna reč *global* se ne bi koristila.

Još jedna od upotreba *for* petlje jeste generisanje nizova (*comprehensions*). Iako je ovo moguće uraditi i uobičajenim *for* petljama, češće se koriste skraćeni zapisi, ilustrovani u narednom primeru.

```
a = [2^i for i in 0:10]
b = [2^i for i in 0:10 if i % 2 == 0]
```

7.3 *While* petlja

Kod neprebrojivih petlji uslov završavanja zavisi od proračuna tokom izvršavanja petlje, odnosno ne zna se tačan broj prolazaka kroz petlju, sve do prestanka važenja uslova. Uslov se proverava na početku petlje (i svake sledeće iteracije) i predstavlja relacioni izraz, čiji je rezultat skalar sa logičkom vrednošću. Naredni primer ilustruje upotrebu *while* petlje u okviru funkcije. [1] Bitno je primetiti da nije korišćena ključna reč *global* prilikom izmene vrednost promenljive n, zbog toga što se petlja nalazi u funkciji, pa je n lokalna promenljiva.

[1] Pitanje: Šta radi ovaj primer, odnosno šta je povratna vrednost funkcije?

```
function primer(level, target)
    n = 1
    while level - 1 / n >= target
        level -= 1 / n
        n += 1
    end
    return n
end
```

Sledeći primer ilustruje funkciju za formiranje Fibonačijevog niza brojeva manjih od vrednosti parametra *limit*, upotrebom *while* petlje. Bitno je primetiti upotrebu funkcije *push!*, koja vrši dodavanje prosleđenog elementa (drugi parametar) u određeni niz (prvi parametar).

```
function fibonaci(limit)
    f = [1, 1]
    n = 2
    while f[n] + f[n-1] < limit
        push!(f, f[n] + f[n-1])
        n += 1
    end
    return f
end
```

8. Rad sa vektorima i matricama

Kao što je već naglašeno u uvodnom delu, *Julia* je preuzela dosta osobina iz matematičkih programskih jezika, najviše iz *Matlab*-a. Ovo je najuočljivije prilikom korišćenja mnogobrojnih naprednih opcija indeksiranja i funkcija za rad sa vektori-

ma i matricama, tako da je upotreba petlji za iteriranje kroz te strukture svedena na minimum.

8.1 Kreiranje vektora i matrica

Kreiranje vektora podrazumeva navođenje njegovih elemenata u uglastim zagradama, tako da se elementi razdvajaju razmacima ili zarezima. Bitna razlika između ova dva načina kreiranja vektora je u tome kako *Julia* kompajlira ove naredbe, odnosno koliko dimenzija sadrži rezultujući vektor. U nastavku je dat jednostavan primer kreiranja vektora na dva navedena načina.

```
v1 = [1, 2, 3, 4, 5]
v2 = [1 2 3 4 5]
```

Izvršavanjem ovih naredbi dobijaju se dve promenljive prikazane na slici 2.

```
julia> v1 = [1, 2, 3, 4, 5]
5-element Vector{Int64}:
 1
 2
 3
 4
 5

julia> v2 = [1 2 3 4 5]
1×5 Matrix{Int64}:
 1  2  3  4  5
```

Slika 2: Kreiranje vektora

Prvo što se može primetiti iz samog formata ispisa jeste to da je v_1 definisan kao vektor kolona, a v_2 kao vektor vrsta. Druga, takođe značajna razlika između ova dva načina definisanja vektora jeste u dimenzijama rezultata: ukoliko pažljivo pogledamo ispis, primetićemo da je tip rezultata ovih naredbi drugačiji: kao rezultat prve naredbe dobija se promenljiva tipa $Vector\{Int64\}$, dok je rezultat druge naredbe tipa $Matrix\{Int64\}$ [1]. Promenljiva v_1 je jednodimenziona, odnosno predstavlja vektor, dok je promenljiva v_2 dvodimenziona, odnosno predstavlja matricu sa jednom vrstom i pet kolona. Ovo je nešto čega bi trebalo da budemo svesni, s obzirom da može dovesti do problema prilikom indeksiranja vektora (posebno logičkog indeksiranja).

Julia omogućava jednostavno generisanje rastućih ili opadajućih nizova ekvidistantnih vrednosti upotrebom opsega. Opsezi su definisani kao posebni tipovi u programskom jeziku *Julia* i definišu se upotrebom operatora ":", kojim se formira sekvenca brojeva u zadatim granicama zatvorenog intervala. Sledeći primer ilustruje kreiranje dva opsega vrednosti u intervalu $[0, 10]$. Promenljiva o_1 kreirana je bez definisanja koraka, odnosno sa podrazumevanom vrednošću koraka koja iznosi jedan. Prilikom kreiranja promenljive o_2, definisan je korak od 0.01, što znači da će vrednosti u tom opsegu biti: $0, 0.01, 0.02, ..., 9.98, 9.99, 10$.

[1] U starijim verzijama *Julia* programskog jezika, ova dva tipa imaju zajedničko ime *Array*, a razlikuju se samo u dimenzijama ($Array\{Int64, 1\}$ i $Array\{Int64, 2\}$)

```
o1 = 0:10
o2 = 0:0.01:10
```

Bitno je razumeti da promenljiva tipa opseg ne sadrži same vrednosti koje taj opseg definiše. Da bi se generisao niz vrednosti definisanih opsegom, potrebno je pozvati funkciju *collect*, što je ilustrovano u sledećem primeru.

```
o = 0:0.5:5
arr = collect(o)
```

Matrice se kreiraju slično kao i vektori, odnosno u uglastim zagradama se navode elementi, tako što se elementi u okviru iste vrste razdvajaju praznim mestima, dok se za prelazak u novu vrstu koristi karakter ";". Naredni primer ilustruje kreiranje matrice 3x3.[2]

[2] Rezultat izvršavanja sledeće naredbe biće tipa *Array{Int64,2}*

```
A = [1 2 3; 4 5 6; 7 8 9]
```

8.2 Indeksiranje vektora i matrica

Treba razjasniti zašto su nam matrice toliko bitne. U današnjem tehničkom/tehnološkom svetu prisutna je ogromna količina podataka. Veliki procenat tih podataka zapisan je upravo u obliku matrice (2D ili 3D).[3] Rad sa podacima se često svodi na rad sa matricama. Zato nam je na raspolaganju ogroman broj funkcija, ali tu je sa druge strane potrebna i matematička podloga koja se odnosi na matrice.

[3] Primer 2D matrice može biti tabela sa rezultatima kolokvijuma, dok je fotografija u boji 3D matrica.

Vektor je takođe matrica kod koje je jedna od dimenzija jednaka jedinici. Dakle, sve što se prikaže za matrice, važi i za vektore.

Osnovni rad sa matricama podrazumeva poznavanje svih (ili većine) načina indeksiranja. Tu razlikujemo tri načina indeksiranja:

- **Jednodimenziono**. Potrebno je znati na koji način su indeksi raspoređeni u matrici. Sledeća matrica ilustruje jednodimenzione indekse u matrici 3x3:

  $$\mathrm{A} = \begin{bmatrix} a_1 & a_4 & a_7 \\ a_2 & a_5 & a_8 \\ a_3 & a_6 & a_9 \end{bmatrix}$$

  Dakle, indeksi su raspoređeni po kolonama. Primer indeksiranja elementa na poziciji 6 u matrici A je $A[6]$. Ovaj oblik indeksiranja se najčešće koristi kod indeksiranja vektora.

- **Dvodimenziono**. Kako kod 2D matrica imamo upravo dve dimenzije (vrstu i kolonu)[4], od nas se očekuje da znamo indeks vrste i indeks kolone na čijem preseku se nalazi traženi element. Sledeća matrica ilustruje dvodimezione indekse u matrici 3x3:

[4] Kod 3D matrica postoji treća dimenzija (list) kojom se opisuje dubina matrice. U našim primerima zadržaćemo se na 2D matricama.

$$\mathrm{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

Indeksiranje elementa koji se nalazi na preseku treće vrste i druge kolone u matrici A je $A[3, 2]$. U uglastoj zagradi na prvom mestu navodi se indeks vrste, a na drugom indeks kolone.

- **Logičko indeksiranje**. Ovaj način indeksiranja je malo specifičniji. Osnovna ideja je da se kreira logička matrica[5] (maska) kojom se definiše pravilo po kome se indeksiraju potrebni elementi. Sa takvom matricom se indeksira matrica nad kojom radimo.

  [5] Matrica tipa *bool* (*BitArray*). Sadrži nule (*false*) i jedinice (*true*).

  Na primer, ukoliko želimo selektovati element iz prethodno definisane matrice A koji se nalazi na preseku treće vrste i druge kolone, kreirali bi masku istih dimenzija kao matrica A koja ima logičku jednicu na poziciji željenog elementa, a ostali elementi bi bili nule.

  Za matricu $\mathrm{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, maska će biti $\mathrm{M} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

  Za ovako pravilno definisanu logičku matricu M, naredba u kodu bi bila $A[M]$. Rezultat ove naredbe bio bi element na poziciji logičke jedinice, tj. broj 8.

  Retko ćemo logičku matricu kreirati "ručno". Češće ćemo to raditi upotrebom nekih uslova ili funkcija, u zavisnosti od potrebe u zadatku.

Primeri koji su navedeni za sva tri načina indeksiranja kao rezultat vraćaju isti element (broj 8) za gore definisanu matricu A. Na ovaj način smo prikazali kako jedan isti element možemo da indeksiramo na tri različita načina. Naravno, neće uvek biti slučaj da možemo na sva tri načina izdvojiti tražene elemente, ali će u skladu sa složenošću zadatka biti potrebno odlučiti se za najefikasniji i najjednostavniji način.

Kod jednodimenzionog i dvodimenzionog načina indeksiranja, indeksi elemenata koje želimo da selektujemo mogu se definisati kao vektori. Sve što je prethodno opisano za vektore, može se iskoristiti i ovde. Recimo, ako želimo da iz matrice A selektujemo sve elemente koji se nalaze na presecima prve i treće vrste sa drugom kolonom, naredba za to bi bila $A[[1, 3], 2]$.

Više primera za rad sa matricama sledi u poglavlju sa rešenim primerima.

8.3 Funkcije za rad sa vektorima i matricama

U ovoj sekciji dat je kratak pregled funkcija za rad sa vektorima i matricama. Najbolji način za upoznavanje sa ovim funkcijama je kroz primere i upotrebom *Julia* dokumentacije i *help* režima rada.

Neke od funkcija koje ćemo koristiti za generisanje vektora i matrica su:

- *ones* - generiše vektor/matricu jedinica željenih dimenzija.
- *zeros* - generiše vektor/matricu nula željenih dimenzija.
- *rand* - generiše vektor/matricu slučajnih vrednosti u opsegu $[0, 1]$

Dodatno, postoji i operator I, koji se koristi za generisanje jedinične matrice.[6] Ovaj operator je moguće koristiti kao funkciju ili u kombinaciji sa nekim od navedenih funkcija za generisanje matrica, što je prikazano u sledećim primerima:

[6] Za korišćenje operatora I potrebno je uključiti paket *LinearAlgebra*.

```
m1 = I(5)
m2 = I + zeros(5, 5)
m3 = Matrix{Bool}(I, 5, 5)
```

Selekcija određenih elemenata matrice može se uraditi na više načina: različitim načinima indeksiranja i pomoću ugrađenih funkcija. Dve korisne funkcije za selekciju gornje i donje trougaone matrice su *tril* i *triu*. Ove funkcije primaju dva parametra: ulaznu matricu i ceo broj k, a vraćaju gornju (*triu*) i donju (*tril*) trougaonu matricu, počevši od k-te dijagonale. Sledeći primeri ilustruju rezultate različitih poziva funkcija *triu* i *tril* nad ulaznom matricom M dimenzija 4x4.

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \xrightarrow{triu(M)} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 6 & 7 & 8 \\ 0 & 0 & 11 & 12 \\ 0 & 0 & 0 & 16 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \xrightarrow{triu(M,\ 1)} \begin{bmatrix} 0 & 2 & 3 & 4 \\ 0 & 0 & 7 & 8 \\ 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \xrightarrow{tril(M,\ -2)} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 \\ 13 & 14 & 0 & 0 \end{bmatrix}$$

Još jedna od često korišćenih funkcija je *reverse*, koja vrši obrtanje elemenata vektora ili matrice. U slučaju rada sa matricama, *reverse* prima još jedan parametar *dims*, koji označava dimenziju po kojoj se vrši obrtanje (vrednost 1 predstavlja obrtanje po vrstama, dok vrednost 2 označava obrtanje matrice po kolonama). Ukoliko se ne prosledi vrednost za parametar *dims*, obrtanje se izvršava i po vrstama i po kolonama. Naredni primeri ilustruju upotrebu *reverse* funkcije za različite vrednosti *dims* parametra.

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \xrightarrow{reverse(M,\ dims=1)} \begin{bmatrix} 13 & 14 & 15 & 16 \\ 9 & 10 & 11 & 12 \\ 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \xrightarrow{reverse(M,\ dims=2)} \begin{bmatrix} 4 & 3 & 2 & 1 \\ 8 & 7 & 6 & 5 \\ 12 & 11 & 10 & 9 \\ 16 & 15 & 14 & 13 \end{bmatrix}$$

Agregatne funkcije se vrlo često koriste u radu sa vektorima i matricama, i njihov zadatak je vraćanje jedne vrednosti za prosleđenu kolekciju vrednosti. Neke od uobičajenih agregatnih funkcija u programskom jeziku *Julia* su: *minimum*, *maximum*, *sum*, *mean*[7], *prod*, itd. Sve navedene funkcije podržavaju rad sa vektorima i matricama. U slučaju pozivanja agregatne funkcije nad matricom, moguće je proslediti i dodatni parametar *dims*, koji govori da li se agregatna funkcija izvršava po vrstama ili kolonama. Bitno je napomenuti da vrednosti parametra *dims* imaju suprotno značenje u poređenju sa funkcijom *reverse*, odnosno vrednost 1 predstavlja pozivanje agregatne funkcije po kolonama, a vrednost 2 po vrstama. Sledeći primeri prikazuju rezultate dobijene upotrebom različitih poziva nekih od agregatnih funkcija.

[7] Za funkciju *mean* potrebno je uključiti paket *Statistics*.

$$M = \begin{bmatrix} 19 & 16 & 9 & 28 \\ 3 & 10 & 15 & 5 \\ 25 & 30 & 19 & 16 \\ 2 & 13 & 20 & 11 \end{bmatrix} \xrightarrow{mean(M)} 15.0625$$

$$M = \begin{bmatrix} 19 & 16 & 9 & 28 \\ 3 & 10 & 15 & 5 \\ 25 & 30 & 19 & 16 \\ 2 & 13 & 20 & 11 \end{bmatrix} \xrightarrow{maximum(M,\ dims=1)} \begin{bmatrix} 25 & 30 & 20 & 28 \end{bmatrix}$$

$$M = \begin{bmatrix} 19 & 16 & 9 & 28 \\ 3 & 10 & 15 & 5 \\ 25 & 30 & 19 & 16 \\ 2 & 13 & 20 & 11 \end{bmatrix} \xrightarrow{maximum(M,\ dims=2)} \begin{bmatrix} 28 \\ 15 \\ 30 \\ 20 \end{bmatrix}$$

Sve funkcije navedene u ovoj sekciji su dizajnirane za rad sa vektorima i matricama, odnosno njihova upotreba na skalarima nema smisla. Međutim, postoji značajan broj funkcija dizajniranih za rad sa skalarima koje se koriste i u radu sa vektorima i matricama. U slučaju da želimo da upotrebimo neku takvu funkciju na elemente vektora ili matrice, neophodna je upotreba tačka-operatora. Sledeći primer ilustruje kreiranje matrice nasumičnih vrednosti upotrebom funkcije *rand*, kao i upotrebu funkcije *round* za zaokruživanje svih elemenata generisane matrice. Bitno je primetiti da se uz funkciju *round* koristi i tačka-operator, zbog toga što ova funkcija kao parametar prima jednu vrednost (skalar), a mi želimo da je primenimo na svaku vrednost generisane matrice dimenzija 5x5.

```
m = round.(rand(5, 5) * 30)
```

9. Primeri sa rešenjima

9.1 Jednostavniji primeri

Sledeći primeri će ilustrovati jednostavnije operacije u radu sa matricama, kao što su selekcija elemenata matrice i korišćenje ugrađenih funkcija namenjenih za rad sa matricama. Treba uzeti u obzir da prikazane ilustracije nisu jedini način za rešavanje zadatka, kao i što korišćene funkcije u zadacima nisu jedine funkcije koje postoje.
U narednim primerima koristiće se sledeća matrica A:

$$\mathrm{A} = \begin{bmatrix} 1 & 4 & -2 & 9 & 6 \\ -1 & 0 & 0 & 3 & 7 \\ 99 & 3 & -3 & 4 & 7 \\ 5 & -6 & 0 & -8 & 3 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

Matrica se može generisati sledećom naredbom:

```
A = [1 4 -2 9 6; -1 0 0 3 7; 99 3 -3 4 7; 5 -6 0 -8 3; 1 2 3 4 5]
```

Primer 1. Iz zadate matrice A selektovati element 99 upotrebom jednodimenzionog indeksiranja. Rezultat smestiti u promenljivu p.

```
p = A[3]
```

Primer 2. Iz zadate matrice A selektovati element 99 upotrebom dvodimenzionog indeksiranja. Rezultat smestiti u promenljivu p.

```
p = A[3, 1]
```

Primer 3. Iz zadate matrice A selektovati element 99 upotrebom logičkog indeksiranja. Rezultat smestiti u promenljivu p.[1]

[1] Ponoviti isti primer korišćenjem funkcije *zeros*.

```
M = [0 0 0 0 0;
     0 0 0 0 0;
     1 0 0 0 0;
     0 0 0 0 0;
     0 0 0 0 0]

M_logical = convert.(Bool, M)

p = A[M_logical]
```

Primer 4. Iz zadate matrice A selektovati prvu vrstu.

```
vrsta = A[1, :]
```

Primer 5. Iz zadate matrice A selektovati poslednju kolonu.

```
kolona = A[:, end]
```

Primer 6. Iz zadate matrice A selektovati sve neparne vrste.

```
nep_vrste = A[1:2:end, :]
```

Primer 7. Iz zadate matrice A selektovati sve pozitivne elemente.[2]

[2] Rezultat naredbe $A .> 0$ je logička matrica, tako da je ovo zapravo logičko indeksiranje.

```
pozitivni = A[A .> 0]
```

Primer 8. Iz zadate matrice A selektovati sve elemente koji se nalaze na opsegu [-5, 5).[3]

[3] Takođe logičko indeksiranje.

```
el_opseg = A[(A .>= -5) .& (A .< 5)]
```

Primer 9. Odrediti maksimalni element matrice A.

```
max_el = maximum(A)
```

Primer 10. Iz zadate matrice A izdvojiti sporednu dijagonalu.

```
using LinearAlgebra # dodavanje neophodnog paketa za funkciju diag()

A_rev = reverse(A, dims = 2)
sd = diag(A_rev)
```

Primer 11. Iz zadate matrice A izdvojiti sve parne elemente.[4]

[4] Još logičkog indeksiranja. Kombinacija sa funkcijom.

```
parni = A[rem.(A, 2) .== 0]
```

Primer 12. U zadatoj matrici A pronaći vrstu sa maksimalnom sumom. Uzeti u obzir situaciju da više vrsta ima istu sumu i da je ona maksimalna.[5]

[5] U čemu se razlikuje rešenje ako ovu situaciju ne uzmemo u obzir?

```
suma_svake_vrste = sum(A, dims = 2)
max_el = maximum(suma_svake_vrste)
vrste = findall(suma_svake_vrste .== max_el)
```

9.2 Složeniji primeri

Primer 1. Napisati funkciju *primer1* koja kao ulazni parametar sadrži kvadratnu matricu A, a kao izlazne vraća:

- skalar s koji predstavlja srednju vrednost elemenata u poslednjoj vrsti matrice A.
- vektor v koji sadrži sve pozitivne elemente sa glavne dijagonale matrice A.

```
# dodavanje neophodnog paketa za funkcije mean() i diag()
using Statistics, LinearAlgebra

function primer1(A)
    s = mean(A[end, :])
    gd = diag(A)
    v = gd[gd .> 0]

    return s, v
end
```

Primer poziva fukcije:

```
A = round.(20 * randn(10, 10))

sk, vek = primer1(A)
```

Primer 2. Napisati funkciju koja određuje maksimalni element matrice A i njegovu poziciju (vrstu i kolonu) u matrici A, bez upotrebe ugrađenih funkcija.

```
function maksimalni(A)
    n, m = size(A)
    max_el = A[1,1]
    vrsta = 1;
    kolona = 1;
    for i = 1:n
        for j = 1:m
            if A[i, j] > max_el
                max_el = A[i, j];
                vrsta = i;
                kolona = j;
            end
        end
    end
    return max_el, vrsta, kolona
end
```

Primer poziva funkcije:

```
A = round.(20 * randn(5, 5))
max_el, vrsta, kolona = maksimalni(A)
```

Primer 3. Napisati funkciju koja određuje kumulativnu sumu[6] vektora.

[6] Kumulativna suma vektora $[\mathbf{1, 2, 3, 4, 5}]$ je vektor $[\mathbf{1, 3, 6, 10, 15}]$

```
function cum_sum(v)
    ret_val = zeros(Int64, length(v))
    ret_val[1] = v[1]
    for i = 2:length(v)
        ret_val[i] = sum(v[1:i])
    end

    return (ret_val)
end
```

Primer poziva funkcije:

```
v = [1, 2, 3, 4, 5]
c = cum_sum(v)
```

Primer 4. Data je tabela T koja sadrži podatke o osobama. Napisati kod koji:

- određuje osobu sa najvećom težinom (ili osobe ako ih je više). Ispisati poruku na terminal koja prikazuje imena osobe(a) i maksimalnu težinu.
- određuje prosečnu visinu i težinu.

Ime	Pol	Starost	Težina	Visina
Ana	ž	20	46	160
Bojan	m	24	52	165
Vlada	m	24	95	195
Gordana	ž	30	57	160
Dejan	m	36	84	185
Zoran	m	22	80	180

```
T = ["Ime" "Pol" "Starost" "Tezina" "Visina";
     "Ana" "z" 20 46 160;
     "Bojan" "m" 24 52 165;
     "Vlada" "m" 24 95 195;
     "Gordana" "z" 30 57 160;
     "Dejan" "m" 36 84 185;
     "Zoran" "m" 22 80 180]

# osoba sa najvecom tezinom i njena tezina
#(paziti ukoliko ima vise osoba sa istom tezinom. U tom slucaju prikazati sve osobe)

podaci = T[2:end, 3:end]

maks_tezina = maximum(podaci[:, 2])
poz = findall(podaci[:, 2] .== maks_tezina)

if length(poz) == 1
    ime = T[poz .+ 1, 1][1]
    println("Osoba sa najvecom tezinom je ", ime, " (", maks_tezina, "kg)")
else
    imena = T[poz .+ 1, 1]
    println("Osobe sa tezinom ", maks_tezina, "kg su: ")

    [println(ime) for ime in imena]
end

# prosecna visina i tezina

prosecna_visina = mean(podaci[:, end])
prosecna_tezina = mean(podaci[:, end-1])
```

Primer 5. Napisati funkciju *primer5* koja određuje srednju vrednost elemenata iznad sporedne dijagonale zadate matrice. Napisati primer poziva funkcije.

```
function primer5(A)
    M = ones(size(A))
    M = triu(M, 1)
    M_rev = reverse(M, dims = 2)
    M_logicko = convert.(Bool, M_rev)
    el_iznad_sporedne = A[M_logicko]

    sred_vred = mean(el_iznad_sporedne)
end
```

Primer poziva funkcije:

```
K = [1 4 -2 9 6; -1 0 0 3 7; 99 3 -3 4 7; 5 -6 0 -8 3; 1 2 3 4 5]
s_v = primer5(K)
```

Zadaci za vežbu

Zadatak 1. Za proizvoljnu kvadratnu matricu A, izdvojiti sve parne kolone.

Zadatak 2. Za proizvoljnu kvadratnu matricu A, izdvojiti sve elemente koji su deljivi sa 9.

Zadatak 3. Za proizvoljnu kvadratnu matricu A, izdvojiti elemente koji se nalaze na preseku parnih vrsta i parnih kolona.

Zadatak 4. Napisati funkciju koja određuje zbir svih elemenata matrice A,

$$\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}$$

gde je m broj vrsta, a n broj kolona, koji imaju osobinu da je zbir indeksa $(i + j)$ paran broj $(A_{11} + A_{13} + ...)$

Zadatak 5. Napisati funkciju koja za zadatu kvadratnu matricu A, određuje:

- vektor m koji se formira od elemenata sa glavne dijagonale matrice A.
- skalar s koji predstavlja srednju vrednost elemenata iznad glavne dijagonale matrice A. (može se koristiti funkcija *mean()* iz programskog paketa *Statistics*)

Zadatak 6. Napisati funkciju koja za zadate kvadratne matrice A i B istih dimenzija određuje:

- vektor m koji se sastoji od elemenata ispod glavne dijagonale matrice A koji su pozitivni celi brojevi deljivi sa 3.
- skalar s koji predstavlja srednju vrednost elemenata sa sporedne dijagonale matrice B koji su veći od srednje vrednosti elemenata sa glavne dijagonale matrice A.

Zadatak 7. Za podatke iz tabele T (Primer 4 - Složeni primeri) napisati kod koji određuje:

- koliko je ženskih, a koliko muških osoba (poželjno je prikazati i njihova imena),
- prosečnu visinu i težinu ženskih osoba,
- prosečnu visinu i težinu muških osoba,
- najstariju i najmlađu osobu,
- standardnu devijaciju za visinu.

Zadatak 8. Napisati funkciju koja određuje poziciju nenultih elemenata proizvoljne matrice. Zadatak rešiti bez korišćenja funkcije *findall*.

Zadatak 9. Napisati funkciju, po uzoru na funkciju *prod*, koja određuje proizvod svih elemenata vektora.

Zadatak 10. Napisati funkciju, po uzoru na funkciju *sum*, koja određuje sumu elemenata proizvoljne matrice. Implementirati opcioni ili imenovani parametar funkcije na osnovu koga će se računati suma elemenata po vrstama ili po kolonama matrice.

Literatura

- *Julia* programski jezik (sajt) https://julialang.org/
- *Think Julia* (*online* knjiga) https://benlauwens.github.io/ThinkJulia.jl/latest/book.html.