UNIVERZA V LJUBLJANI
FGG
Fakulteta za gradbeništvo in geodezijo

Robert Klinc, Matevž Dolenc

Programiranje za inženirje

Rešeni primeri v programskem jeziku Python

Robert Klinc, Matevž Dolenc

Programiranje za inženirje: Rešeni primeri v programskem jeziku Python

univerzitetni učbenik

Ljubljana, oktober 2024

Programiranje za inženirje:
Rešeni primeri v programskem jeziku Python

univerzitetni učbenik

Avtorja: Robert Klinc, Matevž Dolenc

Recenzenti: Tomislav Levanič, Jaka Dujc, Anja Brelih
Jezikovni pregled: Mojca Vilfan
Oblikovanje in prelom: Robert Klinc
Naslovnica: Gašper Mrak
Naslovna slika: Microsoft Copilot

Založnik: Založba Univerze v Ljubljani
(University of Ljubljana Press)
Za založbo: Gregor Majdič, rektor Univerze v Ljubljani
Izdajatelj: Univerza v Ljubljani,
Fakulteta za gradbeništvo in geodezijo
Za izdajatelja: Violeta Bokan Bosiljkov,
dekanja Fakultete za gradbeništvo in geodezijo

Ljubljana, 2024
Prva e-izdaja.
Publikacija je brezplačna.

Publikacija je v digitalni obliki prosto dostopna na: https://ebooks.uni-lj.si
DOI: 10.15292/9789612974275

Kataložni zapis o publikaciji (CIP) pripravili v
Narodni in univerzitetni knjižnici v Ljubljani

COBISS.SI-ID 212389891
ISBN 978-961-297-427-5 (PDF)

Kazalo

Predgovor

> Kar slišim, pozabim. Kar vidim, si zapomnim. Kar naredim, razumem.
>
> — kitajski pregovor

Ne glede na to, da v času nastajanja te zbirke knjig, sploh pa na papirju, pregovorno nihče več ne bere, sva avtorja ugotovila, da je kljub preobilju informacij podobnega čtiva, namenjenega inženirjem, zelo malo. Na to so naju konec koncev opozarjali tudi študenti, ki so si podobne zbirke najbolj želeli. In zato je tudi nastala. Pred vami je tako zbirka nalog za učenje programiranja v programskem jeziku Python. Zbirka je bila pripravljena v pomoč vsem, ki bi se želeli naučiti programirati, še posebej pa študentom pri predmetu Računalništvo in informatika na Fakulteti za gradbeništvo in geodezijo Univerze v Ljubljani.

Programiranje je zanimiv izziv in zahteva vsaj nekaj algoritmičnega razmišljanja, ki se ga ne moremo naučiti s ponavljanjem za drugimi ali pa zgolj z branjem knjig. Programiranja se naučimo s samostojnim reševanjem nalog. S tipkovnico!

Pred vami je torej zbirka osnovnih nalog z rešitvami, napisanimi v programskem jeziku Python, ki vas bo popeljala od samih začetkov do točke, ko boste lahko začeli pridobljeno znanje uporabljati tudi za reševanje svojih nalog in izboljšanje svojih delovnih procesov.

V tem učbeniku oziroma zbirki nalog ne boste našli čisto vseh osnov in v njej ne bodo predstavljeni čisto vsi koncepti, bo pa to dober priročnik za vse, ki se boste želeli priučiti programiranja ali pa svoje znanje programiranja zgolj preveriti oziroma nadgraditi. Ugotovili boste (če morda še ne veste), da se lahko reševanja lotite na (skoraj) neomejeno število načinov. Eni so bolj učinkoviti, drugi so bolj berljivi, tretji so sintaktično bolj pravilni … Ustrezni so (skoraj) vsi, samo da vrnejo pričakovane rezultate. Vsako predstavljeno rešitev naloge torej vzemite kot eno od možnosti in nikakor ne kot edino pravilno!

Sedaj pa veselo na delo.

Srečno!

Robert Klinc in ***Matevž Dolenc***

Napotki za uporabo

Zbirka je razdeljena na dva dela:

- v prvem delu so opisane naloge z bistvenimi namigi ter povezavami do rešitev,
- v drugem delu so predstavljene nekatere rešitve.

Vse rešitve so opremljene z bistvenimi pojasnili. Pri določenih nalogah pa je predstavljenih več možnih rešitev, vse z namenom, da bi bila pot do rešitev čim bolj jasna in razumljiva.

Vsa izvorna koda je objavljena na spletnem mestu GitHub [https://github.com/pzi-si/pzi-src-2/tree/main].

Vse rešitve so prilagojene osnovni stopnji programiranja in bi morale biti razumljive tudi začetnikom.

Predlagava, da si najprej izberete programerski izziv v prvem delu, si ogledate namige in poskusite sami napisati rešitev. Šele ko nalogo rešite, si oglejte tudi predlagane rešitve ter razlike med posameznimi različicami, če obstajajo. V pomoč pri razumevanju so vam lahko tudi bistvena pojasnila pri vsaki od rešitev.

Želiva vam uspešno programiranje.

Poglavje 1. Naloge

1.1. Množenje števil

Pripravite program, v katerem dvema spremenljivkama določite vrednost, nato pa uvedete tretjo, ki je zmnožek prejšnjih dveh. Tretjo spremenljivko izpišete na zaslon.

*Primer: zmnozek = mnozenec * mnozitelj, izpišete spremenljivko* ***zmnozek****.*

Nalogo rešimo tako, da uvedemo novi spremenljivki, jima določimo vrednost, nato pa uvedemo tretjo, ki je zmnožek prejšnjih dveh. Rezultat izpišemo na zaslon s pomočjo funkcije *print()*.

Primer rešitve: Izvorna koda 1.

1.2. Pozdrav

Pripravite program, ki vas bo vprašal po vašem imenu, nato pa vas bo pozdravil.

Primer: program vas vpraša ***Kako vam je ime?*** *in vas po vpisu imena* ***Robert*** *pozdravi s* ***Pozdravljen_a, Robert!****.*

Za reševanje uporabimo funkcijo *input()*, ki nas vpraša po imenu in naš vpis dodeli določeni spremenljivki kot niz znakov (*string*). Ta niz znakov lahko potem združimo s poljubnim besedilom in s pomočjo funkcije *print()* izpišemo na zaslon.

Primer rešitve: Izvorna koda 2.

1.3. Izračun ploščine in obsega kroga

Pripravite program, ki vam bo za podani polmer izračunal ploščino in obseg kroga ter vam izpisal rezultat.

V tem primeru imamo več možnosti reševanja:

- direktno računanje ploščine in obsega s približno vrednostjo konstante π
- uporaba natančnejše vrednosti π (uporaba konstante *math.pi*)
- uporaba alternativnega načina potenciranja (uporaba **)
- uporaba alternativnega načina potenciranja (uporaba funkcije *math.pow()*)
- rešitev z zaokroževanjem (uporaba funkcije *round()*)
- rešitev z interaktivnim vnosom polmera r (uporaba funkcije *input()*)
- rešitev z interaktivnim vnosom polmera r in kontrolo vnosa (uporaba *try/except*)

1.4. Seštevanje števil

Pripravite program, ki vas bo pozval, da vpišete števili, nato pa ju bo seštel in vam vrnil rezultat.

Tukaj je treba uporabnika pozvati k vpisu dveh števil s pomočjo funkcije *input()*, nato pa je treba biti pozoren na podatkovni tip:

- rešitev brez spremembe podatkovnega tipa (napačna)
- rešitev s spremembo podatkovnega tipa (pravilna)
- rešitev s spremembo podatkovnega tipa - drugačen izpis
- rešitev s spremembo podatkovnega tipa - drugačen izpis #2
- končna rešitev s preverjanjem vnosov (uporaba try/except)

1.5. Izračun ploščine in obsega pravokotnika

Pripravite program, ki vas bo pozval, da vpišete dolžini stranic pravokotnika, nato pa bo izračunal ploščino in obseg pravokotnika ter vam vrnil rezultata.

Tukaj je treba uporabnika pozvati k vpisu dolžin stranic s pomočjo

funkcije *input()*:

- rešitev brez preverjanja vnosov
- rešitev s preverjanjem vnosov

1.6. Pretvarjanje med Celzijevimi in Fahrenheitovimi stopinjami

Napišite program, ki mu uporabnik vpiše temperaturo v Fahrenheitovih stopinjah, program pa jo izpiše v Celzijevih (°F → °C).

Pripravite še program, ki računa v nasprotni smeri (°C → °F).

V tem primeru spoznamo uporabo pogojnih stavkov. Pomembni sta formuli za pretvorbo med °C in °F:

- formula za izračun °F → °C: T(°C) = (T(°F) - 32) / 1,8
- formula za izračun °C → °F: T(°F) = 1,8 * T(°C) + 32

Pripravimo lahko tri programe:

- rešitev za pretvorbo Fahrenheitovih stopinj v Celzijeve stopinje
- rešitev za pretvorbo Celzijevih stopinj v Fahrenheitove stopinje
- rešitev s pretvorbo v katero koli smer

1.7. Preverjanje, ali je podano število popoln kvadrat

Napišite program, ki mu podate število, program pa izpiše, ali gre za popoln kvadrat ali ne.

(*Popoln kvadrat je v matematiki pozitivno celo število, ki se ga lahko zapiše kot kvadrat drugega celega števila.*)

Kako lahko preverimo, če je število popoln kvadrat? Tako da preverimo, če je koren tega števila celo število! Pri tem bomo spoznali novi funkciji *is_integer()* in *int()*.

Nalogo lahko rešimo na več načinov:

- s pomočjo modula *math* in funkcije *math.sqrt()*
- brez pomoči modula *math*
- s kontrolami vnosa
- ...

1.8. Delitelji podanega števila

Napišite program, ki mu podate število, program pa sam poišče in izpiše vse delitelje podanega števila. Vsak delitelj naj bo izpisan zgolj enkrat, delitelji pa naj bodo urejeni po velikosti.

Za rešitev te naloge se moramo spomniti, da je za delitelja nekega števila značilno, da je ostanek pri deljenju števila z deliteljem enak 0. Na tem lahko gradimo. Za reševanje bomo uporabili zanko *while*, lahko pa bi nalogo rešili tudi z zanko *for*.

Nalogo lahko rešimo tako, da:

- izpišemo vsak delitelj v svojo vrstico
- izpišemo vsak delitelj v svojo vrstico, a iščemo za absolutno vrednost vnosa
- izpišemo delitelje po vrsti v eno vrstico
- ...

1.9. Praštevilo

Napišite program, ki za podano število pove, ali je praštevilo.

Praštevilo je naravno število (n > 1), ki ima točno dva pozitivna delitelja (faktorja), število 1 in samega sebe.

Naloga je logično nadaljevanje naloge 8. Lahko jo rešimo tako, da preverjamo, koliko je deliteljev nekega števila. Če sta zgolj 2 (1 in število, ki ga preverjamo), smo našli praštevilo. Lahko pa preverimo vsa števila razen 1 in števila, ki ga preverjamo, in če najdemo še en delitelj, vemo, da število ni praštevilo, v nasprotnem primeru pa je.

1.10. Prestopno leto

Napišite program, ki za vpisano letnico pove, ali je (bilo/bo) leto prestopno.

Pravilo za določanje prestopnih let je:

- Leto je prestopno, če je deljivo s 4, razen ...
- ... v primeru, da je leto deljivo s 100, leto ni prestopno, razen ...
- ... v primeru, da je leto deljivo s 400, leto je prestopno.

Zdaj že vidimo, da bo naša rešitev vsebovala nekaj pogojnih stavkov.

1.11. Fibonaccijevo zaporedje

Napiši program, ki izpiše prvih *X* členov Fibonaccijevega zaporedja. Število *X* na poziv vnese uporabnik.

Fibonaccijevo zaporedje se začne s številoma 1, 1, vsak naslednji člen pa dobimo tako, da seštejemo prejšnja dva. 1 in 1 je 2, 1 in 2 je 3, 2 in 3 je 5, 3 in 5 je 8 in tako naprej. Zaporedje se tako začne z 1 1 2 3 5 8 13 21 34 55.

Rešitev bo morala vsebovati pristop k prirejanju spremenljivk:

- rešitev z uvedbo vmesne spremenljivke,
- rešitev z direktnim prirejanjem.

1.12. Računanje statistike

Napišite program, ki od uporabnika bere števila, dokler uporabnik ne vpiše števila 0.

Program naj sproti izpisuje največje vpisano število, najmanjše vpisano število ter povprečje vpisanih števil.

Nalogo rešimo s pomočjo zapisovanja vpisanih števil v polje.

Primer rešitve: naloga 12.

1.13. Seštevanje števk naravnega števila

Sestavite program, ki prebere naravno število in sešteje vse njegove števke, kar ponavlja, dokler ne pride do enomestnega števila. Program naj izpisuje vse vmesne rezultate.

Rešitvi

- z direktnim seštevanjem,
- z uporabo definirane funkcije.

1.14. Pravilno sklanjanje

Sestavite program, ki prebere število in v pravilni slovenščini izpiše, koliko opravljenih izpitov imate.

Težava je s sklanjanjem v slovenščini, saj opravimo 1 izpit, 2 izpita in 3 (ali 4) izpite ter 5 (ali več) izpitov. Pozorni pa moramo biti tudi pri številih, večjih od 100! Opravimo namreč 101 (201, 301, ...) izpit, 102 izpita in 103 (ali 104) izpite ter 105 (ali več) izpitov.

Rešitev naloge: naloga 14.

1.15. Fakulteta števila

Sestavi program, ki izračuna fakulteto vnesenega števila.

Rešitve

- z računanjem in prikazom vmesnih korakov,
- z uporabo definirane funkcije in prikazom vmesnih korakov,
- z uporabo vgrajene funkcije za izračun fakultete (in brez prikaza vmesnih korakov).

1.16. Risanje smrečice

Napiši program, ki od uporabnika dobi naravno število, nato pa nariše *smrečico* z znakom *.

Smrečica naj ima toliko vrstic, kolikor jih uporabnik definira z vnosom naravnega števila.

Rešitve

- smrečica, poravnana levo,
- smrečica, poravnana desno,
- smrečica, poravnana sredinsko.

1.17. Obrat naravnega števila

Sestavi program, ki prebere naravno število in ga izpiše v obratnem vrstnem redu.

Rešitve

- splošna rešitev,
- rešitev samo za cela števila,

- rešitev samo za naravna števila,
- rešitev za naravna števila (krajši način).

1.18. Je besedilo palindrom?

Sestavi program, v katerem uporabnik vpiše niz znakov, program pa pove, ali je vpisani niz znakov palindrom ali ne.

Pomoč: Palindrom je niz znakov, ki se enako prebere z leve in z desne.

Rešitvi

- (polovična) rešitev,
- (končna) rešitev.

1.19. Največji skupni delitelj podanih števil

Sestavi program, ki izračuna največji skupni delitelj števil a in b in izpiše vse vmesne rezultate Evklidovega algoritma.

Pomoč: Evklidov algoritem [https://sl.wikipedia.org/wiki/Evklidov_algoritem]

Rešitvi

- rešitev,
- rešitev za več kot 2 podani števili.

1.20. Ugibanje celega števila med 0 in 100

Sestavi program, s katerim ugibamo celo število med 0 in 100.

Program naj si naključno izbere število med 0 in 100, uporabnik pa naj v čim manj poskusih ugane, katero število je izbral.

Rešitvi

- rešitev brez pomoči,
- rešitev s pomočjo.

1.21. Ali trikotnik obstaja?

Sestavi program, ki bo prebral tri realna števila, nato pa preveril, ali obstaja trikotnik s takimi dolžinami stranic. Če obstaja, naj program izračuna njegovo ploščino in obseg.

Namig: Heronova formula [https://sl.wikipedia.org/wiki/Heronova_formula]

Rešitvi

- rešitev z vnosom vsake stranice posebej,
- rešitev z enkratnim vnosom.

Poglavje 2. Rešitve

2.1. Množenje števil

Pripravite program, v katerem dvema spremenljivkama določite vrednost, nato pa uvedete tretjo, ki je zmnožek prejšnjih dveh. Tretjo spremeljivko izpišete na zaslon.

*Primer: mnozenec, mnozitelj, zmnozek = mnozenec * mnozitelj, izpišete spremenljivko* ***zmnozek****.*

2.1.1. Rešitev

Izvorna koda 1. naloga001.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga001.py]

```
 1 #!/usr/bin/env python
 2 # -*- coding: utf-8 -*-
 3
 4 """ Program, ki zmnoži podani števili. """
 5
 6 # Podamo številki, ki ju bomo zmnožili
 7 mnozenec = 2
 8 mnozitelj = 4
 9
10 # zmnožimo števili
11 zmnozek = mnozenec * mnozitelj
12
13 # izpišemo vsoto
14 print(zmnozek)
15 #print(f"Rezultat: {mnozenec} * {mnozitelj} = {zmnozek}")
```

Če program zaženete, bo program izpisal zgolj rešitev. Če pa odstranite znak # v vrstici 15, bo izpis podoben spodnjemu.

```
Rezultat: 2 * 4 = 8
```

Tabela 1. Razlaga

vrstica	komentar
1	Dobra praksa je, da vsako datoteko začnemo z navedbo, da gre za program, napisan v programskem jeziku Python. Tako lahko v tej vrstici najdemo deklaracijo, ki se ne spreminja in je lahko za vsako datoteko enaka. V računalništvu se shebang (imenovan tudi hashbang, hashpling, pound bang ali crunchbang) nanaša na znaka "#!", kadar sta to prva dva znaka v ukazu za tolmača kot prva vrstica besedilne datoteke. V operacijskem sistemu, podobnem Unixu, nalagalnik programa prisotnost teh dveh znakov razume kot znak, da je datoteka skripta, in poskuša to skripto izvesti z uporabo tolmača, določenega v preostanku prve vrstice v datoteki. V našem primeru gre za tolmača oziroma prevajalnik, ki prevaja ukaze, zapisane v programskem jeziku Python.
2	V tej vrstici so navodila, ki aplikaciji, v kateri ste odprli datoteko s programsko kodo, pove, da mora znake v datoteki interpretirati s kodno tabelo UTF-8.
4	S tremi znaki " označujemo večvrstični niz znakov. Če se nahaja na začetku vrstice, Python vse, kar je vmes, interpretira kot komentar in tega ne izvaja.
6	Z znakom # se začenjajo enovrstični komentarji. Vse, kar se nahaja desno od znaka #, Python obravnava kot komentar.
7	Tukaj uvedemo spremenljivko *mnozenec*, ki ji dodelimo vrednost 2.
8	Tukaj uvedemo spremenljivko *mnozitelj*, ki ji dodelimo vrednost 4.
11	Uvedemo spremenljivko *zmnozek*, za katero določimo, da je zmnožek spremenljivk *mnozenec* in *mnozitelj*.
14	Vrednost spremenljivke *zmnozek* izpišemo na zaslon z uporabo funkcije ***print***.
15	Če želite lepši izpis, lahko rezultat zapišete v kompleksnejši obliki. Če ste pozorni, je v izvorni kodi vrstica komentirana. Če želite, da se izpiše, je treba znak # odstraniti.

2.2. Pozdrav

Pripravite program, ki vas bo vprašal po vašem imenu, nato pa vas bo pozdravil.

*Primer: program vas vpraša **Kako vam je ime?** in vas po vpisu imena **Robert** pozdravi s **Pozdravljen_a, Robert!**.*

2.2.1. Rešitev

Izvorna koda 2. naloga002.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga002.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """ Program, ki vas vpraša po imenu, nato pa vas pozdravi. """
5
6 # povprašamo po imenu
7 ime = input("Kako ti je ime? ")
8
9 # pozdravimo
10 print(f"Pozdravljen_a, {ime}!")
```

Tabela 2. Razlaga

vrstica	komentar
7	Da nam lahko uporabnik vnese vrednost, ki jo priredimo spremenljivki ime, uporabimo vgrajeno funkcijo *input()*. Funkcija nam vrne vrednost tipa *string* (niz znakov).
10	Izpišemo pozdrav.

Izpis programa bo podoben naslednjemu.

```
Kako ti je ime? Herman Celjski
Pozdravljen_a, Herman Celjski!
```

2.3. Izračun ploščine in obsega kroga

Pripravite program, ki vam bo za podani polmer izračunal ploščino in obseg kroga ter vam izpisal rezultat.

2.3.1. Rešitev #1

Izvorna koda 3. naloga003a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga003a.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """ Program za računanje ploščine in obsega kroga. """
5
6 # določimo polmer
7 r = 5
8
9 # izračunamo ploščino kroga
10 ploscina = 3.14 * r * r
11
12 # izračunamo obseg kroga
13 obseg =  2 * 3.14 * r
14
15 # izpišemo ploščino in obseg kroga
16 print(f"Ploščina kroga s polmerom {r} je {ploscina}.")
17 print(f"Obseg kroga s polmerom {r} je {obseg}.")
```

Tabela 3. Razlaga

vrstica	komentar
4	S tremi znaki " označujemo večvrstične komentarje. Vse, kar je vmes, Python interpretira kot komentar in tega ne izvaja.
6	Z znakom # se začenjajo enovrstični komentarji. Vse, kar se nahaja desno od znaka #, Python obravnava kot komentar.
7	Tukaj uvedemo spremenljivko *r* in ji dodelimo vrednost 5.

vrstica	komentar
10	Uvedemo spremenljivko *ploscina*, ki ji dodelimo ploščino, izračunano po formuli $\pi * r^2$.
13	Uvedemo spremenljivko *obseg*, ki ga izračunamo po formuli $2 * \pi * r$.
16 in 17	Izpišemo rezultata.

Če program zaženete, bo izpisal rešitev v naslednji obliki.

```
Ploščina kroga s polmerom 5 je 78.5.
Obseg kroga s polmerom 5 je 31.400000000000002.
```

Opazite lahko, da je vrednost spremenljivke *obseg* drugačna, kot bi pričakovali. Z vašim programom ni prav nič narobe! To se zgodi zaradi numerične napake samega programskega jezika. Vrednosti s plavajočo decimalno vejico (tip *float*) na splošno nimajo natančne dvojiške predstavitve. To je posledica načina, kako centralna procesna enota (CPU) računalnika obdeluje podatke s plavajočo vejico (decimalna številka). Zaradi tega lahko pride do določene izgube natančnosti in nekatere operacije s plavajočo vejico lahko dajo nepričakovane rezultate.

Takšno obnašanje je lahko posledica enega od naslednjih dejavnikov:

- Dvojiška (binarna) predstavitev decimalnega števila morda ni natančna.
- Med uporabljenimi števili prihaja do neujemanja tipov (na primer uporaba števil tipa *float* in *integer*).

Več o samem fenomenu si lahko preberete tukaj [https://docs.python.org/3/tutorial/floatingpoint.html].

Pozor

Sam program lahko seveda precej izboljšamo:

- Za potenciranje lahko uporabimo drugačen način zapisa.
- Namesto približka vrednosti π lahko uporabimo natančnejšo vrednost.
- Vrednosti lahko zaokrožimo.
- Polmer lahko podamo interaktivno.

2.3.2. Rešitev #2: uporaba natančnejše vrednosti π

Izvorna koda 4. naloga003b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga003b.py]

```
 1 #!/usr/bin/env python
 2 # -*- coding: utf-8 -*-
 3
 4 """ Program za računanje ploščine in obsega kroga. """
 5
 6 # vključimo modul math
 7 import math
 8
 9 # določimo polmer
10 r = 5
11
12 # izračunamo ploščino kroga
13 ploscina = math.pi * r * r
14
15 # izračunamo obseg kroga
16 obseg =  2 * math.pi * r
17
18 # izpišemo ploščino in obseg kroga
19 print(f"Ploščina kroga s polmerom {r} je {ploscina}.")
20 print(f"Obseg kroga s polmerom {r} je {obseg}.")
```

Če primerjamo **rešitev 1** in **rešitev 2**, opazimo, da smo v drugem poskusu dodali vrstici **6** in **7** ter spremenili vrstici **13** in **16**. V vrstici **7** smo vključili Python modul *math*, ki nam (poleg ostalega) omogoča uporabo matematičnih konstant. Brez te vrstice ne bi mogli uporabiti matematične konstante *math.pi*, ki nam da natančnejšo vrednost π in boljši končni rezultat.

Program sedaj vrne bolj natančen rezultat.

```
Ploščina kroga s polmerom 5 je 78.53981633974483.
Obseg kroga s polmerom 5 je 31.41592653589793.
```

Še vedno pa so možne izboljšave (vsaj pri zakroževanju, saj nas ne zanima toliko decimalk).

2.3.3. Rešitev #3: uporaba alternativnega načina potenciranja

Izvorna koda 5. naloga003c.py *[https://github.com/pzi-si/pzi-src-2/blob/main/naloga003c.py]*

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """ Program za računanje ploščine in obsega kroga. """
5
6 # vključimo modul math
7 import math
8
9 # določimo polmer
10 r = 5
11
12 # izračunamo ploščino kroga
13 ploscina = math.pi * r**2
14
15 # izračunamo obseg kroga
16 obseg =  2 * math.pi * r
17
18 # izpišemo ploščino in obseg kroga
19 print(f"Ploščina kroga s polmerom {r} je {ploscina}.")
20 print(f"Obseg kroga s polmerom {r} je {obseg}.")
```

Če primerjamo **rešitev 2** in **rešitev 3** opazimo, da smo tokrat spremenili vrstico **13**, kjer smo namesto množenja *r * r* uporabili zapis *r**2*, kar je v programskem jeziku Python enakovredno zapisu r^2. Zapisa sta enakovredna in izpis je identičen.

2.3.4. Rešitev #4: uporaba alternativnega načina potenciranja #2

Ker smo že v **rešitvi 2** vključili modul *math*, lahko za potenciranje uporabimo tudi njegovo vgrajeno funkcijo *math.pow()*. Tako lahko v **vrstici 13** polmer *r* potenciramo v obliki *math.pow(r,2)*. Funkcija *math.pow()* potrebuje 2 parametra: prvi je vrednost, ki jo potenciramo, drugi pa označuje potenco. Tudi ta oblika zapisa je enakovredna obema prejšnjima, tako da je izpis identičen.

Izvorna koda 6. naloga003d.py *[https://github.com/pzi-si/pzi-src-2/blob/main/naloga003d.py]*

```
1 #!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-

""" Program za računanje ploščine in obsega kroga. """

# vključimo modul math
import math

# določimo polmer
r = 5

# izračunamo ploščino kroga
ploscina = math.pi * math.pow(r,2)

# izračunamo obseg kroga
obseg =  2 * math.pi * r

# izpišemo ploščino in obseg kroga
print(f"Ploščina kroga s polmerom {r} je {ploscina}.")
print(f"Obseg kroga s polmerom {r} je {obseg}.")
```

2.3.5. Rešitev #5: zaokroževanje

Izvorna koda 7. naloga003e.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga003e.py]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" Program za računanje ploščine in obsega kroga. """

# vključimo modul math
import math

# določimo polmer
r = 5

# izračunamo ploščino kroga
ploscina = math.pi * math.pow(r,2)
ploscina = round(ploscina,2)

# izračunamo obseg kroga
```

```
17 obseg =  2 * math.pi * r
18 obseg = round(obseg,2)
19
20 # izpišemo ploščino in obseg kroga
21 print(f"Ploščina kroga s polmerom {r} je {ploscina}.")
22 print(f"Obseg kroga s polmerom {r} je {obseg}.")
```

Običajno ne potrebujemo tako natančnih vrednosti in jih želimo zakrožiti na manj decimalnih mest. V ta namen lahko uporabimo vgrajeno funkcijo *round*, kar smo naredili v vrsticah **14** in **18**. Funkcija *round()* zahteva dva parametra: prvi parameter je številčna vrednost, drugi pa želeno število decimalnih mest. Izpis se seveda spremeni.

```
Ploščina kroga s polmerom 5 je 78.54.
Obseg kroga s polmerom 5 je 31.42.
```

2.3.6. Rešitev #6: interaktivni vnos polmera r

Izvorna koda 8. naloga003f.py *[https://github.com/pzi-si/pzi-src-2/blob/main/naloga003f.py]*

```
 1 #!/usr/bin/env python
 2 # -*- coding: utf-8 -*-
 3
 4 """ Program za računanje ploščine in obsega kroga. """
 5
 6 # vključimo modul math
 7 import math
 8
 9 # določimo polmer
10 r = input("Vnesi polmer kroga: ")
11 # pretvorimo r v številko
12 r = float(r)
13
14 # izračunamo ploščino kroga
15 ploscina = math.pi * math.pow(r,2)
16 ploscina = round(ploscina,2)
17
18 # izračunamo obseg kroga
19 obseg =  2 * math.pi * r
20 obseg = round(obseg,2)
```

```
21
22 # izpišemo ploščino in obseg kroga
23 print(f"Ploščina kroga s polmerom {r} je {ploscina}.")
24 print(f"Obseg kroga s polmerom {r} je {obseg}.")
```

Da ne bi bilo treba vedno spreminjati programa, ko želimo izračun za drugačno vrednost polmera *r*, lahko program spremenimo tako, da nas interaktivno vpraša po vrednosti polmera. V ta namen uporabimo vgrajeno funkcijo *input*, ki uporabnika pozove k vnosu polmera. To smo storili v vrstici **10**. Pri tem je treba biti pozoren, saj vgrajena funkcija *input* vedno vrne spremenljivko tipa *string*. To pomeni, da moramo dobljeno spremenljivko spremeniti v enega od številskih tipov (*int*, *float*), saj nam bo v nasprotnem primeru program vrnil napako. To smo storili v vrstici **12**. Izpis je sedaj podoben spodnjemu.

```
Vnesi polmer kroga: 5
Ploščina kroga s polmerom 5.0 je 78.54.
Obseg kroga s polmerom 5.0 je 31.42.
```

Pozor

Kljub vsem spremembam pa program še vedno ni popoln, saj ne preverja vnosov. To lahko preverite tako, da takrat, ko vas pozove k vnosu polmera *r*, namesto številke vpišete črko. Program se bo v takšnem primeru ustavil z napako.

Primer izpisa, če vnesemo črko namesto številke.

```
Vnesi polmer kroga: r
Traceback (most recent call last):
  File "naloga003f.py", line 12, in <module>
    r = float(r)
        ^^^^^^^^
ValueError: could not convert string to float: 'r'
```

Seveda lahko pripravimo program, ki bo obravnaval tudi to izjemo.

2.3.7. Rešitev #6: interaktivni vnos polmera r s kontrolo vnosa

Izvorna koda 9. naloga003g.py *[https://github.com/pzi-si/pzi-src-2/blob/main/naloga003g.py]*

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """ Program za računanje ploščine in obsega kroga. """
5
6 # vključimo modul math
7 import math
8
9 # določimo polmer
10 r = input("Vnesi polmer kroga: ")
11
12 # poskusimo pretvoriti r v številko
13 try:
14     r = float(r)
15 except:
16     print(f"'{r}' ni številka!")
17     quit()
18
19 # izračunamo ploščino kroga
20 ploscina = math.pi * math.pow(r,2)
21 ploscina = round(ploscina,2)
22
23 # izračunamo obseg kroga
24 obseg =  2 * math.pi * r
25 obseg = round(obseg,2)
26
27 # izpišemo ploščino in obseg kroga
28 print(f"Ploščina kroga s polmerom {r} je {ploscina}.")
29 print(f"Obseg kroga s polmerom {r} je {obseg}.")
```

Razlika med Izvorna koda 8 in Izvorna koda 9 je zgolj v vrsticah **13-17**. Program v vrstici **14** poskusi pretvoriti vnos iz podatkovnega tipa *string* v tip *float*. Če mu to ne uspe, v vrstici **16** izpiše, da vnosna vrednost ne predstavlja številke in se v vrstici **17** ustavi. Če je z vnosom vse v redu, program deluje tako kot Izvorna koda 8.

Izpis v primeru napačnega vnosa je sedaj podoben spodnjemu.

```
Vnesi polmer kroga: r
'r' ni številka!
```

2.4. Seštevanje števil

Pripravite program, ki vas bo pozval, da vpišete števili, nato pa ju bo seštel in vam vrnil rezultat.

2.4.1. Rešitev #1

Izvorna koda 10. naloga004a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga004a.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """ Program, ki sešteje podani števili. """
5
6
7 # Izpišemo, za kakšen program gre
8 print("*"*40)
9 print("*"*5,"Program za seštevanje števil","*"*5)
10 print("*"*40)
11
12 # Podamo številki, ki ju seštevamo
13 sestevanec1 = input('Vnesite prvo število: ')
14 sestevanec2 = input('Vnesite drugo število: ')
15
16 # seštejemo števili
17 vsota = sestevanec1 + sestevanec2
18
19 # izpišemo vsoto
20 print(f"Rezultat: {sestevanec1} + {sestevanec2} = {vsota}")
```

Tabela 4. Razlaga

vrstica	komentar
7	Funkcijo *print()* že poznamo, velja pa spomniti, da sintaksa *"*" * 40* pomeni, da bomo niz znakov *"*"* ponovili 40-krat. Povedano drugače, izpis funkcije *print("RI"*3)* bi bil *RIRIRI*. Naša sintaksa zgoraj bo torej pomenila, da bomo izpisali 40 znakov *.
12	Uporabnika pozovemo, da vpiše prvo število, ki ga priredimo spremenljivki z imenom *sestevanec1*.
13	Uporabnika pozovemo, da vpiše drugo število, ki ga priredimo spremenljivki z imenom *sestevanec2*.
16	Uvedemo novo spremenljivko *vsota*, ki dobi vrednost seštevka spremenljivk *sestevanec1* in *sestevanec2*.
19	Izpišemo vsoto.

Če program zaženete, bo izpis podoben spodnjemu.

```
****************************************
***** Program za seštevanje števil *****
****************************************
Vnesite prvo število: 1
Vnesite drugo število: 2
Rezultat: 1 + 2 = 12
```

Zakaj ne sešteje vrednosti? Zato, ker nam Python funkcija *input* vrne niz znakov (vrednost tipa *string*) in ne enega od številskih tipov (*integer*, *float* ...). S seštevanjem dveh vrednosti v 16. vrstici kode (Izvorna koda 10) smo tako sestavili dva niza znakov (postavili enega k drugemu) in ne sešteli. To se zgodi tudi, če vpišemo številke, saj funkcija input v vsakem primeru vrne vrednost tipa *string*.

Rešitev je pretvorba podatkovnega tipa *string* v enega od številskih tipov (*integer* ali *float*). Če pretvorimo v *integer*, bo program deloval, a zgolj za cela števila, pri vnosu decimalnih števil bo vrnil napako, zato bomo pretvorili v številski tip *float*.

2.4.2. Rešitev #2

Razlika med Izvorna koda 10 in Izvorna koda 11 je zgolj v vrsticah 16 in 17, kjer smo

spremenili tipa spremenljivk *sestevanec1* in *sestevanec2* iz *string* v *float*. Vse ostalo ostane enako.

Izvorna koda 11. naloga004b.py *[https://github.com/pzi-si/pzi-src-2/blob/main/naloga004b.py]*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" Program, ki sešteje podani števili. """


# Izpišemo, za kakšen program gre
print("*"*40)
print("*"*5,"Program za seštevanje števil","*"*5)
print("*"*40)

# Podamo številki, ki ju seštevamo
sestevanec1 = input('Vnesite prvo število: ')
sestevanec2 = input('Vnesite drugo število: ')

# spremenimo tip spremenljivke iz 'string' v 'float'
sestevanec1 = float(sestevanec1)
sestevanec2 = float(sestevanec2)

# seštejemo števili
vsota = sestevanec1 + sestevanec2

# izpišemo vsoto
print(f"Rezultat: {sestevanec1} + {sestevanec2} = {vsota}")
```

Program po tej spremembi sešteje obe podani števili.

```
****************************************
***** Program za seštevanje števil *****
****************************************
Vnesite prvo število: 1
Vnesite drugo število: 2
Rezultat: 1.0 + 2.0 = 3.0
```

2.4.3. Rešitev #3

Izvorna koda 12. naloga004c.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga004c.py]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" Program, ki sešteje podani števili. """


# Izpišemo, za kakšen program gre
print("*"*40)
print("*"*5,"Program za seštevanje števil","*"*5)
print("*"*40)

# Podamo številki, ki ju seštevamo
sestevanec1 = input('Vnesite prvo število: ')
sestevanec2 = input('Vnesite drugo število: ')

# spremenimo tip spremenljivke iz 'string' v 'float'
sestevanec1 = float(sestevanec1)
sestevanec2 = float(sestevanec2)

# seštejemo števili
vsota = sestevanec1 + sestevanec2

# izpišemo vsoto
print("Rezultat: {} + {} = {}".format(sestevanec1,sestevanec2
,vsota))
```

Drugačna je zgolj zadnja vrstica kode.

2.4.4. Rešitev #4

Izvorna koda 13. naloga004d.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga004d.py]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" Program, ki sešteje podani števili. """
```

```
 5
 6
 7 # Izpišemo, za kakšen program gre
 8 print("*"*40)
 9 print("*"*5,"Program za seštevanje števil","*"*5)
10 print("*"*40)
11
12 # Podamo številki, ki ju seštevamo
13 sestevanec1 = input('Vnesite prvo število: ')
14 sestevanec2 = input('Vnesite drugo število: ')
15
16 # spremenimo tip spremenljivke iz 'string' v 'float'
17 sestevanec1 = float(sestevanec1)
18 sestevanec2 = float(sestevanec2)
19
20 # seštejemo števili
21 vsota = sestevanec1 + sestevanec2
22
23 # izpišemo vsoto
24 print("Rezultat: "+str(sestevanec1)+" + "+str(sestevanec2)+" = "
   +str(vsota))
```

Tako primer Izvorna koda 12 kot tudi Izvorna koda 13 se od Izvorna koda 11 razlikujeta zgolj v vrstici 23, kjer s pomočjo funkcije *print* izpišemo niz znakov (*string*), ki ga vsakič pripravimo drugače. Vsi načini so enakovredni, gre le za berljivost izvorne kode. Bodite pozorni na primer Izvorna koda 13, kjer v omenjeni vrstici uporabimo tudi funkcijo *str*, ki spremeljivko tipa *float* spremeni v spremenljivko tipa *string*! To storimo zato, ker je treba pri programiranju operirati s spremenljivkami istega tipa in v tem primeru za izpis združujemo nize znakov.

V svojih programih lahko uporabljate kateri koli način. Izpis vseh treh programov je enak.

Pozor

Kljub tej spremembi program še vedno ne deluje pravilno v vseh primerih, saj ne preverja vnosov. Da je res tako, lahko preverite tudi sami, tako da namesto številk vnesete črke ali pa decimalne številke zapišete z vejico. Program se bo v takšnem primeru ustavil z napako. Seveda lahko zajamemo tudi primere napačnih vnosov.

2.4.5. Rešitev #5: preverjanje vnosov

Izvorna koda 14. naloga004e.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga004e.py]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" Program, ki sešteje podani števili. """

# Izpišemo, za kakšen program gre
print("*"*40)
print("*"*5,"Program za seštevanje števil","*"*5)
print("*"*40)

# Podamo številki, ki ju seštevamo
sestevanec1 = input('Vnesite prvo število: ')

# poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
try:
    sestevanec1 = float(sestevanec1)
except:
    print(f"Prva podana vrednost ni število. Podali ste {sestevanec1}.")
    quit()

sestevanec2 = input('Vnesite drugo število: ')

# poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
try:
    sestevanec2 = float(sestevanec2)
except:
    print(f"Druga podana vrednost ni število. Podali ste {sestevanec2}.")
    quit()

# seštejemo števili
vsota = sestevanec1 + sestevanec2

# izpišemo vsoto
print(f"Rezultat: {sestevanec1} + {sestevanec2} = {vsota}")
```

V tej rešitvi preverjamo tudi uporabnikove vnose. Vse prejšnje rešitve (Izvorna koda 10, Izvorna koda 11, Izvorna koda 12 in Izvorna koda 13) namreč odpovedo, če uporabnik namesto številke vnese kakšen drug znak. Zato smo v tej rešitvi uvedli preverjanje izjem. Tako v **vrstici 15** povemo, da bomo v **vrstici 16** poskusili spremenljivko *sestevanec1* pretvoriti iz tipa *string* v tip *float*:

- Če bo pretvorba uspešna, se bo naš program izvajal od vrstice 20 dalje (izpustil bo del kode, ki se nahaja pod izjemo (*except*)).
- Če pretvorba ne bo uspešna, bo izvedel kodo, ki se nahaja v bloku *except* (izvedel bo kodo v **vrstici 18** ter **vrstici 19**). Najprej bo torej izpisal obvestilo, da podana vrednost ni število, takoj zatem pa bo prekinil izvajanje programa (v ta namen smo uporabili funkcijo *quit()*, ki prekine izvajanje).

Preverjanje ponovimo tudi za vnos druge številke (vrstice **24-28**).

2.5. Izračun ploščine in obsega pravokotnika

Pripravite program, ki vas bo pozval, da vpišete dolžini stranic pravokotnika, nato pa bo izračunal ploščino in obseg pravokotnika ter vam vrnil rezultata.

2.5.1. Rešitev #1: računanje brez preverjanja vnosov

Izvorna koda 15. naloga005a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga005a.py]

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """ Program za računanje ploščine in obsega pravokotnika. """
5
6  # Izpišemo, za kakšen program gre
7  print("*"*64)
8  print("*"*5,"Program za računanje ploščine in obsega pravokotnika
   ","*"*5)
9  print("*"*64)
10
11 # Podamo stranici pravokotnika
12 stranicaA = input('Vnesite dolžino stranice a: ')
13 stranicaB = input('Vnesite dolžino stranice b: ')
14
15 # poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
16 stranicaA = float(stranicaA)
17 stranicaB = float(stranicaB)
18
19 # izračun obsega in ploščine
20 obseg = 2*stranicaA + 2*stranicaB
21 ploscina = stranicaA * stranicaB
22
23 # izpišemo ploščino in obseg
24 print(f"Ploščina pravokotnika s stranicama {stranicaA} in
   {stranicaB} je {ploscina}.")
25 print(f"Obseg pravokotnika s stranicama {stranicaA} in {stranicaB}
   je {obseg}.")
```

Če program zaženete, bo izpis podoben spodnjemu.

```
****************************************************************
***** Program za računanje ploščine in obsega pravokotnika *****
****************************************************************
Vnesite dolžino stranice a: 2
Vnesite dolžino stranice b: 4
Ploščina pravokotnika s stranicama 2.0 in 4.0 je 8.0.
Obseg pravokotnika s stranicama 2.0 in 4.0 je 12.0.
```

Tabela 5. Razlaga

vrstica	komentar
12	Uporabnika pozovemo, da vpiše dolžino stranice a, vrednost pa priredimo spremenljivki z imenom *stranicaA*.
13	Uporabnika pozovemo, da vpiše dolžino stranice b, vrednost pa priredimo spremenljivki z imenom *stranicaB*.
16	Spremenljivko *stranicaA* s pomočjo vgrajene funkcije *float()* spremenimo iz podatkovnega tipa *string* v tip *float*.
17	Spremenljivko *stranicaB* s pomočjo vgrajene funkcije *float()* spremenimo iz podatkovnega tipa *string* v tip *float*.
20	Uvedemo novo spremenljivko *obseg* in ji dodelimo vrednost rezultata formule za izračun obsega pravokotnika.
21	Uvedemo novo spremenljivko *ploscina* in ji dodelimo vrednost rezultata formule za izračun ploščine pravokotnika.
24	Izpišemo rezultat.
25	Izpišemo rezultat.

Kljub tej spremembi program, tako kot tudi v primeru 2.4: seštevanje števil, še vedno ne deluje v vseh primerih povsem pravilno, saj ne preverja vnosov. Da je res tako, lahko preverite tudi sami, tako da namesto številk za dolžino stranice a ali b vnesete črke ali pa decimalne številke zapišete z vejico. Program se bo v takšnem primeru ustavil z napako.

Napaka v primeru vnosa črke namesto številke je podobna naslednji.

```
****************************************************************
***** Program za računanje ploščine in obsega pravokotnika *****
****************************************************************
Vnesite dolžino stranice a: r
Vnesite dolžino stranice b: k
Traceback (most recent call last):
  File "naloga005a.py", line 16, in <module>
    stranicaA = float(stranicaA)
                ^^^^^^^^^^^^^^^^
ValueError: could not convert string to float: 'r'
```

Seveda lahko zajamemo tudi primere napačnih vnosov.

2.5.2. Rešitev #2: računanje s preverjanjem vnosov

Izvorna koda 16. naloga005b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga005b.py]

```
 1 #!/usr/bin/env python
 2 # -*- coding: utf-8 -*-
 3
 4 """ Program za računanje ploščine in obsega pravokotnika. """
 5
 6 # Izpišemo, za kakšen program gre
 7 print("*"*64)
 8 print("*"*5,"Program za računanje ploščine in obsega pravokotnika
   ","*"*5)
 9 print("*"*64)
10
11 # Podamo stranici pravokotnika
12 stranicaA = input('Vnesite dolžino stranice a: ')
13
14 # poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
15 try:
16     stranicaA = float(stranicaA)
17 except:
18     print(f"Podali ste '{stranicaA}', kar ni število.")
19     quit()
```

```
20
21 stranicaB = input('Vnesite dolžino stranice b: ')
22
23 # poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
24 try:
25     stranicaB = float(stranicaB)
26 except:
27     print(f"Podali ste '{stranicaB}', kar ni število.")
28     quit()
29
30 # izračun obsega in ploščine
31 obseg = 2*stranicaA + 2*stranicaB
32 ploscina = stranicaA * stranicaB
33
34 # izpišemo ploščino in obseg
35 print(f"Ploščina pravokotnika s stranicama {stranicaA} in
   {stranicaB} je {ploscina}.")
36 print(f"Obseg pravokotnika s stranicama {stranicaA} in {stranicaB}
   je {obseg}.")
```

Razlika med Izvorna koda 15 in Izvorna koda 16 je zgolj v vrsticah **14-19** in **23-28**. Sedaj že vemo: program v vrsticah **16** in **25** poskusi pretvoriti vnos iz podatkovnega tipa *string* v tip *float*. Če mu to ne uspe, v vrsticah **18** in **27** izpiše, da tega ne more storiti, in se v vrstici **19** ali pa **28** ustavi. Če je z vnosom vse v redu, program deluje tako kot Izvorna koda 15.

Sedaj je izpis v primeru napačnega vnosa podoben naslednjemu.

```
**********************************************************************
***** Program za računanje ploščine in obsega pravokotnika *****
**********************************************************************
Vnesite dolžino stranice a: r
Podali ste 'r', kar ni število.
```

2.6. Pretvarjanje med Celzijevimi in Fahrenheitovimi stopinjami

Napišite program, ki mu uporabnik vpiše temperaturo v Fahrenheitovih stopinjah, program pa jo izpiše v Celzijevih (°F → °C). Med temperaturama pretvarjamo po formuli °C = 5/9 (°F – 32).

Pripravite še program, ki računa v nasprotni smeri (°C → °F).

2.6.1. Rešitev #1: °F → °C

Izvorna koda 17. naloga006a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga006a.py]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#T(°C) = (T(°F) - 32)/1,8
#T(°F) = 1,8 T(°C) + 32

# Izpišemo, za kakšen program gre
print("*"*73)
print("*"*5,"Program za pretvarjanje stopinj Fahrenheita v stopinje Celzija","*"*5)
print("*"*73)

# Podajte
temperaturaF = input('Vnesite °F: ')

# poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
try:
    temperaturaF = float(temperaturaF)
except:
    print(f"Podana vrednost ni število. Podali ste {temperaturaF}.")
    quit()

# pretvorimo F v C
temperaturaC = (temperaturaF - 32)*5/9
```

```
24
25 # zaokrožimo temperaturo v °C na 1 decimalko
26 temperaturaC = round(temperaturaC,1)
27
28 # izpišemo rezultat
29 print(f"{temperaturaF} °F = {temperaturaC} °C")
```

Če program zaženete, bo izpis podoben spodnjemu.

```
**********************************************************************
*****
***** Program za pretvarjanje stopinj Fahrenheita v stopinje Celzija
*****
**********************************************************************
*****
Vnesite °F: 56
56.0 °F = 13.3 °C
```

V programu ni nič novega, vse smo spoznali že v prejšnjih nalogah:

- Uporabnika najprej pozovemo, da vpiše temperaturo v °F (**vrstica 17**).
- Vnos (vrednost spremenljivke *temperaturaF*) poskusimo spremeniti iz tipa *string* (ki ga vrne funkcija *input()*) v tip *float*. Če pretvorba ni uspešna, program izpiše, da vpisana vrednost ni številka, in se prekine, v nasprotnem primeru pa se izvajanje nadaljuje. Vse to se zgodi v **vrsticah 20-24**.
- Če se izvajanje programa ni prekinilo (kar pomeni, da je bil vnos številčen), pretvorimo °F v °C (**vrstica 27**).
- V **vrstici 30** vrednost spremenljivke *temperaturaC* s pomočjo funkcije *round* zaokrožimo na eno decimalno mesto.
- Rezultat izpišemo (**vrstica 29**).

2.6.2. Rešitev #2: °C → °F

Izvorna koda 18. naloga006b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga006b.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
```

```
#T(°C) = (T(°F) - 32)/1,8
#T(°F) = 1,8 T(°C) + 32

# Izpišemo, za kakšen program gre
print("*"*73)
print("*"*5,"Program za pretvarjanje stopinj Celzija v stopinje Fahrenheita","*"*5)
print("*"*73)

# Podajte
temperaturaC = input('Vnesite °C: ')

# poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
try:
    temperaturaC = float(temperaturaC)
except:
    print(f"Podana vrednost ni število. Podali ste {temperaturaC}.")
    quit()

# pretvorimo F v C
temperaturaF = (temperaturaC)*1.8+32

# zaokrožimo C na 1 decimalko
temperaturaF = round(temperaturaF,1)

# izpišemo rezultat
print(f"{temperaturaC} °C = {temperaturaF} °F")
```

Izpis programa:

```
***************************************************************************
*****
***** Program za pretvarjanje stopinj Celzija v stopinje Fahrenheita *****
***************************************************************************
*****
Vnesite °C: 30
30.0 °C = 86.0 °F
```

2.6.3. Rešitev #3: Pretvorba v obe smeri

Seveda so možne tudi rešitve, kjer lahko v en program združimo pretvorbi v obe smeri - združimo Izvorna koda 17 in Izvorna koda 18 v skupen program.

Ena od možnih rešitev je naslednja.

Izvorna koda 19. naloga006c.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga006c.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 # Izpišemo, za kakšen program gre
5 print("*"*54)
6 print("*"*5,"Program za pretvarjanje °C v °F in obratno","*"*5)
7 print("*"*54)
8
9 # navodilo
10 print("Podajte vrednost, ki jo želite pretvoriti.")
11 print("Dodajte C, če je vrednost v °C, in F, če je vrednost v °F.
   Primer: '23 C'")
12
13 # uporabnik poda vrednost
14 vnos = input('Vnesite stopinje: ')
15
16 # odstranimo morebitne presledke na začetku in koncu vnosnega niza
   znakov
17 vnos = vnos.strip()
18
19 # preberemo zadnji znak
20 lestvica = vnos[-1]
21
22 # preberemo stopinje
23 temperatura = vnos[:-1]
24
25 # odstranimo morebitne presledke na začetku in koncu vnosnega niza
   znakov
26 temperatura = temperatura.strip()
27
28 # poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
29 try:
```

```
30      temperatura = float(temperatura)
31  except:
32      print(f"Podana vrednost ni število. Podali ste {vnos}.")
33      quit()
34
35  # preverimo, ali pretvarjamo °F v °C ali °C v °F
36  if lestvica == "C" or lestvica == "c":
37      # pretvorimo F v C
38      temperaturaF = (temperatura)*1.8+32
39
40      # izpišemo rezultat pretvorbe
41      print(f"{temperatura} °C = {temperaturaF} °F")
42  elif lestvica == "F" or lestvica == "f":
43      # pretvorimo F v C
44      temperaturaC = (temperatura - 32)*5/9
45
46      # zaokrožimo C na 1 decimalko
47      temperaturaC = round(temperaturaC,1)
48
49      # izpišemo rezultat
50      print(f"{temperatura} °F = {temperaturaC} °C")
51  else:
52      # ne pretvarjamo ne F in ne C
53      print("Niste določili, po kateri lestvici je vrednost, ki ste
    jo podali.")
```

Program je precej daljši, saj vsebuje kar nekaj kontrol in dodatne kode.

Tabela 6. Razlaga

vrstica	komentar
11	Navodilo uporabniku pove, da mora poleg temperature vpisati tudi enoto (C za °C in F za °F). Dodan je tudi primer vnosa.
17	V tej vrstici s pomočjo funkcije *strip()* iz spremenljivke *unos* odstranimo morebitne presledke pred in za nizom znakov. To pomeni, da če uporabnik namesto '30 C' vnese ' 30 C ', niz znakov spremenimo tako, da bo vrednost spremenljivke *unos* enaka '30 C'.

vrstica	komentar
20	Tukaj uvedemo spremenljivko *lestvica*, ki ji priredimo vrednost, ki je enaka zadnjemu znaku v nizu znakov (*string*) *vnos*. Z drugimi besedami: 'odrežemo' zadnji znak in ga priredimo spremenljivki *lestvica*. Če torej uporabnik vnese vrednost '30 C', bo po tej vrstici vrednost spremenljivke *lestvica* enaka 'C'.
23	V tej vrstici storimo podobno kot v vrstici **20**, le da uvedemo spremenljivko *temperatura* in ji priredimo vrednost, ki je enaka vsem znakom v nizu znakov (*string*) *vnos*, razen zadnjega. Z drugimi besedami: spremenljivki *vnos* odstranimo zadnji znak in ostalo priredimo spremenljivki *temperatura*. Če torej uporabnik vnese vrednost '30 C', bo po tej vrstici vrednost spremenljivke *temperatura* enaka '30 ' (bodite pozorni na presledek za številko).
26	Ker po vrstici **23** lahko ostanejo neželeni presledki, ki onemogočajo pretvorbo iz niza znakov (*string*) v decimalno število (*float*), jih moramo odstraniti. V ta namen ponovno uporabimo funkcijo *strip()*.
29-33	Poskusimo pretvoriti tip spremenljivke *temperatura* v decimalno število. Če nam ne uspe, izvajanje programa prekinemo.
36-41	Preverimo, če je vrednost spremenljivke *lestvica* enaka veliki ali mali črki 'C'. Če je, je uporabnik vnesel temperaturo v °C, zato jo pretvorimo v °F in rezultat izpišemo. V tem delu je koda enaka Izvorna koda 18.
42-50	Če vrednost spremenljivke *lestvica* ni enaka veliki ali mali črki 'C', je pa enaka veliki ali mali črki 'F', je uporabnik vnesel temperaturo v °F, zato pretvorimo v °C in rezultat izpišemo. V tem delu je koda enaka Izvorna koda 17.
51-53	Če se program izvede do sem, uporabnik ni vnesel predvidene oznake 'F' ali 'C' in se zato program brez pretvorbe konča.

Izpis je približno podoben naslednjemu.

```
******************************************************
***** Program za pretvarjanje °C v °F in obratno *****
******************************************************
Podajte vrednost, ki jo želite pretvoriti.
Dodajte C, če je vrednost v °C, in F, če je vrednost v °F. Primer:
'23 C'
Vnesite stopinje: 30   F
30.0 °F = -1.1 °C
```

Kot lahko opazite, zna naš program računati tudi v primeru, da je presledkov med številko in enoto več.

Če uporabnik ne vnese oznake lestvice (F ali C), se program konča na naslednji način.

```
******************************************************
***** Program za pretvarjanje °C v °F in obratno *****
******************************************************
Podajte vrednost, ki jo želite pretvoriti.
Dodajte C, če je vrednost v °C, in F, če je vrednost v °F. Primer:
'23 C'
Vnesite stopinje: 23
Niste določili, po kateri lestvici je vrednost, ki ste jo podali.
```

2.7. Preverjanje, ali je podano število popoln kvadrat

Napišite program, ki mu podate število, program pa izpiše, ali gre za popoln kvadrat ali ne.

2.7.1. Rešitev #1: s pomočjo modula *math* in funkcije *math.sqrt()*

Izvorna koda 20. naloga007a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga007a.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """ Program, ki preveri, ali je podano število popoln kvadrat. """
5
6 # Izpišemo, za kakšen program gre
7 print("*"*53)
8 print("*"*5,"Program za preverjanje popolnih kvadratov","*"*5)
9 print("*"*53)
10
11 # Uvozimo potrebne Python module
12 import math
13
14 # Uporabnika pozovemo, da vpiše število, ki ga bomo preverili
15 stevilo = input("Vpišite število: ")
16
17 # Lepotni izpis
18 print("-"*53)
19
20 # Izpišemo število, ki ga je uporabnik vnesel
21 print(f"Število, ki ga preverjamo, je: {stevilo}")
22
23 # Lepotni izpis
24 print("="*53)
25
26 # Spremenljivko število pretvorimo v tip 'float'
27 stevilo_stevilo = float(stevilo)
28
```

```
29 # Izračunamo koren podanega števila
30 koren = math.sqrt(stevilo_stevilo)
31
32 # Preverimo, če je izračunan koren tipa 'integer'
33 if koren.is_integer():
34     # ker vemo, da je koren tipa integer, ga pretvorimo v ta tip
35     koren = int(koren)
36     # izpišemo besedilo
37     print(f"Podano število {stevilo} je popoln kvadrat števila
   {koren}!")
38 else:
39     # izpišemo obvestilo, da podano število ni popoln kvadrat
40     print(f"Podano število {stevilo} ni popoln kvadrat.")
41 # Lepotni izpis
42 print("-"*53)
```

Posebnost programa je uporaba funkcije *math.sqrt()* v **vrstici 30**, ki vrne koren števila. Seveda je za to, da lahko funkcijo *math.sqrt()* sploh uporabimo, treba vključiti modul *math*, kar smo storili v **vrstici 12**.

Prav tako sta novi funkciji *is_integer()* ter *int()*. Funkcijo *is_integer()* smo uporabili v **vrstici 33**, da smo preverili, če je koren, izračunan v **vrstici 30**, celo število. Če je, potem imamo popoln kvadrat in lahko neznanko *stevilo* pretvorimo v številski tip *integer*, kar storimo v **vrstici 35** s funkcijo *int()*. To storimo zgolj zato, da nam program vrne celo število (kar je logično, če gre za popoln koren) in ne izpisuje decimalk.

Ostanejo nam samo še ustrezni izpisi.

Če program zaženete, bo izpis podoben spodnjemu.

```
*****************************************************
***** Program za preverjanje popolnih kvadratov *****
*****************************************************
Vpišite število: 9
-----------------------------------------------------
Število, ki ga preverjamo, je: 9
=====================================================
Podano število 9 je popoln kvadrat števila 3!
-----------------------------------------------------
```

2.7.2. Rešitev #2: brez pomoči modula *math*

Če ne želite vključiti modula *math*, lahko rešitev poiščete drugače.

Izvorna koda 21. naloga007b.py *[https://github.com/pzi-si/pzi-src-2/blob/main/naloga007b.py]*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" Program, ki preveri, ali je podano število popoln kvadrat brez vključenega modula math. """

# Izpišemo, za kakšen program gre
print("*"*53)
print("*"*5,"Program za preverjanje popolnih kvadratov","*"*5)
print("*"*53)

# Uporabnika pozovemo, da vpiše število, ki ga bomo preverili
stevilo = input("Vpišite število: ")

# Lepotni izpis
print("-"*53)

# Izpišemo število, ki ga je uporabnik vnesel
print(f"Število, ki ga preverjamo, je: {stevilo}")

# Lepotni izpis
print("="*53)

# Spremenljivko število pretvorimo v tip 'float'
stevilo_stevilo = float(stevilo)

# Izračunamo koren podanega števila
koren = stevilo_stevilo**0.5

# Preverimo, če je izračunan koren tipa 'integer'
if koren.is_integer():
    # ker vemo, da je koren tipa integer, ga pretvorimo v ta tip
    koren = int(koren)
    # izpišemo besedilo
    print(f"Podano število {stevilo} je popoln kvadrat števila
```

```
    {koren}!")
35 else:
36     # izpišemo obvestilo, da podano število ni popoln kvadrat
37     print(f"Podano število {stevilo} ni popoln kvadrat.")
38 # Lepotni izpis
39 print("-"*53)
```

Razlik med Izvorna koda 20 in Izvorna koda 21 ni veliko. Opazite lahko, da v tem primeru nismo vključili modula *math* in da smo v **vrstici 27** za korenjenje uporabili vgrajeno možnost potenciranja (za kar uporabimo dva znaka **). Rezultat (in tudi izpis) je enak Izvorna koda 20.

Vendar pa tako Izvorna koda 20 in Izvorna koda 21 odpovesta, če namesto številke vnesete črko. Izpis bo v tem primeru podoben naslednjemu.

```
*****************************************************
***** Program za preverjanje popolnih kvadratov *****
*****************************************************
Vpišite število: rk
-----------------------------------------------------
Število, ki ga preverjamo, je: rk
=====================================================
Traceback (most recent call last):
  File "naloga007b.py", line 24, in <module>
    stevilo_stevilo = float(stevilo)
                      ^^^^^^^^^^^^^^
ValueError: could not convert string to float: 'rk'
```

Seveda znamo popraviti tudi to.

2.7.3. Rešitev #3: s kontrolami vnosa

Razlika je zgolj v **vrsticah 27 do 35**, kjer uvedemo kontrolo vnosa. Če uporabnik vnese neštevilčno vrednost, se funkcija *float()* v **vrstici 29** ne more izvesti, zato program izpiše obvestilo, da je uporabnik podal niz znakov in zato preverjanje ni smiselno. Program se nato ustavi.

Izvorna koda 22. naloga007c.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga007c.py]

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """ Program, ki preveri, ali je podano število popoln kvadrat s
   kontrolami. """
5
6  # Izpišemo, za kakšen program gre
7  print("*"*53)
8  print("*"*5,"Program za preverjanje popolnih kvadratov","*"*5)
9  print("*"*53)
10
11 # Uvozimo potrebne Python module
12 import math
13
14 # Uporabnika pozovemo, da vpiše število, ki ga bomo preverili
15 stevilo = input("Vpišite število: ")
16
17 # Lepotni izpis
18 print("-"*53)
19
20 # Izpišemo število, ki ga je uporabnik vnesel
21 print(f"Število, ki ga preverjamo, je: {stevilo}")
22
23 # Lepotni izpis
24 print("="*53)
25
26 # Preverimo, ali je uporabnik vpisal število ali ne
27 try:
28     # Spremenljivko število poskusimo pretvoriti v tip 'float'
29     stevilo_stevilo = float(stevilo)
30 except:
31     # Če pretvorba v tip 'float' ni uspešna, izpiši obvestilo in
   končaj program
32     print("Podali ste niz znakov namesto števila.\nPreverjanje ni
   smiselno!")
33     # Lepotni izpis
34     print("-"*53)
35     quit()
```

```
36
37
38 # Izračunamo koren podanega števila
39 koren = math.sqrt(stevilo_stevilo)
40
41 # Preverimo, če je izračunan koren tipa 'integer'
42 if koren.is_integer():
43     # ker vemo, da je koren tipa integer, ga pretvorimo v ta tip
44     koren = int(koren)
45     # izpišemo besedilo
46     print(f"Podano število {stevilo} je popoln kvadrat števila
   {koren}!")
47 else:
48     # izpišemo obvestilo, da podano število ni popoln kvadrat
49     print(f"Podano število {stevilo} ni popoln kvadrat.")
50 # Lepotni izpis
51 print("-"*53)
```

Izpis je v takšnem primeru podoben spodnjemu.

```
******************************************************
***** Program za preverjanje popolnih kvadratov *****
******************************************************
Vpišite število: Robert
------------------------------------------------------
Število, ki ga preverjamo, je: Robert
======================================================
Podali ste niz znakov namesto števila.
Preverjanje ni smiselno!
------------------------------------------------------
```

Če je pretvorba v **vrstici 29** uspešna, se program izvede tako, kot se je v primeru Izvorna koda 20 in Izvorna koda 21.

2.8. Delitelji podanega števila

Napišite program, ki mu podate število, program pa sam poišče in izpiše vse delitelje podanega števila. Vsak delitelj naj bo izpisan zgolj enkrat, delitelji pa naj bodo urejeni po velikosti.

Vprašajmo se, kako bi lahko nalogo sploh rešili. Načinov je seveda mnogo, a če izkoristimo moč računalnika in poiščemo najpreprostejšo, lahko rešitev poiščemo tako, da se sprehodimo po vseh celih številih od 1 do vpisane in preverjamo, ali je rezultat deljenja vpisanega števila in trenutnega števila celo število. Tako dobimo vse možne delitelje.

Nalogo bi lahko rešili s pomočjo zanke *for*, a v tem primeru smo izbrali zanko *while*.

2.8.1. Rešitev #1: izpis vsakega delitelja v novi vrstici

Izvorna koda 23. naloga008a.py *[https://github.com/pzi-si/pzi-src-2/blob/main/naloga008a.py]*

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  print("*"*43)
5  print("*** Iskanje deliteljev podanega števila ***")
6  print("*"*43)
7
8  # Uporabnika pozovemo, da vpiše število, katerega delitelje bomo
   iskali
9  stevilo = input("Vpišite število, za katerega želite poiskati vse
   delitelje: ")
10
11 # preverimo, ali je podano število tipa 'integer'
12 try:
13     stevilo = int(stevilo)
14 except:
15     print("Niste podali celega števila!")
16     quit()
17
```

```
18 print(f"Delitelji števila {stevilo} so:")
19
20 # gremo po deliteljih od 1 dalje
21 trenutno_stevilo = 1
22 while trenutno_stevilo <= stevilo:
23     #izračunamo ostanek pri deljenju števila s trenutnim številom
24     ostanek = stevilo % trenutno_stevilo
25     # preverimo, kakšen je ostanek deljenja izbranega števila z i
26     if ostanek == 0:
27         print(trenutno_stevilo)
28     trenutno_stevilo = trenutno_stevilo + 1
```

Tabela 7. Razlaga

vrstica	komentar
12-16	V teh vrsticah preverjamo, če je vnešeno število sploh celo število (tipa *integer*) in ne decimalno število ali pa recimo niz znakov. V tem primeru iskanje deliteljev nima smisla in izvajanje programa prekinemo z obvestilom, da uporabnik ni podal celega števila.
21	V tej vrstici uvedemo novo spremenljivko *trenutno_stevilo* in ji dodelimo vrednost 1. To spremenljivko potrebujemo zato, da vemo, kateri delitelj trenutno preverjamo, ter tudi zato, da lahko vrednost trenutnega števila ob koncu vsakega obhoda zanke povečamo za 1 (to storimo v **vrstici 28**).
22	Začnemo izvajati zanko *while*. Ta zanka se izvaja tako dolgo, dokler je vrednost spremenljivke *trenutno_stevilo* manjša ali enaka vrednosti spremenljivke *stevilo*, to je številke, ki jo je uporabnik vnesel in ki jo preverjamo.
24	Tukaj preverjamo, kakšen je ostanek pri deljenju števila, ki ga preverjamo (spremenljivke *stevilo*) s trenutnim številom (spremenljivka *trenutno_stevilo*). Ta ostanek nam v programskem jeziku Python da operator %.
26	Če je ostanek enak 0 (ga ni), pomeni, da smo našli delitelj podanega števila in ga v **vrstici 27** izpišemo.
28	Povečamo trenutno število (spremenljivka *trenutno_stevilo*) za 1 in gremo v nov krog izvajanja zanke *while*. Namesto zapisa v tej vrstici bi lahko zapisali tudi '*trenutno_stevilo += 1*'. Oba zapisa sta enakovredna, seveda pa je slednji krajši.

Program deluje! Če ga zaženete, bo izpis podoben spodnjemu (seveda je odvisno od

tega, kakšno število mu podate).

```
****************************************
*** Iskanje deliteljev podanega števila ***
****************************************
Vpišite število, za katerega želite poiskati vse delitelje: 123
Delitelji števila 123 so:
1
3
41
123
```

Če vpišete decimalno število ali pa niz znakov, se program tudi prav lepo ustavi.

```
****************************************
*** Iskanje deliteljev podanega števila ***
****************************************
Vpišite število, za katerega želite poiskati vse delitelje: 4.5
Niste podali celega števila!
```

```
****************************************
*** Iskanje deliteljev podanega števila ***
****************************************
Vpišite število, za katerega želite poiskati vse delitelje: Robert
Niste podali celega števila!
```

Pozor

Kljub vsemu program ni popoln. Da je res tako, lahko opazite, če mu podate negativno celo število.

V tem primeru program ne bo vrnil napake, temveč se bo izvedel, a ne bo našel nobenega delitelja. Zakaj?

Razlog je v naši zanki, ki se začne v **vrstici 22**. Zanka *while* se namreč v takšnem primeru izvaja, vse dokler je vrednost spremenljivke *trenutno_stevilo* manjša ali enaka vrednosti spremenljivke *stevilo*. V spornem primeru je vrednost spremenljivke *stevilo* negativna,

trenutno_stevilo pa ima vrednost 1. Pogoj je torej izpolnjen, še preden se zanka izvede prvič.

Seveda bi lahko program popravili na več načinov, a mi ga bomo tako, da bomo negativno število spremenili v pozitivno. To lahko storimo s pogojnimi stavki (če npr. je število negativno, ga pomnoži z -1), lahko pa uporabimo vgrajeno funkcijo *abs()*, ki za podano število vrne njegovo absolutno vrednost, kar smo storili v Izvorna koda 24. Rešitev je seveda še več.

2.8.2. Rešitev #2: izpis deliteljev za absolutno vrednost celega števila

Izvorna koda 24. naloga008b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga008b.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 print("*"*43)
5 print("*** Iskanje deliteljev podanega števila ***")
6 print("*"*43)
7
8 # Uporabnika pozovemo, da vpiše število, katerega delitelje bomo
  iskali
9 stevilo = input("Vpišite število, za katerega želite poiskati vse
  delitelje: ")
10
11 # preverimo, ali je podano število tipa 'integer'
12 try:
13     stevilo = int(stevilo)
14 except:
15     print("Niste podali celega števila!")
16     quit()
17
18 # preverjali bomo absolutno vrednost podanega števila
19 stevilo = abs(stevilo)
20
21 # izpišemo prvo vrstico izpisa
22 print(f"Delitelji števila {stevilo} so:")
23
```

```
24 # gremo po deliteljih od 1 dalje
25 trenutno_stevilo = 1
26 while trenutno_stevilo <= stevilo:
27     #izračunamo ostanek pri deljenju števila s trenutnim številom
28     ostanek = stevilo % trenutno_stevilo
29     # preverimo, kakšen je ostanek deljenja izbranega števila z i
30     if ostanek == 0:
31         print(trenutno_stevilo)
32     trenutno_stevilo = trenutno_stevilo + 1
```

Ob pozornem pregledu in primerjavi Izvorna koda 23 in Izvorna koda 24 opazimo, da je razlika (če zanemarimo komentarje) zgolj v **vrstici 19**, kjer smo zapisali, naj program od te vrstice dalje išče delitelje za absolutno vrednost (pozitivno celo število) vnosa.

Program sedaj deluje tudi za negativna števila, a vrne delitelje njegove absolutne vrednosti.

```
*****************************************
*** Iskanje deliteljev podanega števila ***
*****************************************
Vpišite število, za katerega želite poiskati vse delitelje: -16
Delitelji števila 16 so:
1
2
4
8
16
```

2.8.3. Rešitev #3: izpis deliteljev v eni vrstici

Tako Izvorna koda 23 kot Izvorna koda 24 izpišeta delitelje po vrsticah, kar je prostorsko potratno, mi pa seveda znamo program popraviti tako, da bodo delitelji izpisani v eni vrstici.

Izvorna koda 25. naloga008c.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga008c.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
```

```
print("*"*43)
print("*** Iskanje deliteljev podanega števila ***")
print("*"*43)

# Uporabnika pozovemo, da vpiše število, katerega delitelje bomo iskali
stevilo = input("Vpišite število, za katerega želite poiskati vse delitelje: ")

# preverimo, ali je podano število tipa 'integer'
try:
    stevilo = int(stevilo)
except:
    print("Niste podali celega števila!")
    quit()

# preverjali bomo absolutno vrednost podanega števila
stevilo = abs(stevilo)

# gremo po deliteljih od 1 dalje
trenutno_stevilo = 1

# uvedemo spremenljivko, kamor bomo zapisovali rezultate
delitelji = ""

while trenutno_stevilo <= stevilo:
    #izračunamo ostanek pri deljenju števila s trenutnim številom
    ostanek = stevilo % trenutno_stevilo
    # preverimo, kakšen je ostanek deljenja izbranega števila z i
    if ostanek == 0:
        delitelji = delitelji + str(trenutno_stevilo)
        # vejico izpišemo samo, če ne dodajamo zadnjega števila v nizu
        if trenutno_stevilo != stevilo:
            delitelji +=", "
    trenutno_stevilo = trenutno_stevilo + 1

#izpišemo delitelje
# začnemo izpis rezultatov
print(f"Delitelji števila {stevilo} so: {delitelji}")
```

Tabela 8. Razlaga

vrstica	komentar
25	Uvedemo novo spremenljivko *delitelji* in ji dodelimo prazen niz znakov. Spremenljivki bomo v zanki *while* dodajali najdene delitelje, vse naenkrat pa izpisali v **vrstici 40**.
32	Ko najdemo delitelj, ga dodamo k prejšnjim (dodamo k nizu *delitelji*). Pri tem uporabimo funkcijo *str()*, saj je spremenljivka *trenutno_stevilo* tipa *integer* (celo število). To vrstico bi lahko zapisali tudi kot '*trenutno_stevilo += str(trenutno_stevilo)*'. Oba zapisa sta enakovredna.
34	Preverimo, če *trenutno_stevilo* **ni** enako spremenljivki *stevilo*. Če je, to pomeni, da smo prišli do konca deliteljev in vejice v **vrstici 35** ne dodamo. Če ni enako, pa pomeni, da pričakujemo vsaj še en delitelj, zato jo bomo dodali.
35	Če v **vrstici 34** *trenutno_stevilo* **ni** enako spremenljivki *stevilo*, dodamo vejico.
40	Izpišemo besedilo in delitelje.

Izpis je sedaj podoben spodnjemu.

```
*****************************************
*** Iskanje deliteljev podanega števila ***
*****************************************
Vpišite število, za katerega želite poiskati vse delitelje: -12
Delitelji števila 12 so: 1, 2, 3, 4, 6, 12
```

2.9. Praštevilo

Napišite program, ki za podano število pove, ali je praštevilo.

2.9.1. Rešitev

Izvorna koda 26. naloga009.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga009.py]

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """ Program, ki za podano število pove, ali je praštevilo. """
5
6 # Uporabnika pozovemo, da vpiše število, ki ga preverjamo
7 stevilo = input("Vpišite število, za katerega vas zanima, ali gre
  za praštevilo: ")
8
9 # preverimo, ali je podano število tipa 'integer'
10 try:
11     stevilo = int(stevilo)
12 except:
13     print("Niste podali celega števila!")
14     quit()
15
16 # uvedemo novo spremenljivko "napaka", ki ji določimo vrednost
  "False"
17 napaka = False
18
19 # za vsa števila od 2 do izbranega števila - 1 preverimo, ali je
  delitelj števila
20 for i in range(2,stevilo):
21     # če je trenutno število i delitelj, spremenimo vrednost
  spremenljivke napaka
22     if stevilo % i == 0:
23         napaka = True
24
25 # če ima vrednost TRUE, potem izpišemo, da iskano število NI
  praštevilo
```

```
26 if napaka == True:
27     print(f"Število {stevilo} NI praštevilo!")
28 else:
29     print(f"Število {stevilo} JE praštevilo!")
```

Tabela 9. Razlaga

vrstica	komentar
17	Uvedemo novo spremenljivko *napaka*, ki ji določimo začetno vrednost "False". S tem smo ji določili podatkovni tip *boolean*, kar pomeni, da ima lahko zgolj vrednost *0* ali *1* oziroma *True* ali *False*.
20	Za vsa cela števila v obsegu od 2 do (ampak brez) vrednosti *stevilo* bomo preverjali, ali je trenutno število (ki je v našem primeru vrednost spremenljivke *i*) delitelj *stevilo*.
22	Preverjamo, če je *i* delitelj *stevilo*.
23	Če v **vrstici 22** ugotovimo, da je *i* delitelj, spremenimo vrednost spremenljivke *napaka* v *True*, To pomeni, da smo našli še en delitelj, ki ni 1 in ni podano število, kar pomeni, da nimamo praštevila.
26	Preverimo vrednost spremenljivke *napaka*. Če ima vrednost *True*, vemo, da smo našli še vsaj en dodaten delitelj in podano število ni praštevilo, kar izpišemo v **vrstici 27**.
28	Ker nismo našli nobenega dodatnega delitelja, vemo, da je podano število praštevilo, kar izpišemo v **vrstici 29**.

Izpis programa, če je podano število praštevilo.

```
Vpišite število, za katerega vas zanima, ali gre za praštevilo: 127
Število 127 JE praštevilo!
```

Če podano število ni praštevilo, je izpis seveda drugačen.

```
Vpišite število, za katerega vas zanima, ali gre za praštevilo: 128
Število 128 NI praštevilo!
```

Delujejo pa seveda tudi kontrole.

```
Vpišite število, za katerega vas zanima, ali gre za praštevilo:
Robert
Niste podali celega števila!
```

2.10. Prestopno leto

Napišite program, ki za vpisano letnico pove, ali je (bilo/bo) leto prestopno.

2.10.1. Rešitev

Izvorna koda 27. naloga010.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga010.py]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

print("******************************")
print("*** Ali je leto prestopno? ***")
print("******************************")

# pozovemo uporabnika, naj vpiše letnico
letnica = input("Vpiši letnico: ")

# preverimo, če je vpisana res letnica
try:
    letnica = int(letnica)
    # omejimo letnice na obseg od -9999 do 9999
    if letnica > 9999 or letnica < -9999:
        print("Niste vnesli letnice!")
        quit()
except:
    print("Niste vnesli letnice!")
    quit()

# uvedemo spremeljivko prestopno
prestopno = False

# če je letnica deljiva s 400, je leto prestopno
if letnica % 400 == 0:
    prestopno = True
# če ni deljiva s 400, je pa deljiva s 4 in 100, potem leto ni
  prestopno
elif letnica % 4 == 0 and letnica % 100 == 0:
```

```
30      prestopno = False
31  # letnica je deljiva s 4, ni pa hkrati deljiva s 100 -> leto je
    prestopno
32  elif letnica % 4 == 0:
33      prestopno = True
34
35  if prestopno:
36      print(f"Leto {letnica} JE prestopno!")
37  else:
38      print(f"Leto {letnica} NI prestopno!")
```

Tabela 10. Razlaga

vrstica	komentar
13	Preverimo, če je podana letnica tipa *integer* oziroma celo število.
15	Omejimo letnice na obseg od -9999 do 9999.
23	Uvedemo spremenljivko *prestopno* in ji določimo vrednost *False*. Tako predvidevamo, da vnesena letnica ne označuje prestopnega leta, razen če bomo ugotovili drugače.
26	Začnemo s preverjanjem. Najprej preverimo, če je letnica deljiva s 400, saj potem vemo, da je leto prestopno.
29	Če letnica ni deljiva s 400, je pa hkrati deljiva s 4 in 100, potem leto ni prestopno.
32	Če je letnica deljiva s 4 ter ni deljiva s 100 in 400, potem je leto prestopno. Delitelja 400 in 100 smo preverili že v prejšnjih dveh pogojih.
35	Preverimo, če ima spremenljivka *prestopno* vrednost (pomeni, da nima vrednosti 0 ali *False*). Če ima, potem je leto prestopno in to izpišemo na zaslon. Ta vrstica bi lahko bila zapisana tudi kot '*if prestopno == True:*'. Zapisa sta enakovredna.
38	Če leto ni prestopno, to izpišemo na zaslon.

Primer izpisa programa.

```
*****************************
*** Ali je leto prestopno? ***
*****************************
Vpiši letnico: 2023
Leto 2023 NI prestopno!
```

2.11. Fibonaccijevo zaporedje

Napiši program, ki izpiše prvih *X* členov Fibonaccijevega zaporedja. Število *X* na poziv vnese uporabnik.

2.11.1. Rešitev #1: z vmesno spremenljivko

Izvorna koda 28. naloga011a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga011a.py]

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  Napiši program, ki izpiše prvih _X_ členov Fibonaccijevega
   zaporedja. Število _X_ na poziv vnese uporabnik.
6  """
7
8  # Uporabnika pozovemo, da vpiše število, koliko členov
   Fibonaccijevega zaporedja naj prikažemo
9  stevilo = input("Vpišite, koliko členov Fibonaccijevega zaporedja
   vas zanima: ")
10
11 # preverimo, ali je podano število tipa 'integer'
12 try:
13     stevilo = int(stevilo)
14 except:
15     print("Niste podali celega števila!")
16     quit()
17
18 #ker moramo vedno poznati dve števili zaporedja, uvedemo dve novi
   neznanki
19 stevilo_ena = 1
20 stevilo_dva = 1
21 # za vsa števila od 1 do izbranega števila + 1 izračunamo in
   izpišemo
22 for i in range(1,stevilo+1):
23     print(stevilo_ena)
24     # uvesti moramo novo spremeljivko, da si začasno zapomni
```

```
   vrednost spremenljivke stevilo_ena
25     stevilo_nic = stevilo_ena
26     stevilo_ena = stevilo_dva
27     stevilo_dva = stevilo_nic+stevilo_ena
```

Tabela 11. Razlaga

vrstica	komentar
19-20	Ker moramo vedno poznati dve števili zaporedja, uvedemo dve novi neznanki: *stevilo_ena* in *stevilo_dva* ter obema priredimo/določimo vrednost 1, ker sta to prva dva člena Fibonaccijevega zaporedja.
22	Izvedemo zanko za vse vrednosti od 1 do *stevilo* (ki ga je vnesel uporabnik) + 1. Zakaj *stevilo* povečamo za 1? Ker nam funkcija *range(zacetek,konec)* vrne vrednosti po formuli r[i] = začetek + korak*i, kjer je i >= 0 in r[i] < konec. Korak je v našem primeru *1*, saj ga nismo določili drugače.
23	Izpišemo vrednost spremenljivke *stevilo_ena*.
25	Vrednost spremenljivke *stevilo_ena* začasno zapišemo, kar storimo tako, da jo priredimo spremenljivki *stevilo_nic*. To moramo storiti zato, ker bi drugače trenutno vrednost spremenljivke *stevilka_ena* prepisali in izgubili, ga pa potrebujemo, da izračunamo naslednje število.
26	Spremenljivka *stevilo_ena* zdaj dobi vrednost spremenljivke *stevilo_dva* (premaknemo se po Fibonaccijevem zaporedju za eno mesto v desno).
27	Nova vrednost spremenljivke *stevilo_dva* pa je seštevek vrednosti spremenljivk *stevilka_ena* in *stevilka_nic*. Zato smo uvedli vmesno spremenljivko.

Primer izpisa programa za prvih 5 členov zaporedja.

```
Vpišite, koliko členov Fibonaccijevega zaporedja vas zanima: 5
1
1
2
3
5
```

2.11.2. Rešitev #2: z dvojnim prirejanjem

Izvorna koda 29. naloga011b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga011b.py]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Napiši program, ki izpiše prvih _X_ členov Fibonaccijevega zaporedja. Število _X_ na poziv vnese uporabnik.
"""

# Uporabnika pozovemo, da vpiše število, koliko členov Fibonaccijevega zaporedja naj prikažemo
stevilo = input("Vpišite, koliko členov Fibonaccijevega zaporedja vas zanima: ")

# preverimo, ali je podano število tipa 'integer'
try:
    stevilo = int(stevilo)
except:
    print("Niste podali celega števila!")
    quit()

#ker moramo vedno poznati dve števili zaporedja, uvedemo dve novi neznanki
stevilo_ena = stevilo_dva = 1
# za vsa števila od 1 do izbranega števila + 1 izračunamo in izpišemo
for i in range(1,stevilo+1):
    print(stevilo_ena)
    stevilo_ena, stevilo_dva = stevilo_dva,stevilo_ena+stevilo_dva
```

Tabela 12. Razlaga

vrstica	komentar
19	Za razliko od rešitve a v tem primeru ne določamo/prirejamo vsake spremenljivke posebej, ampak lahko to v programskem jeziku *Python* storimo hkrati, saj je vrednost obeh spremenljivk (*stevilka_ena* in *stevilka_dva*) enaka, to je 1.
23	Tudi v zanki lahko obe vrednosti vsakič izračunamo hkrati, kar pomeni, da ne potrebujemo vmesne spremenljivke kot v rešitvi a. Kratka razlaga je, da programski jezik *Python* najprej izračuna vrednosti na desni strani enačaja in jih nato priredi spremenljivkam na levi.

Izpis programa je enak prejšnjemu.

2.12. Računanje statistike

Napišite program, ki od uporabnika bere števila, dokler uporabnik ne vpiše števila 0. Program naj sproti izpisuje največje vpisano število, najmanjše vpisano število ter povprečje vpisanih števil.

2.12.1. Rešitev

Izvorna koda 30. naloga012.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga012.py]

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  Napiši program, ki od uporabnika bere števila, dokler uporabnik ne
   vpiše števila 0. Program naj sproti izpisuje največje, najmanjše
   in povprečje števil, vpisanih do sedaj.
6  """
7
8  # Izpišemo, za kakšen program gre
9  print("*"*43)
10 print("*"*5,"Program za računanje statistike","*"*5)
11 print("*"*43)
12
13 # ustvarimo nov prazen seznam
14 stevila = []
15
16 while True:
17     # uporabnika prosimo, naj vpisuje števila
18     stevilo = input('Vnesi število: ')
19
20     # če je vpisano število 0 naj se program prekine.
21     if stevilo == "0":
22         print("Pritisnili ste 0 in tako končali program.")
23         quit()
24
25     # preverimo, ali je uporabnik vnesel število
26     try:
27         stevilo = float(stevilo)
```

```
28      except:
29          print("Niste vnesli števila!")
30          quit()
31
32      stevila.append(stevilo)
33      print(f"V seznamu so števila: {stevila}")
34      print(f"Največje število = {max(stevila)}")
35      print(f"Najmanjše število = {min(stevila)}")
36      print(f"Povprečje števil v seznamu = {sum(stevila)/len(
    stevila)}")
37      print("-"*40)
```

Tabela 13. Razlaga

vrstica	komentar
14	Za našo rešitev bomo uporabili seznam (angl. *list*), zato uvedemo nov seznam *stevila*, ki je na začetku seveda prazen.
16	***While True*** je način oblikovanja zanke v programskem jeziku Python, ki omogoča, da se blok kode ponavlja v nedogled. Pogosto se uporablja v povezavi z ukazom *break*, ki omogoča izhod iz zanke pod določenimi pogoji. V našem primeru je izhod iz zanke v vrstici **23** in se izvede ob pogoju, da uporabnik vnese število **0**. Kakšna je razlika med *break* in *quit*? Kadar uporabimo *break*, se bo koda za zanko po izhodu iz zanke izvedla. Če uporabimo *quit*, se program na tistem mestu konča. V našem primeru smo uporabili *quit*, ker kode za zanko *while* nimamo.
18	Uporabnika pozovemo k vpisu številke, ki jo priredimo spremenljivki *stevilka*. Pazite na razliko med *stevilka* in *stevilke* (v vrstici **14**). Gre namreč za dve različni spremenljivki.
21-23	Če uporabnik vnese število **0**, izpišemo obvestilo (**vrstica 22**) ter izvajanje programa prekinemo (**vrstica 23**). Če uporabnik vpiše kaj drugega, se ta blok kode preskoči.
26-30	V tem bloku kode preverimo, če je uporabnik vnesel število. Če ga ni, izpišemo sporočilo (**vrstica 29** in izvajanje programa prekinemo (**vrstica 30**)).
32	Če je vnos prestal vse prejšnje kontrole, število dodamo na seznam *stevila*. To storimo zato, ker bomo uporabili funkcije, vgrajene v sam programski jezik, ki delujejo s seznami. Lahko bi računali tudi na roke.

vrstica	komentar
33	Izpišemo števila, ki jih je uporabnik že vnesel in so trenutno na seznamu *stevila*.
34	Izpišemo največje število na seznamu *stevila*. V ta namen uporabimo vgrajeno funkcijo *max()*, ki vrne največjo vrednost na seznamu.
35	Izpišemo najmanjše število na seznamu *stevila*. V ta namen uporabimo vgrajeno funkcijo *min()*, ki vrne najmanjšo vrednost na seznamu.
36	Izpišemo povprečje števil na seznamu *stevila*. V ta namen uporabimo vgrajeni funkciji *sum()* (vrne vsoto vseh števil na seznamu) in *len()* (vrne število elementov na seznamu).

Če program zaženete, bo izpis podoben spodnjemu.

```
****************************************
***** Program za računanje statistike *****
****************************************
Vnesi število: 5
V seznamu so števila: [5.0]
Največje število = 5.0
Najmanjše število = 5.0
Povprečje števil v seznamu = 5.0
--------------------------------------
Vnesi število: 6
V seznamu so števila: [5.0, 6.0]
Največje število = 6.0
Najmanjše število = 5.0
Povprečje števil v seznamu = 5.5
--------------------------------------
Vnesi število: 4
V seznamu so števila: [5.0, 6.0, 4.0]
Največje število = 6.0
Najmanjše število = 4.0
Povprečje števil v seznamu = 5.0
--------------------------------------
Vnesi število: 0
Pritisnili ste 0 in tako končali program.
```

Pozor

Program deluje v skladu z navodili in vsebuje tudi kontrole vnosa. Če uporabnik vpiše črko namesto števila, se program ustavi.

```
****************************************
***** Program za računanje statistike *****
****************************************
Vnesi število: programiranje
Niste vnesli števila!
```

Kako bi morali program popraviti, da bi izpisal zgolj obvestilo, da ni bilo vneseno število, ta vnos zanemaril in deloval naprej? Ni vam treba napisati niti ene nove vrstice kode!

Namig

Izbrišite **vrstico 30** (*quit()*), blok kode 32-37 pa prestavite za kontrolo *try* (za **vrstico 27** in pred *except*).

2.13. Seštevanje števk naravnega števila

Sestavite program, ki prebere naravno število in sešteje vse njegove števke, kar ponavlja, dokler ne pride do enomestnega števila.

Program naj izpisuje vse vmesne rezultate.

2.13.1. Rešitev #1: z direktnim seštevanjem

Izvorna koda 31. naloga013a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga013a.py]

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  """
5  Seštevanje števk naravnega števila
6  """
7
8  # uporabnika prosimo, naj vpiše številko
9  stevilo = input('Vnesi večmestno celo število: ')
10
11 try:
12     stevilo = int(stevilo)
13 except:
14     print("Niste vnesli celega števila!")
15     quit()
16
17 while stevilo > 9:
18     sestevek = 0
19     stevilo_string = str(stevilo)
20     for i in range(0,len(stevilo_string)):
21         sestevek += int(stevilo_string[i])
22     print(sestevek)
23     stevilo = sestevek
```

Tabela 14. Razlaga

vrstica	komentar
17-22	Začnemo z zanko *while*, ki jo bomo ponavljali, dokler ne bomo dobili enomestnega števila (manjšega od 10, oziroma drugače - ponavljamo, dokler je spremenljivka *stevilo* večja od 9).
18	Vsakič, ko gre zanka v nov krog, nastavimo spremenljivko *sestevek* na vrednost 0.
19	Uvedemo novo spremenljivko *stevilo_string* in mu priredimo vrednost spremenljivke *stevilo*, pri čemer tip spremenljivke s pomočjo funkcije *str()* spremenimo v *string* (niz znakov). To storimo zato, da lahko dobimo dolžino vnesenega niza znakov, saj funkcija *len()* ne deluje s številčnimi tipi (*int*,*float* ,*double*).
20-21	Vse števke od prve do zadnje (od mesta 0 do konca niza) seštejemo. Bodite pozorni, da moramo v **vrstici 21** niz znakov *string* spet spremeniti v številko, kar storimo s pomočjo funkcije *int()*. Vsako številko dodamo vrednosti spremenljivke *sestevek*.
22	Izpišemo vrednost spremenljivke *sestevek*.
23	Vrednost spremenljivke *stevilo* spremenimo in ji določimo vrednost spremenljivke *sestevek*, zato da bomo v naslednji iteraciji seštevali števke novega števila. Če bo vrednost spremenljivke *stevilo* manjše od 10, se bo zanka *while* prekinila in se ne bo izvedla.

Primer izpisa:

```
Vnesi večmestno celo število: 99999
45
9
```

2.13.2. Rešitev #2: z uporabo definirane funkcije

Izvorna koda 32. naloga013b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga013b.py]

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Seštevanje števk naravnega števila
"""

# uporabnika prosimo, naj vpiše številko
stevilo = input('Vnesi večmestno celo število: ')

try:
    stevilo = int(stevilo)
except:
    print("Niste vnesli celega števila!")
    quit()

def sestej(x):
    sestevek = 0
    x_string = str(x)
    for i in range(0,len(x_string)):
        sestevek += int(x_string[i])
    return sestevek

while stevilo > 9:
    stevilo = sestej(stevilo)
    print(stevilo)
```

Tabela 15. Razlaga

vrstica	komentar
17-22	Tokrat uporabimo možnost priprave (definiranja) lastne funkcije, ki bo seštevala vrednosti. Gre za splošni zapis fukcij, ki jih v programski kodi navadno večkrat uporabljamo in jih zato definiramo kot svojo funkcijo. Ta mora biti definirana, preden jo prvič kličemo v programski kodi, saj se ta izvaja po vrsti in mora imeti definirano funkcijo **v spominu**. Dobra praksa je, da se vse definicije funkcij, ki jih imamo v programski kodi, uvrsti na začetek, takoj za tem, ko uvozimo vse morebitne dodatne knjižnice, ki jih uporabljamo. Šele po vseh definicijah pišemo programsko kodo, ki jo izvajamo. Opazimo lahko, da je funkcija *sestej()* precej podobna vrsticam 17-21 rešitve a, razlika je zgolj v dodatni **vrstici 22**, saj rešitve funkcije ne izpišemo, ampak jo vrnemo (s pomočjo *return*). Ime spremenljivke *stevilo_string* smo namenoma spremenili v *x_string*, ker smo spremenljivko, ki jo podamo kot parameter funkcije *sestej()*, poimenovali *x* in ne *stevilo*, da ne bi prišlo do pomote.
24-26	Zanko *while* izvajamo ob enakih pogojih kot prej, razlika je zgolj v tem, da sedaj v zanki zgolj pokličemo prej uvedeno (pripravljeno, določeno) funkcijo *sestej()*, ki ji podamo vsakokratno število, funkcija nam vrne rezultat, ki ga potem v **vrstici 26** izpišemo.

Izpis programa je identičen prejšnjemu.

2.14. Pravilno sklanjanje

Sestavite program, ki prebere število in v pravilni slovenščini izpiše, koliko opravljenih izpitov imate. Bodite pozorni pri številih, večjih od 100!

2.14.1. Rešitev

Pri tem reševanju imamo 4 različne možnosti sklanjanja:

- izpit (za 1 opravljen izpit)
- izpita (za 2 opravljena izpita)
- izpite (za 3 ali 4 opravljene izpite)
- izpitov (za 0 ali več kot 4 opravljene izpite)

Ker se nam te 4 možnosti ponavljajo tudi kasneje (npr. 101, 201, 1001, 10201), moramo najti vzorec. Kaj je skupnega vsem tem številom?

Skupen jim je ostanek pri deljenju s 100:

- Če je ostanek 1, izberemo sklon **izpit**.
- Če je ostanek 2, izberemo sklon **izpita**.
- Če je ostanek 3 ali 4, izberemo sklon **izpite**.
- V vseh ostalih primerih izberemo sklon **izpitov**.

Izvorna koda 33. naloga014.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga014.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """
5 Pravilno sklanjanje
6 """
7
8 # uporabnika prosimo, naj vpiše število opravljenih izpitov
9 stevilo = input('Vnesi število opravljenih izpitov: ')
10
11 try:
```

```
    stevilo = int(stevilo)
except:
    print("Niste vnesli celega števila!")
    quit()

if stevilo%100 == 1:
    besedilo = "izpit"
elif stevilo%100 == 2:
    besedilo = "izpita"
elif stevilo%100 == 3 or stevilo%100 == 4:
    besedilo = "izpite"
else:
    besedilo = "izpitov"

print(f"Opravili ste {stevilo} {besedilo}.")
```

Če program zaženete, bo izpis podoben spodnjemu.

```
Vnesi število opravljenih izpitov: 103
Opravili ste 103 izpite.
```

2.15. Fakulteta števila

Sestavi program, ki izračuna fakulteto vnesenega števila.

2.15.1. Rešitev #1: z računanjem in prikazom vmesnih korakov

Izvorna koda 34. naloga015a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga015a.py]

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  print("-"*21)
5  print("| Fakulteta števila |")
6  print("-"*21)
7
8  # uporabnika prosimo, naj vpiše število
9  stevilo = input('Vnesi število: ')
10
11 try:
12     stevilo = int(stevilo)
13 except:
14     print("Niste vnesli celega števila!")
15     quit()
16
17 if stevilo < 0:
18     print("Ne morem izračunati fakultete negativnega števila!")
19     quit()
20
21 if stevilo == 0 or stevilo == 1:
22     print(f"{stevilo}! = 1")
23 else:
24     izpis = []
25     n = stevilo
26     fakulteta = 1
27     while n > 0:
28         izpis.append(str(n))
29         fakulteta *= n
30         n -= 1
```

```
31
32      izpis = ' * '.join(izpis)
33      print(f"{stevilo}! = {izpis} = {fakulteta}")
```

Tabela 16. Razlaga

vrstica	komentar
11-15	Preverimo, če je uporabnik vnesel celo število (**vrstica 12**). Če ni vnesel celega števila (lahko je vnesel decimalno število, črke, polje, slovar ali kakršen koli drug poljuben niz), ga na to opozorimo (**vrstica 14**) in program prekinemo (**vrstica 15**).
17-19	Preverimo, če je uporabnik morda vnesel negativno število. Če ga je vnesel, izpišemo, da izračun ni možen (**vrstica 18**), ter program prekinemo (**vrstica 19**).
21-22	Če uporabnik vpiše število **0** ali **1**, je rezultat ena. Ker gre za posebnost (sploh če računamo **0!**), to rešimo s pogojnim stavkom.
23	Uporabnik je vnesel celo število, večjo od 1 (vse ostale smo izločili v korakih zgoraj), zato lahko izračunamo fakulteto števila.
24	Uvedemo spremenljivko *izpis* in ji priredimo prazen seznam (polje), v katerega bomo vpisovali vmesne rezultate.
25	Uvedemo spremenljivko *n*, ki ima na začetku enako vrednost kot spremenljivka *stevilo*. Zakaj potrebujemo novo spremenljivko? Ker ima spremenljivka *stevilo* vrednost, ki jo je vnesel uporabnik, spremenljivka *n* pa se bo spreminjala, ko bomo vneseno število manjšali proti 1 in računali fakulteto. Potrebujemo torej obe spremenljivki.
26	Uvedemo spremenljivko *fakulteta*, ki bo imela vrednost izračunane fakultete vnesenega števila. Nastavimo ji začetno vrednost 1, s katero bomo množili vsa ostala števila.
27-30	Zanka, ki se naj izvaja, dokler je vrednost spremenljivke *n* večja od 0 (se pravi od vrednosti spremenljivke *stevilo* (**vrstica 25**) do 1).
28	Vsakič na seznam *izpis* dodamo trenutno vrednost spremenljivke *n* (prva bo enaka vrednosti vpisanega števila *stevilo*, zadnja bo 1, po tem se bo zanka prekinila, saj bo vrednost spremenljivke *n* enaka 0 in pogoj zanke *while* v **vrstici 27** ne bo več izpolnjen).

vrstica	komentar
29	Trenutno vrednost spremenljivke *fakulteta* množimo s trenutno vrednostjo spremenljivke *n*. Po zaključeni zanki bo imela tako spremenljivka *fakulteta* vrednost fakultete števila. To vrstico bi lahko zapisali tudi daljše v obliki *fakulteta = fakulteta * n*. Oba zapisa sta enakovredna.
30	Vrednost spremenljivke *n* po koncu izračuna zmanjšamo za 1 in zanka se ponovno zažene. To vrstico bi lahko zapisali tudi kot *n = n - 1*. Oba zapisa sta enakovredna.
32	Ko se zanka prekine, lahko vse vrednosti v seznamu (polju) *izpis* uporabimo za pripravo izpisa vmesnih korakov. Ker smo uporabili seznam, lahko uporabimo funkcijo *join()*. Ta nam vse elemente na seznamu poveže v niz znakov, med posamezne elemente pa doda niz znakov, ki ga podamo na začetku (v našem primeru je to niz znakov " * " (presledek zvezdica presledek)). Na tak način dobimo lep izpis vseh vmesnih korakov.

Primeri izpisa:

```
---------------------
| Fakulteta števila |
---------------------
Vnesi število: 6
6! = 6 * 5 * 4 * 3 * 2 * 1 = 720
```

```
---------------------
| Fakulteta števila |
---------------------
Vnesi število: -3
Ne morem izračunati fakultete negativnega števila!
```

```
---------------------
| Fakulteta števila |
---------------------
Vnesi število: 0
0! = 1
```

```
---------------------
| Fakulteta števila |
---------------------
Vnesi število: python
Niste vnesli celega števila!
```

2.15.2. Rešitev #2: z uporabo definirane funkcije in izpisom vmesnih rezultatov

Izvorna koda 35. naloga015b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga015b.py]

```
 1 #!/usr/bin/env python3
 2 # -*- coding: utf-8 -*-
 3
 4 print("-"*21)
 5 print("| Fakulteta števila |")
 6 print("-"*21)
 7
 8 # uporabnika prosimo, naj vpiše število
 9 stevilo = input('Vnesi število: ')
10
11 def fakulteta(stevilo):
12     try:
13         stevilo = int(stevilo)
14         if stevilo < 0:
15             rezultat = "Ne morem izračunati fakultete negativnega
   števila!"
16         elif stevilo == 0 or stevilo == 1:
17             rezultat = f"{stevilo}! = 1"
18         else:
19             izpis = []
20             n = stevilo
21             fakulteta = 1
22             while n > 0:
23                 izpis.append(str(n))
24                 fakulteta *= n
25                 n -= 1
26
27             izpis = ' * '.join(izpis)
```

```
28             rezultat = f"{stevilo}! = {izpis} = {fakulteta}"
29     except:
30         rezultat = "Niste vnesli celega števila!"
31     return rezultat
32 
33 print(fakulteta(stevilo))
```

Razlika med rešitvijo a in rešitvijo b je v tem, da v drugem primeru uporabimo možnost priprave (definiranja) funkcije. Funkcija *fakulteta()* potrebuje zgolj 1 parameter, to je število, za katerega želimo izračunati fakulteto in katerega vrednost priredimo spremenljivki *stevilo*. Funkcija ima povsem enake sestavne dele kot rešitev a in v **vrstici 31** vrne rezultat. Rezultat je izpis, ki ga lahko direktno izpišemo, kar storimo v **vrstici 33**.

Izpis programa je enak prejšnjemu.

2.15.3. Rešitev #3: z uporabo vgrajene funkcije za izračun fakultete

Izvorna koda 36. naloga015c.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga015c.py]

```
 1 #!/usr/bin/env python3
 2 # -*- coding: utf-8 -*-
 3 
 4 print("-"*21)
 5 print("| Fakulteta števila |")
 6 print("-"*21)
 7 
 8 # uvozimo modul math
 9 import math
10 
11 # uporabnika prosimo, naj vpiše število
12 stevilo = input('Vnesi število: ')
13 
14 try:
15     stevilo = int(stevilo)
16 except:
17     print("Niste vnesli celega števila!")
18     quit()
19 
20 if stevilo < 0:
21     print("Vnesli ste negativno število!")
```

```
22      quit()
23
24  print(f"{stevilo}! = {math.factorial(stevilo)}")
```

Opazimo lahko, da je ta program najkrajši (in morda najlažji za razumeti), ne vsebuje pa vseh vmesnih korakov.

Za reševanje uporabimo vgrajeno funkcijo *math.factorial()*. Če jo želimo uporabiti, moramo vključiti modul *math*, kar storimo v **vrstici 9**. Ohranimo kontroli vnosa za preverjanje, če je vneseno celo število (**vrstice 14-18**), ter za preverjanje, če je uporabnik vnesel pozitivno število (**vrstice 20-22**), saj v nasprotnem primeru funkcija *math.factorial()* vrne napako.

Če uporabnik res vnese pozitivno celo število, lahko fakulteto izračunamo in jo izpišemo (**vrstica 24**).

Primer izpisa:

```
----------------------
| Fakulteta števila |
----------------------
Vnesi število: 6
6! = 720
```

2.16. Risanje smrečice

Napiši program, ki od uporabnika dobi naravno število, nato pa nariše *smrečico* z znakom *.

Smrečica naj ima toliko vrstic, kolikor jih uporabnik definira z vnosom naravnega števila.

2.16.1. Rešitev #1: smrečica, poravnana levo

Izvorna koda 37. naloga016a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga016a.py]

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  print("*"*20)
5  print("* Risanje smrečice *")
6  print("*"*20)
7
8  # uporabnika prosimo, naj vpiše število
9  stevilo = input('Vnesi višino smrečice: ')
10
11 try:
12     stevilo = int(stevilo)
13 except:
14     print("Niste vnesli celega števila!")
15     quit()
16
17 if stevilo < 0:
18     stevilo = -stevilo
19     print(f"-{stevilo} sem za vas pretvoril v {stevilo}.")
20
21 # poravnano levo
22 n = 1
23 while n <= stevilo:
24     print("*"*n)
25     n += 1
```

Tabela 17. Razlaga

vrstica	komentar
9	Uporabnika pozovemo, naj vnese višino smrečice.
11-15	Preverimo, če je uporabnik vnesel celo število (**vrstica 12**). Če ni vnesel celega števila (lahko je vnesel decimalno število, črke, polje, slovar ali kakršen koli drug poljuben niz), ga na to opozorimo (**vrstica 14**) in program prekinemo (**vrstica 15**).
17-19	Preverimo, če je uporabnik morda vnesel negativno število. Če ga je vnesel, izpišemo, da izračun ni možen (**vrstica 18**), ter program prekinemo (**vrstica 19**).
22-25	Vpeljemo spremenljivko *n* in ji priredimo vrednost 1 (**vrstica 22**). Nato pričnemo z izvajanjem zanke, ki naj teče od trenutne vrednosti spremenljivke *n* do trenutka, ko bo večja od vrednosti spremenljivke *stevilo* (**vrstica 23**). V vsakem koraku najprej izpišemo toliko zvezdic, kolikor je vrednost spremenljivke *n*, kar naredimo tako, da niz znakov množimo z vrednostjo *n* (**vrstica 24**). Ne pozabite, da funkcija *print* izvede izpis v eni vrstici, vsak naslednji bo v novi vrstici. Na koncu vsake zanke vrednost spremenljivke *n* povečamo za 1 (**vrstica 25**).

Primer izpisa:

```
********************
* Risanje smrečice *
********************
Vnesi višino smrečice: 6
*
**
***
****
*****
******
```

2.16.2. Rešitev #2: smrečica, poravnana desno

Izvorna koda 38. naloga016b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga016b.py]

```
1 #!/usr/bin/env python3
```

```
 2 # -*- coding: utf-8 -*-
 3
 4 print("*"*21)
 5 print("* Risanje smrečice *")
 6 print("*"*21)
 7
 8 # uporabnika prosimo, naj vpiše število
 9 stevilo = input('Vnesi višino smrečice: ')
10
11 try:
12     stevilo = int(stevilo)
13 except:
14     print("Niste vnesli celega števila!")
15     quit()
16
17 if stevilo < 0:
18     stevilo = -stevilo
19     print(f"-{stevilo} sem za vas pretvoril v {stevilo}.")
20
21 # poravnano desno
22 n = 1
23 while n <= stevilo:
24     print(" "*(stevilo-n),"*"*n,sep="")
25     n += 1
```

Rešitev a in rešitev b se razlikujeta zgolj v **vrstici 24** (če smo natančni, je drugačen tudi komentar v **vrstici 21**). Tokrat je izpis s funkcijo *print* nekoliko bolj težaven.

Če želimo smrečico poravnati desno, bomo morali na začetku vrstice izpisovati prazna mesta. Vemo, da:

- bodo v zadnji izpisani vrstici izpolnjena vsa mesta in ne bo prisotnih praznih mest,
- bodo v prvi vrstici prazna vsa izpisana mesta razen enega (skrajno desnega),
- bo vseh izpisanih vrstic toliko, kot znaša spremenljivka *stevilo*,
- funkcija *print* izpiše več posameznih vrednosti, ki jih ločimo med sabo z vejico, med njimi pa funkcija sama izpiše niz znakov, določen z vrednostjo parametra *sep* (privzeta vrednost parametra *sep* = " " (prazno mesto)),
- funkcija *print* omogoča, da nastavimo znak, ki ga vstavi med posamezne parametre znotraj funkcije.

Glede na vse navedeno se lotimo izpisa:

- Najprej izpišemo število praznih mest. Praznih mest je toliko, kot je vrednost spremeljivke *stevilo*, zmanjšana za številko vrstice, v kateri se nahajamo, torej za vrednost spremenljivke *n*. Število praznih mest " " torej množimo z vrednostjo _(stevilo - n).
- Nato izpišemo zvezdice. Zvezdic izpišemo toliko, kot je številka vrstice, v kateri se nahajamo, torej kot je vrednost spremenljivke *n*. Na tem mestu se lahko testiramo. Če izpisujemo 10 vrstic, bo v prvi vrstici 9 praznih mest in 1 polno, v drugi 8 praznih in 2 polni itd. Algoritem torej deluje.
- Ostane še tretji parameter, to je *sep=""*. Z njim nastavimo funkcijo *print* tako, da med posameznimi parametri v izpisu namesto praznega mesta (" ") ne vstavi nič. Tako imamo celotno kontrolo izpisa.

Primer izpisa:

```
**********************
* Risanje smrečice *
**********************
Vnesi višino smrečice: 5
    *
   **
  ***
 ****
*****
```

2.16.3. Rešitev #3: smrečica, poravnana sredinsko

Izvorna koda 39. naloga016c.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga016c.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 print("*"*21)
5 print("* Risanje smrečice *")
6 print("*"*21)
7
8 # uporabnika prosimo, naj vpiše število
9 stevilo = input('Vnesi višino smrečice: ')
```

```
10
11 try:
12     stevilo = int(stevilo)
13 except:
14     print("Niste vnesli celega števila!")
15     quit()
16
17 if stevilo < 0:
18     stevilo = -stevilo
19     print(f"-{stevilo} sem za vas pretvoril v {stevilo}.")
20
21 # poravnano sredinsko
22 n = 1
23 while n <= stevilo:
24     stevilo_zvezdic = 2 * n - 1
25     najvecje_stevilo = 2 * stevilo - 1
26     stevilo_praznih = (najvecje_stevilo - stevilo_zvezdic) / 2
27     stevilo_praznih = int(stevilo_praznih)
28     print(" "*stevilo_praznih,"*"*stevilo_zvezdic,sep="")
29     n += 1
```

Ta primer se od rešitve a in rešitve b spet razlikuje zgolj od **vrstice 24** dalje (če znova zanemarimo komentar v **vrstici 21**). V tem primeru moramo v vsaki vrstici izračunati število praznih mest na začetku tako, da bo zvezdica izpisana na sredini.

Tabela 18. Razlaga

vrstica	komentar
24	Vpeljemo spremenljivko *stevilo_zvezdic* in mu priredimo vrednost *2 * n - 1*. To storimo zato, ker v izpisu zvezdic ne moremo poravnati drugače kot eno nad drugo. To pomeni, da bomo imeli v drugi vrstici izpisa namesto dveh smrečic 3, v tretji 5 itd.
25	Izračunamo največje število zvezdic (spremenljivka *najvecje_stevilo*), ki ga dobimo tako, da na mesto vrednosti spremenljivke *n* vstavimo vrednost spremenljivke *stevilo*. Tako dobimo število zapolnjenih mest v zadnji vrstici, kar bo podlaga za centriranje ostalih izpisov.

vrstica	komentar
26	Izračunamo število praznih mest na začetku vrstice (spremenljivka *stevilo_praznih*), ker moramo ta prazna mesta izpisati. To storimo tako, da od največjega števila zapolnjenih mest (spremenljivka *najvecje_stevilo*) odštejemo število zvezdic (spremenljivka *stevilo_zvezdic*), potem pa to delimo z 2. Zakaj? Ker je polovica teh praznih mest levo od zvezdic, polovica pa desno.
27	Uporabimo funkcijo *int*, ki poskrbi, da je dobljeno število (spremenljivka *stevilo_praznih*) vedno celo. Ta vrstica je do neke mere nepotrebna, saj nam prejšnji dve formuli vedno vrneta liho število, razlika dveh lihih števil pa je vedno soda, a tako smo lahko še dodatno prepričani, da bo dobljena vrednost res cela in da se program ne bo končal z napako.
28	Ostane nam zgolj še izpis. Če ste pozorni, izpisujemo prazna mesta zgolj levo, ne pa desno. Če želite, jih lahko izpišete tudi desno, a razlika prostemu očesu ne bo vidna (saj boste izpisovali prazna mesta).

Primer izpisa:

```
**********************
* Risanje smrečice *
**********************
Vnesi višino smrečice: 5
    *
   ***
  *****
 *******
*********
```

2.17. Obrat naravnega števila

Sestavi program, ki prebere naravno število in ga izpiše v obratnem vrstnem redu.

2.17.1. Rešitev #1: splošna rešitev

Izvorna koda 40. naloga017a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga017a.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """ Sestavi program, ki prebere naravno število in ga izpiše v
  obratnem vrstnem redu.  """
5
6 # Pozovemo uporabnika, naj poda število
7 stevilo = input("Podajte število, ki ga želite obrniti: ")
8
9 # preštejemo, koliko znakov ima niz
10 dolzina = len(stevilo)
11
12 n = 1
13 obrat = ""
14 while n <= dolzina:
15     obrat += stevilo[dolzina-n]
16     n+=1
17
18 # izpišemo obratno število
19 print(f"Obrat števila '{stevilo}' je '{obrat}'.")
```

Primer izpisa:

```
Podajte število, ki ga želite obrniti: 12345
Obrat števila '12345' je '54321'.
```

Tabela 19. Razlaga

vrstica	komentar
7	Uporabnika pozovemo, naj poda število, ki ga želi obrniti.
10	Da bomo lahko uporabili zanko, moramo vedeti, kako dolgo je število (koliko števk ima), zato uporabimo funkcijo *len()*. Pri tem velja opozorilo, da čeprav smo uporabnika pozvali k vpisu števila, lahko vpiše karkoli in ker funkcija *input* vrne niz znakov, lahko uporabimo funkcijo *len*, sicer bi se morali znajti drugače.
12	Uvedemo spremenljivko *n*, ki jo bomo potrebovali za premikanje od prve števke do zadnje. Spremenljivki *n* določimo vrednost 1.
13	Uvedemo spremenljivko *obrat*, ki jo bomo potrebovali za sestavljanje novega števila. Spremenljivki *obrat* določimo začetno vrednost "" (prazen niz znakov).
14	Nastavimo zanko, ki naj se izvaja, dokler spremenljivka *n* ne bo manjša ali enaka spremenljivki *dolzina*. Z drugimi besedami: zanka naj se izvaja, dokler ne bomo prišli od prvega znaka niza do zadnjega.
15	Spremenljivki *obrat* dodamo znak, ki se nahaja za *n* mest stran od zadnjega znaka. Trik v tem koraku je, da lahko vsak niz znakov (*string*) uporabljamo kot seznam (*list*) posameznih znakov. Tako je potem neko število seznam, v katerem se nahajajo posamezne števke celotnega števila. Če potem obračamo recimo številko *12345*, se lahko premikamo po posameznih števkah: *1*, *2*, *3*, *4* in *5*. V zanki bomo tako v prvem koraku spremenljivki *obrat* dodali števko *5*, ki pa je na mestu, označenem z *indeksom* 4. Zakaj 4? Ker se mesta v seznamih začnejo šteti z 0 in ne 1. Pri številki *12345* je torej števka *1* na mestu *0*.
16	Premaknemo se za 1 števko naprej.
19	Izpišemo rezultat.

Tako pripravljen program deluje za kateri koli vpisani niz znakov, kar pomeni, da ne deluje tako, kot bi si želeli.

```
Podajte število, ki ga želite obrniti: FGG
Obrat števila 'FGG' je 'GGF'.
```

Če želimo, da bo program deloval samo za naravna števila, ga moramo dopolniti.

2.17.2. Rešitev #2: primer rešitve, ki deluje samo za cela števila

Izvorna koda 41. naloga017b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga017b.py]

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  """ Sestavi program, ki prebere naravno število in ga izpiše v
   obratnem vrstnem redu.  """
5
6  ## OPOMBA: deluje samo za cela števila
7
8  # Pozovemo uporabnika, naj poda število
9  stevilo = input("Podajte število, ki ga želite obrniti: ")
10
11 try:
12     tmp = int(stevilo)
13 except:
14     print("Niste vnesli celega števila!")
15     quit()
16
17 # preštejemo, koliko znakov ima niz
18 dolzina = len(stevilo)
19
20 n = 1
21 obrat = ""
22 while n <= dolzina:
23     obrat += stevilo[dolzina-n]
24     n+=1
25
26 # izpišemo vsoto
27 print(f"Obrat števila '{stevilo}' je '{obrat}'.")
```

Rešitev a dopolnimo s preverjanjem tipa spremenljivke (**vrstice 11-15**). Če uporabnik ne vpiše naravnega (celega) števila, se program z obvestilom o napačnem vpisu ustavi. Primer izpisa ob napaki:

```
Podajte število, ki ga želite obrniti: FGG
Niste vnesli celega števila!
```

Tako pripravljen program deluje za katero koli vpisano celo število, tudi negativno, ki ga bo izpisal celo napačno.

```
Podajte število, ki ga želite obrniti: -123
Obrat števila '-123' je '321-'.
```

To pomeni, da moramo program še dopolniti.

2.17.3. Rešitev #3: primer rešitve, ki deluje samo za naravna števila

Izvorna koda 42. naloga017c.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga017c.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """ Sestavi program, ki prebere naravno število in ga izpiše v
  obratnem vrstnem redu.  """
5
6 ## OPOMBA: deluje samo za cela števila
7
8 # Pozovemo uporabnika, naj poda število
9 stevilo = input("Podajte število, ki ga želite obrniti: ")
10
11 if stevilo.isdigit() == False:
12     print("Niste vnesli naravnega števila!")
13     quit()
14
15 # preštejemo, koliko znakov ima niz
16 dolzina = len(stevilo)
17
18 n = 1
19 obrat = ""
20 while n <= dolzina:
21     obrat += stevilo[dolzina-n]
22     n+=1
23
24 # izpišemo vsoto
25 print(f"Obrat števila '{stevilo}' je '{obrat}'.")
```

Čeprav bi lahko nadgradili rešitev b s pogojnim stavkom, ki bi preverjal, ali je vneseno število pozitivno ali negativno, smo raje predstavili rešitev z uporabo metode *isdigit()* (**vrstice 11-13**). Z njo preverimo, če so vsi znaki v nizu številke. Če se na primer v podanem nizu nahaja znak - ali pa recimo črka, decimalna vejica ali pika, metoda vrne vrednost *False*. Če metoda vrne vrednost *True*, smo lahko prepričani, da se v podanem nizu nahajajo samo številke. Druge kontrole niso potrebne. Izpis je popolnoma enak prejšnjemu.

2.17.4. Rešitev #4: primer rešitve, ki deluje samo za naravna števila (krajši način)

Izvorna koda 43. naloga017d.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga017d.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """ Sestavi program, ki prebere naravno število in ga izpiše v
  obratnem vrstnem redu.  """
5
6 # Pozovemo uporabnika, naj poda število
7 stevilo = input("Podajte naravno število, ki ga želite obrniti: ")
8
9 if stevilo.isdigit() == False:
10     print("Niste vnesli naravnega števila!")
11     quit()
12
13 # obrnemo
14 obrat = stevilo[::-1]
15
16 # izpišemo vsoto
17 print(f"Obrat števila '{stevilo}' je '{obrat}'.")
```

Ta rešitev je najkrajša in izkorišča možnosti in bližnjice programskega jezika Python.

Opazimo lahko, da je namesto zanke, s katero smo operirali v prvih treh primerih, v **vrstici 14** uporabljen poseben ukaz oziroma operator **[::-1]**. S tem ukazom povemo, da želimo ustvariti nov niz znakov, ki se začne na koncu prvotnega niza in se pomika proti začetku (to je poziciji *0*) s korakom *-1*, torej nazaj (v drugem posebnem primeru bi se lahko pomikal tudi naprej, odvisno od tega, kaj želimo doseči). Izpis je drugače enak prejšnjemu.

2.18. Je besedilo palindrom?

Sestavi program, v katerem uporabnik vpiše niz znakov, program pa pove, ali je vpisani niz znakov palindrom ali ne.

Naloga je zelo podobna nalogi 17, predvsem rešitvi a, a zahteva dodatne kontrole.

2.18.1. Rešitev #1: (polovična) rešitev

Izvorna koda 44. naloga018a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga018a.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 print("**********************************")
5 print("*** Ali je besedilo palindrom? ***")
6 print("**********************************")
7
8 # pozovemo uporabnika, naj vpiše besedilo
9 besedilo = input("Vpiši potencialen palindrom: ")
10
11 # obrnemo besedilo
12 obrnjeno_besedilo = besedilo[::-1]
13
14 # primerjamo besedili
15 if besedilo == obrnjeno_besedilo:
16     print(f"Besedilo '{besedilo}' JE palindrom!")
17 else:
18     print(f"Besedilo '{besedilo}' NI palindrom!")
```

Primer izpisa:

```
**********************************
*** Ali je besedilo palindrom? ***
**********************************
Vpiši potencialen palindrom: pericareženeracirep
Besedilo 'pericareženeracirep' JE palindrom!
```

```
**********************************
*** Ali je besedilo palindrom? ***
**********************************
Vpiši potencialen palindrom: perica reže raci rep
Besedilo 'perica reže raci rep' NI palindrom!
```

Tabela 20. Razlaga

vrstica	komentar
9	Uporabnika pozovemo, naj poda besedilo, ki ga bomo preverjali.
12	Podano besedilo obrnemo, podobno, kot smo storili pri rešitvi d naloge 17.
15-18	Preverimo, če sta vrednosti spremenljivk *besedilo* ter *obrnjeno_besedilo* enaki, ter na podlagi primerjave izpišemo, da besedilo je ali pa ni palindrom.

Kot vidimo, tako pripravljen program sicer deluje, a ne tako, kot smo vajeni. Kot vidimo zgoraj, za besedilo 'pericareženeracirep' program pravilno ugotovi, da je palindrom, a tega ne ugotovi, če besedilo zapišemo v obliki 'perica reže raci rep', saj moramo presledke postaviti drugam. Zato rešitev izboljšajmo.

2.18.2. Rešitev #2: (končna) rešitev

Izvorna koda 45. naloga018b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga018b.py]

```
 1 #!/usr/bin/env python3
 2 # -*- coding: utf-8 -*-
 3
 4 """
 5 Uporabnik vpiše niz znakov, program pa pove, ali je vpisani niz
   palindrom ali ne.
 6 Palindrom je niz znakov, ki se enako prebere z leve in z desne.
 7 """
 8
 9 ## tole deluje: pericareŽeracirep
10 ## tole ne deluje: perica reže raci rep
11
12 print("**********************************")
```

```
13 print("*** Ali je besedilo palindrom ***")
14 print("*********************************")
15
16
17 # pozovemo uporabnika, naj vpiše besedilo
18 besedilo = input("Vpiši potencialen palindrom: ")
19
20 # odstranimo presledke in vse velike črke spremenimo v male
21 cisto_besedilo = besedilo.replace(" ","").lower()
22 cisto_besedilo = cisto_besedilo.replace(".","").replace(","",""
   ).replace("!","")
23
24 # obrnemo besedilo
25 obrnjeno_besedilo = cisto_besedilo[::-1]
26
27 # primerjamo besedili
28 if cisto_besedilo == obrnjeno_besedilo:
29     print(f"Besedilo '{besedilo}' JE palindrom!")
30 else:
31     print(f"Besedilo '{besedilo}' NI palindrom!")
```

Rešitev a popravimo tako, da dodamo **vrstici 21 in 22**. V **vrstici 21** najprej vpeljemo spremenljivko *cisto_besedilo*. Tej spremenljivki nato priredimo vrednost spremenljivke *besedilo*, ki ji najprej odstranimo presledke (s pomočjo metode *replace()*), nato pa še vse črke zmanjšamo (s pomočjo metode *lower()*).

Ker razmišljamo vnaprej, v **vrstici 22** nizu znakov odstranimo še ločila. Pozorni moramo biti še na **vrstici 25 in 28**, saj operiramo s spremenljivko *cisto_besedilo* in ne *besedilo*. Ostalo ostane enako, program pa sedaj deluje pravilno.

Primer izpisa:

```
*********************************
*** Ali je besedilo palindrom ***
*********************************
Vpiši potencialen palindrom: Perica Reže Raci REP!
Besedilo 'Perica Reže Raci REP!' JE palindrom!
```

2.19. Največji skupni delitelj podanih števil

Sestavi program, ki izračuna največji skupni delitelj števil a in b in izpiše vse vmesne rezultate Evklidovega algoritma.

2.19.1. Rešitev #1: rešitev

Izvorna koda 46. naloga019a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga019a.py]

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import math
5
6  # Izpišemo, za kakšen program gre
7  print("*"*60)
8  print("*"*5,"Program za izračun največjega skupnega delitelja","*
   "*5)
9  print("*"*60)
10
11 # definirajmo funkcijo, ki nam bo preverila, če sta vnešeni
   števili celi
12 def preveriStevilke(stevilka):
13     try:
14         stevilka = int(stevilka)
15         return stevilka
16     except:
17         print(f"'{stevilka}' ni številka.")
18         quit()
19
20 # Pozovemo uporabnika, naj poda dve celi števili
21 stevilo1 = input("Podajte prvo celo število: ")
22 # preverimo števili
23 stevilo1 = preveriStevilke(stevilo1)
24
25 stevilo2 = input("Podajte drugo celo število: ")
26 stevilo2 = preveriStevilke(stevilo2)
27
```

```
28 print("-"*60)
29
30 # določimo spremenljivki a in b
31 a,b = stevilo1,stevilo2
32
33 # Evklidov algoritem
34 while b:
35     n = math.floor(a/b)
36     print(f"{a} = {n} * {b} + {a%b}")
37     a,b=b,a%b
38 print(f"Največji delitelj števil {stevilo1} in {stevilo2} je {a}.
   ")
```

Tabela 21. Razlaga

vrstica	komentar
12-18	Definiramo funkcijo *preveriStevilke()*, ki preveri, če sta vneseni števili celi. Funkcijo pripravimo zato, ker bomo preverjali 2 vnosa. Če uporabnik ne vnese celih števil, funkcija vrne napako in ustavi izvajanje.
21,25	Uporabnika pozovemu k vnosu števil, za kateri bomo iskali največji skupni delitelj.
23,26	Preverimo, če je uporabnik res vnesel števili. Pri tem uporabimo funkcijo *preveriStevilke()*.
31	Uvedemo novi spremenljivki *a* in *b*, za zamenjavo pa uporabimo dvojno prirejanje. Namesto *a,b=stevilo1,stevilo2* v **vrstici 32** bi denimo lahko napisali *a = stevilo1* in v novo vrstico *b = stevilo2*.
34-37	Uporabimo Evklidov algoritem. Zanj uporabimo zanko *while*, ki se izvaja tako dolgo, dokler ima spremenljivka *b* vrednost, ki je večja od 0. Uporabimo tudi funkcijo *math.floor()*, ki količnik *(a/b)* zaokroži na najbližje celo število navzdol. Bodite pozorni, da smo zaradi uporabe te funkcije v **vrstici 4** morali uvoziti modul *math*. Morda spomnimo še to, da nam izraz *a%b* vrne ostanek pri deljenju *a* z *b*.
38	Ko se zanka zaključi (ko je b = 0, kar pomeni, da je a največji delitelj, saj ostanka pri deljenju v prejšnjem koraku ni bilo), izpišemo rezultat.

Primer izpisa:

```
************************************************************
***** Program za izračun največjega skupnega delitelja *****
************************************************************
Podajte prvo celo število: 27
Podajte drugo celo število: 63
------------------------------------------------------------
63 = 2 * 27 + 9
27 = 3 * 9 + 0
Največji delitelj števil 27 in 63 je 9.
```

2.19.2. Rešitev #2: rešitev za več kot 2 podani števili

Izvorna koda 47. naloga019b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga019b.py]

```
 1 #!/usr/bin/env python3
 2 # -*- coding: utf-8 -*-
 3
 4 import math
 5
 6 # Izpišemo, za kakšen program gre
 7 print("*"*60)
 8 print("*"*5,"Program za izračun največjega skupnega delitelja","*
   "*5)
 9 print("*"*60)
10
11 # definirajmo funkcijo, ki nam bo preverila, če sta vnešeni
   števili celi
12 def preveriStevilke(stevilka):
13     try:
14         stevilka = int(stevilka)
15         return stevilka
16     except:
17         print(f"'{stevilka}' ni število.")
18         quit()
19
20 # Pozovemo uporabnika, naj poda števila
21 stevila = input("Podajte števila (ločite jih z vejico): ")
```

```
22
23 # sestavimo seznam s števili
24 stevila_string = stevila.split(",")
25
26 # uvedemo seznam stevila
27 stevila = []
28
29 # preverimo števila in jih dodamo v seznam, če ni napake
30 for i in stevila_string:
31     i = preveriStevilke(i.replace(" ",""))
32     stevila.append(i)
33
34 # izpišemo vnešena števila
35 print(f"Vnesli ste števila: {stevila}")
36 print("-"*60)
37
38 i = 1
39 while i < len(stevila):
40     if i == 1:
41         a,b = stevila[i],stevila[i-1]
42     else:
43         a,b = stevila[i],delitelj
44     a_original,b_original = a,b
45
46     print(f"Preglejujem števili {a} in {b}.")
47     # Evklidov algoritem
48     while b:
49         n = math.floor(a/b)
50         print(f"{a} = {n} * {b} + {a%b}")
51         a,b=b,a%b
52     delitelj = a
53     print(f"Največji skupni delitelj števil {a_original} in
   {b_original} je {delitelj}.")
54     print("-"*60)
55     i += 1
56
57 print(f"Največji skupni delitelj števil {', '.join(stevila_string
   )} je {delitelj}.")
```

Ta rešitev temelji na rešitvi a z dvema bistvenima popravkoma:

- pri vnosu uporabnik števila vnese tako, da jih loči z vejicami,
- iskanje največjega skupnega delitelja za več kot 2 števili.

Tabela 22. Razlaga bistvenih delov

vrstica	komentar
24	Vpeljemo spremenljivko *stevila_string* in ji priredimo seznam, ki ga pripravimo tako, da vpisane vrednosti (spremenljivka *stevila*) ločimo po vejicah.
27	Prejšnjo spremenljivko *stevila* spremenimo v prazen seznam.
30-32	Preverimo vse vrednosti v seznamu *stevila_string*, če so res števila. Če naletimo na vrednost, ki ni celo število, se program ustavi.
31	Preverjamo trenutno število, a pri tem iz vrednosti spremenljivke *i* odstranimo presledke. Tako uvedemo še eno kontrolo in uporabniku dovolimo, da pri naštevanju za vejicami uporablja presledke.
32	Če se program pri preverjanju vrednosti v **vrstici 31** ni ustavil (če se ni ustavil zaradi kontrol v definirani funkciji *preveriStevilke*), potem dobljeno število dodamo na seznam *stevila*.
38-55	V tem sklopu izvajamo evklidov algoritem, kot smo to počeli v rešitvi a. Razlika je v tem, da ne vemo, koliko števil bo uporabnik vpisal, zato se po seznamu premikamo od začetka do konca. Trik je v tem, da najprej iščemo največji delitelj prvih dveh vpisanih števil, v vsakem naslednjem koraku pa preverjamo zgolj največji skupni delitelj med naslednjim številom ter deliteljem iz prejšnjega koraka. Spremenljivki *a_original* ter *b_original*, ki ju uvedemo v **vrstici 44**, sta namenjeni zgolj izpisu v **vrstici 53**, saj bi drugače začetni vrednosti izgubili.
57	Pri izpisu uporabimo metodo *join()*, ki elemente podanega seznama pretvori v niz znakov tako, da med posamezne elemente vstavi niz, ki mu ga podamo (v našem primeru je to ", ").

Primer izpisa:

```
************************************************************
***** Program za izračun največjega skupnega delitelja *****
************************************************************
Podajte števila (ločite jih z vejico): 14,77, 35,  777
Vnesli ste števila: [14, 77, 35, 777]
------------------------------------------------------------
Preglejujem števili 77 in 14.
77 = 5 * 14 + 7
14 = 2 * 7 + 0
Največji skupni delitelj števil 77 in 14 je 7.
------------------------------------------------------------------------
---------------------
Preglejujem števili 35 in 7.
35 = 5 * 7 + 0
Največji skupni delitelj števil 35 in 7 je 7.
------------------------------------------------------------------------
---------------------
Preglejujem števili 777 in 7.
777 = 111 * 7 + 0
Največji skupni delitelj števil 777 in 7 je 7.
------------------------------------------------------------------------
---------------------
Največji skupni delitelj števil 14, 77,  35,   777 je 7.
```

2.20. Ugibanje celega števila med 0 in 100

Sestavi program, s katerim ugibamo celo število med 0 in 100.

Program naj si naključno izbere število med 0 in 100, uporabnik pa naj v čim manj poskusih ugane, katero število je izbral.

2.20.1. Rešitev #1: rešitev brez pomoči

Izvorna koda 48. naloga020a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga020a.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 print("*********************************")
5 print("***** Igra: ugani številko! *****")
6 print("*********************************")
7
8 # vključim modul random, da bom lahko izbral naključen integer
9 import random
10
11 # izberem iskano številko
12 iskanaStevilka = random.randint(0,100)
13
14 steviloPoskusov = 0
15
16 #nastavim začetno število na vrednost, ki zagotovo ni enaka iskani
17 stevilka = -1
18
19 while stevilka != iskanaStevilka:
20     stevilka = input("Ugani številko med 0 in 100: ")
21     steviloPoskusov += 1
22     if stevilka.isnumeric():
23         stevilka = int(stevilka)
24         if stevilka != iskanaStevilka:
25             print(f"Poskus št. {steviloPoskusov}. {stevilka} ni
   prava številka.")
26             print(f"Poskusi znova!")
27         elif stevilka == iskanaStevilka:
```

```
28              print(f"Bravo! V {steviloPoskusov}. poskusu si uganil
    številko {stevilka}!")
29              print(f"Igra je končana!")
30      else:
31          print(f"Poskus št. {steviloPoskusov}. Vnesel si '{
    stevilka}', kar ni cela številka.")
32          print(f"Poskusi znova!")
```

Tabela 23. Razlaga

vrstica	komentar
9	Vključimo modul *random*, saj ga bomo potrebovali, da si bo program izbral naključno število.
12	Uvedemo novo spremenljivko *iskanaStevilka* in ji določimo naključno vrednost med 0 in 100. Za to uporabimo funkcijo *randint()*, ki je del modula *random*.
14	Uvedemo spremenljivko *steviloPoskusov* in določimo začetno število poskusov 0.
17	Uvedemo spremenljivko *stevilka* in ji določimo začetno vrednost, ki zagotovo ni enaka iskani.
19	Začnemo izvajati zanko *while*, ki se izvaja tako dolgo, dokler število, ki ga uporabnik vnese, ni enako številu, ki si ga je izbral program.
20	Uporabnika pozovemo, naj vpiše številko med 0 in 100.
21	Število poskusov povečamo za 1.
22	Če vnesena vrednost vsebuje samo številke, gremo naprej, v nasprotnem primeru se izvede **vrstica 31** in se uporabnik pozove k vpisu nove številke.
23	Tip spremenljivke *stevilka* spremenimo v tip *integer*, saj iz **vrstice 22** vemo, da so vnesene samo številke.
24-29	Preverjamo, če je vnesena vrednost enaka iskani. Če je, program končamo, če ni, se program izvaja naprej.

Primer izpisa:

```
**********************************
***** Igra: ugani številko! *****
**********************************
Ugani številko med 0 in 100: 0
Poskus št. 1. 0 ni prava številka.
Poskusi znova!
Ugani številko med 0 in 100: 1
Poskus št. 2. 1 ni prava številka.
Poskusi znova!
Ugani številko med 0 in 100: 2
Poskus št. 3. 2 ni prava številka.
Poskusi znova!
Ugani številko med 0 in 100: 3
Bravo! V 4. poskusu si uganil številko 3!
Igra je končana!
```

2.20.2. Rešitev #2: rešitev s pomočjo

Izvorna koda 49. naloga020b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga020b.py]

```
 1 #!/usr/bin/env python3
 2 # -*- coding: utf-8 -*-
 3
 4 print("**********************************")
 5 print("***** Igra: ugani številko! *****")
 6 print("**********************************")
 7
 8 # vključim modul random, da bom lahko izbral naključen integer
 9 import random
10
11 # izberem iskano številko
12 iskanaStevilka = random.randint(0,100)
13
14 steviloPoskusov = 0
15
16 stevilka = -1
17
```

```
18 while stevilka != iskanaStevilka:
19     stevilka = input("Ugani število med 0 in 100: ")
20     steviloPoskusov += 1
21     if stevilka.isnumeric():
22         stevilka = int(stevilka)
23         if stevilka != iskanaStevilka:
24             if stevilka > iskanaStevilka:
25                 pomoc = "Iskana številka je manjša!"
26             else:
27                 pomoc = "Iskana številka je večja!"
28             print(f"Poskus št. {steviloPoskusov}. {stevilka} ni
   prava številka.")
29             print(f"{pomoc} Poskusi znova!")
30         elif stevilka == iskanaStevilka:
31             print(f"Bravo! V {steviloPoskusov}. poskusu si uganil
   številko {stevilka}!")
32             print(f"Igra je končana!")
33         print("-"*33)
34     else:
35         print(f"Poskus št. {steviloPoskusov}. Vnesel si '{
   stevilka}', kar ni cela številka.")
36         print(f"Poskusi znova!")
```

Pri tej rešitvi uporabniku pomagamo tako, da mu v vsakem koraku povemo, ali je iskana številka manjša ali večja od vpisane. Tako ga usmerjamo k rešitvi.

Od rešitve a se razlikuje zgolj v **vrsticah 24 do 27** (ter izpisu pomoči v **vrstici 29**).

Primer izpisa:

```
**********************************
***** Igra: ugani številko! *****
**********************************
Ugani številko med 0 in 100: 3
Poskus št. 1. 3 ni prava številka.
Iskana številka je večja! Poskusi znova!
---------------------------------
Ugani številko med 0 in 100: 50
Poskus št. 2. 50 ni prava številka.
Iskana številka je manjša! Poskusi znova!
```

```
---------------------------------
Ugani številko med 0 in 100: 30
Bravo! V 3. poskusu si uganil številko 30!
Igra je končana!
---------------------------------
```

2.21. Ali trikotnik obstaja?

Sestavi program, ki bo prebral tri realna števila, nato pa preveril, ali obstaja trikotnik s takimi dolžinami stranic. Če obstaja, naj program izračuna njegovo ploščino in obseg.

2.21.1. Rešitev #1: rešitev z vnosom vsake stranice posebej

Izvorna koda 50. naloga021a.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga021a.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 # Izpišemo, za kakšen program gre
5 print("*"*64)
6 print("*"*5,"Karakteristike trikotnika","*"*5)
7 print("*"*64)
8
9 # Podamo stranice pravokotnika
10 a = input('Vnesite dolžino stranice a: ')
11 b = input('Vnesite dolžino stranice b: ')
12 c = input('Vnesite dolžino stranice c: ')
13
14 # definirajmo funkcijo, ki nam bo preverila vnesene stranice
15 def preveriStevilke(stevilka):
16     try:
17         stevilka = float(stevilka)
18         return stevilka
19     except:
20         print(f"'{stevilka}' ni število.")
21         quit()
22
23 print("-"*64)
24
25 # poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
26 a = preveriStevilke(a)
27 b = preveriStevilke(b)
28 c = preveriStevilke(c)
29
```

```
# uvedemo spremenljivko napaka tipa 'boolean', ki ima privzeto vrednost 'false'
napaka = False

# preverimo, če trikovnik s podanimi stranicami obstaja
if a + b > c:
    print(f"a + b > c: Pogoj JE izpolnjen -> {a} + {b} > {c}")
else:
    print(f"a + b > c: Pogoj NI izpolnjen -> {a} + {b} < {c}")
    napaka = True

if b + c > a:
    print(f"b + c > a: Pogoj JE izpolnjen -> {b} + {c} > {a}")
else:
    print(f"b + c > a: Pogoj NI izpolnjen -> {b} + {c} < {a}")
    napaka = True

if a + c > b:
    print(f"a + c > b: Pogoj JE izpolnjen -> {a} + {c} > {b}")
else:
    print(f"a + c > b: Pogoj NI izpolnjen -> {a} + {c} < {b}")
    napaka = True

print("-"*64)
if napaka == True:
    print(f"Trikotnik s podanimi stranicami {a}, {b} in {c} ne obstaja.")
else:
    obseg = a + b + c

    #polobseg
    s = obseg / 2

    # izračun površine po Heronovi formuli
    povrsina = (s*(s-a)*(s-b)*(s-c))**0.5

    # izpis
    print(f"Trikotnik s podanimi stranicami {a}, {b} in {c} ima obseg {obseg} in površino {povrsina}.")
```

Tabela 24. Razlaga

vrstica	komentar
10-12	Uporabnika pozovemo k vnosu vseh treh stranic in vpisane vrednosti priredimo spremenljivkam *a*, *b* in *c*.
15-21	Definiramo funkcijo *preveriStevilke()*, ki za vpisano vrednost preveri, ali gre za decimalno število ali ne. Najprej poskusi vpisani niz znakov pretvoriti v decimalno število (z uporabo funkcije *float()*, **vrstica 17**). Če je pretvorba uspešna, vrne številko (**vrstica 18**), v nasprotnem primeru izpiše napako (**vrstica 20**) in program ustavi (**vrstica 21**).
26-28	Za vsako od spremenljivk (*a*, *b* in *c*) preverimo, če gre za število.
31	Uvedemo spremenljivko *napaka*, ki je tipa *boolean* in ima privzeto vrednost *False*. V nadaljevanju jo bomo uporabili za preverjanje vseh pogojev.
34-50	Preverimo, če trikotnik s podanimi dolžinami stranic obstaja. Pri vsakem od pogojev izpišemo, če je pogoj izpolnjen. Če ni izpolnjen, vrednost spremenljivke *napaka* spremenimo v *True*, kar pomeni, da trikotnik s podanimi dolžinami stranic ne obstaja.
53-65	Preverimo, če je vrednost spremenljivke *napaka* enaka *True* (**vrstica 53**). V tem primeru izpišemo obvestilo o tem, da trikotnik s podanimi stranicami ne obstaja (**vrstica 54**). Če vrednost spremenljivke *napaka* ni enaka *True* (kar pomeni, da je enaka *False*), to pomeni, da tak trikotnik obstaja, zato zanj izračunamo obseg (**vrstica 56**) in površino (**vrstica 62**) ter rezultat z obvestilom izpišemo (**vrstica 65**).

Primer izpisa, če trikotnik obstaja:

```
****************************************************************
***** Karakteristike trikotnika *****
****************************************************************
Vnesite dolžino stranice a: 3
Vnesite dolžino stranice b: 4
Vnesite dolžino stranice c: 5
----------------------------------------------------------------
a + b > c: Pogoj JE izpolnjen -> 3.0 + 4.0 > 5.0
b + c > a: Pogoj JE izpolnjen -> 4.0 + 5.0 > 3.0
a + c > b: Pogoj JE izpolnjen -> 3.0 + 5.0 > 4.0
----------------------------------------------------------------
```

```
Trikotnik s podanimi stranicami 3.0, 4.0 in 5.0 ima obseg 12.0 in
površino 6.0.
```

Primer izpisa ob napaki:

```
**********************************************************************
***** Karakteristike trikotnika *****
**********************************************************************
Vnesite dolžino stranice a: 1
Vnesite dolžino stranice b: 2
Vnesite dolžino stranice c: b
----------------------------------------------------------------------
'b' ni številka.
```

Primer izpisa, če trikotnik ne obstaja:

```
**********************************************************************
***** Karakteristike trikotnika *****
**********************************************************************
Vnesite dolžino stranice a: 1
Vnesite dolžino stranice b: 2
Vnesite dolžino stranice c: 3
----------------------------------------------------------------------
a + b > c: Pogoj NI izpolnjen -> 1.0 + 2.0 < 3.0
b + c > a: Pogoj JE izpolnjen -> 2.0 + 3.0 > 1.0
a + c > b: Pogoj JE izpolnjen -> 1.0 + 3.0 > 2.0
----------------------------------------------------------------------
Trikotnik s podanimi stranicami 1.0, 2.0 in 3.0 ne obstaja.
```

2.21.2. Rešitev #2: rešitev z enkratnim vnosom

Izvorna koda 51. naloga021b.py [https://github.com/pzi-si/pzi-src-2/blob/main/naloga021b.py]

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 # Izpišemo, za kakšen program gre
```

```
print("*"*64)
print("*"*5,"Karakteristike trikotnika","*"*5)
print("*"*64)

# Podamo stranice pravokotnika
stranice = input('Vnesite dolžine stranic trikotnika, ki jih loči z vejico: ')

# Podani niz znakov pretvorimo v seznam
stranice = stranice.split(",")

# Preverimo, koliko delov ima seznam
if len(stranice) != 3:
    print("Podali ste več ali manj kot 3 stranice!")
    quit()

# če so tri stranice
a = stranice[0]
b = stranice[1]
c = stranice[2]

# definirajmo funkcijo, ki nam bo preverila vnesene stranice
def preveriStevilke(stevilka):
    try:
        stevilka = float(stevilka)
        return stevilka
    except:
        print(f"{stevilka} ni število.")
        quit()

print("-"*64)

# poskusimo spremeniti tip spremenljivke iz 'string' v 'float'
a = preveriStevilke(a)
b = preveriStevilke(b)
c = preveriStevilke(c)

# uvedemo spremenljivko napaka tipa 'boolean', ki ima privzeto vrednost 'false'
napaka = False
```

```
43
44 # preverimo, če trikovnik s podanimi stranicami obstaja
45 if a + b > c:
46     print(f"a + b > c: Pogoj JE izpolnjen -> {a} + {b} > {c}")
47 else:
48     print(f"a + b > c: Pogoj NI izpolnjen -> {a} + {b} < {c}")
49     napaka = True
50
51 if b + c > a:
52     print(f"b + c > a: Pogoj JE izpolnjen -> {b} + {c} > {a}")
53 else:
54     print(f"b + c > a: Pogoj NI izpolnjen -> {b} + {c} < {a}")
55     napaka = True
56
57 if a + c > b:
58     print(f"a + c > b: Pogoj JE izpolnjen -> {a} + {c} > {b}")
59 else:
60     print(f"a + c > b: Pogoj NI izpolnjen -> {a} + {c} < {b}")
61     napaka = True
62
63 print("-"*64)
64 if napaka == True:
65     print(f"Trikotnik s podanimi stranicami {a}, {b} in {c} ne
   obstaja.")
66 else:
67     obseg = a + b + c
68
69     #polobseg
70     s = obseg / 2
71
72     # izračun površine po Heronovi formuli
73     povrsina = (s*(s-a)*(s-b)*(s-c))**0.5
74
75     # izpis
76     print(f"Trikotnik s podanimi stranicami {a}, {b} in {c} ima
   obseg {obseg} in površino {povrsina}.")
```

Pri tej rešitvi uporabnika ne pozovemo k vnosu vrednosti za vsako stranico posebej, ampak ga pozovemo, da to stori v enem koraku. Od rešitve a je ta različica drugačna od **vrstice 10** do **vrstice 23**.

Tabela 25. Razlaga

vrstica	komentar
10	Uporabnika pozovemo k vnosu vseh treh stranic, ki jih mora ločiti z vejico.
13	Vpeljemo spremenljivko *stranice* in ji priredimo vrednosti, ki jih dobimo tako, da vpisani niz znakov razdelimo po vejicah. Spremenljivka *stranice* je torej seznam (podatkovni tip *list*).
16-18	Preverimo, če ima seznam *stranice* več ali manj elementov kot 3. Če število elementov ne ustreza 3, izpišemo poročilo o napaki (**vrstica 17**) in izvajanje prekinemo (**vrstica 32**).
21-23	Če se izvajanje v **vrstici 32** ne prekine, potem vemo, da ima seznam točno 3 elemente, zato uvedemo spremenljivke *a*, *b* in *c* in jim priredimo prvo (indeks 0, **vrstica 21**), drugo (indeks 1, **vrstica 22**) ter tretjo (indeks 2, **vrstica 23**) vrednost na seznamu *stranice*.

Primer izpisa:

```
**********************************************************************
***** Karakteristike trikotnika *****
**********************************************************************
Vnesite dolžine stranic trikotnika, ki jih loči z vejico: 4,   5,  7
----------------------------------------------------------------------
a + b > c: Pogoj JE izpolnjen -> 4.0 + 5.0 > 7.0
b + c > a: Pogoj JE izpolnjen -> 5.0 + 7.0 > 4.0
a + c > b: Pogoj JE izpolnjen -> 4.0 + 7.0 > 5.0
----------------------------------------------------------------------
Trikotnik s podanimi stranicami 4.0, 5.0 in 7.0 ima obseg 16.0 in
površino 9.797958971132712.
```

Pozor

Če ste pozorni, pri pretvarjanju vnosnega niza znakov najprej v seznam in potem v decimalno število nikjer ne odstranjujemo presledkov. To ni treba, saj za odvečne presledke poskrbi uporabljena funkcija *float()*.

Poglavje 3. Dodatni viri

Vsi dodatni viri so prosto dostopni na spletni strani http://pzi.si. Spodaj je seznam dodatnih virov:

- Izvorna koda programov v knjigi.
- Povezave do opisov uporabljenih konstant in funkcij ter pomembnih razlag:
  - Python3 funkcija *abs()* [https://docs.python.org/3/library/functions.html?highlight=abs#abs]
  - Python3 funkcija *float()* [https://docs.python.org/3/library/functions.html#float]
  - Python3 funkcija *input()* [https://docs.python.org/3/library/functions.html#input]
  - Python3 funkcija *int()* [https://docs.python.org/3/library/functions.html?highlight=int#int]
  - Python3 funkcija *is_integer()* [https://docs.python.org/3/library/stdtypes.html?highlight=is_integer#float.is_integer]
  - Python3 konstanta *math.pi* [https://docs.python.org/3/library/math.html?highlight=math%20pi#math.pi]
  - Python3 funkcija *math.pow()* [https://docs.python.org/3/library/math.html?highlight=math%20pi#math.pow]
  - Python3 funkcija *print()* [https://docs.python.org/3/library/functions.html#print]
  - Python3 funkcija *round()* [https://docs.python.org/3/library/functions.html?highlight=round#round]
  - Python3 funkcija *str()* [https://docs.python.org/3/library/functions.html#func-str]
  - Python3 funkcija *strip()* [https://docs.python.org/3/library/stdtypes.html?highlight=strip#str.strip]
  - Python3 funkcija *quit()* [https://docs.python.org/3/library/constants.html?highlight=quit#quit]
  - Python3 ravnanje z izjemami (try-except) [https://www.w3schools.com/python/python_try_except.asp]
- Povezave na uporabne spletne vire:
  - Programerski krožek na Gimnaziji Vič [https://prog.gimvic.org/krozekDokumentacija/_build/html/python.html]

Robert Klinc, Matevž Dolenc:

Programiranje za inženirje

Rešeni primeri v programskem jeziku Python

Zbirka nalog z naslovom *Programiranje za inženirje: Rešeni primeri v programskem jeziku Python* je smiseln nabor nalog, s katerimi bodo bodoči inženirji pridobili osnovna znanja programiranja v programskem jeziku Python. Avtorja knjige sta se zbirke nalog lotila zelo sistematično in zahtevnost programerskih nalog stopnjujeta od najbolj preprostih do zahtevnejših.
Skupaj sta pripravila 21 nalog, ki jih najprej podata kot izziv za študente, da jih rešijo sami, nato pa jim ponudita rešitve nalog. Menim, da je zbirka nalog dobro izhodišče za vsakogar, ki se želi poučiti o programskem jeziku Python, dobiti občutek zanj in morda na novo pridobljena znanja tudi nadgraditi z bolj zahtevnimi programerskimi pristopi v programskem okolju Python.

prof. dr. Tomislav Levanič

Univerzitetni učbenik *Programiranje za inženirje: Rešeni primeri v programskem jeziku Python* avtorjev Roberta Klinca in Matevža Dolenca je kakovostno pripravljena zbirka nalog za učenje programiranja v Pythonu, namenjena tako začetnikom kot tistim, ki želijo svoje znanje osvežiti ali nadgraditi.
Učbenik se osredotoča na praktično učenje skozi reševanje nalog, ki bralce spodbudi, da se najprej sami lotijo reševanja nalog, nato pa si ogledajo predlagane rešitve, ki so opremljene z obširno razlago in podrobnimi opisi posameznih vrstic kode. Dodana vrednost učbenika je v predstavitvi več možnih rešitev za isti problem, kar bralcu omogoča, da se spozna z različnimi pristopi k programiranju. Učbenik je bil napisan s poudarkom na potrebah študentov pri predmetu Računalništvo in informatika na Fakulteti za gradbeništvo in geodezijo Univerze v Ljubljani, vendar je uporaben tudi za inženirje drugih smeri.

doc. dr. Jaka Dujc

Učbenik *Programiranje za inženirje: Rešeni primeri v programskem jeziku Python* študentom omogoča spoznavanje s programskim jezikom skozi praktične primere, ki jih študent sam rešuje. Pri vsaki nalogi so podane rešitve z razlago posameznih vrstic programske kode, da študent razume sintakso jezika. Podani rešeni primeri so primerni za začetnike kot tudi za tiste, ki osnove Pythona že poznajo. Podane naloge so uporabne in aplikativne za reševanje programerskih izzivov v gradbeništvu.

asist. Anja Brelih

UNIVERZA V LJUBLJANI | FGG Fakulteta za gradbeništvo in geodezijo