Zorka **NOVAK PINTARIČ**
Sanja **POTRČ**
Miloš **BOGATAJ**

Python v kemiji in kemijskem inženirstvu:
Učbenik za predmet

RAČUNALNIŠTVO V KEMIJI

Univerzitetna založba
Univerze v Mariboru

Univerza v Mariboru

Fakulteta za kemijo
in kemijsko tehnologijo

Python v kemiji in kemijskem inženirstvu

Učbenik za predmet Računalništvo v kemiji

Avtorji
Zorka Novak Pintarič
Sanja Potrč
Miloš Bogataj

November 2025

Naslov *Title*
Python v kemiji in kemijskem inženirstvu
Python in Chemistry and Chemical Engineering

Podnaslov *Subtitle*
Učbenik za predmet Računalništvo v kemiji
Textbook for the Course Computer Science in Chemistry

Avtorji *Authors*
Zorka Novak Pintarič
(Univerza v Mariboru, Fakulteta za kemijo in kemijsko tehnologijo)

Sanja Potrč
(Univerza v Mariboru, Fakulteta za kemijo in kemijsko tehnologijo)

Miloš Bogataj
(Univerza v Mariboru, Fakulteta za kemijo in kemijsko tehnologijo)

Recenzija *Review*
Matevž Pogačnik
(Univerza v Ljubljani, Fakulteta za elektrotehniko)

Marko Bračko
(Univerza v Mariboru, Fakulteta za kemijo in kemijsko tehnologijo; Institut Jožef Stefan)

Lektoriranje *Language editing*
Agnes Kojc

Tehnična urednika *Technical editors*
Zorka Novak Pintarič
(Univerza v Mariboru, Fakulteta za kemijo in kemijsko tehnologijo)

Jan Perša
(Univerza v Mariboru, Univerzitetna založba)

Oblikovanje ovitka *Cover designer*
Jan Perša
(Univerza v Mariboru, Univerzitetna založba)

Grafike na ovitku *Cover graphics*
Turning vanes, avtor: NASA, Great Images in NASA, javna domena, 1990

Grafične priloge *Graphic material*
Vsi viri so lastni, če ni navedeno drugače.
Novak Pintarič, Potrč, Bogataj (avtorji), 2025

Založnik *Published by*
Univerza v Mariboru
Univerzitetna založba
Slomškov trg 15, 2000 Maribor, Slovenija
https://press.um.si, zalozba@um.si

Izdajatelj *Issued by*
Univerza v Mariboru
Fakulteta za kemijo in kemijsko tehnologijo
Smetanova ulica 17, 2000 Maribor, Slovenija
https://fkkt.um.si, fkkt@um.si

Izdaja *Edition*
Prva izdaja

Vrsta publikacije *Publication type*
E-knjiga

Izdano *Published at*
Maribor, Slovenija, november 2025

Dostopno na *Available at*
https://press.um.si/index.php/ump/catalog/book/1068

NOO | NAČRT ZA OKREVANJE IN ODPORNOST
50 let Univerza v Mariboru
REPUBLIKA SLOVENIJA
MINISTRSTVO ZA VISOKO ŠOLSTVO, ZNANOST IN INOVACIJE
Financira Evropska unija NextGenerationEU

Ime projekta *Project name* Predlog pilotnih projektov Univerze v Mariboru za zelen in odporen prehod v Družbo 5.0. Zelena kemija za prehod v Družbo 5.0 UM - FKKT UM

Številka projekta *Project number* 3330-22-3521

Financer projekta *Project financier* Republika Slovenija, Ministrstvo za visoko šolstvo, znanost in inovacije; Evropska unija – NextGenerationEU.

Projekt sofinancirata Republika Slovenija, Ministrstvo za visoko šolstvo, znanost in inovacije, in Evropska unija – NextGenerationEU. Projekt se izvaja skladno z načrtom v okviru razvojnega področja Pametna, trajnostna in vključujoča rast, komponente Krepitev kompetenc, zlasti digitalnih in tistih, ki jih zahtevajo novi poklici in zeleni prehod (C3 K5), za ukrep investicija F. Izvajanje pilotnih projektov, katerih rezultati bodo podlaga za pripravo izhodišč za reformo visokega šolstva za zelen in odporen prehod v družbo 5.0: projekt Pilotni projekti za prenovo visokega šolstva za zelen in odporen prehod.

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

004.43:66(0.034.2)

NOVAK-Pintarič, Zorka
Python v kemiji in kemijskem inženirstvu : učbenik za predmet Računalništvo v kemiji [Elektronski vir] / avtorji Zorka Novak Pintarič, Sanja Potrč, Miloš Bogataj. - E-knjiga. - Maribor : Univerza v Mariboru, Univerzitetna založba, 2025

Način dostopa (URL): https://press.um.si/index.php/ump/catalog/book/1068

ISBN 978-961-299-075-6 (PDF)
doi: 10.18690/um.fkkt.7.2025
COBISS.SI-ID 255670275

ISBN 978-961-299-075-6 (pdf)

DOI https://doi.org/10.18690/um.fkkt.7.2025

Cena Brezplačni izvod

Odgovorna oseba založnika *For publisher* Prof. dr. Zdravko Kačič
rektor Univerze v Mariboru

Citiranje *Attribution* Novak Pintarič, Z., Potrč, S., Bogataj, M., (2025). *Python v kemiji in kemijskem inženirstvu: učbenik za predmet Računalništvo v kemiji*. Univerza v Mariboru, Univerzitetna založba. doi: 10.18690/um.fkkt.7.2025

Kazalo

Seznam kratic

IDLE integrirano razvojno in učno okolje (angl. Integrated Development and Learning Environment)

VSC Visual Studio Code

Seznam simbolov

m masa (g)

M molska masa (g/mol)

n množina (mol)

p tlak (Pa)

R splošna plinska konstantna (J/(mol·K))

T temperatura (K)

V volumen (m^3)

1 Uvod

V današnjem času hitro napredujočih tehnologij ima računalništvo ključno vlogo v številnih znanstvenih disciplinah, med drugim tudi v kemiji in kemijskem inženirstvu. Poznavanje računalniškega programiranja študentom kemije in kemijskega inženirstva odpira vrata do novih raziskovalnih metod, analize podatkov ter reševanja kompleksnih problemov. Programski jezik *Python* je na tem področju še posebej uporaben zaradi svoje enostavnosti, fleksibilnosti in obsežnih knjižnic, kot so *NumPy*, *SciPy* ter *Matplotlib*, ki omogočajo učinkovito obdelavo podatkov, simulacije kemijskih procesov in vizualizacijo rezultatov. Tako lahko kemiki in kemijski inženirji s pomočjo programiranja v *Pythonu* poenostavijo raziskovalne postopke, avtomatizirajo rutinska opravila in pridobijo globlje vpoglede v svoje eksperimentalne podatke.

Obvladovanje osnov programiranja je za študente kemije in kemijskega inženirstva izjemno pomembno, čeprav računalništvo ni njihovo primarno področje študija. Razlogi za to so naslednji:

- Analiza podatkov. Kemijske raziskave in procesi pogosto generirajo velike količine podatkov. Programiranje omogoča učinkovito zbiranje, analizo, vizualizacijo in interpretacijo teh podatkov, kar je ključnega pomena za sprejemanje utemeljenih znanstvenih zaključkov.
- Avtomatizacija rutinskih nalog. Številne naloge v kemijskih laboratorijih so ponavljajoče in časovno zahtevne. S programiranjem lahko te procese avtomatizirajo, kar prihrani čas in zmanjša možnost človeških napak.
- Modeliranje in simulacije. Programiranje omogoča ustvarjanje matematičnih modelov in simulacij kemijskih procesov, kar pomaga pri napovedovanju obnašanja sistemov v različnih pogojih, ne da bi bilo treba izvajati drage in dolgotrajne eksperimente.
- Interdisciplinarno sodelovanje. Znanje programiranja olajša komunikacijo in sodelovanje z računalničarji, inženirji ter drugimi strokovnjaki, kar je vse bolj pomembno v sodobnem interdisciplinarnem znanstvenem okolju.
- Karierne priložnosti. Sposobnost programiranja poveča zaposljivost študentov kemije in kemijskega inženirstva. Mnoga delovna mesta v industriji, akademskem svetu in raziskovalnih inštitutih zahtevajo osnovno poznavanje programiranja.
- Reševanje kompleksnih problemov. Programiranje razvija logično mišljenje in sposobnost algoritemskega pristopa k reševanju problemov, kar je bistveno za uspešno reševanje kompleksnih znanstvenih izzivov.

Programiranje ne izboljšuje le raziskovalnih in delovnih procesov, temveč širi obzorja ter diplomantom kemije in kemijskega inženirstva odpira nove priložnosti.

1.1 Programiranje: zakaj, kaj to sploh je in kako?

»Programiranje je vsaj delno umetniški izdelek in tako kot pri likovni umetnosti ali glasbi obstaja več stilov in več poti do končnega rezultata. Vsak, ki programira, si izbere oz. iznajde svoj stil in izbere svojo pot. Nekateri stili pa bi vseeno morali biti zakonsko prepovedani.« M.Q.B.J.

Na vprašanje *zakaj* smo na široko odgovorili že v uvodu. Kratek odgovor je, da se znebimo mukotrpnega, pogosto ponavljajočega se, dostikrat »peš« neizvedljivega izvajanja računskih operacij in da zmanjšamo možnost napake pri izračunih.

Za programiranje pogosto uporabimo analogijo, da je kot pisanje recepta, ki mu računalnik slepo sledi in ga izvede korak za korakom. Poudarek je na besedi *slepo*. Če je recept napačen, bo tudi rezultat napačen ali ga sploh ne bo. Slednje je pogosto bolje, saj v takih primerih vemo, da je s programom nekaj hudo narobe.

Programiranje ni samo končni izdelek, tj. delujoč program v nekem programskem jeziku, ampak je proces. Proces, ki zajema vse od definicije in razumevanja problema, ki ga želimo rešiti; premlevanja idej, kako zadani problem rešiti z uporabo računalnika; zasnove poteka programa v človeku berljivem načinu (psevdokode); pisanja programa v izbranem programskem jeziku; testiranja programa ter odpravljanja napak; optimiranja programske kode itd. Vsi našteti koraki so pomembni, če želimo, da končni izdelek deluje pravilno, zanesljivo in učinkovito. Morda nas premami, da kakšnega preskočimo in prihranimo na času, vendar je zanimivo, da se površnost (preskok) pri katerem koli koraku po navadi kaznuje v naslednjem. Kazni se po verigi vsaj seštevajo, če že ne množijo. Da se v kar največji meri izognemo takim nevšečnostim, podajamo nekaj nasvetov, kako pristopiti k programiranju.

Prvi korak: Dobro, ampak resnično dobro razumevanje problema, ki ga želimo rešiti. Če je problem nekdo že definiral – naj bo to v obliki besedila (npr. besedilne naloge) ali v obliki nekakšnega diagrama – je nujno, da problem res dobro razumemo. Pri tem koraku moramo določiti, kaj so parametri in njihove vrednosti; rešitev katere enačbe ali sistema enačb nam bo dala rešitev problema. Preveriti moramo konsistentnost merskih enot, npr. naj bo temperatura podana v K ali °C; naj bo plinska konstanta podana v L·Pa/(mol·K) ali L·kPa/(mol·K)? Določiti moramo rešitev našega problema. Je to ena sama vrednost? Je to morda niz vrednosti, npr. časovni profil koncentracije reaktanta? Če je problem nekdo samo omenil med pogovorom, močno priporočamo, da ga, preden se lotimo česar koli drugega, zapišemo v obliki besedila ali diagrama.

Nasvet: Pogosto se bo zgodilo, da moramo v prvem koraku sami poiskati ustrezne enačbe, vrednosti parametrov itd. V takem primeru uporabimo preverjene spletne vire (boj se Wikipedije), enciklopedije, tiskane knjige in njihove e-različice. V prvem koraku pozabimo na programski jezik. Tega tukaj ne potrebujemo.

Drugi korak: Psevdokoda. Psevdokoda je prvi prevod zadanega problema v strukturiran redosled navodil. In kot vedno; če je prvi prevod površen, so vsi naslednji prevodi še toliko slabši. Psevdokoda je zapis poteka programa, ki je berljiv nam in ne računalniku, zato je načeloma neodvisna od programskega jezika, v katerem bo zapisan naš končni program.

Namen psevdokode je dvojni. Prvotno je namenjena orisu poteka programa, tj. ključnih korakov programa. Tipično zajema dva bloka:

1) priglasitev parametrov in njihovih vrednosti;
2) jasen zapis vrstnega reda izračunov ter izpisov rezultatov.

Drugi namen psevdokode je, da jo uporabljamo kot vodilo pri dejanskem pisanju programa v izbranem programskem jeziku.

Čeprav psevdokoda ni dejanski program, je dobra psevdokoda več kot polovica rešitve zadanega problema. Vse, kar nam še manjka do delujočega programa, je nekdo, ki obvlada kakršen koli programski jezik. Če je psevdokoda dobra, ji bo lahko sledil in jo prevedel v delujoč program, četudi ni strokovnjak s področja problema, ki ga želimo rešiti.

Nasvet: Raje porabimo minuto ali dve več pri izdelavi dobre psevdokode kot pa uro, dan ali dva več pri pisanju dejanskega programa zaradi tega, »ker se nam pač ni dalo« in »bomo že nekako sproti.«

Tretji korak: Pisanje programa v izbranem programskem jeziku. Pri tem koraku so pomembne tri stvari: poznavanje sintakse programskega jezika (ukazov, rezerviranih besed, zank); doslednost poimenovanja parametrov in spremenljivk ter komentarji. Poznavanje sintakse, vsaj osnovne, pridobimo relativno hitro, še posebej, ker je nabor osnovnih ukazov in računskih operatorjev zelo podoben v večini programskih jezikov. Razen v npr. res obskurnih programskih jezikih je znak za množenje `*`, znaka za potenciranje pa `**` ali `^`. Nekoliko več časa pa potrebujemo za obvladovanje konceptov na višji ravni, kot so funkcije, objekti itd. A tudi to nam z nekaj truda zleze pod kožo.

Nekateri programski jeziki so občutljivi na velike in male črke, drugi spet ne. Pri prvih velja, da npr. parameter `a` ni enak `A` – to sta dva različna parametra oz. spremenljivki. Pri drugih sta `a` in `A` enaka – ena sama stvar. V vsakem primeru pa je izredno modro, da smo konsistentni pri pisanju imen spremenljivk, parametrov, funkcij itd. Nekaj več o dovoljenih imenih teh konstruktov bomo povedali v naslednjih razdelkih, vendar naj bodo ta takšna, da nam nekaj povedo. Če že ne nam, pa vsaj tistemu, ki bo za nami brskal po kodi, jo nadgrajeval, ponovno uporabil ali kaj podobnega. No, morda tudi nam, ko bomo kodo pogledali čez teden ali dva. Če nismo bili dosledni pri poimenovanju, bomo namreč popolnoma izgubljeni in bomo imeli občutek, da program pišemo na novo. Zgolj za ilustracijo: z vidika programskega jezika lahko temperaturo, izraženo v kelvinih, poimenujemo npr. kot: `T; t; temp; tempK; x; tovornjak` itd. Katero izmed poimenovanj bo najmanj dvoumno in nam bo dalo največ informacij, če se vrnemo k programu čez leto dni?

Večina programskih jezikov omogoča komentiranje. Komentarji so opombe, ki ne vplivajo na program, so pa v veliko pomoč pri iskanju napak in razumevanju programa. Komentarji niso nujno potrebni, so pa vedno dobrodošli, če so zapisani smiselno in z namenom. Recimo, da želimo izvesti izračun, ki zahteva vrednost plinske konstante R. To lahko naredimo na dva načina. Prvič jo priglasimo kot: `R = 8.314`.

Drugič pa kot: `R = 8.314 #Plinska konstanta v LkPa/(molK)`, pri čemer je vse v vrstici za znakom `#` (in vključno z njim) komentar. Diskusijo o tem, kaj je bolje in zakaj, prepuščamo vam.

Enačbe v naravoslovnih in inženirskih vedah so lahko zelo dolge. Z uvedbo dodatnih spremenljivk lahko enačbo razbijemo na več krajših in bolj obvladljivih členov. S tem močno zmanjšamo možnost napak in povečamo preglednost programske kode.

Nasvet: V večini primerov lahko zaženemo program, preden je dokončan, mnogokrat že po prvi zapisani vrstici. Seveda pri tem ne moremo pričakovati končnega rezultata, lahko pa preverimo, ali je sintaksa pravilna. Lahko dodamo tudi »vmesne izpise«, da preverimo, ali računalnik »pravilno razume« naše ukaze. Pogosti vmesni zagoni in sprotno preverjanje, če se le da vrstico za vrstico, omogočajo sprotno odpravljanje napak, saj je teh po navadi zelo malo in so najverjetneje prisotne zaradi napak v zadnji zapisani vrstici.

Četrti korak: Testiranje programa. Je najzabavnejši del, sploh v primeru, ko program ne javi napake, a ugotovimo, da je rezultat napačen. Kanček dvoma v rezultat je vedno zaželen, če že ne nujen. Kako pa vemo, da je rezultat napačen? Najprej preverimo očitne stvari, kot so predznaki in velikostni red rezultatov. Če je cilj našega programa izračunati tlak plina in nam program vrne negativno vrednost, je zagotovo nekaj narobe. Pogosto si lahko pomagamo tudi z izračunom »peš«, pri čemer uporabimo poenostavljen izračun, kot je npr. uporaba idealne enačbe stanja namesto npr. Peng-Robinsonove enačbe stanja, ki potencialno nastopa v našem

programu. Rezultati ne bodo enaki, vendar morajo biti dovolj podobni, vsaj glede velikostnega reda in predznaka rezultata.

Nasvet: Lahko se zgodi, da program deluje za en niz vhodnih podatkov, če pa spremenimo samo enega – ni potrebna velika sprememba –, program neha delovati. Pogosto je v takem primeru razlog numerične narave, kot je npr. deljenje z 0. Zato preverimo vse imenovalce in se prepričajmo, ali se lahko pripeti kaj takega. Če najdemo kak člen v enačbi, ki je s tega vidika problematičen, lahko imenovalcu prištejemo majhno vrednost npr. 10^{-6}. S tem ne bomo bistveno vplivali na vrednost člena, ko so vse spremenljivke znotraj »normalnih« vrednosti. Bomo pa preprečili deljenje z 0.

Peti korak: Optimiranje programske kode. Ta faza v večini primerov, ki jih bomo reševali v sklopu nalog med študijem, ni pomembna. Je pa pomembna, ko želimo, da so izračuni hitri; da program ne zasede preveč spomina in da se ne »sesuje« pri različnih vhodnih podatkih, še posebej, ko obdelujemo količine podatkov, merjenih v GB. V takih primerih lahko navidezno majhna sprememba kode pomeni tudi velikostni red hitrejšega izvajanja programa.

In še zadnji nasvet: Pri delu z računalnikom, predvsem pa pri programiranju, se izogibajte vseh črk, ki niso v angleški abecedi (š, č, ć, ž, đ, ä, ø itd.). To velja tako za imena spremenljivk, parametrov kot za imena datotek in map. Pri poimenovanju datotek in map se izogibajte presledkom, pikam in podobnim ločilom. Edini varni simbol, ki v imenih nakazuje na presledek, je podčrtaj _.

1.2 Pretvarjanje iz desetiškega v dvojiški sistem in obratno

Računalniki temeljijo na dvojiškem sistemu, saj jih je lažje elektronsko realizirati z dvema stanjema (vklopljen/izklopljen, 1/0). Vsak podatek, ki ga obdeluje računalnik, je predstavljen v dvojiški obliki, bodisi gre za številke, besedilo, slike ali zvok. Seveda nihče od nas ne razmišlja v dvojiškem sistemu niti ne poskuša zapisati enega samega ukaza v tem sistemu, pa vendar je koristno poznati osnove pretvarjanja iz desetiškega v dvojiški sistem in obratno.

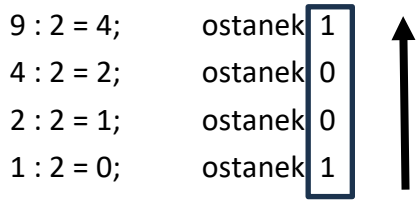
1.2.1 Pretvarjanje iz desetiškega v dvojiški sistem

Za pretvarjanje celega števila iz desetiškega v dvojiški sistem sledimo naslednjim korakom:

1. **Deljenje z 2**: Vzemi desetiško število in ga deli z 2.
2. **Zapiši rezultat in ostanek**: Zapiši rezultat in ostanek deljenja, ki je 0 ali 1.
3. **Ponavljaj deljenje**: Rezultat deljenja (brez ostanka) ponovno deli z 2 in zapiši ostanek.
4. **Ponavljaj korake**: Ponavljaj deljenje, dokler rezultat deljenja ni 0.
5. **Obratni vrstni red ostankov**: Ostanke, ki si jih dobil, zapiši v obratnem vrstnem redu. To je dvojiško število.

Primer 1-1:

Pretvori število 9 iz desetiškega v dvojiški sistem.



Rezultat: $9_{(10)} = 1001_{(2)}$

Za pretvarjanje realnega števila iz desetiškega v dvojiški sistem sledimo naslednjim korakom:

1. **Pretvorba celega dela**: Vzemi celi del števila in ga pretvori, kot je opisano v prejšnjem primeru.
2. **Decimalni del**: Vzemi decimalni del realnega števila.
3. **Množenje z 2**: Decimalni del množi z 2.
4. **Zapiši celi del rezultata**: Zapiši celi del rezultata množenja, ki je 0 ali 1.
5. **Posodobi decimalni del**: Posodobi decimalni del tako, da odšteješ celi del rezultata od rezultata množenja.
6. **Ponavljaj množenje**: Ponavljaj množenje z 2, dokler decimalni del ni 0 ali dokler ne dosežeš želene natančnosti.
7. **Zapiši rezultat**: Zapiši cele dele rezultatov množenj v nespremenjenem vrstnem redu.
8. **Združi** celi in decimalni del.

Primer 1-2:

Pretvori število 10,625 iz desetiškega v dvojiški sistem.

Najprej pretvorimo celi del:

10 : 2 = 5; ostanek 0
5 : 2 = 2; ostanek 1
2 : 2 = 1; ostanek 0
1 : 2 = 0; ostanek 1

Rezultat: $10_{(10)} = 1010_{(2)}$

Nato pretvorimo decimalni del:

0,625 · 2 = 1,25; celi del je 1
0,25 · 2 = 0,50; celi del je 0
0,50 · 2 =1,0; celi del je 1

Rezultat: $0{,}625_{(10)} = 101_{(2)}$

Združimo rezultat celega in decimalnega dela: $10{,}625_{(10)} = 1010{,}101_{(2)}$

1.2.2 Pretvarjanje iz dvojiškega v desetiški sistem

Za lažje razumevanje najprej spomnimo, kako deluje desetiški sistem. Npr. število 1315,287 predstavlja vsoto mnogokratnikov števila deset na določene potence:

$$1315{,}287 = 1 \cdot 10^3 + 3 \cdot 10^2 + 1 \cdot 10^1 + 5 \cdot 10^0 + 2 \cdot 10^{-1} + 8 \cdot 10^{-2} + 7 \cdot 10^{-3}$$

Na podoben način lahko zapišemo dvojiško število kot vsoto mnogokratnikov števila dve na ustrezne potence.

Primer 1-3:

Pretvori število 11001,0111 iz dvojiškega v desetiški sistem.

$$11001{,}0111_{(2)} = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4}$$
$$= 25{,}4375_{(10)}$$

1.2.3 Realna števila in floating-point napake

Pomembno je, da se zavedamo, da vsak naš izračun, ki ga podamo v desetiškem sistemu, računalnik prevede v dvojiški sistem, izvede izračun v dvojiškem sistemu in rezultat pretvori nazaj v desetiški sistem (dvojno pretvarjanje). To ima posledice, ki se jim, če ne delamo z dovoljšno mero natančnosti, lahko precej čudimo. Nenazadnje lahko privedejo tudi do računske nenatančnosti (beri napak). Poglejmo ulomek 1/10. V desetiškem sistemu je to precej pohlevno število s končnim številom decimalnih mest $0{,}1_{(10)}$. Če pa ga pretvorimo v dvojiški sistem, postane $0{,}00011001100110011\ \ldots_{(2)}$.

Računalniki namreč shranjujejo realna števila v dvojiškem zapisu z omejenim številom bitov. To pomeni, da marsikatero število, ki je v desetiškem sistemu »natančno«, v računalniku nima popolne predstavitve. Posledica so majhna odstopanja pri računanju, ki jim pravimo *floating-point* napake. Te napake se lahko pokažejo na različne načine: na primer pri seštevanju decimalnih števil rezultat ni vedno do potankosti takšen, kot bi pričakovali, pri primerjavah števil dobimo nepričakovane rezultate, pri ponavljajočih izračunih se lahko odstopanja seštevajo in postanejo opazna. Zato moramo biti previdni, ko delamo z realnimi števili in po potrebi uporabiti zaokroževanje ali posebne knjižnice za večjo natančnost. Moderni programski jeziki in sodobna računalniška arhitektura napake pri pretvarjanju k sreči omilijo do te mere, da skorajda nimajo vpliva na natančnost praktičnih inženirskih izračunov.

Primer 1-4:

Kaj se izpiše kot rezultat seštevanja števil 0,1 in 0,2?

```
print(0.1+0.2)
```

Izpiše se: `0.30000000000000004`

V 4. poglavju bomo spoznali, kako dosežemo izpis rezultata z zaokroževanjem na želeno število decimalnih mest brez nadležne štirice na koncu.

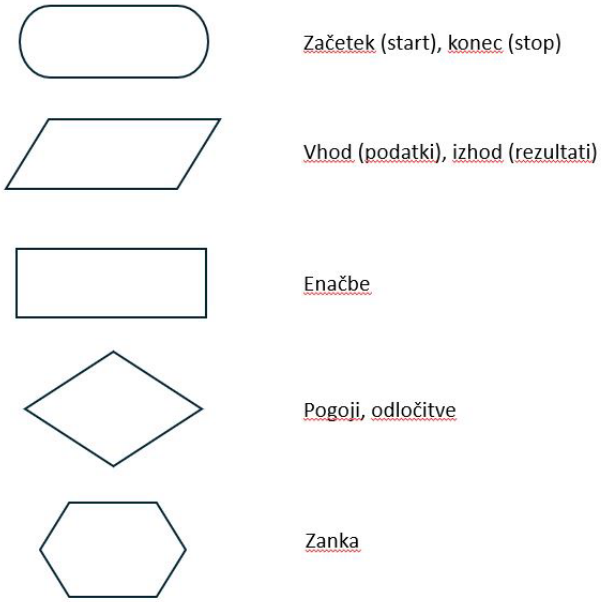
1.3 Algoritemski pristop k reševanju problemov v naravoslovju in inženirstvu

Računalniški program je zaporedje različnih ukazov, ki se praviloma izvajajo po vrsti, eden za drugim. V kemiji in kemijskem inženirstvu se računalniški programi najpogosteje uporabljajo za reševanje problemov, ki vsebujejo številne izračune in bi bili brez programa zelo zamudni. Pisanje programske kode oz. programiranje pa ni prvi korak, ki se ga lotimo pri reševanju problema. Za sistemsko ali algoritemsko reševanje problemov so pomembni naslednji koraki:

a) Definiranje problema. Jasno opišemo problem; kaj želimo doseči z njegovim reševanjem in kako nameravamo priti do rešitve.
b) Analiza problema in modeliranje. Zberemo enačbe, s katerimi bomo reševali problem, ter definiramo, katere vhodne podatke potrebujemo in katere rezultate bomo dobili z reševanjem enačb. Posebna pozornost velja merskim enotam fizikalnih veličin.
c) Izdelava algoritma oz. diagrama poteka. Shematsko predstavimo potek reševanja in pri tem predvidimo vrstni red reševanja posameznih enačb, saj nekatere enačbe potrebujejo vhodne podatke, ki so rezultati drugih enačb.
d) Programiranje. Model pretvorimo iz matematične oblike v računalniški program.

Algoritem predstavlja zaporedje korakov ali navodil, ki opisujejo, kako rešiti določen problem ali opraviti neko nalogo. Algoritmi so osnova za pisanje računalniških programov, saj določajo, kako naj program izvaja določene operacije.

Diagram poteka je vizualna predstavitev algoritma, ki prikazuje, kako se povezujejo posamezne naloge in v kakšnem vrstnem redu se izvajajo. Z uporabo različnih simbolov, kot so ovali, pravokotniki, rombi, so predstavljeni začetek in konec programa; izračunavanje spremenljivk in druge operacije; odločitve in povezave med njimi. Diagrami poteka so zelo uporabni pri poučevanju računalništva in programiranja, saj omogočajo lažje razumevanje logičnega toka algoritmov. Simboli, ki se uporabljajo za predstavitev korakov pri reševanju problema, so predstavljeni na sliki 1-1.



Slika 1-1: Simboli za diagram poteka

Primer 1-5:

a) Definiranje problema. Izračunajmo množino (mol) in volumen (L) za 100 g dušika pri temperaturi 23 °C in tlaku 1,2 atm.

b) Matematični model:

$$pV = nRT \tag{1}$$

kjer je:

p tlak (Pa)

V volumen (m^3)

n množina (mol)

R splošna plinska konstantna (J/(mol·K))

T temperatura (K)

$$n = \frac{m}{M} \qquad \left[\frac{\mathrm{g}}{1}\,\frac{\mathrm{mol}}{\mathrm{g}} = \mathrm{mol}\right] \tag{2}$$

kjer je:

m masa (g)
M molska masa (g/mol)

Iz enačbe (1) sledi enačba za volumen plina:

$$V = \frac{n \cdot R \cdot T}{p} \qquad \left[\frac{\mathrm{mol}}{1} \cdot \frac{\mathrm{J}}{\mathrm{mol \cdot K}} \cdot \frac{\mathrm{K}}{1} \cdot \frac{1}{\mathrm{Pa}} = \frac{\mathrm{J}}{\mathrm{Pa}} = \frac{\mathrm{Nm}}{1} \cdot \frac{\mathrm{m^2}}{\mathrm{N}} = \mathrm{m^3}\right] \tag{3}$$

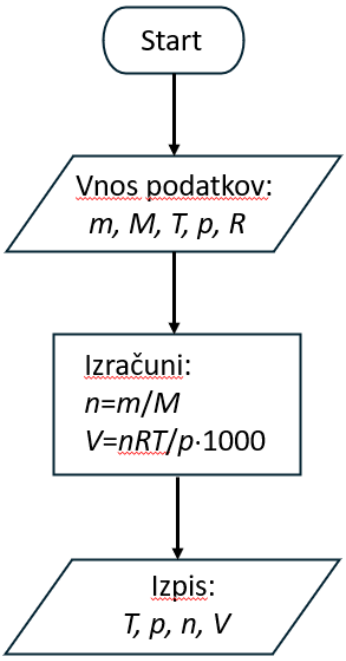
Če želimo rezultat izraziti v litrih, dodamo v enačbi (3) še množenje s 1000, saj ima 1 m^3 1000 L. Ker v enačbi (3) potrebujemo tlak v Pa, pretvorimo 1,2 atm v Pa, pri čemer je pretvornik 1 atm = 101 325 Pa.

$$1{,}2\ \mathrm{atm}\,\frac{101325\ \mathrm{Pa}}{1\ \mathrm{atm}} = 121590\ \mathrm{Pa}\ \ \text{oz.}\ \ 1{,}21590\ \mathrm{E{+}05\ Pa} \tag{4}$$

Za temperaturo vemo, da stopinjam Celzija prištejemo 273 in dobimo temperaturo v kelvinih. Glede na enačbe modela lahko strnemo vhodne podatke in izračunane rezultate:

Vhodni podatek	Vrednost in enota	Rezultat	Enota
masa, *m*	100 g	množina, *n*	mol
molska masa, *M*	28 g/mol	volumen, *V*	L
temperatura, *T*	296 K		
tlak, *p*	121 590 Pa		
splošna plinska konstanta, *R*	8,314 J/(mol·K)		

c) Diagram poteka:



Slika 1-2: Diagram poteka za primerPrimer 1-5

d) Programiranje. Ker še ne poznamo programskega jezika, zapišimo psevdokodo, ki sledi korakom v diagramu poteka in omogoča enostavno implementacijo v katerem koli programskem jeziku.

```
ZAČETEK

VNOS PODATKOV
    Preberi m   # masa plina v g
    Preberi M   # molekulska masa plina v g/mol
    Preberi T   # temperatura v K
    Preberi p   # tlak v Pa
    Preberi R   # splošna plinska konstanta 8.314 J/(molK)

IZRAČUNI
    Izračunaj množino n = m/M              # množina v mol
    Izračunaj volumen V = n*R*T/p*1000     # prostornina L

IZPIS
    Izpiši T    # temperatura v K
    Izpiši p    # tlak v Pa
    Izpiši n    # množina v mol
    Izpiši V    # prostornina v L

KONEC
```

1.4 Problemi 1. poglavja

Problem 1-1:

Pretvorite naslednja števila iz desetiškega sistema v dvojiški.

134

86,09375

112,1

Problem 1-2:

Pretvorite naslednja števila iz dvojiškega sistema v desetiški.

11001101

10101010,10

11110111110,1111

Problem 1-3:

Pripraviti želimo 100 ml raztopine NaOH z množinsko koncentracijo 0,1 mol/L. Izračunajte maso NaOH, ki jo je treba zatehtati (g), in masno koncentracijo pripravljene raztopine (g/L). Narišite diagram poteka in zapišite psevdokodo za ta problem.

Problem 1-4:

20 gramov natrijevega klorida (NaCl) raztopimo v 500 mL vode. Predpostavite, da je gostota raztopine 1,0 kg/L in da se volumen med raztapljanjem ne spremeni. Molska masa NaCl je 58,44 g/mol. Napišite psevdokodo za izračun masne in množinske koncentracije raztopine.

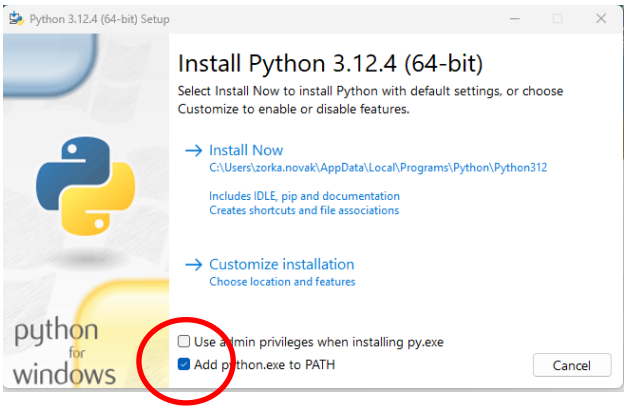
Problem 1-5:

Pri sežigu 2 mol metana (CH_4) se sprosti 890 kJ toplote. Napišite psevdokodo za izračun entalpijske spremembe za sežig 5 mol metana.

2 Namestitev okolja Python

2.1 Python v okolju IDLE

Okolje *Python* je prosto dostopno na spletni strani: https://www.python.org/downloads/. Na tej strani naložimo zadnjo verzijo na svoj računalnik in v mapi *Downloads* dvokliknemo na program. Ob začetku namestitve označimo *Add python.exe to PATH* in sledimo korakom instalacije.

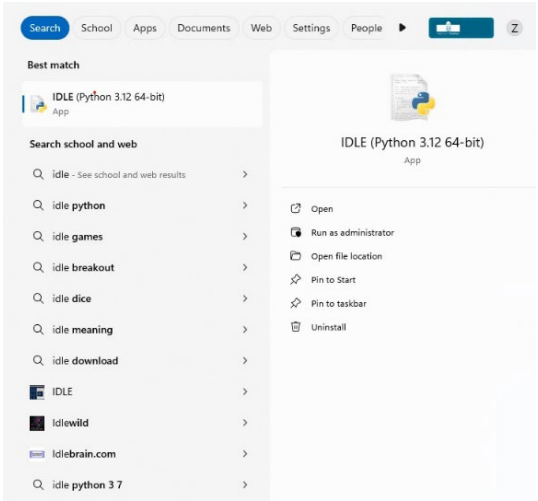


Slika 2-1: Namestitev programskega okolja *Python*

Za programiranje potrebujemo še integrirano razvojno okolje oz. IDE (angl. *Integrated Development Environment*), ki ga olajša. Na voljo je veliko okolij, npr. IDLE, VSC, *PyCharm* in druga. Uporaba okolja IDLE je opisana v nadaljevanju.

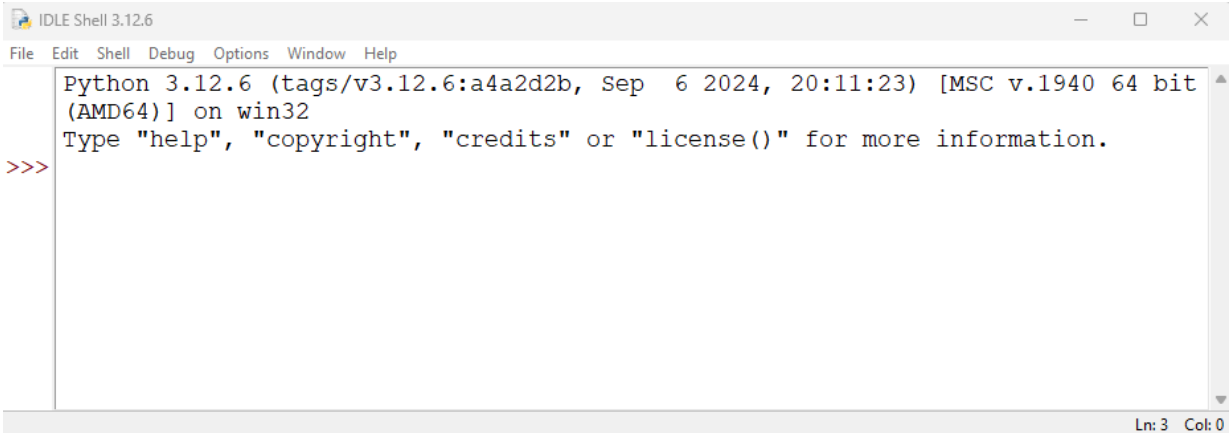
2.1.1 IDLE

Okolje IDLE dobimo že v okviru namestitve okolja *Python*. Je integrirano razvojno okolje, ki je del standardne distribucije *Pythona* in omogoča enostavno pisanje, urejanje, izvajanje ter odpravljanje napak v *Pythonovih* programih. Zaženemo ga tako, da v meniju *Start* okolja *Windows* poiščemo IDLE (*Python* ...) in kliknemo nanj.



Slika 2-2: Zagon *Pythona* v okolju IDLE

Pojavi se osnovno okno, v katerem lahko začnemo uporabljati *Python*.



Slika 2-3: Osnovno okno *Python* v okolju IDLE

2.1.2 Dva načina dela

V okolju IDLE lahko uporabljamo *Python* na dva načina:

1. z ukazno vrstico oz. kot nekakšen kalkulator, ki ne omogoča shranjevanja v datoteko;
2. z datoteko, ki jo lahko shranimo.

V prvem primeru v ukazno vrstico, ki ima na levi strani oznako `>>>`, vpišemo želen ukaz in stisnemo *enter*. Za izpisovanje vrednosti spremenljivk in besedil uporabimo ukaz `print`, ki je podrobneje opisan v podpoglavju 5.1.

```
>>> a=2
>>> b=4.2
>>> c=a*b
>>> print(a,b,c)
```

V istem oknu dobimo po zadnjem ukazu izpisan rezultat:

```
>>> a=2
>>> b=4.2
>>> c=a*b
>>> print(a,b,c)
2 4.2 8.4
```

V drugem primeru odpremo datoteko z ukazom `File, New File`. Odpre se dodatno okno z urejevalnikom, v katerega pišemo ukaze. Za boljšo preglednost lahko vklopimo številčenje vrstic z ukazom `Options, Show Line Numbers`. Datoteko shranimo z ukazom `Save As` pod želeno ime, pri čemer dobi končnico `.py`. Izvajanje programa zaženemo z ukazom `Run, Run Module`. Npr. program:

```
mesec = "marec"
dan = 27
print("Danes je", dan,".", mesec)
```

bo dal potem, ko ga shranimo in zaženemo, izpis v oknu IDLE:

```
Danes je 27 . marec.
```

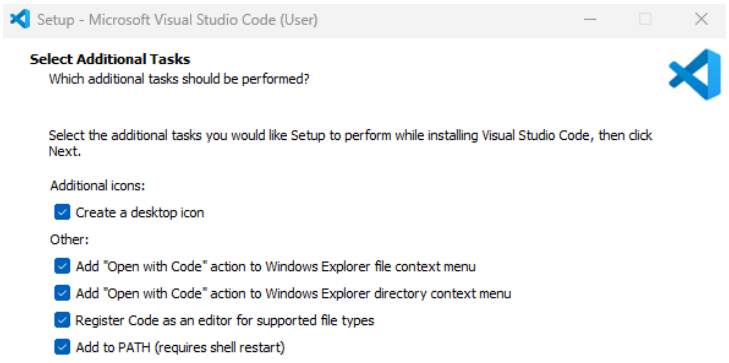
2.2 PyCharm

Najdemo ga na strani https://www.jetbrains.com/pycharm/, kjer izberemo neplačljivo verzijo *PyCharm Community Edition* (ne *PyCharm Professional*!). Ko je aplikacija naložena v mapi *downloads*, z dvoklikom začnemo njeno instalacijo. Med postopkom instalacije označimo kvadratek pri *Add bin folder to the PATH* in po želji tudi *Create Desktop Shortcut.* Ko je namestitev končana, je potreben ponovni zagon računalnika. Po ponovnem zagonu zaženemo program z dvoklikom ikone na namizju, če smo jo ustvarili, sicer ga poiščemo z iskalnikom. Ob prvem zagonu se nam ponudi možnost kreiranja projekta, ki ga poimenujemo z imenom po svoji izbiri. Projekt predstavlja lokacijo, na kateri bodo vse naše datoteke, ki jih bomo ustvarili v sklopu tega projekta. Ob ponovnem zagonu *PyCharm* bomo imeli na voljo projekt in vse v njem ustvarjene datoteke, tako da jih ne bo treba iskati po računalniku. Po želji si nastavimo svetlo ozadje (4 vodoravne črtice zgoraj levo, *Settings*, *Appearance*, *Theme Light*).

S tem je *PyCharm* pripravljen za programiranje. Datoteko, v katero bomo pisali program, odpremo z *New, Python File* in ji damo želeno ime. V levem delu okna je brskalnik, v katerem vidimo seznam datotek v projektu. V desnem oknu je *editor*, v katerem pišemo kodo. Hkrati imamo lahko odprtih več datotek, katerih imena so vidna v vrstici nad *editorjem* in lahko preklapljamo med njimi. Kodo zaženemo s pritiskom zelenega znaka ▷ zgoraj desno; ob tem se datoteka tudi avtomatsko shrani. Ob zagonu programa se pojavi spodnje okno (*Run Tool Window*), v katerem vidimo rezultate.

2.3 VSC

Za urejanje in zagon kode *Python* lahko uporabimo tudi brezplačni urejevalnik VSC (*Visual Studio Code*), ki ga namestimo s strani: https://code.visualstudio.com/. V mapi *Downloads* dvokliknemo na program in začnemo z namestitvijo. Ko pridemo do okna *Select Additional Tasks*, označimo vse gumbe:

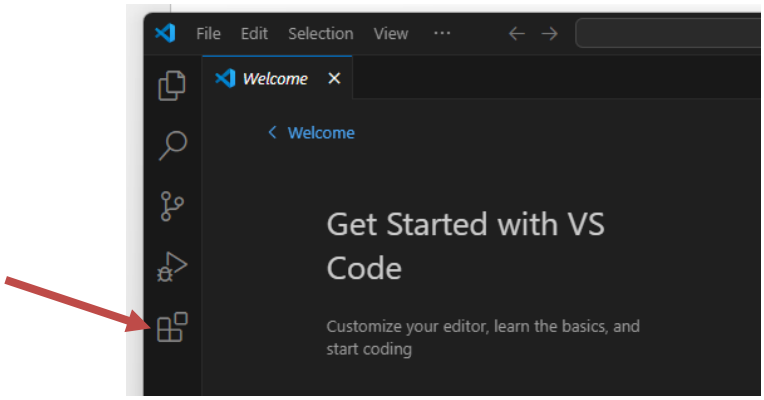


Slika 2-4: Namestitev urejevalnika VSC

Ko je namestitev končana, se odpre aplikacija VSC – običajno okno *Welcome*. Sledimo še naslednjim korakom:

1. V levem navpičnem meniju kliknemo na ikono »`Extensions`« (slika 2-5) in v oknu »`Search`« vpišemo »`Python`«. Kliknemo na zadetek *Python* in nato na gumb »`Install`«.

2. Nastavimo si še *Python* za naš interpreter, tako da hkrati pritisnemo CTRL + SHIFT + P. V oknu za iskanje poiščemo »`Python: Select Interpreter`« in med ponujenimi izberemo *Python*, ki smo ga namestili pred tem.
3. Za začetek programiranja odpremo novo datoteko: `File, New File` in označimo *Python File*.
4. Odpre se datoteka brez imena (*Untitled*-1), ki jo shranimo s pritiski na `File, Save As` pod želeno ime. Datoteka dobi končnico `.py`, ki je značilna za datoteke *Python*. V to datoteko začnemo zapisovati program.
5. Program zaženemo s pritiskom na gumb ▷ `(Run Python File)` v zgornjem desnem kotu.
6. Rezultati so prikazani v oknu pod programom.



Slika 2-5: Gumb *Extensions*

2.4 Thonny

Za začetnike je primerno tudi okolje *Thonny*, ki ga prenesemo s strani https://thonny.org/. Ko je program naložen v mapi prenosov, z dvoklikom izvedemo njegovo namestitev. Delo poteka v dveh glavnih oknih; zgornje je urejevalnik za programiranje, v spodnjem se prikažejo rezultati.

2.5 Spletne različice

Za resno učenje programiranja je potrebna namestitev programa na računalnik, občasno pa lahko pridejo prav tudi spletne različice, npr. za uporabo na tablici. Veliko orodij najdemo s spletnim brskalnikom, nekaj možnosti je naslednjih:

- *Online Python* (preprosto spletno okolje za hitre preizkuse);
- *Programiz* (interaktivne lekcije in primeri);
- *Replit* (okolje za projekte in deljenje kode);
- *Google Colab* (zvezki (*notebooks*) za *Python*);
- in drugi, ki jih lahko poiščete sami.

3 Spremenljivke in vrste podatkov

Spremenljivke shranjujejo različne podatke, ki so v glavnem treh vrst:

a) numerični;
b) znakovni oz. tekstovni (*string*);
c) logični.

Ime spremenljivke je lahko sestavljeno iz črk, številk in podčrtaja. Na prvem mestu morata biti črka ali podčrtaj. Presledki in drugi posebni znaki v imenih spremenljivk niso dovoljeni. Prav tako za imena spremenljivk ni dovoljeno uporabiti ključnih besed programskega jezika *Python*, kot so `if`, `break`, `try` ipd. Priporočeno je, da so imena spremenljivk kratka in čim bolj opisna, da se takoj spomnimo, kaj predstavljajo. Program razlikuje med velikimi in malimi črkami, tako da npr.

`ime_spremenljivke` ni enako kot `Ime_spremenljivke`.

3.1 Numerične spremenljivke

So za kemike in kemijske inženirje najbolj uporabna vrsta spremenljivk. Z njimi shranjujemo številčne vrednosti in izvajamo aritmetične operacije za izračun želenih vrednosti.

Primer 3-1:

V naslednjem programu se spremenljivka imenuje `ime_sprem` in ji je najprej dodeljena numerična vrednost `24`. Nato se izračuna dvakratnik te vrednosti.

```
ime_sprem = 24
print(ime_sprem)
ime_sprem = ime_sprem*2
print(ime_sprem)
```

Po zagonu programa se izpiše:

```
24
48
```

Primer 3-1 ponazarja, da je vrstni red ukazov zelo pomemben. Ukazi se izvajajo po vrsti eden za drugim. V prvi vrstici je bila spremenljivki `ime_sprem` dodeljena vrednost 24, ki se je nato v tretji vrstici »prepisala« z dvakratnikom prejšnje vrednosti. Tako je prvotna vrednost pozabljena.

Nekatere spremenljivke shranjujejo vrednosti, ki se v programu ne spreminjajo, zato jih imenujemo konstante. V programu jih lahko podamo kar s številskimi vrednostmi, lahko pa tudi z imenom; podobno kot spremenljivke, le da njihovih vrednosti praviloma ne spreminjamo več. Splošni plinski konstanti lahko na primer v programu dodelimo ime, enako kot kateri koli numerični spremenljivki, vendar se njena vrednost ne bo spreminjala:

```
R = 8.314
```

V nadaljevanju programa konstanto z imenom `R`, ki ima vrednost 8,314, uporabljamo v izračunih, kjer jo potrebujemo, npr. v plinski enačbi.

Primer 3-2:

Kaj izračuna naslednji program?

```
p=1.21590E+05
T=296
m=100
M=28
R=8.314
n=m/M
V=n*R*T/p*1000
print(T, p, n, V)
```

Po zagonu programa se izpiše:

```
296 121590.0 3.571428 72.284609
```

3.2 Znakovne spremenljivke (nizi)

Znakovne spremenljivke ali nizi (angl. *strings*) so zaporedja znakov oz. besedila, ki jih želimo v programu. Njihove vrednosti se podajo v narekovajih; uporabimo lahko enojni ali dvojni narekovaj.

Primer 3-3:

V naslednjem programu se spremenljivka imenuje `Ime_mesta` in ji je najprej dodeljena znakovna vrednost `"Maribor"`, ki se nato v tretji vrstici prepiše z znakovno vrednostjo `"Ljubljana"`.

```
Ime_mesta = "Maribor"
print(Ime_mesta)
Ime_mesta="Ljubljana"
print(Ime_mesta)
```

Izpis:

```
Maribor
Ljubljana
```

Primer 3-4:

Znakovne spremenljivke lahko združujemo s spajanjem (*concatenation*), za kar uporabimo znak `+`.

```
ime_predmeta = "Racunalnistvo v kemiji"
stanje_predmeta = " je ok"
skupaj = ime_predmeta + stanje_predmeta
print(skupaj)
```

Izpis:

```
Racunalnistvo v kemiji je ok
```

Če želimo združiti izpis znakovne in numerične spremenljivke s spajanjem, moramo numerično spremenljivko pretvoriti v znakovno z ukazom `str`.

Primer 3-5:

```
a="Los Angeles "
b=2028
print (a + str(b))
```

Izpis:

```
Los Angeles 2028
```

Ena od značilnosti *Pythona*, ki ga ločuje od mnogih drugih programskih jezikov, je uporaba metod, vezanih neposredno na objekte, npr. spremenljivke, sezname. To pomeni, da lahko na objektu (spremenljivki) neposredno kličemo funkcije, ki izvajajo določene operacije na tem objektu.

Primer 3-6:

Uporabna funkcija je `replace`, s katero nadomestimo del znakovne spremenljivke.

```
Ime_fakultete = "Fakulteta za farmacijo"
print(Ime_fakultete)
print(Ime_fakultete.replace("farmacijo","medicino"))
```

Izpis:

```
Fakulteta za farmacijo
Fakulteta za medicino
```

3.3 Logične spremenljivke

Poleg znakovnih in numeričnih spremenljivk poznamo še logične spremenljivke, ki lahko zavzamejo le dve vrednosti: `True` ali `False`.

Primer 3-7:

```
ime_sprem = True
print(ime_sprem)
```

Po zagonu programa se izpiše:

```
True
```

Pomen in uporabo logičnih spremenljivk bomo podrobneje spoznali pozneje.

3.4 Problemi 3. poglavja

Problem 3-1:

Ali so imena spremenljivk pravilna? Če niso, utemeljite.

```
answeR
J+329
Lambda_stop
X-ray
2145
Try_1
```

Problem 3-2:

Ali so pravilni naslednji stavki v programskem jeziku *Python*? Če niso, utemeljite.

```
a = b + c
b + c = A
a = a + a
A+B = C+D
Ab = 5x
oseba_ime = "Ana"+"Marko"
oseba_ime= Ana + Marko
```

Problem 3-3:

Kaj se izpiše kot rezultat naslednjega programa?

```
var_ime = "Fakulteta"
var_ime = "za kemijo"
var_ime = "in kemijsko tehnologijo"
print (var_ime)
```

Preuredite program tako, da se bodo izpisali vsi trije nizi.

Problem 3-4:

Ugotovite, zakaj koda ne deluje in jo popravite.

```
x = y + 5
y = 10
```

Problem 3-5:

Popravite napake v enačbah.

a) `10 = x + 5`

b) `y + 5 = 12`

Problem 3-6:

Najdite napake v programih in jih popravite.

a)
```
1st_variable = 10
my-variable = 5
total@sum = 1st_variable + my-variable
print (total@sum)
```

b)
```
if = 20
x = if + 10
print (x)
```

c)
```
age = 25
message = "Star sem " + age + " let."
print (message)
```

Problem 3-7:

Zapišite program, ki bo spremenljivki *x* dodelil vrednost 33.9 in spremenljivki *y* vrednost 3.0. Nato naj izračuna vrednost spremenljivke *N* kot vsoto podanih spremenljivk *x* in *y* ter vrednost spremenljivke *Z* kot razliko *x* in *y*. Na koncu naj se izpišejo vrednosti vseh štirih spremenljivk.

4 Aritmetični izrazi

Aritmetični izrazi so računski izrazi, ki vsebujejo konstante, spremenljivke in aritmetične operatorje. Konstante in spremenljivke so bile definirane v prejšnjem poglavju, aritmetični operatorji pa so dobro znane računske operacije:

`+`	seštevanje
`–`	odštevanje
`*`	množenje
`/`	deljenje
`**`	potenciranje

V aritmetičnem izrazu mora biti obvezno na levi strani ena sama spremenljivka, na desni strani pa je matematični izraz, ki predstavlja formulo, s katero izračunamo vrednost spremenljivke:

```
spremenljivka = aritmetični izraz
```

Vse spremenljivke, ki se uporabijo v aritmetičnem izrazu na desni strani, morajo biti pred tem definirane oz. izračunane.

Primer 4-1:

V spodnjem programu se v prvi in drugi vrstici dodelita vrednosti spremenljivkama `x` in `y`, nato se v tretji vrstici z aritmetičnim izrazom izračuna vrednost spremenljivke `z`, ki je vsota kvadratov podanih spremenljivk.

```
x = 3
y = 4
z = x**2 + y**2
print(x,y,z)
```

Po zagonu programa se izpiše:

```
3 4 25
```

4.1 Vrstni red izvajanja aritmetičnih operacij

Pravilen vrstni red računskih operacij je ključen za izračun pravilnega rezultata. Najvišjo prioriteto pri izračunih imajo oklepaji, zato izraze v oklepajih izračunamo najprej. Računske operacije si sledijo po naslednjem vrstnem redu: potenciranje, množenje in deljenje, ki sta enakovredni operaciji, ter seštevanje in odštevanje, ki sta prav tako enakovredni operaciji. Če so v aritmetičnem izrazu prisotne enakovredne operacije, se račun izvaja od leve proti desni.

Vrstni red izračunov je torej naslednji:

1. oklepaji;
2. potenciranje;
3. množenje/deljenje, ki sta enakovredni operaciji;
4. seštevanje/odštevanje, ki sta enakovredni operaciji.

Primer 4-2:

Izračunajte vrednosti spodnjih izrazov, če je: $a = 2$; $b = 4$; $c = 10$; $d = 12$, tako da prikažete pravilni vrstni red izračunov. Zapišite program in primerjajte rezultate.

a) $x = b + \frac{d - c}{a}$

b) $x = a \cdot \frac{d}{b} c^a - d$

c) $x = \left(a \frac{d}{b} c \right)^a - d$

```
a=2
b=4
c=10
d=12
x=b+(d-c)/a
print(x)
x=a*d/b*c**a-d
print(x)
x=(a*d/b*c)**a-d
print(x)
```

Izpis:

```
5.0
588.0
3588.0
```

Primer 4-3:

Izračunajte vrednosti spodnjih izrazov, če je: $a = 12$; $b = 3$; $c = 4$, tako da prikažete pravilni vrstni red izračunov. Zapišite program in primerjajte rezultate.

a) $x = \frac{a}{b} \cdot c$

b) $x = \frac{a}{b \cdot c}$

```
a=12
b=3
c=4
x=a/b*c
print(x)
x=a/(b*c)
print(x)
```

Izpis:

```
16.0
1.0
```

4.2 Vgrajene matematične funkcije

Osnovne matematične funkcije v *Pythonu* so zbrane v tabeli 4-1.

Tabela 4-1: Osnovne vgrajene matematične funkcije

Absolutna vrednost	`abs(x)`
Potenciranje	`pow(x, y)`
Največja vrednost	`max(x1, x2, x3 ...)`
Najmanjša vrednost	`min(x1, x2, x3 ...)`
Zaokroževanje na najbližje celo število (ali na določeno število decimalnih mest)	`round(x)`
Pretvori realno število v celo	`int(x)`
Ostanek pri deljenju (modulo)	`x % y` (vrne ostanek pri deljenju števila *x* z *y*)

Primer 4-4:

Kaj se izpiše pri vsakem `print` stavku?

```
stevilo = -3.5
print(abs(stevilo))
n=3
m=pow(n,5)
print(m)
najvec= max(2, 5, -6.4, 12)
najmanj= min(2, 5, -6.4, 12)
print("Najvecja vrednost: " + str(najvec) + ". ")
print("Najmanjsa vrednost: " + str(najmanj)+ ". ")
stevilo = 3.7
print(round(stevilo))
```

Po zagonu programa se izpiše:

```
3.5
243
Najvecja vrednost: 12.
Najmanjsa vrednost: -6.4.
4
```

Za nekatere preostale funkcije je treba vključiti v program naslednji ukaz, v katerem zvezdica `*` predstavlja vse funkcije oz. vrednosti, npr. sin, cos, število π ipd.

```
from math import *
```

Nabor najznačilnejših osnovnih matematičnih funkcij je prikazan v tabeli 4-2. Če v programu potrebujemo le nekaj specifičnih funkcij, npr. funkciji kvadratnega korena in naravni logaritem, ju lahko uvozimo iz knjižnice `math` z naslednjim ukazom:

```
from math import sqrt, log
```

Tabela 4-2: Vgrajene matematične funkcije, ki zahtevajo uvoz knjižnice `math`

Pretvorba v celo število navzdol	`floor(x)`
Pretvorba v celo število navzgor	`ceil(x)`
Kvadratni koren	`sqrt(x)`
Naravni logaritem	`log(x)`
Desetiški logaritem	`log10(x)`
Kotne funkcije (koti v radianih)	`sin(x), cos(x), tan (x)`
Krožne funkcije (koti v radianih)	`asin(x), acos(x), atan (x)`
Eksponentna funkcija e^x	`exp(x)`

Primer 4-5:

Kaj se izpiše?

```
from math import *
stevilo = 3.4
print(floor(stevilo))
print(ceil(stevilo))
n=sqrt(25)
print(n)

x=log(100)
y=log10(100)
print("Naravni je " + str(x) + ". ")
print("Desetiški je " + str(y)+ ". ")

alfa=60
alfarad=alfa*3.14159/180
x=cos(alfarad)
print(x)
```

Po zagonu programa se izpiše:

```
3
4
5.0
Naravni je 4.60517.
Desetiški je 2.0.
0.50000
```

Primer 4-6:

Primerjava funkcij `floor` in `int` ter `ceil` in `round`. Kaj se izpiše?

```
from math import *
a=5.8
b=-5.8
print(floor(a),int(a))
print(floor(b),int(b))
c=3.6
d=-3.6
print(ceil(c),round(c))
print(ceil(d),round(d))
```

Po zagonu programa se izpiše:

```
5 5
-6 -5
4 4
-3 -4
```

4.3 Problemi 4. poglavja

Problem 4-1:

Prikažite vrstni red izračunov za naslednji izraz, če je *A* = 15.3; *B* = –7.4; *J* = 8; *L* = 32:

$$X = A + J*(L/3)**2 + B/4*(L/A)$$

Problem 4-2:

V *Pythonu* zapišite naslednje izraze in uporabite ustrezne matematične funkcije.

a) $\sqrt{5 \cdot x^2 + 8 \cdot y^2}$

b) $\sin(x - 2y) + e^{x \cdot y} - |x^2 - y^2|$

Problem 4-3:

Najdite napake v računalniških zapisih na desni strani:

	Matematični zapis	Zapis v *Pythonu*
a)	$p = \frac{x \cdot y}{z \cdot 1000}$	`p = x*y/z*1000`
b)	$z = 2x + 3/y + e^{-2x}$	`z = 2x+3/y+e**(-2*x)`
c)	$z = \sqrt{2x+1} + (2x+1)^{-3}$	`z =sqrt2x-1+(2*x+1)**-3`
d)	$a = \sqrt{c^2 - b^2}$	`a = (c**2-b*2)**(2/1)`

Problem 4-4:

Kaj se izpiše kot rezultat?

```
from math import *
x=22.2
y=44.4
a=sqrt(x)
print(a)
b=sqrt(x+y)
print(b)
```

Problem 4-5:

Kaj se izpiše kot rezultat?

```
from math import *
xv=1000
a=log(xv)
b=log10(xv)
print(a)
print(b)
```

Problem 4-6:

Z uvedbo dodatnih spremenljivk razstavite enačbi v več enostavnejših enačb; pri tem predpostavite, da so vse spremenljivke na desni strani že podane. Pri prvem primeru vsak ulomek zamenjajte z novo spremenljivko. Pri drugem primeru poskušajte doseči, da ima vsak ulomek samo eno spremenljivko v imenovalcu in samo eno v števcu.

a)

$$p = \frac{RT}{V_m - b} - \frac{a(T)}{V_m(V_m + b)}$$

b)

$$p = \frac{RT}{V_m - b} - \frac{a(T)}{V_m(V_m + \eth b) + \zeta b(V_m - b)}$$

Problem 4-7:

Napišite program, ki bo izračunal dolžino hipotenuze v pravokotnem trikotniku, če sta kateti dolgi 3 cm in 4 cm. Izpišite izračunano dolžino hipotenuze.

Problem 4-8:

Napišite program, ki bo izračunal cos 60° in sin 30° ter izpisal oba rezultata. Kotne stopinje je treba pretvoriti v radiane. 180° = 3.141593 rad. Namig: število π lahko v *Pythonu* prikličemo s simbolom `pi`.

5 Izpis in vnos podatkov

5.1 Ukaz print

V prejšnjih poglavjih smo že uporabili ukaz `print` za enostavno izpisovanje vrednosti spremenljivk. V tem poglavju je prikazanih še nekaj drugih načinov uporabe tega ukaza. Poleg vrednosti posameznih spremenljivk lahko izpišemo tudi vrednosti računskih izrazov, besedilo ali združimo izpis besedil in števil. Pri tem mora biti tekst, ki ga želimo izpisati, v narekovajih. Uporabimo lahko enojni ali dvojni narekovaj.

Primer 5-1:

```
print(12)
print(3*3)
print("Število atomov je", 3*2)
```

Po zagonu programa se izpiše:

```
12
9
Število atomov je 6
```

Če v *Pythonu* uporabimo ukaz `print` z več argumenti, v izpisu privzeto vstavi med argumente en presledek.

Primer 5-2:

```
a = 3
b = 4
c = a*b**2
print("Vrednosti so:", a, b, c)
```

Po zagonu programa se izpiše:

```
Vrednosti so: 3 4 48
```

Če želimo, da so vrednosti izpisane vsaka v svoji vrstici, lahko za vsako spremenljivko uporabimo svoj ukaz `print` ali uporabimo ukaz `'\n'` (še enkrat omenimo, da lahko uporabimo tako enojni kot dvojni narekovaj):

```
print("Vrednosti so:",'\n', a,'\n', b,'\n', c)
```

ali

```
print("Vrednosti so:", a, b, c, sep='\n')
```

Po zagonu programa se izpiše:

```
Vrednosti so:
 3
 4
 48
```

Če želimo, da so vrednosti zapisane v isti vrstici s presledkom treh mest, zapišemo:

```
print("Vrednosti so:", a, b, c, sep="   ")
```

Primer 5-3:

Naslednji primer prikazuje koristnost uporabe spremenljivk. Če namreč vrednosti spremenljivk spremenimo na enem mestu v programu, se sprememba uveljavi povsod v programu, kjer nastopa ime te spremenljivke.

```
oseba_ime='Ana'
oseba_starost='32'
print('Oseba se imenuje', oseba_ime)
print('in je stara', oseba_starost, 'let.')
print('Vsec ji je ime', oseba_ime,'.')
```

Po zagonu programa se izpiše:

```
Oseba se imenuje Ana
in je stara 32 let.
Vsec ji je ime Ana .
```

Če bi spremenili ime in starost osebe v prvih dveh vrsticah programa, bi se izpis prilagodil novim podatkom.

V stavku `print` lahko namesto vejic uporabimo tudi znak `+` (concatenation). V zgornjem primeru npr. drugi `print` stavek:

```
print('in je stara ' + oseba_starost + ' let.')
```

Opozoriti velja, da je v programu spremenljivka `oseba_starost` podana kot znakovna spremenljivka (`'32'`), saj je v narekovajih. Če je ne bi podali v narekovajih, bi jo program obravnaval kot numerično spremenljivko in bi morali izpis v zgornjem `print` stavku dopolniti z dodatnim ukazom `str`, ki numerično spremenljivko spremeni v znakovno:

```
print('in je stara ' + str(oseba_starost) + ' let.')
```

5.2 Aritmetični vnos

Aritmetični vnos podatkov pomeni, da spremenljivki pripišemo vrednost s prireditvenim stavkom, ki je del programa. To pomeni, da če želi uporabnik spremeniti vrednost spremenljivke, mora poseči neposredno v program.

Primer 5-4:

```
ime = "Ana"
Pi = 3.14159
print(ime, Pi)
```

Po zagonu programa se izpiše:

```
Ana 3.14159
```

5.3 Interaktivni vnos

Uporabnik lahko vstavi vrednosti spremenljivk interaktivno prek ekrana z ukazom `input`. Ta način pomeni, da uporabnik ob vstavljanju podatkov ne posega v program. Slabost tega pristopa je, da lahko postane nepregleden v primeru velikega števila podatkov.

Primer 5-5:

```
ime = input("Vstavite svoje ime: ")
print("Zdravo, " + ime + "!")
```

Po zagonu programa, ko uporabnik vstavi svoje ime, npr. Janez Novak, se izpiše:

```
Vstavite svoje ime: Janez Novak
Zdravo, Janez Novak!
```

Vnos z ukazom `input` vedno privzame, da so vstavljene vrednosti nizi, torej znakovne spremenljivke. Če jih v programu želimo uporabljati kot numerične vrednosti, jih moramo spremeniti iz znakovnih v numerične; to storimo s funkcijo `int` za cela števila in `float` za realna števila:

Primer 5-6:

```
x=input("Vnesi prvo celo število: ")
y=input("Vnesi drugo celo število: ")
rezultat = int(x)+int(y)
print(rezultat)
```

Primer 5-7:

```
x=input("Vnesi prvo realno število: ")
y=input("Vnesi drugo realno število: ")
rezultat = float(x)+float(y)
print(rezultat)
```

Funkcijo `float` lahko uporabimo tako za realna kot za cela števila, seveda pa bo program vneseno celo spremenljivko potem obravnaval kot realno.

Primer 5-8:

```
a=input("Vnesi celo število: ")
b=input("Vnesi realno število: ")
a=float(a)
b=float(b)
c=a+b
print(a,b,c)
```

Če vnesemo npr. števili 4 in 6,3, se po zagonu programa izpiše:

```
4.0 6.3 10.3
```

Če želimo preprečiti neustrezen vnos uporabnika, uporabimo strukturo `try-except`.

Primer 5-9:

Vzemimo, da mora uporabnik vstaviti celo število, pa namesto tega vstavi realno število ali tekst. V običajnem zapisu bi se izvajanje programa »nasilno« prekinilo. To preprečimo z naslednjo obliko:

```
try:
    stevilo = input("Vnesi celo stevilo: ")
    stevilo = int(stevilo)
    print(stevilo)
except ValueError:
    print("Napacen vnos")
```

Beseda `ValueError` označuje napačen tip podatka. Značilna napaka je tudi deljenje z nič oz. `ZeroDivisionError`, za kar lahko ustvarimo dodaten `except` stavek.

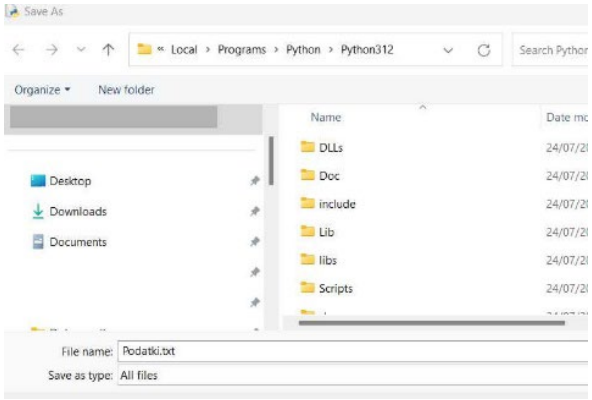
5.4 Branje podatkov iz datoteke

V primeru velikega števila vhodnih podatkov, ki se pogosto spreminjajo, je najugodnejši vnos iz datoteke. Na ta način uporabnik spreminja samo tekstovno datoteko s podatki in ne posega v programsko kodo.

5.4.1 Tvorjenje datoteke s podatki

Kako v urejevalniku IDLE tvorimo podatkovno datoteko? Z ukazoma `File` in `New File` odpremo novo datoteko, v katero vpišemo podatke. Nato jo shranimo z ukazoma `File` in `Save As` ter ob tem vpišemo:

- *File name*: ime datoteke, npr. Podatki.txt.
- *Save as type*: izberemo *All files* (slika 5-1).



Slika 5-1: Tvorjenje podatkovne datoteke

Datoteka s podatki mora biti shranjena v istem direktoriju kot program *Python*; v nasprotnem primeru je treba v programu vpisati pot do datoteke.

5.4.2 Povezava podatkovne datoteke s programsko datoteko

Splošni ukaz v programski kodi za odpiranje datoteke, iz katere želimo prebrati podatke, je:

```
Ime_spremenljivke = open("Ime_datoteke", "Način_dostopa")
```

Način dostopa je lahko:

- Samo branje ("r"): odpre datoteko za branje.
- Izpis in branje ("r+"): odpre datoteko za branje in pisanje.

Datoteko je po branju priporočeno zapreti, za kar uporabimo ukaz:

```
Ime_spremenljivke.close()
```

Za branje podatkov lahko uporabimo ukaze `read, readline` ali `readlines`:

- Stavek `read` prebere celotno vsebino datoteke kot eno znakovno vrednost in je primeren, ko želimo prebrati celotno vsebino datoteke naenkrat. Primeren je za majhne datoteke, saj vrne vsebino kot en seznam in ne loči vrstic.
- Ukaz `readline` prebere eno vrstico v datoteki.
- Ukaz `readlines` prebere celotno vsebino datoteke, jo razdeli na posamezne vrstice in tvori seznam iz vrstic. Vsaka vrstica je en element seznama. O seznamih bomo več govorili v 8. poglavju.

Primer 5-10:

Datoteka s podatki se imenuje Podatki.txt in je shranjena v istem direktoriju kot koda *Python*:

```
23
1.025
8.314
```

Koda v *Pythonu*:

```
vsebina = open("Podatki.txt", "r")
f=vsebina.readlines()
vsebina.close()
temperatura=float(f[0])
tlak=float(f[1])
R=float(f[2])
print(temperatura, tlak, R)
```

Po zagonu se izpiše:

```
23.0 1.025 8.314
```

Opomba: Program prebere vrednosti v datoteki Podatki.txt kot znake, zato je uporabljen ukaz `float`, ki spremenljivkam dodeli numerične vrednosti.

Primer 5-11:

Podatke v podatkovni datoteki lahko zapišemo tudi v isti vrstici, ločene s presledki. V tem primeru za branje vrstice uporabimo ukaz `readline` in `split`, ki prebrano vrstico razdeli na posamezne podatke.

Datoteka s podatki se imenuje Podatki.txt in je shranjena v istem direktoriju kot koda *Python*:

```
Irena 165 62.3
```

Koda v *Pythonu*:

```
vsebina = open("Podatki.txt", "r")
f=vsebina.readline().split()
vsebina.close()
ime= f[0]
visina=float(f[1])
masa=float(f[2])
print(ime, visina, masa)
```

Izpis:

```
Irena 165.0 62.3
```

Primer 5-12:

Datoteka s podatki se imenuje testread.txt in je shranjena v istem direktoriju kot koda *Python*:

```
Zdravo.
To je test.
2024
```

Koda v *Pythonu*:

```
fp=open('testread.txt', 'r')
a=fp.read()
fp.close()
print(a)
```

Po zagonu se izpiše:

```
Zdravo.
To je test.
2024
```

Če bi uporabili ukaz `readline`, bi se izpisala le prva vrstica datoteke testread.txt:

```
Zdravo.
```

Če bi uporabili ukaz `readlines`, bi bil izpis:

```
['Zdravo.\n', 'To je test.\n', '2024']
```

Izpis z uporabo ukaza `readlines` vsebuje tudi znake za prelom vrstice (\n), ki jih odstranimo z ukazom `strip()` in lahko dostopamo do posameznega elementa seznama.

Primer 5-13:

```
fp=open('testread.txt', 'r')
a=fp.readlines()
fp.close()
a=[line.strip() for line in a]
print(a)
```

Po zagonu se izpiše:

```
['Zdravo.', 'To je test.', '2024']
```

5.5 Izpis podatkov v datoteko

Osnovni ukaz za izpis podatkov `print` smo že spoznali. Rezultate izpiše na zaslon. V tem poglavju je prikazan izpis podatkov v datoteko.

Splošni ukaz za odpiranje datoteke, v katero želimo izpis rezultatov, je:

```
Ime_spremenljivke = open("Ime_datoteke", "Način_dostopa")
```

Način dostopa je lahko:

- Samo izpis (`'w'`): odpre datoteko za pisanje. Če datoteka obstaja, prepiše podatke. Kreira datoteko, če še ne obstaja.
- Izpis in branje (`'w+'`): odpre datoteko za pisanje in branje. Če datoteka obstaja, prepiše podatke.
- Dodajanje (`'a'`): odpre datoteko za izpisovanje. Kreira datoteko, če še ne obstaja. Podatki se izpišejo za obstoječimi podatki v datoteki.

Primer 5-14:

```
a=3
datoteka = open("Rezultati.txt", "w")
print('Halo \n', a, file=datoteka)
datoteka.close()
```

Po zagonu se v datoteki Rezultati.txt izpiše:

```
Halo
3
```

Datoteko z rezultati najdemo v istem direktoriju, kot je program.

5.6 Problemi 5. poglavja

Problem 5-1:

Kakšen bo izpis naslednjega programa?

```
a = 3
b = 4
c = a*b**2
d="mesto Maribor"
print("Izpis bo:","\n", a, b,"\n", c,"\n", d)
print("mesto\nLjubljana")
```

Problem 5-2:

Podani sta spremenljivki `a=3` in `b=4`. Uporabite ukaz `print`, da dobite naslednji izpis, pri čemer morate v stavku `print` uporabiti imeni spremenljivk *a* in *b*, ne njunih vrednosti:

```
c=3**4
```

Problem 5-3:

Predpostavite:

```
a=2
b=3
c=4
```

Zapišite ukaz `print`, da dosežete naslednje izpise:

a) `2 3 4`

b) `4 3 2`

c) `234`

d) `2, 3, 4`

e)
```
2
3
4
```

Problem 5-4:

Predpostavite, da je v datoteki `input.txt` 10 vrstic besedila. Izdelajte program, ki prebere besedilo iz datoteke input.txt in podatke zapiše v datoteko `output.txt`.

Problem 5-5: (po 9. poglavju)

Predpostavite, da je v datoteki `input.txt` 10 vrstic besedila. Izdelajte program, ki prebere besedilo iz datoteke input.txt in v datoteko `output.txt` zapiše vsako drugo vrstico.

6 Oblikovanje izpisa

Oblikovanje izpisa omogoča, da oblikujemo razmike med izpisanimi vrednostmi, število decimalnih mest ipd. Formatiranje je v *Pythonu* zelo obsežno področje. Obstaja celo notranji »mini-language« za številne možnosti formatiranja. V nadaljevanju je prikazanih nekaj osnovnih primerov; več informacij je na voljo na spletu (ključne besede za iskanje: Python format specification mini-language, f-strings, format specifiers, string formatting).

6.1 Število decimalnih mest

Realno število oblikujemo za izpis z določenim številom decimalnih mest na dva načina:

a) Določimo skupno število vseh števk, ki jih želimo v številu;
b) določimo, koliko števk želimo za decimalno piko.

6.1.1 Skupno število števk v številu

Za določitev skupnega števila števk uporabimo dvopičje, ki mu sledi pika skupaj s celim številom, ki predstavlja število števk brez decimalne pike. To postavimo v zavit oklepaj, torej: `{:.n}`, kjer je `n` skupno število števk brez decimalne pike. Ob formatiranju se izvede zaokroževanje.

Primer 6-1:

Izpišimo število na dve decimalni mesti.

```
x=4863.4363091
print("{:.6}".format(x))
```

Po zagonu se izpiše:

```
4863.44
```

Uporabimo lahko tudi `f-string`, ki da povsem enak izpis in ima splošno obliko: `f" "` (dovoljena sta mali `f` in veliki `F`), v katero med narekovaja vstavimo `{spremenljivka:.n}` ter poljuben tekst:

```
x=4863.4363091
print(f"{x:.6} je rezultat.")
```

Izpis:

```
4863.44 je rezultat.
```

Primer 6-2:

Izpis števila v svojo vrstico.

```
x=4863.4363091
a=14.2345
print(f"{x:.6}")
print(f"{a:.5}")
```

ali en `print` stavek namesto dveh, v katerem uporabimo znak za novo vrstico `\n`:

```
print(f"{x:.6}\n{a:.5}")
```

Po zagonu se izpiše:

```
4863.44
14.235
```

Primer 6-3:

Če definiramo manjše ali enako število števk, kot jih vsebuje celoštevilski del števila, dobimo eksponentni zapis.

```
x=4863.4363091
print(f"{x:.3}")
```

Po zagonu se izpiše eksponentna oblika števila s tremi uporabljenimi števkami:

```
4.86e+03
```

6.1.2 Število decimalnih mest

Za določitev števila decimalnih mest uporabimo zapis z orodjem `f-string` in znotraj tega ob spremenljivki, dvopičju ter piki še dodatno črko `f`, pred katero je navedena številka `n`, ki pomeni točno določeno število števk za decimalno piko, torej splošno: `{spremenljivka:.nf}`.

Primer 6-4:

```
x=4863.4363091
print(f"{x:.3f}")
```

Po zagonu se izpiše:

```
4863.436
```

Če bi uporabili samo črko `f` brez definiranega števila decimalnih mest, je privzetih 6 decimalnih mest.

Primer 6-5:

```
x=4863.4363091
print(f"{x:f}")
```

Po zagonu se izpiše:

```
4863.436309
```

6.2 Ločevanje tisočic

Zaradi lažjega branja pri velikih številih tisočice pogosto ločujemo z znaki, kot so pika, vejica ali presledek.

Primer 6-6:

```
x=1000000
print(f"{x:,}")
print(f"{x:,}".replace("," , " "))
```

Po zagonu se izpiše:

```
1,000,000
1 000 000
```

To lahko tudi združimo z določitvijo števila decimalnih mest, pri čemer zapišemo znak za ločevanje, npr. vejico, pred število decimalnih mest.

Primer 6-7:

```
x=4863.4363091
print(f"{x:,.3f}")
```

Po zagonu se izpiše:

```
4,863.436
```

6.3 Oblikovanje števil kot odstotki

Število lahko zapišemo kot odstotek, tako da črko `f` zamenjamo z znakom `%`, številka pred njim pa označuje število decimalnih mest.

Primer 6-8:

```
x=0.34567
print(f"{x:.2%}")
```

Po zagonu se izpiše:

```
34.57%
```

Primer 6-9:

```
a=34
b=116
print(f"{a/b:.4%}")
```

Po zagonu se izpiše:

```
29.3103%
```

6.4 Poravnava

Pri oblikovanju izpisa zaradi preglednosti pogosto želimo določeno število presledkov med izpisanimi spremenljivkami, pa tudi, da so te poravnane levo, desno ipd.

Primer 6-10:

Navaden `print` stavek izpiše imena treh kemikalij, ločena s po enim presledkom.

```
print("etanol","metanol","formaldehid")
```

Izpis:

```
etanol metanol formaldehid
```

Če želimo za vsako ime rezervirati določeno minimalno število mest in ga poravnati levo, uporabimo znak `:<n`, kjer je `n` število mest.

```
print(f"{"etanol":<10}{"metanol":<10}{"formaldehid":<10}")
```

Izpis:

```
etanol     metanol    formaldehid
```

V nekaterih urejevalnikih se v tem primeru pojavi konflikt med dvojnimi narekovaji v f-*stringu* in pri navajanju znakovnih spremenljivk. Temu se izognemo z uporabo različnih narekovajev, npr.:

```
print(f"{'etanol':<10}{'metanol':<10}{'formaldehid':<10}")
```

Za desno poravnavo bi uporabili znak `:>10`. Število predstavlja minimalno širino polja. Če je dejanska beseda daljša od določene širine, bo vseeno prikazana v celoti in polje bo razširjeno tako, da bo vsebovalo celotno besedo.

6.5 Izpisovanje teksta in števil

Spoznali smo že skupno izpisovanje teksta in števil. Uporabimo lahko tudi oblikovanje z orodjem `f-string`, tako da za črko `f` ali `F` med narekovaji zapišemo vse, kar želimo izpisati. Na mestu, kamor želimo vstaviti vrednost spremenljivke, uporabimo zavite oklepaje `{}` in vanje postavimo ime spremenljivke.

Primer 6-11:

```
mesec = "marec"
dan = 27
print(f"Danes je {dan}. {mesec}.")
```

Izpis:

```
Danes je 27. marec.
```

6.6 Komentarji

Komentarji so vrstice v programu, ki jih *Python* ignorira; to pomeni, da se ne izvedejo. Pomembni so za razvijalce ali uporabnike programa, da podajo kratko razlago, pojasnilo ali drugo opombo. Vrstico, v kateri bomo zapisali komentar, začnemo z znakom `#`. Komentar lahko zapišemo tudi na koncu vrstice. Komentarji se ob zagonu ne izpišejo.

Primer 6-12:

Kaj se izpiše?

```
# Program izracuna absolutno vrednost stevila.
# Uporabnik vstavi stevilo interaktivno.
# Rezultat se izpise na 4 decimalna mesta.

a=input("Vnesi število: ")
a = float(a)
```

```
b=abs(a)        # izracun absolutne vrednosti
print(f"Absolutna vrednost je {b:.4f}")
```

Primer 6-13:

Za vnos daljših komentarjev prek več vrstic lahko pred in za komentarjem uporabimo znak, sestavljen iz treh narekovajev (`'''` ali """), med katerimi ne sme biti presledkov.

```
'''
Program izracuna absolutno vrednost stevila. Uporabnik
vstavi stevilo interaktivno. Rezultat se izpise na 4
decimalna mesta.
'''
```

6.7 Problemi 6. poglavja

Problem 6-1:

Izpiši število 4.56744 kot:

a) samo celi del števila.
b) Samo decimalni del števila.
c) Navzgor zaokroženo celo število.
d) Navzdol zaokroženo celo število.
e) Število z dvema decimalnima mestoma, zaokroženo navzdol (4.56).
f) Število z dvema decimalnima mestoma, zaokroženo navzgor (4.57).
g) Kot 4,56744.

Problem 6-2:

Zapiši stavek `print` za izpis naslednjih števil: 12.34567 na tri decimalna mesta; 152.1 kot celo število; –1115.3 na dve decimalni mesti. Števila naj bodo izpisana v isti vrstici, poravnana levo; za vsako število predvidimo 10 mest.

Problem 6-3:

Želimo, da uporabnik vnese spremenljivko tipa *float*. Na to smo ga prijazno opozorili. Razmislite in naštejte možne stvari, ki lahko grejo narobe pri vnosu? Namig: uporabnik bo vnesel vse razen tega, kar se pričakuje od njega.

7 Pogojni stavki

Pogojni stavki so ključni del vsakega programskega jezika, saj omogočajo, da program sprejema odločitve na osnovi določenih pogojev. Ti stavki omogočajo, da se različni deli programa izvedejo glede na to, ali določeni pogoji veljajo ali ne. Pogojne stavke pogosto uporabljamo tudi v vsakdanjem življenju, npr.:

Zbudim se,
če sem lačna,
    jem zajtrk.
Odidem od doma,
če je oblačno,
    vzamem dežnik,
če ni,
    vzamem sončna očala.
Grem v restavracijo,
če želim meso,
    naročim zrezek,
če ne in želim testenine,
    naročim špagete,
če ne,
    naročim solato.

7.1 Primerjalni in logični operatorji

Za zapis pogojev uporabljamo primerjalne in logične operatorje. S prvimi primerjamo spremenljivke ali aritmetične izraze (tabela 7-1), z drugimi kombiniramo več pogojev (tabela 7-2).

Tabela 7-1: Primerjalni operatorji

==	Enako
!=	Ni enako
>	Večje
>=	Večje ali enako
<	Manjše
<=	Manjše ali enako

Tabela 7-2: Logični operatorji

`or`	vsaj eden od pogojev mora biti izpolnjen
`and`	oba oz. vsi pogoji morajo biti izpolnjeni
`not`	izpolnjeno mora biti nasprotje pogoja

Logični operatorji imajo različne prioritete, kar vpliva na vrstni red izvajanja izrazov. Najvišjo prioriteto ima operator `not`, nato `and` in nazadnje `or`.

Primer 7-1:

Prioritete logičnih operatorjev.

```
a = 4
b = 5
c = 7
result = not a==3 or b>c and c>a
print(result)
```

Izpis: `True`

Najprej se izvede izraz `not a==3`, ki je `True`, nato se izvede `b>c and c>a`, ki je `False`. Na koncu se izvede `True or False`, kar da končni rezultat `True`.

Pri kompleksnih izrazih je priporočljivo uporabiti oklepaje za boljšo preglednost in preprečevanje napak, recimo v zgornjem primeru:

```
result = (not a==3) or (b>c and c>a)
```

7.2 Zapis pogojnih stavkov

V *Pythonu* so pogojni stavki zapisani s ključnima besedama `if` in `else`. Za `if` zapišemo pogoj(e) in vrstico zaključimo z dvopičjem. Sledijo vrstice, ki se izvedejo, če je pogoj izpolnjen. Zapisane so zamaknjeno. Zatem sledi beseda `else`, ki je prav tako zaključena z dvopičjem in ji zamaknjeno sledijo vrstice, ki se izvedejo, če pogoj ni izpolnjen. Ko so stavki v bloku `else` končani, se nadaljnje vrstice zapisujejo nezamaknjeno, kar je znak, da je struktura `if-else` končana.

Splošna oblika:

```
if pogoj:
    ukazi, ki se izvedejo, če je pogoj izpolnjen (true)
else:
    ukazi, ki se izvedejo, če pogoj ni izpolnjen (false)
```

Več pogojev lahko sprogramiramo s strukturo, ki med ukaza `if` in `else` doda še enega ali več ukazov `else-if` (kratko elif), tj. strukturo `if-elif-else`, ki ima naslednjo obliko:

```
if pogoj_1:
    ukazi, ki se izvedejo, če je pogoj_1 izpolnjen (true)
elif pogoj_2:
    ukazi, ki se izvedejo, če je pogoj_2 izpolnjen (true)
elif pogoj_3:
    ukazi, ki se izvedejo, če je pogoj_3 izpolnjen (true)
...

else:
    ukazi, ki se izvedejo, če noben pogoj ni izpolnjen (false)
```

Primer 7-2:

Pogojni stavek z numeričnimi spremenljivkami.

```
a=4
if a==4:
    a=2*a
else:
    a=5*a
print(a)
```

Izpis:

```
8
```

Primer 7-3:

V čem se naslednja različica programa razlikuje od predhodne in kaj se izpiše?

```
a=4
if a==4:
    a=2*a
else:
    a=5*a
    print(a)
```

Izpis: Nič se ne izpiše. Zakaj ne?

Primer 7-4:

Če obstaja specifično zaporedje ukazov samo za primer, ko je pogoj izpolnjen, lahko v strukturi pogojnega stavka `else` tudi izpustimo.

```
starost = 20
if starost >= 18:
    print("Oseba je polnoletna.")
```

Izpis:

```
Oseba je polnoletna.
```

Primer 7-5:

Več pogojev z logičnimi operatorji.

```
a=4
b=12
if a==5 or b==5:
    c=0

else:
    c=a+b
print(c)
```

Izpis:

```
16
```

Primer 7-6:

Pogojni stavek z logičnimi spremenljivkami.

```
je_krog = True
je_moder = False
if je_krog and je_moder:
    print("To je moder krog.")
else:
    print("To ni moder krog.")
```

Izpis:

```
To ni moder krog.
```

Primer 7-7:

Sprogramirajmo kalkulator, v katerega bo uporabnik vnesel dve števili in želeno aritmetično operacijo +, −, / ali *.

```
num1=float(input("Enter first number: "))
op=input("Enter operator: ")
num2=float(input("Enter second number: "))

if op == "+":
    print(num1 + num2)
elif op == "-":
    print(num1-num2)
elif op == "/":
    print(num1/num2)
elif op == "*":
    print(num1*num2)
else:
    print("Invalid operator")
```

7.3 Problemi 7. poglavja

Problem 7-1:

Zapišite pogojne stavke:

a) Če je *n* enak 0, se spremenljivki *x* dodeli vrednost 10. Izpiši vrednost spremenljivke *x.*
b) Če je *a* večje od *b*, izpiši vrednost spremenljivke *a*.
c) Če je *x* večji od *y* in *a* manjši od *b*, se spremenljivki *c* dodeli vrednost *z*. Izpiši vrednost spremenljivke *c.*
d) Če je *X* večji od 0, dobi spremenljivka *Y* vrednost $\sqrt{X}$. Izpiši vrednosti spremenljivk *X* in *Y.*

Problem 7-2:

Zapišite pogojne stavke:

a) Če je *a* enak *b* in *b* enak *c,* izpiši »a je enak c«. V nasprotnem primeru izpiši »a je morda enak c«.
b) Če je *b* enak 2a, izpiši »b = 2a«. V nasprotnem primeru izpiši »b != 2a«.
c) Če je *b* delitelj števila *a*, izračunaj kvocient *c* in izpiši »b je delitelj a« ter »kvocient je«, čemur naj sledi izpis vrednosti spremenljivke c*.* V nasprotnem primeru naj se izpiše »b ni delitelj a«.

Problem 7-3:

Zapišite pogojne stavke:

Če je *a* Na ali K in *b* H_2O, izpiši »Pazi, eksplozija!«, drugače pa izpiši »Vrei je … verjetno«.

Problem 7-4:

Zapišite pogojne stavke:

Če *a* ni K in ne Na in tudi ne H_2O ali če *b* ni K in ne Na in tudi ne H_2O, izpiši »Ne prepoznam spojine!«.

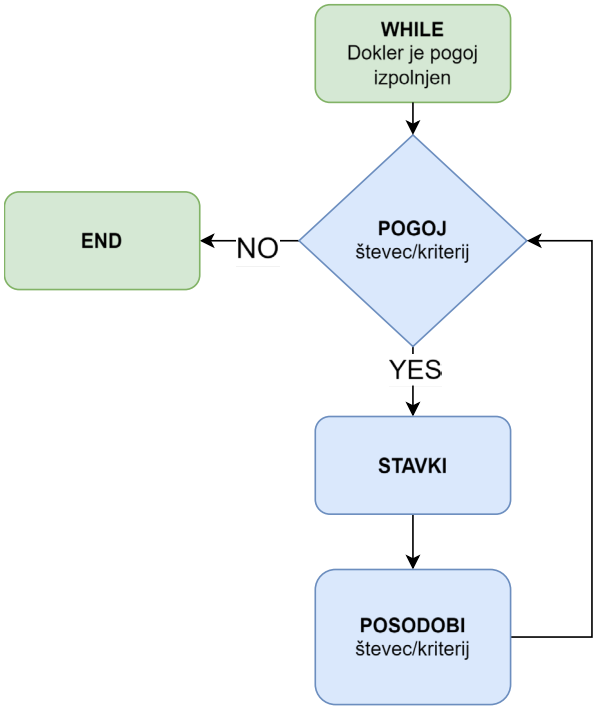
V nasprotnem primeru pa: če je katera koli od kombinacij *a* in *b* taka, da je nevarnost, da pride do eksplozije (*a* je Na ali *a* je K in *b* je H_2O) ali (*a* je H_2O in *b* je Na ali *b* je K), izpiši »Pazi, eksplozija!«, drugače pa izpiši »Vrei je … verjetno«.

8 Zanke

Zanke uporabimo, ko želimo določeno zaporedje ukazov izvesti večkrat zapored. Takšno zaporedje ukazov zapišemo v zanki, ki se ponavlja, dokler je izpolnjen določen pogoj (zanka *while*) ali točno določeno število *krat* (zanka *for*). S tem se izognemo temu, da bi za vsako ponovitev ukazov pisali isto zaporedje stavkov.

8.1 Zanka while

Ključna beseda te zanke je `while`, ki ji sledita določen pogoj in dvopičje. S pogojem definiramo, do kdaj se bo izvajalo zaporedje ukazov (telo zanke), ki sledi vrstici `while`. S pogojem ni definirano točno število prehodov skozi zanko, ampak se ukazi v telesu zanke ponavljajo tako dolgo, dokler je pogoj izpolnjen (*true*). Če oz. ko pogoj ni več izpolnjen (*false*), se prehodi skozi zanko končajo in program nadaljuje izvajanje v prvi vrstici za zanko (slika 8-1). Z modro barvo so prikazani deli zanke, ki jih moramo definirati sami: pogoj, izvršljivi stavki in kako naj se kriterij posodobi oz. spremeni po vsakem prehodu skozi zanko.



Slika 8-1: Struktura zanke `while`

Splošna oblika programa z zanko `while` je naslednja:

```
while pogoj:
    telo zanke (zaporedje ukazov,ki se izvajajo, dokler je
    pogoj izpolnjen)
    posodobitev pogoja
nadaljevanje programa (prva nezamaknjena vrstica)
```

Ukazi, ki tvorijo telo zanke, so obvezno zapisani zamaknjeno, kar se zgodi avtomatsko, ko stisnemo *enter* za prvo vrstico zanke z besedo `while`, s pogojem in z dvopičjem.

Primer 8-1:

```
i=6
while i <= 10:
    print(i)
    i=i+1
print("Konec")
```

Po zagonu se izpiše:

```
6
7
8
9
10
Konec
```

Opomba: ekvivalentna oblika zapisu `i=i+1` je `i+=1`. Operator `+=` se imenuje *addition assignment operator* in v našem primeru poveča isto spremenljivko za 1.

Primer 8-2:

```
skrito_stevilo = "4"
ugibam = ""
while ugibam != skrito_stevilo:
   ugibam = input("Ugani celo stevilo med 1 in 5: ")
print("Cestitke!")
```

Opomba: vrstica `ugibam = ""` predstavlja inicializacijo spremenljivke `ugibam` kot prazno vrstico. Za inicializacijo bi lahko uporabili tudi število, ki ni enako vrednosti spremenljivke, `skrito_stevilo`, namesto prazne vrstice. To moramo storiti zato, da pred prvim prehodom zanke `while` program pozna neko vrednost spremenljivke, s katero primerja `skrito_stevilo` in tako omogoči vstop v zanko ter prikaz uporabniškega vnosa.

Primer 8-3:

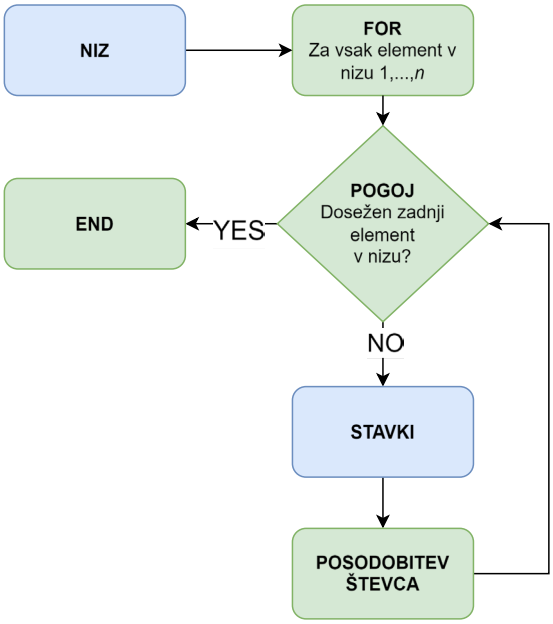
Zapis sodih števil med 2 in 20.

```
i = 2
while i <= 20:
    print(i)
    i=i+2
```

8.2 Zanka for

To zanko uporabimo za vnaprej določeno število ponovitev ukazov. Zanko shematsko prikazujemo na sliki 8-2. Podobno kot na sliki 8-1 so tudi tukaj z modro barvo prikazani deli zanke, ki jih moramo definirati sami. V primeru zanke `for` so to niz, prek katerega se izvrši zanka, in izvršljivi stavki.

Uporaba zanke `for` je tesno povezana s seznami, zato bomo na tem mestu predstavili samo njeno osnovno uporabo; več bo prikazano v poglavju 9 Seznami.



Slika 8-2: Struktura zanke `for`

Splošna oblika zanke `for` je naslednja:

```
for element v nizu:
    telo zanke (zaporedje ukazov,ki se izvajajo, dokler ni
    dosežen zadnji element v nizu)
nadaljevanje programa (prva nezamaknjena vrstica)
```

Ključna beseda je `for`, ki ji sledi na določen način izraženo točno število ponovitev oz. prehodov zanke (slika 8-2), vrstica pa se zaključi z dvopičjem. Ukazi, ki tvorijo telo zanke, so zapisani zamaknjeno. Zadnja vrstica, ki je zapisana zamaknjeno, je znak, da se zanka v tej vrstici zaključi. Prva nezamaknjena vrstica zatem predstavlja nadaljevanje programa.

Za definiranje števila prehodov skozi zanko pogosto uporabljamo funkcijo `range`:

- ukaz `range(n)` ustvari zaporedje celih števil od 0 do *n*-1 s korakom po 1;
- ukaz `range(m,n)` pa zaporedje od *m* do *n*-1 s korakom po 1;
- če želimo drugačen korak od 1, je ukaz `range(m,n,k)`, ki ustvari zaporedje celih števil med *m* in *n*-1 s korakom *k*.

Primer 8-4:

S funkcijo `range(5)` ustvarimo zaporedje naravnih števil med 0 in 4 (4 je 5 minus 1). V vsaki iteraciji števec *i* prevzame vrednost enega od teh števil. Ob vsakem prehodu skozi zanko se z ukazom `print` izpiše trenutna vrednost števca *i*. Vrednosti se izpišejo vsaka v svoji vrstici.

```
for i in range(5):
    print(i)
```

Izpis:

```
0
1
2
3
4
```

Primer 8-5:

V tem primeru s funkcijo `range(1,5)` ustvarimo zaporedje števil med 1 in 4 (4 = 5 – 1).

```
for i in range(1,5):
    print(i)
```

Izpis:

```
1
2
3
4
```

Primer 8-6:

V zanko lahko vključimo pogojni stavek. Ob vsakem prehodu skozi zanko se preveri pogoj.

```
for n in range(5):
    if n == 0:
        print("Prva iteracija")
    else:
        print("Ni prva iteracija")
```

Izpis:

```
Prva iteracija
Ni prva iteracija
Ni prva iteracija
Ni prva iteracija
Ni prva iteracija
```

Primer 8-7:

Program naj prebere pet števil iz datoteke PODAT2.DAT, nato pa z uporabo zanke `for` izračuna vsoto teh števil. Vsako število se prebere posebej ob prehodu skozi zanko in se ne shranjuje v seznam.

Datoteka PODAT2.DAT

```
25
30
10
40
15
```

Program v *Pythonu*:

```
fp=open('PODAT2.DAT', 'r')
total=0

for i in fp:
    a=float(i)
    total=total+a
print(total)
```

Izpis:

```
120.0
```

Opomba: Dve vrstici v telesu zanke lahko združimo v eno. S tem ni potrebe po vpeljavi spremenljivke `a` (vse ostalo v programu ostane enako):

```
total=total+float(i)
```

Pri primeru 8-7 smo za iteriranje po datoteki uporabili zanko `for`. *Python* samodejno prebere datoteko po vrsticah. Števec *i* ob vsakem prehodu prevzame vrednost števila v naslednji vrstici v datoteki. To pomeni, da ni treba posebej klicati funkcij `read()` ali `readlines()`. Vsaka iteracija zanke prebere eno vrstico iz datoteke, kar poenostavi delo in zmanjša porabo pomnilnika, saj *Python* v vsakem trenutku obravnava le eno vrstico.

Primer 8-8:

Ta primer povezuje zanko in pogojni stavek. Iz datoteke OCENE.DAT preberemo ocene za pet študentov. Nato izpišemo določen tekst za študente, ki so dosegli pozitivno oceno, in drug tekst za študente, ki je niso.

Datoteka s podatki OCENE.DAT:

```
87.7
55.
75
60
44.8
```

Program:

```
belezka=open('OCENE.DAT', 'r')
for i in belezka:
    ocena = float(i)
    if ocena >= 60:
        print(ocena, "Opravil")
        print("---------------")
    else:
        print(ocena, "Ni Opravil")
        print("---------------")
```

Izpis:

```
87.7 Opravil
---------------
55.0 Ni Opravil
---------------
75.0 Opravil
---------------
60.0 Opravil
---------------
44.8 Ni Opravil
---------------
```

8.3 Problemi 8. poglavja

Problem 8-1:

Izračunajmo tlak idealnega plina pri temperaturi 0 °C za množine od 1 do 5 mol s korakom po 1 mol. Vhodne podatke splošno plinsko konstanto, temperaturo in volumen podamo v datoteki TLAK.DAT.

Problem 8-2:

Preračunajte stopinje Fahrenheita od 0 do 212 v Celzije s korakom po 2. Izpišejo naj se vsi pari vrednosti stopinj Fahrenheita in Celzija.

Problem 8-3:

Zapišite svoje ime 10-krat.

Problem 8-4:

Napišite program, ki bo izpisal vsa neparna števila od 1 naprej, manjša od 100.

Problem 8-5:

Za cela števila od 1 do 15 napišite program, v katerem boste izpisali tabelo števil in kvadratne ter kubične vrednosti teh števil.

Problem 8-6:

Napišite program za pretvarjanje milj od 10 do 100 s korakom po 10 v kilometre. Pretvornik je: 1 milja = 1,6 km. Izpišejo naj se vsi pari vrednosti v miljah in kilometrih.

Problem 8-7:

Kaj se izpiše kot rezultat programa?

```
i=3
while i <=8:
    vrednost=i-3
    print(i, vrednost)
    i=i+1
```

Problem 8-8:

Kaj se izpiše kot rezultat programa?

```
for k in range(100,2001,100):
    stoletje=(k-100)/100 + 1
    print("Leto: ", k," Stoletje:  ", int(stoletje))
```

Problem 8-9:

Kaj je rezultat naslednjega programa?

```
for i in range(11):
    if(i>7):
        print(i)
```

Problem 8-10:

V podjetju so zaposleni trije delavci. Za vsakega delavca so na voljo podatki: ime in priimek, število opravljenih mesečnih ur in vrednost ure. Če delavec dela 174 ur ali manj, se mu plača izračuna kot produkt števila ur in vrednosti ure. Če dela več kot 174 ur, se mu 174 ur ovrednoti z osnovno vrednostjo ure, medtem ko se ure nad 174 obravnavajo kot nadure in so ovrednotene s 130 % vrednosti ure. Napišite program, v katerem izračunajte in izpišite: ime in priimek; znesek osnovne plače v € in znesek nadur za vsakega delavca v €. Podatki so naslednji in naj bodo shranjeni v datoteki DELAVCI.DAT:

Ime in priimek	Število ur	Vrednost ure (€)
Ime1 Priimek1	185	10,44
Ime2 Priimek2	145	13,58
Ime3 Priimek3	174	18,22

Problem 8-11:

Sprogramirajte kodo, kjer mora uporabnik v največ treh poskusih uganiti skrito celo število med 1 in 5.

9 Seznami

Do sedaj smo pod imenom spremenljivke shranjevali samo eno vrednost. Pogosto imamo opravka z velikim številom podobnih podatkov, s katerimi želimo izvajati podobne izračune, npr. izračunati tlak plina pri različnih temperaturah. Pri tem uporabimo vedno iste enačbe, a različne vhodne podatke. V tem primeru za spremenljivke uporabimo sezname (polja, vektorje, angl. list).

9.1 Uvod v sezname

Seznam definiramo z oglatimi oklepaji, v katerih naštejemo elemente, ločene z vejico. Vsakemu elementu pripada njegov indeks, ki je celo število in se začne z 0, 1, 2, 3 itd.

Primer 9-1:

Seznam `x` vsebuje tri elemente, to so števila 6, 3 in 8. Prvemu elementu seznama, to je številu 6, pripada indeks 0; številu 3 pripada indeks 1 in številu 8 indeks 2.

```
x=[6, 3, 8]
print(x)
a=x[0]
b=x[1]
c=x[2]
print("a = ",a)
print("b = ",b)
print("c = ",c)
```

Po zagonu se izpiše:

```
[6, 3, 8]
a = 6
b = 3
c = 8
```

Primer 9-2:

```
prijatelji=["Jana","Marko","Ana","Irena"]
print(prijatelji)
print(prijatelji[0])
print(prijatelji[2])
print(prijatelji[-1])
print(prijatelji[1:3])
```

Po zagonu se izpiše:

```
['Jana', 'Marko', 'Ana', 'Irena']
Jana
Ana
Irena
['Marko', 'Ana']
```

Poleg seznamov obstajajo v *Pythonu* še terke (*tupples*), ki se definirajo podobno kot seznami, le da se namesto oglatih uporabijo okrogli oklepaji. Pomembna razlika je ta, da so elementi terk fiksni, torej jih ni mogoče spreminjati.

9.2 Uporaba funkcij za sezname

Funkcije omogočajo, da dobimo različne informacije o seznamih ali jih spreminjamo.

9.2.1 Informacije o seznamih

Uporabne funkcije za sezname so `max`, `min`, `sum`, `len`, s katerimi dobimo največjo in najmanjšo vrednost seznama, vsoto števil v seznamu in število elementov seznama. To so splošne, samostojne funkcije in delujejo z različnimi vrstami podatkov (ne le s seznami), zato ime seznama nastopa v njih kot argument funkcije:

Primer 9-3:

```
stevila=[6, 3, 8, 12, -5]
st_min=min(stevila)
st_max=max(stevila)
vsota=sum(stevila)
print(st_min, st_max, vsota)
povprecje=sum(stevila)/len(stevila)
print(povprecje)
dolzina_seznama=len(stevila)
print(dolzina_seznama)
```

Po zagonu se izpiše:

```
-5 12 24
4.8
5
```

Omenimo še funkcijo `count`, ki vrne število pojavljanj določenega elementa v seznamu. Ta funkcija je funkcija objekta, v tem primeru seznama, zato je zapis za uporabo funkcije na seznamu nekoliko drugačen kot pri prejšnjih funkcijah.

Primer 9-4:

Preštejmo, koliko krat se v seznamu `stevila` ponovi število 3.

```
stevila = [1, 2, 3, 4, 3, 8, 3, 12, 7, 3]
ponovitve = stevila.count(3)
print(ponovitve)
```

Izpis:

```
4
```

9.2.2 Spreminjanje seznamov

Elemente seznamov lahko spreminjamo, dodajamo, odvzemamo ipd.

Primer 9-5:

Začetni seznam *prijatelji* ima tri znakovne spremenljivke `Jana, Marko, Ana`. Jani pripada indeks 0; Marku 1 in Ani 2.

```
prijatelji=["Jana","Marko","Ana"]
print(prijatelji)
```

Po zagonu se izpiše:

```
['Jana', 'Marko', 'Ana']
```

Spremenljivko z indeksom 1 (`Marko`) zamenjamo z `Gregor` tako, da zgornjemu programu pred stavek `print` dodamo naslednjo vrstico:

```
prijatelji[1]= "Gregor"
```

Po zagonu se izpiše:

```
['Jana', 'Gregor', 'Ana']
```

Trenutnemu seznamu elementov dodamo element `Mojca` in tako seznam razširimo s treh na štiri elemente oz. indekse: z 0, 1, 2 na 0, 1, 2, 3. Na seznamu `prijatelji` uporabimo ukaz `append` (pred stavkom `print`), ki doda element na konec seznama.

```
prijatelji.append("Mojca")
```

Po zagonu se izpiše:

```
['Jana', 'Gregor', 'Ana', 'Mojca']
```

V zgornji seznam elementov vrinemo dodatni element `Janez` na mesto z indeksom 2, kar pomeni med elementa `Gregor` in `Ana`. Na seznamu `prijatelji` uporabimo ukaz `insert`, tako da dodamo naslednjo vrstico:

```
prijatelji.insert(2, "Janez")
```

Po zagonu se izpiše:

```
['Jana', 'Gregor', 'Janez', 'Ana', 'Mojca']
```

V zgornji seznam elementov lahko dodamo dodatni seznam, tako da na seznamu `prijatelji` uporabimo ukaz `extend`:

```
stevila=[3, 12]
prijatelji.extend(stevila)
```

Po zagonu se izpiše:

```
['Jana', 'Gregor', 'Janez', 'Ana', 'Mojca', 3, 12]
```

Če bi namesto ukaza `extend` uporabili `append`, bi bil izpis naslednji, saj ukaz `append` doda celotni objekt kot en element:

```
['Jana', 'Gregor', 'Janez', 'Ana', 'Mojca', [3, 12]]
```

To lahko potrdimo s funkcijo `len`, ki pove, koliko elementov je v seznamu. Ukaz:

```
a=len(prijatelji)
print(a)
```

bo dal v primeru `extend` vrednost 7, v primeru `append` pa 6.

Pozicijo določenega elementa v seznamu, tj. pripadajoči indeks elementa, podaja funkcija `index`.

Primer 9-6:

Določimo pozicijo oz. pripadajoči indeks elementa `"Janez"`:

```
prijatelji=["Jana","Gregor","Janez","Ana"]
print(prijatelji.index("Janez"))
```

Izpis:

```
2
```

Nekaj drugih uporabnih funkcij za spreminjanje seznamov je še:

- `remove` (odstrani naveden element);
- `pop` (odstrani zadnji element);
- `count` (vrne število navedenih elementov);
- `reverse` (zamenja vrstni red elementov);
- `clear` (izprazni seznam);
- `copy` (ustvari kopijo seznama);
- `sort` (razporedi znakovne elemente po abecedi oz. numerične spremenljivke po velikosti) itd.

9.3 Ustvarjanje seznamov

9.3.1 Ustvarjanje seznamov s funkcijo range

Za ustvarjanje seznamov je uporabna funkcija `range`, ki ustvari numerične vrednosti v določenem območju, vendar sama po sebi ne ustvari seznama iz generiranih vrednosti. Če želimo vrednosti iz območja, ki smo ga definirali s funkcijo `range`, pretvoriti v seznam, uporabimo ukaz `list`.

Primer 9-7:

Definirajmo naravna števila med 5 in 10 ter jih pretvorimo v seznam.

```
seznam=range(5,11)
x=list(seznam)
print(x)
```

Ali zgoščeno:

```
x=list(range(5,11))
print(x)
```

Po zagonu se izpiše:

```
[5, 6, 7, 8, 9, 10]
```

Omenimo še funkcijo `map`, ki omogoča uporabo določene funkcije na vsakem elementu seznama in ima obliko: `map(funkcija,seznam)`.

Primer 9-8:

Seznam `x` vsebuje naravna števila med 5 in 10. Ustvarimo seznam `y`, katerega elementi bodo števila, ki so naravni logaritmi števil seznama `x`:

```
from math import *
seznam=range(5,11)
x=list(seznam)
y=map(log,x)
z=list(y)
print(x)
print(z)
```

Ali zgoščeno:

```
from math import *
x=list(range(5,11))
z=list(map(log,x))
print(x)
print(z)
```

Izpis v obeh primerih je:

```
[5, 6, 7, 8, 9, 10]
[1.6094,1.7917,1.9459,2.0794,2.1972,2.3025]
```

Omenimo še funkcijo `zip`, ki združi istoležne elemente seznamov v pare oz. n-terice in tvori sezname n-teric ter ima obliko: `zip(seznam1,seznam2, ...)`.

Primer 9-9:

Izračunajmo povprečne vrednosti istoležnih elementov seznamov `a` in `b` z uporabo funkcije `zip`, pri čemer naj bo rezultat v obliki seznama. Seznam `povpr` na začetku inicializiramo kot prazen seznam, v katerega nato ob vsakem prehodu zanke dodamo element seznama:

```
a=[1,2,3]
b=[5,6,7]
c = list(zip(a,b))
print(c)
povpr=[]
for x, y in c:
    d=(x+y)/2
    povpr.append(d)
print(povpr)
```

Izpis:

```
[(1, 5), (2, 6), (3, 7)]
[3.0, 4.0, 5.0]
```

Program za izračun povprečne vrednosti istoležnih elementov seznamov `a` in `b` lahko zapišemo tudi brez funkcije `zip`. V tem primeru je treba definirati dimenzijo seznama `povpr`:

```
a=[1,2,3]
b=[5,6,7]
povpr=[None]*3          # ali povpr=[0,0,0]

for i in range(len(a)):
    povpr[i]=(a[i]+b[i])/2
print(povpr)
```

Primer 9-10:

Seznam `a` vsebuje soda števila med 2 in 20. Izpišimo posamezne elemente tega seznama:

```
seznam=range(2,21,2)
a=list(seznam)
i = 0
while i < len(a):
    print(a[i])
    i = i + 1
```

9.3.2 Ustvarjanje seznamov z vgrajeno zanko for (List Comprehension)

List comprehension je sintaktična oblika v *Pythonu*, ki omogoča ustvarjanje novih seznamov na jedrnat način. Omogoča ustvarjanje seznamov z uporabo obstoječih iterabilnih objektov (kot so seznami, nizi, množice itd.) s pomočjo enovrstičnega izraza. Ta metoda je pogosto bolj jedrnata in hitrejša od tradicionalne zanke `for`. Splošna oblika je:

```
[izraz for element in seznam if condition]
```

Primer 9-11:

Ustvarjanje seznama kvadratov števil od 0 do 9:

```
kvadrati = [x*x for x in range(10)]
print(kvadrati)
```

Izpis:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

To nalogo bi s klasično zanko `for` rešili tako:

```
kvadrati=[None]*10
for x in range(10):
    kvadrati[x]=x*x
print(kvadrati)
```

in z zanko `while` tako:

```
kvadrati=[None]*10
x=0
while x < 10:
    kvadrati[x]=x*x
    x=x+1
print(kvadrati)
```

V prvi vrstici smo priglasili prazen seznam z imenom `kvadrati` in dimenzijo 10, s čimer programu povemo, da bo imel seznam `kvadrati` 10 elementov.

Če želimo elemente seznama `kvadrati` izpisati vsakega v svojo vrstico, prestavimo stavek `print` v telo zanke in navedemo indeks elementa, ki naj se izpiše v vsaki iteraciji; ta je enak števcu iteracij:

```
kvadrati=[None]*10
for x in range(10):
    kvadrati[x]=x*x
    print(kvadrati[x])
```

Primer 9-12:

Tvorimo seznam sodih števil od 0 do 9 z imenom `soda`:

```
soda = [x for x in range(10) if x%2==0]
print(soda)
```

Izpis:

```
[0, 2, 4, 6, 8]
```

Ta seznam bi s klasično zanko `for` ustvarili tako:

```
soda=[]
for i in range(0,5):
   x=2*i
   soda.append(x)
print(soda)
```

ali tako:

```
soda=[]
for i in range(0,10,2):
   soda.append(i)
print(soda)
```

ali brez zanke:

```
soda=list(range(0,10,2))
print(soda)
```

V obeh primerih z zanko `for` smo v prvi vrstici priglasili prazen seznam z ukazom `soda=[]`. Nato pa smo ob vsakem prehodu skozi zanko v seznam dodali po en element z ukazom `append`.

Primer 9-13:

Seznam `x` vsebuje tri elemente, `y` pa bomo ustvarili kot nov seznam, katerega elementi so dvakratniki elementov seznama `x`.

a) Brez zanke. Ker bomo elemente seznama `y` določili z računanjem vsakega elementa posebej, moramo seznam `y` najprej inicializirati kot prazen seznam z ustrezno dimenzijo:

```
x=[6, 3, 8]
y=[None]*3
y[0]=2*x[0]
y[1]=2*x[1]
y[2]=2*x[2]
print(x)
print(y)
```

Izpis:

```
[6, 3, 8]
[12, 6, 16]
```

Ta način je zamuden, saj moramo vsak element seznama izračunati s svojo enačbo v svoji vrstici, zato je bolje uporabiti zanke.

b) Klasična zanka `for`:

```
x=[6, 3, 8]
y=[None]*3
for i in range(3):
    y[i]=2*x[i]
print(x)
print(y)
```

c) Vgrajena zanka `for` (ne zahteva inicializacije seznama `y`; vsi elementi tega seznama se tvorijo v eni vrstici neposredno v seznam):

```
x=[6, 3, 8]
y=[2*stevilo for stevilo in x]
print(x)
print(y)
```

Primer 9-14:

Iz datoteke preberimo pet števil in jih pretvorimo v seznam petih numeričnih vrednosti ter izračunajmo njihovo vsoto. Primerjajte s primerom 8-7. Uporabimo ukaz `float`, saj program števila iz datoteke prebere kot znakovne spremenljivke.

Datoteka PODAT2.DAT:

```
25
30
10
40
15
```

Program v *Pythonu*:

```
fp=open('PODAT2.DAT', 'r')
a=[float(i) for i in fp]
```

```
total = sum(a)
print(a)
print(total)
```

Po zagonu se izpiše:

```
[25.0, 30.0, 10.0, 40.0, 15.0]
```

Vrstica »`a=[float(i) for i in fp]`« vsebuje vgrajeno zanko `for`, s čimer smo iz vrednosti, prebranih iz datoteke, direktno ustvarili seznam `a`; ukaz `readlines` ni potreben.

9.4 Uporaba seznamov v kombinaciji z zanko for

Števec zanke `for` lahko po seznamu prehaja (iterira) na dva načina:

1. Števec iterira neposredno po elementih seznama in v posamezni iteraciji prevzame vrednost elementa seznama, ne indeksa;
2. števec iterira po indeksih seznama in v vsaki iteraciji prevzame vrednosti indeksa.

Prvi način uporabimo, ko želimo neposredno iterirati po elementih seznama in ne potrebujemo dostopa do njihovih indeksov. Drugi način uporabimo, ko želimo iterirati po indeksih elementov seznama in nato dostopati do elementov seznama z uporabo teh indeksov.

Primer 9-15:

V tem primeru števec `i` iterira neposredno po elementih seznama `vrednosti`.

```
vrednosti=[15,25,35,45]
for i in vrednosti:
    print(i)
```

Izpis:

```
15
25
35
45
```

V posamezni iteraciji je števec `i` prevzel vrednost elementa seznama in ne indeksa:

Prvi prehod zanke:	`i`	je 15.
Drugi prehod zanke:	`i`	je 25.
Tretji prehod zanke:	`i`	je 35.
Četrti prehod zanke:	`i`	je 45.

Primer 9-16:

Tudi v tem primeru v zanki `for` iteriramo neposredno po elementih seznama:

```
alkani=["metan","etan","propan","butan"]
for i in alkani:
    print(i)
```

V posamezni iteraciji je števec `i` prevzel vrednost elementa seznama:

Prvi prehod zanke: `i` je "metan".
Drugi prehod zanke: `i` je "etan".
Tretji prehod zanke: `i` je "propan".
Četrti prehod zanke: `i` je "butan".

Primer 9-17:

V tem primeru želimo v zanki `for` iterirati po indeksih seznama in nato dostopati do elementov seznama z uporabo teh indeksov:

```
alkani=["metan","etan","propan","butan"]
for i in range(len(alkani)):
    print(alkani[i])
print(alkani[1])
```

Izpis:

```
metan
etan
propan
butan
etan
```

V posamezni iteraciji je števec `i` prevzel vrednost indeksa:

Prvi prehod zanke: `i` je 0; `alkani[0]` = "metan".
Drugi prehod zanke: `i` je 1; `alkani[1]` = "etan".
Tretji prehod zanke: `i` je 2; `alkani[2]` = "propan".
Četrti prehod zanke: `i` je 3; `alkani[3]` = "butan".

Opomba: Ukaz »`print(alkani[1])`« ni del zanke!

Primer 9-18:

V primeru 8-7 v 8. poglavju smo iz datoteke PODAT2.DAT prebrali vsako število posebej ob prehodu skozi zanko in iz njih nismo tvorili seznama. Lahko pa preberemo tudi vsa števila naenkrat in jih pretvorimo v seznam s klasično zanko `for`, tako da ob vsakem prehodu skozi zanko dodamo po en element. V drugem delu programa seštejemo elemente seznama.

```
fp=open('PODAT2.DAT', 'r')
x=fp.readlines() #preberemo vse vrstice
a=[]             #definiramo prazen seznam

for i in x:
    a.append(float(i)) #dodajamo elemente
print(a)

total = 0
for i in range(5): #izračunamo vsoto števil
    total = total + a[i]
print(total)
```

Izpis:

```
[25.0, 30.0, 10.0, 40.0, 15.0]
120.0
```

V prvem delu programa lahko za tvorjenje seznama uporabimo tudi vgrajeno zanko `for`.

```
fp=open('PODAT2.DAT', 'r')
a=fp.readlines()
a=[float(i) for i in a]
print(a)
```

Tudi drugi del programa lahko zapišemo na več načinov, npr.:

```
total = 0
for i in a:
    total = total + i
print(total)
```

Ker je pet števil definiranih kot seznam `a`, lahko uporabimo tudi funkcijo `sum`, ki sešteje elemente seznama. Tako za seštevanje ne potrebujemo zanke `for`.

```
total = sum(a)
print(total)
```

Primer 9-19:

V laboratoriju so tri posode s tekočino. V prvi posodi je 2 L vode (ρ = 1 kg/L); v drugi so 3 L etanola (ρ = 0,789 kg/L); v tretji posodi je 1 L glicerola (ρ = 1,26 kg/L). Napišite program, ki bo računal maso in množino snovi v vsaki posodi. Pri tem uporabite numerični vnos in sezname za vhodne podatke ter rezultate.

```
tekocine=["voda","etanol","glicerol"]
volumen=[2, 3, 1]
gostota=[1.000,0.789,1.260]
mmasa=[18,46,92.1]
masa=[0,0,0]        #inicializacija seznama
mnozina=[0,0,0]     #inicializacija seznama

for k in range(len(tekocine)):
    masa[k]=volumen[k]*gostota[k]
    mnozina[k]=masa[k]/mmasa[k]*1000
    print(tekocine[k], masa[k], mnozina[k])
```

Če želimo lepši izpis, nadomestimo `print` stavek z naslednjim:

```
print(f"{tekocine[k]:<8}{masa[k]:>8.3f}{mnozina[k]:>10.3f}")
```

V tem primeru bo izpis:

```
voda       2.000   111.111
etanol     2.367    51.457
glicerol   1.260    13.681
```

Pojasnimo izpis na primeru prve vrstice: element seznama `voda` zaseda 4 mesta levo poravnano, zato za tem sledijo 4 prazna mesta; `2.000` zaseda 5 mest desno poravnano, zato so pred tem še 3 prazna mesta; `111.111` zaseda 7 mest desno poravnano, zato so pred tem še 3 prazna mesta.

9.5 Problemi 9. poglavja

Problem 9-1:

Kaj se izpiše kot rezultat programa?

```
spojina=["voda","etanol","metanol"]
ax=[100, 80, 78]
by=[1000, 800, 780]

for i in range (len(spojina)):
    a=ax[i]*by[i]
    print(spojina[i],a)
```

Problem 9-2:

Kaj se izpiše kot rezultat programa?

```
spojina=["voda","etanol","metanol"]
ax=[100, 80, 78]
by=[1000, 800, 780]
a=[]

for i in range (len(ax)):
    a.append(ax[i]*by[i])
print(spojina,a)
```

Problem 9-3:

Kaj se izpiše kot rezultat programa?

```
podatki=[("voda",2,3,4),("etanol",5,6,7),("metanol",1,5,8)]
R=8.314
nvse=[]

for(ime,x,y,z) in podatki:
    n=x*y*R/z
    print(ime,n)
    nvse.append(n)
print(nvse)
```

Problem 9-4:

Preuredite Primer 9-19 treh kemikalij tako, da bo branje podatkov potekalo iz datoteke Trikemikalije.dat in da v programu ne bomo uporabili seznamov.

Problem 9-5:

Kaj se izpiše kot rezultat naslednjega programa?

```
X=[11.1,22.2,33.3,44.4]
total=0
for i in range (len(X)):
     total=total + X[i]
print(total)
```

Krajši zapis:

```
X=[11.1,22.2,33.3,44.4]
print(sum(X))
```

Problem 9-6:

Kaj se izpiše kot rezultat naslednjega programa?

```
num=[None]*7
for i in range(1,7):
     num[i]=i*i
     print(i,num[i])
```

Krajši zapis:

```
for i in range(1,7):
     print(i,i*i)
```

Problem 9-7:

Kaj se izpiše kot rezultat naslednjega programa?

```
N1=[50,60,70]
N2=[70,80,90]
N3=[None]*len(N1)
for i in range(len(N1)):
     N3[i]=(N1[i]+N2[i])/2
     print(i+1,N1[i],N2[i],int(N3[i]))
```

S formatiranim print stavkom:

```
print(f"{(i+1):<5}{N1[i]:<5}{N2[i]:<5}{int(N3[i]):<5}")
```

Problem 9-8:

Kakšne so razlike pri izpisih naslednjih programov?

Podatkovna datoteka Podatki.dat (pazite, da ne bo odvečnih praznih vrstic):

```
10 20 30
40 50 60
70 80 90
```

Varianta 1:

```
file=open('Podatki.dat', 'r')
vsebina=file.readlines()
file.close()
for line in vsebina:
    print(line)
```

Varianta 2:

```
vrstice=[]
file=open('Podatki.dat', 'r')
for line in file:
    x=line.split()
    y=[float(i) for i in x]
    vrstice.append(y)
file.close()
print(vrstice)
```

Opomba: z ukazom `for line in file` program bere po eno vrstico naenkrat, zato ukaz `read` ali `readline` ni potreben.

Kaj se izpiše, če programu variante 2 dodamo še naslednje:

```
for line in vrstice:
    a=sum(line)
    print(a,line)
```

Problem 9-9:

Izračunajmo povprečja istoležnih elementov seznamov N1, N2 in N3.

Seznam	Indeks		
	0	1	2
N1	10	20	30
N2	60	70	80
N3	90	100	110

Podatki naj bodo shranjeni v datoteki Podatki.txt:

```
10 20 30
60 70 80
90 100 110
```

10 Funkcije

Funkcija je zbirka določenih ukazov, ki izvajajo določeno nalogo oz. zaporedje ukazov. Uporabimo jo takrat, ko se neko zaporedje ukazov v programu izvede večkrat. Na ta način istega zaporedja ukazov ni treba pisati znova, ampak samo pokličemo želeno funkcijo. Funkciji dodelimo ime in jo v glavnem programu kličemo z njenim imenom. Definirana mora biti, preden jo kličemo.

10.1 Splošna oblika funkcije

V programu, v katerega je vključena funkcija (ali več funkcij), običajno govorimo o več delih: en del imenujemo glavni program, drugi del je funkcija (ali več funkcij). Ukazi v glavnem programu se izvajajo eden za drugim, medtem ko se funkcija izvede takrat, ko je poklicana v glavnem programu. Takrat se izvajanje iz glavnega programa prestavi v funkcijo, tako da se izvedejo ukazi v telesu funkcije. Nato se izvajanje vrne nazaj v glavni program in nadaljuje s prvo vrstico za klicem funkcije.

Splošna oblika funkcije je naslednja:

```
def ime_funkcije(seznam vhodnih spremenljivk):
    ukaz1
    ukaz2
    return seznam izhodnih spremenljivk
```

V glavnem programu kličemo funkcijo tako:

```
seznam izhodnih spremenljivk = ime_funkcije(seznam vhodnih
spremenljivk)
```

V seznamih vhodnih in izhodnih spremenljivk ločimo spremenljivke z vejico. Ukazi, ki tvorijo telo funkcije, morajo biti zamaknjeni. Vrstica, ki ni zamaknjena, ni del funkcije. Ukaz `return` v funkciji ni obvezen, če v funkciji izvedemo določene ukaze, potem pa neodvisno od njih nadaljujemo s programom. Ukaz `return` je potreben, ko hočemo prenesti izračunane vrednosti iz funkcije v glavni program in jih tam tudi uporabiti. Z vrstico `return` se funkcija konča; po njej niso več možni ukazi, ki bi bili del funkcije.

Funkcija ni nujno del istega programa, v katerem pišemo kodo in v katerem potrebujemo izračune te funkcije. Lahko je zapisana v ločeni datoteki *Python*; v tem primeru jo uvozimo v *file*, v katerem jo potrebujemo, z ukazom:

```
from ime_datoteke import ime_funkcije
```

Na ta način lahko ustvarimo svojo knjižnico funkcij za izračune, ki jih pogosto uporabljamo. Poleg tega je na voljo veliko že pripravljenih knjižnic. Standardne knjižnice so ob namestitvi programa že vgrajene in so na lokaciji programskega okolja *Python* v mapi *Lib*; dostopne so neposredno po zagonu. Knjižnice, ki niso del osnovne namestitve, najdemo na spletu. V *Python* jih instaliramo z ukazom *pip*, kot je prikazano v poglavju 11 Knjižnice. Za celoten pregled knjižnic uporabite uradno dokumentacijo *Python* (na spletu poiščite Python Module Index).

10.2 Uporaba funkcij

Primer 10-1:

Vrstni red ukazov, ko je vključena funkcija.

```
def pozdrav(ime):
   print("Lep pozdrav, " + ime)

print("Zacetek")
pozdrav("Mojca")
print("Konec")
```

Izpis:

```
Zacetek
Lep pozdrav, Mojca
Konec
```

Primer 10-2:

Pomen uporabe funkcije.

```
def pozdrav(ime):
   print("Lep pozdrav, ", ime)

pozdrav("Marko")
pozdrav("Vesna")
pozdrav("Ina")
```

Izpis:

```
Lep pozdrav, Marko
Lep pozdrav, Vesna
Lep pozdrav, Ina
```

Primer 10-3:

Funkcija z več vhodnimi spremenljivkami.

```
def pozdrav(ime, leta):
   print("Lep pozdrav, " + ime + ", star/a " + str(leta))

pozdrav("Marko", 21)
pozdrav("Vesna", 25)
```

Izpis:

```
Lep pozdrav, Marko, star/a 21
Lep pozdrav, Vesna, star/a 25
```

Primer 10-4:

Funkcija z ukazom `return` za izračun kuba števila.

```
def kub(num):
   return num * num * num
```

```
print(kub(3))
print(kub(4))
print(kub(5))
```

Izpis:

```
27
64
125
```

Primer 10-5:

Funkcija z več izhodnimi spremenljivkami in ukazom `return`. Pretvorimo 67 stopinj Fahrenheita v stopinje Celzija in kelvine.

```
def pretvorba(a):
    b = 5 / 9 * (a - 32)
    c = b + 273.15
    return b, c

tempF = 67
tempC, tempK = pretvorba(tempF)
print(tempF, tempC, tempK)
```

Izpis:

```
67 19.444444444444446 292.59444444444443
```

Primer 10-6:

Pretvorimo stopinje Fahrenheita med 0 in 212 s korakom po 2 v stopinje Celzija in kelvine ter za to uporabimo funkcijo.

```
def pretvorba(a):
    b = 5 / 9 * (a - 32)
    c = b + 273.15
    return b, c

tempF = 0
while tempF <= 212:
    tempC, tempK = pretvorba(tempF)
    print(f"{tempF:.0f}" + ' ' * 3 + f"{tempC:.2f}" + ' ' * 3 +
f"{tempK:.2f}")
    tempF = tempF + 2
```

Izpis:

```
0   -17.78   255.37
2   -16.67   256.48
4   -15.56   257.59
6   -14.44   258.71
8   -13.33   259.82
10   -12.22   260.93
12   -11.11   262.04
14   -10.00   263.15
16   -8.89   264.26
itd.
```

10.3 Problemi 10. poglavja

Problem 10-1:

Napišimo program, ki bo izračunal parne tlake za tri spojine pri 30 °C z uporabo Antoinove enačbe:

$$p = \exp\left(a - \frac{b}{T+c}\right)$$

kjer je *p* parni tlak (mmHg); *T* temperatura (K) in *a*, *b*, *c* konstante, značilne za vsako spojino.

Za izračun parnega tlaka naj bo uporabljena funkcija; kot rezultat naj se izpiše ime spojine in njen parni tlak v bar.

Podatki so naslednji:

Spojina	*a*	*b*	*c*
Etanol	18.5242	3578.91	-50.5
Voda	18.3036	3816.44	-46.13
Metanol	18.5875	3626.55	-34.29

Program zapišite z:

a) aritmetičnim vnosom podatkov;

b) branjem podatkov iz datoteke.

Problem 10-2:

Za trojice števil, ki so podane v datoteki, naj se izračuna povprečje z uporabo funkcije. Trojice števil so: 100, 200, 300 in 6, 8, 9, ter 13, 15, 17.

Problem 10-3:

Zapišimo program za potenciranje, ki bo izračunal potenco nekega števila, ne da bi uporabili vgrajene matematične funkcije, ampak tako, da število množimo samim s seboj tolikokrat, kolikor znaša definiran eksponent, npr. x^3 = *x***x***x*. Število in celi eksponent definiramo v glavnem programu, za potenciranje pa zapišimo lastno funkcijo.

Preuredimo program tako, da bo uporabnik vstavil osnovo in eksponent interaktivno. Funkcija `potenciranje` ostane enaka kot prej, spremeni se preostali del programa.

Problem 10-4:

Sprogramirajte funkcijo, ki bo v tekstu, ki ga vstavi uporabnik, zamenjala vse samoglasnike v črko g, npr. pes v pgs; korak v kgrgk.

11 Knjižnice

Knjižnice v *Pythonu* predstavljajo ključni element povečanja učinkovitosti in produktivnosti programiranja. Z uporabo knjižnic lahko razvijalci enostavno dostopajo do obsežnih zbirk predpripravljenih funkcij in orodij, kar jim omogoča hitrejše reševanje kompleksnih problemov. Knjižnice v *Pythonu* pokrivajo širok spekter področij, od znanstvenega računanja in obdelave podatkov do razvoja spletnih aplikacij ter umetne inteligence. Zaradi bogatega nabora razpoložljivih knjižnic je *Python* postal eden najpriljubljenejših programskih jezikov med razvijalci po svetu. Najpriljubljenejše knjižnice so npr. *NumPy, Matplotlib, Pandas, SciPy, scikit-learn* idr.

11.1 Namestitev knjižnic

Namestitev knjižnic se v vsakem okolju nekoliko razlikuje. V tem učbeniku podajamo prikaz namestitve knjižnic za okolji IDLE in *Pycharm*.

11.1.1 IDLE

V oknu *IDLE Shell* ugotovimo lokacijo, kjer je instaliran *Python*:

```
>>> import sys
>>> sys.executable
```

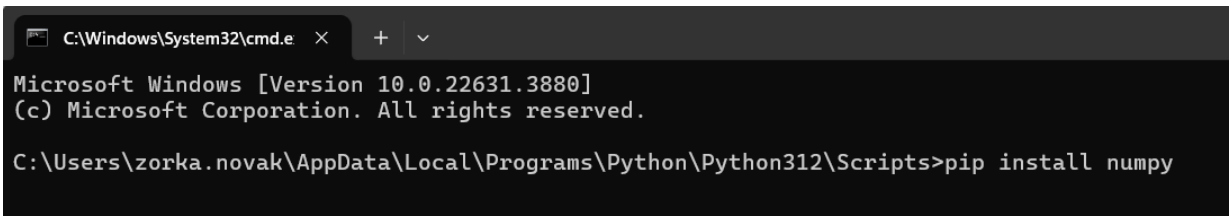
Izpiše se pot: npr. 'C:\\Users\ime.priimek\\AppData\\Local\Progams\\Python\\Python312

Odpremo *File Explorer* in poiščemo to pot. Če katere mape na poti ne vidimo, v meniju *View* poiščemo opcijo *Show* in omogočimo *Hidden items.*

Ko pridemo na lokacijo programa (v zgornjem primeru *Python312*), izberemo mapo *Scripts*.

V zgornji vrstici kliknemo na lokacijo, da se obarva cela pot in prek tega napišemo ukaz cmd.

Odpre se komandno okno na lokaciji mape *Scripts*, v katerem zapišemo ukaz: `pip install numpy` (slika 11-1). Podobno namestimo druge knjižnice, ki jih zatem lahko uvozimo in uporabimo v *Python* kodah.



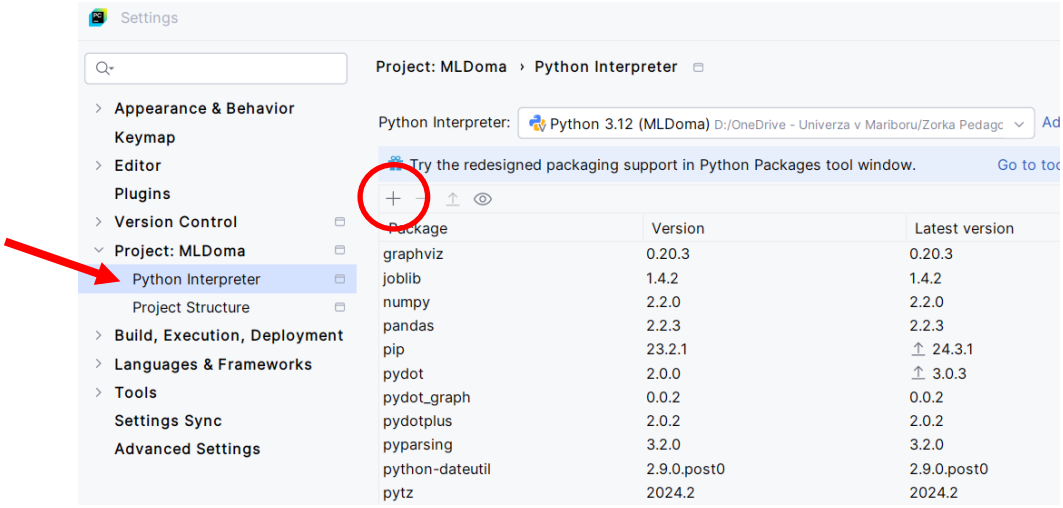
Slika 11-1: Namestitev knjižnic v okolju IDLE

11.1.2 PyCharm

V okolju *PyCharm* uporabimo ukaze:

File > *Settings* > ime projekta > *Python Interpreter*.

Stisnemo znak + in v iskalniku poiščemo knjižnice, ki jih potrebujemo, npr. *numpy, joblib* in druge (slika 11-2). Knjižnico označimo in stisnemo gumb *Install Package*.



Slika 11-2: Iskanje in namestitev knjižnic v *PyCharm*

11.2 Knjižnica Numpy

Knjižnica *numpy* omogoča številne operacije z večdimenzionalnimi seznami, matrikami; statistične izračune; generator naključnih števil ipd. Je zelo uporabna na področju strojnega učenja (angl. *Machine Learning*).

Primer 11-1:

Definiranje seznama *numpy* .

```
import numpy as np
a=np.arange(10)
print(a)
```

Izpis:

```
[0 1 2 3 4 5 6 7 8 9]
```

Opomba: za definiranje seznama bi lahko uporabili običajno *Pythonovo* sintakso, tj.:

```
a=[0 1 2 3 4 5 6 7 8 9],
```

vendar je ob uporabi knjižnice *numpy* bolj priporočljiva uporaba seznama *numpy*, saj ima večjo hitrost pri izračunih; boljšo učinkovitost pri porabi pomnilnika; omogoča več operacij po seznamih brez uporabe zank ipd. Knjižnica *numpy* je optimizirana za matematične operacije seznamov in za delo z večdimenzionalnimi podatki. Podpira vektorizirane operacije, kar omogoča hitre izračune.

Primer 11-2:

Primer 9-13 lahko rešimo elegantneje s knjižnico *numpy* brez uporabe zanke.

```
import numpy as np
x=np.array([6, 3, 8])
y=2*x
print(x)
print(y)
```

Izpis:

```
[6 3 8]
[12  6 16]
```

Primer 11-3:

Izračun ničel polinoma $x^3 + 2x^2 - 5x - 6$.

```
import numpy as np
coefficients=[1, 2, -5, -6]
nicle = np.roots(coefficients)
print(nicle)
```

Izpis:

```
[ 2. -3. -1.]
```

Primer 11-4:

Statistične funkcije.

```
import numpy as np

# create an array
array1 = np.array([1, 3, 5, 7, 9, 11, 13, 15, 17, 19])

# compute mean
mean = np.mean(array1)
print("Mean:",mean)

# compute standard deviation
std_dev = np.std(array1)
print("Standard deviation:",f"{std_dev:.4f}")

# compute the 25th percentile of the array
result1 = np.percentile(array1, 25)
print("25th percentile:",result1)


# compute the 75th percentile of the array
result2 = np.percentile(array1, 75)
print("75th percentile:",result2)
```

Izpis:

```
Mean: 10.0
Standard deviation: 5.7446
25th percentile: 5.5
75th percentile: 14.5
```

11.3 Knjižnica Chemics

Knjižnica *chemics* vsebuje funkcije za reševanje kemijskih problemov, kot so lastnosti plinov; lastnosti plinskih zmesi; lastnosti reaktantov in produktov kemijske reakcije; pretvarjanje masnih deležev v množinske; sestava biomase itd. (https://chemics.readthedocs.io/en/latest/installation.html)

Namestimo jo v mapo *scripts* z ukazom: `pip install chemics`.

Primer 11-5:

Pretvorba masnih deležev v množinske za 5 komponent.

```
import chemics as cm
y = [0.36, 0.16, 0.20, 0.28] #masni delezi
mw = [12.011, 1.008, 15.999, 14.007] #molske mase
mn_del= cm.massfrac_to_molefrac(y, mw) #pretvorba
print(mn_del) #mnozinski delezi
```

Izpis:

```
[0.13550366 0.7176078  0.05651515 0.0903734 ]
```

Primer 11-6:

Gostota dušika pri 500 °C in 1 atm.

```
import chemics as cm
gas = cm.Gas("N2", 773)
rho = gas.density()
mu = gas.viscosity()
print("Nitrogen gas properties at 773 K and 101,325 Pa")
print(f"density {rho:.4f} kg/m3")
print(f"viscosity {mu:.2f} P")
```

Izpis:

```
Nitrogen gas properties at 773 K and 101,325 Pa
density 0.4416 kg/m3
viscosity 363.82 P
```

Primer 11-7:

Lastnosti reaktantov in produktov kemijske reakcije

```
import chemics as cm
ce = cm.ChemicalEquation('2 HCl + 2 Na -> 2 NaCl + H2')
print(ce.is_balanced())    # True, ce je reakcija urejena
print(ce.rct_properties)   # vrne lastnosti reaktantov
print(ce.prod_properties) # vrne lastnosti produktov
```

Izpis:

```
True
                HCl         Na
moles           2.0        2.0
species         HCl         Na
molwt        36.458      22.99
mass         72.916      45.98
molfrac         0.5        0.5
massfrac   0.613275   0.386725

                NaCl         H2
moles            2.0        1.0
species         NaCl         H2
molwt          58.44      2.016
mass          116.88      2.016
molfrac     0.666667   0.333333
massfrac    0.983044   0.016956
```

Primer 11-8:

Sestava biomase na osnovi ultimativne analize: masni delež ogljika 0,534; masni delež vodika 0,06.

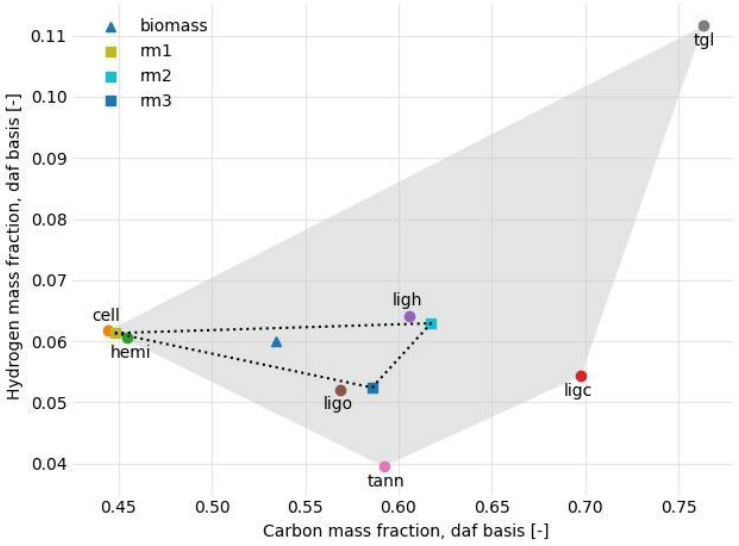
```
import chemics as cm
import matplotlib.pyplot as plt
yc=0.534
yh=0.06
bc = cm.biocomp(yc, yh, printcomp=True)

fig, ax = plt.subplots(tight_layout=True)
cm.plot_biocomp(ax, yc, yh, bc['y_rm1'], bc['y_rm2'],
bc['y_rm3'])

plt.show()
```

Izpis:

```
basis    cell    hemi   ligc    ligh    ligo   tann    tgl
x_daf    0.4118  0.2745  0.0627  0.1529  0.0981  0.0000  0.0000
x_wet    0.4118  0.2745  0.0627  0.1529  0.0981  0.0000  0.0000
y_daf    0.2936  0.1595  0.0713  0.2934  0.1822  0.0000  0.0000
y_wet    0.2936  0.1595  0.0713  0.2934  0.1822  0.0000  0.0000
y_wetash 0.2936  0.1595  0.0713  0.2934  0.1822  0.0000  0.0000
```



12 Risanje grafov

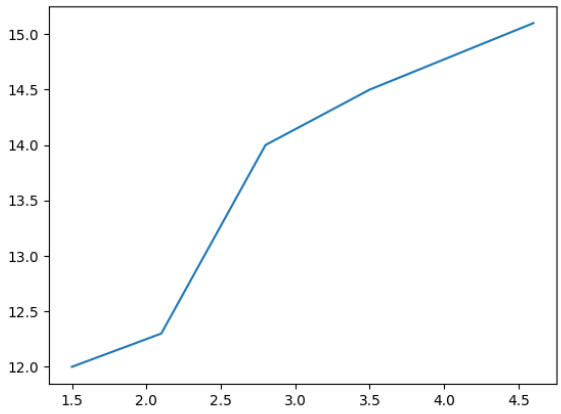
Za risanje grafov uporabimo knjižnico *matplotlib*, ki jo uvozimo v program v komandnem oknu na lokaciji mape *Scripts* z ukazom: `pip install matplotlib`, kot je podrobneje opisano v poglavju 11.1. Za risanje grafov uporabimo ukaz `plt.plot`.

Primer 12-1:

Podane so točke (x,y). Narišimo graf teh točk.

```
import matplotlib.pyplot as plt
x=[1.5, 2.1, 2.8, 3.5, 4.6]
y=[12.0, 12.3, 14.0, 14.5, 15.1]
plt.plot(x,y)
plt.show()
```

Po zagonu se izriše:

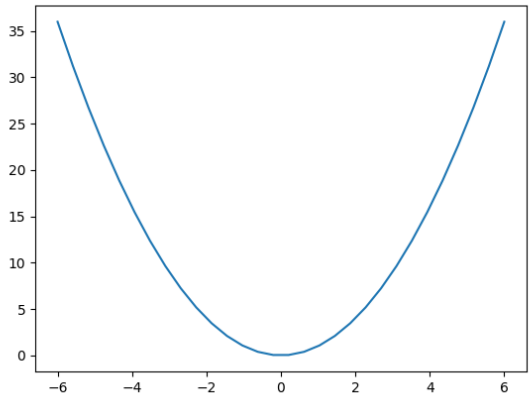


Primer 12-2:

Narišimo funkcijo x^2 med vrednostma –6 in 6. Za generiranje vrednosti neodvisne spremenljivke *x* med –6 in 6 uporabimo knjižnico `numpy` in funkcijo `linspace`, ki ustvari zahtevano število enakomerno razporejenih točk med spodnjo in zgornjo mejo. Za izris grafa v posebnem oknu je uporabljen ukaz `plt.show()`.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-6,6,30)
y = x**2
plt.plot(x, y)
plt.show()
```

Po zagonu se izriše:



Graf lahko opremimo še z naslovom grafa in naslovoma osi ter dodatkom mrežnih črt:

```
plt.title("Graf kvadratne funkcije")
plt.xlabel("x-os")
plt.ylabel("y-os")
plt.grid(True)
```

Te ukaze zapišemo pred ukazom `plt.show()`. V oklepaju lahko pri tekstih dodamo velikost črk, npr. `fontsize=14`.

Nastavimo lahko tudi barvo in obliko krivulje, debelino črt, obliko markerjev ipd. Za barve se praviloma uporabi prva črka, npr. b-modra; g-zelena; r-rdeča; m-magenta; y-rumena; w-bela; k-črna.

Za vrsto črte se uporabijo znaki: - polna črta; -- črtkana črta; -. črta-pika; : pikčasta črta. Za točke na grafu (markerje) se uporabijo: . pika; , vejica; o krog; v trikotnik, obrnjen navzdol; ^ trikotnik, obrnjen navzgor; s kvadrat; * zvezdica; + plus; D romb itd.

Za te nastavitve razširimo ukaz `plt.plot`:

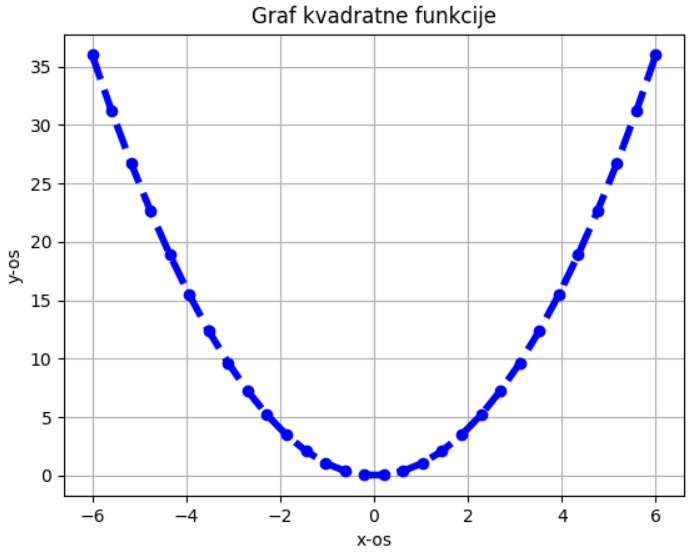
```
plt.plot(x, y, 'r--', linewidth=4)
plt.plot(x, y, 'bo')
```

V prvem primeru bo krivulja črtkana črta rdeče barve in nekoliko debelejša zaradi nastavljene debeline črte. V drugem primeru bodo namesto črte modri krogi.

Tako izpolnjen program se glasi:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-6,6,30)
y = x**2
plt.plot(x, y, 'b--o', linewidth=4)
plt.title("Graf kvadratne funkcije")
plt.xlabel("x-os")
plt.ylabel("y-os")
plt.grid(True)
plt.show()
```

Po zagonu se izriše graf:



Različne barve črte in markerjev določimo z ukazom:

```
plt.plot(x, y, color='r', linestyle='--', marker='o',
         markerfacecolor='b', markeredgecolor='b')
```

S tem ukazom bo krivulja rdeča črtkana črta, markerji pa modri krogci.

13 Podatkovna znanost v Pythonu

Podatkovna znanost postaja nepogrešljivo orodje v sodobni kemiji in kemijskem inženirstvu, saj korenito spreminja načine analiziranja, obdelave ter razumevanja podatkov. Tradicionalne metode pogosto ne zadoščajo več za obvladovanje kompleksnosti in obsežnosti podatkov, ki jih generirajo sodobne kemijske raziskave ter industrijski procesi. Z uporabo naprednih algoritmov strojnega učenja in umetne inteligence lahko sedaj:

Napovedujemo vedenje sistemov. Namesto dolgotrajnih in dragih eksperimentov lahko z modeliranjem in s simulacijami, ki temeljijo na podatkih, predvidimo lastnosti novih materialov, potek kemijskih reakcij in obnašanje kompleksnih sistemov. To omogoča hitrejši razvoj novih materialov, katalizatorjev in farmacevtskih učinkovin.

Optimiramo procese. V kemijski industriji je optimizacija ključnega pomena za učinkovitost, zmanjšanje stroškov in varovanje okolja. Podatkovna znanost omogoča optimizacijo proizvodnih procesov, nadzor kakovosti, napovedovanje napak ter vzdrževanje opreme.

Pridobivamo globlji vpogled v kompleksne kemijske pojave. S pomočjo vizualizacije podatkov in naprednih statističnih metod lahko odkrivamo skrite vzorce ter povezave v kompleksnih kemijskih podatkih. To omogoča boljše razumevanje temeljnih kemijskih mehanizmov in razvoj novih teorij.

Avtomatiziramo eksperimente. Podatkovna znanost omogoča razvoj avtomatiziranih eksperimentalnih platform, ki združujejo robote, računalnike in specializirano programsko opremo za samodejno izvajanje eksperimentov. S tem hitreje in učinkoviteje generiramo rezultate, kar omogoča preizkušanje večjega števila hipotez.

Podatkovna znanost temelji na več zaporednih korakih, ki omogočajo razvoj napovednih modelov za:

- *Vnos podatkov*. Zbiranje podatkov iz različnih virov, npr. *Excela*, CSV-datoteke, senzorjev.
- *Čiščenje podatkov*. Odstranjevanje napak, nepopolnosti in skritih nepravilnosti v podatkih.
- *Razdelitev na učno in testno množico*. Podatke razdelimo na dve skupini: eno za učenje modela (trening), drugo za oceno delovanja modela (test).
- *Kreiranje modela*. Uporabimo matematične in statistične metode za oblikovanje napovednega modela.
- *Učenje modela*. Model se uči iz podatkov z uporabo algoritmov strojnega učenja.
- *Napovedovanje*. Model uporablja naučena pravila za napovedovanje novih rezultatov.
- *Evalvacijo in izboljšave*. Izvede se analiza točnosti modela ter možnosti za izboljšanje z optimizacijo parametrov.

V okolju *Python* za izvedbo podatkovnih analiz in razvoja modelov pogosto uporabljamo naslednje knjižnice:

- *Numpy* – za matematične operacije in delo s tabelami števil.
- *Pandas* – za učinkovito obdelavo in analizo podatkov v tabelarni obliki.
- *Matplotlib* – za vizualizacijo podatkov in rezultatov.
- *Scikit-Learn* – knjižnica, namenjena strojnemu učenju, ki omogoča enostavno kreiranje, učenje in evalvacijo modelov.

13.1 Primer – napovedovanje agregatnega stanja

Primer prikazuje izgradnjo modela, ki ga naučimo, da na osnovi vrednosti molske mase in temperature napoveduje agregatno stanje spojine. Tabela z majhno bazo podatkov je v prilogi 14.3. Vsebuje 25 podatkov za tri atribute: molsko maso (mmasa); temperaturo (temp) in agregatno stanje (agrst). Podatke prenesemo v *Excel* in shranimo z imenom `spojine.csv`.

Za delo si ustvarimo projekt v okolju *PyCharm*. Za namestitev knjižnic uporabimo ukaze:

File > *Settings* > ime projekta > *Python Interpreter*.

Stisnemo znak + in v iskalniku poiščemo knjižnice, ki jih potrebujemo, ter jih namestimo; to so `pandas`, `scikit-learn` in `joblib`. Tudi datoteka `spojine.csv` mora biti na lokaciji ustvarjenega projekta, tako kot bodo kode *Python*, ki jih bomo ustvarili v nadaljevanju.

13.1.1 Gradnja in učenje napovednega modela

V tem podpoglavju bomo zgradili preprost model za napovedovanje agregatnega stanja spojine na osnovi dveh vhodnih parametrov: molske mase spojine in temperature. Uporabimo algoritem odločitvenega drevesa (`DecisionTreeClassifier`) iz knjižnice `Scikit-Learn`.

Odločitveno drevo je intuitivna metoda strojnega učenja, ki na osnovi vhodnih podatkov gradi drevesno strukturo odločitev, s katero napoveduje različne izide. Uporaba preprostega modela, kot je odločitveno drevo, omogoča hitro analizo in napovedovanje z dobro preglednostjo rezultatov. Model je posebej primeren za začetnike, saj omogoča intuitivno razumevanje procesa učenja iz podatkov. V kemijskem inženirstvu lahko podobne pristope uporabimo za napovedovanje lastnosti materialov, optimizacijo procesov ali klasifikacijo spojin glede na njihove lastnosti.

Osnovni model v kodi *Python* je prikazan v nadaljevanju in sledi naslednjim korakom:

Priprava podatkov. Podatke preberemo iz datoteke `spojine.csv`, ki vsebuje stolpce za molsko maso (mmasa), temperaturo (temp) in agregatno stanje (agrst) (priloga 14.3).

Razdelitev podatkov. Vhodni podatki (mmasa in temp) predstavljajo neodvisne spremenljivke `X`. Generiramo jih s funkcijo `drop` tako, da iz vseh prebranih podatkov datoteke izločimo stolpec z opisom agregatnega stanja. Ta stolpec tvori odvisno spremenljivko `y`.

Gradnja modela in učenje. Z uporabo metode `DecisionTreeClassifier()` zgradimo model, ki se bo učil iz podanih podatkov. Za učenje uporabimo funkcijo `fit`, ki model nauči prepoznavati povezave med vhodnimi parametri in izhodom (agregatnim stanjem).

Napovedovanje. Na koncu izvedemo testno napoved s funkcijo `predict` za specifične vrednosti vhodnih parametrov, npr. za etanol molska masa = 46 g/mol, temperatura = -14 °C, in preverimo, kateri izhod, tj. agregatno stanje, napove model.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

# branje podatkov iz datoteke
spojine_data = pd.read_csv('spojine.csv')

# neodvisne spremenljivke
X = spojine_data.drop(columns=['agrst'])
```

```
# odvisna spremenljivka
y = spojine_data['agrst']

# DecisionTreeClassifier model
model = DecisionTreeClassifier()

# ucenje modela
model.fit(X, y)

# napoved
predictions = model.predict([[46, -14]])

# vhodni podatki za napoved (pretvorjeni v DataFrame)
#new_data = pd.DataFrame([[46, -14]], columns=['mmasa', 'temp'])
#predictions = model.predict(new_data)

print(predictions)
```

Agregatno stanje, ki ga napove model: `['tekoce']`

V nadaljevanju bomo ta model nadgradili z oceno točnosti in vizualizacijo modela, kar nam bo omogočilo boljšo analizo njegove učinkovitosti.

13.1.2 Ocena točnosti napovednega modela

V tem podpoglavju bomo osnovni model razširili z uvedbo delitve podatkov na učno in testno množico ter dodali oceno točnosti modela. Ocena točnosti omogoča, da preverimo, kako dobro deluje model pri napovedovanju rezultatov za nove podatke. Model v kodi *Python* je prikazan v nadaljevanju. Novi so naslednji koraki:

Delitev podatkov. Celoten nabor podatkov razdelimo v dve skupini s funkcijo `train_test_split` iz knjižnice `Scikit-Learn`:

- učno množico: npr. 80 % podatkov se uporabi za učenje modela;
- testno množico: npr. 20 % podatkov se uporabi za preverjanje točnosti napovedi.

Ocena točnosti. Za merjenje uspešnosti modela uporabimo funkcijo `accuracy_score`, ki izračuna, kako pogosto je model pravilno napovedal rezultate v testni množici, saj ta vsebuje podatke, ki jih model še ni videl med učenjem. Parameter *klasifikacijska točnost* je izražen kot razmerje med številom pravilnih napovedi in skupnim številom napovedi v testni množici. Zavzame lahko vrednosti med 0 in 1, kjer vrednost 1 pomeni popolno točnost.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

spojine_data = pd.read_csv('spojine.csv')
X = spojine_data.drop(columns=['agrst'])
y = spojine_data['agrst']

# 80 % podatkov za ucenje in 20 % za testiranje
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = DecisionTreeClassifier()
```

```
model.fit(X_train, y_train)
predictions = model.predict(X_test)

# napoved točnosti med 0 in 1, ob zagonih je lahko drug rezultat
score = accuracy_score(y_test, predictions)
print(score)
```

Vrednost parametra točnosti se lahko ob vsakem zagonu nekoliko razlikuje, saj model vsakič vzame drugo množico podatkov za učenje. Z zmanjševanjem deleža podatkov v učni množici se parameter točnosti zmanjšuje, torej slabša.

13.1.3 Uporaba naučenega modela

V tem podpoglavju bomo prikazali, kako shranimo že naučen model in ga pozneje uporabimo za napovedovanje brez ponovnega učenja. Shranjevanje modela je koristno, ko model potrebujemo večkrat ali ko proces treniranja zahteva veliko časa in računalniškega napora. Model shranimo v datoteko in pozneje preprosto naložimo za uporabo. Za shranjevanje modelov se v *Pythonu* pogosto uporablja knjižnica `joblib`, ki omogoča enostavno serializacijo objektov. V spodnjem programu je naučen model shranjen v datoteko `spojinetren.joblib`.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import joblib

spojine_data=pd.read_csv('spojine.csv')
X = spojine_data.drop(columns=['agrst'])
y = spojine_data['agrst']
model = DecisionTreeClassifier()
model.fit(X, y)

#shranjevanje naucenega modela
joblib.dump(model, 'spojinetren.joblib')
```

Za napovedovanje izhodov za nove vhodne podatke lahko zdaj uporabimo že naučen model `spojinetren.joblib`. Naložimo ga z uporabo funkcije `joblib.load`.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import joblib

# uporabimo shranjen model
model = joblib.load('spojinetren.joblib')

predictions = model.predict([[46, -14]])

# vhodni podatki za napoved (pretvorjeni v DataFrame)
#new_data = pd.DataFrame([[46, -14]], columns=['mmasa', 'temp'])
#predictions = model.predict(new_data)

print(predictions)
```

13.1.4 Vizualizacija

Vizualizacija modela je ključna za razumevanje, kako algoritem sprejema odločitve na osnovi vhodnih podatkov. V tem podpoglavju bomo prikazali, kako lahko z uporabo knjižnice `Scikit-Learn` in funkcije `export_graphviz` vizualiziramo odločitveno drevo, ki smo ga ustvarili ter natrenirali v prejšnjih korakih.

S funkcijo `export_graphviz` izvozimo model v besedilno datoteko `tree_spojine.dot`. Ta datoteka vsebuje opis strukture drevesa, vključno z vozlišči, mejnimi pogoji za razvejanje in razvrstitev podatkov v končne izide, tj. agregatno stanje.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

spojine_data=pd.read_csv('spojine.csv')
X = spojine_data.drop(columns=['agrst'])
y = spojine_data['agrst']

model = DecisionTreeClassifier()
model.fit(X, y)

tree.export_graphviz(model, out_file='tree_spojine.dot',
                     feature_names=['mmasa', 'temp'],
                     class_names=sorted(y.unique()),
                     label='all',
                     rounded=True,
                     filled=True)
```

Vsebino datoteke `tree_spojine.dot` najlažje pretvorimo v grafično obliko v spletnih verzijah aplikacije `Graphviz`, ki so prosto dostopne na spletu in omogočajo izvoz grafike v ustreznem formatu, npr. tukaj. Rezultat je drevo na sliki 13.1.

Model odločitvenega drevesa je uporaben za hitro napovedovanje in je preprost za razumevanje, vendar vseh izhodov ne napove vedno pravilno. Točnost modela je odvisna od več dejavnikov:

Kakovosti podatkov. Če so podatki nepopolni, pomanjkljivi ali vsebujejo napake, model težje prepozna vzorce in posledično napake v napovedih niso redkost.

Velikosti nabora podatkov. Pri majhnih naborih podatkov, kot je v našem primeru, se model lahko nauči specifičnih vrednosti namesto splošnih pravil, kar lahko vodi v preveč prilagojen model (*overfitting*).

Preproste strukture modela. Odločitveno drevo je relativno enostaven model, ki je primeren za začetne analize, vendar ni vedno dovolj zmogljiv za kompleksnejše probleme z veliko spremenljivkami ali nelinearnimi zvezami.

Slika 13-1: Odločitveno drevo

14 Priloge

14.1 Najpogostejše napake v sintaksi

Pri pisanju programske kode se pogosto pojavljajo napake v sintaksi. Spodaj navajamo seznam najpogostejših napak in opozoril, ki nam jih *Python* sporoči, ko pride do napake.

1. Pozabljena dvopičja (:) pri definiciji blokov (npr. `if, for, while, def, class`)
   Primer:
   ```
   if x > 5
       print("X je večji od 5")
   ```
   Opozorilo: `SyntaxError: invalid syntax`
   Razlaga: *Python* zahteva dvopičje (:) na koncu vrstic, ki uvajajo blok kode.

2. Nepravilni zamik
   Primer:
   ```
   if x > 5:
   print("X je večji od 5")
   ```
   Opozorilo: `IndentationError: expected an indented block`
   Razlaga: V *Pythonu* so bloki kode določeni z zamikom, običajno s štirimi presledki ali tabulatorjem. Vse vrstice v istem bloku morajo biti pravilno poravnane.

3. Nepravilna uporaba oklepajev
   Primer:
   ```
   print("Pozdrav"
   ```
   Opozorilo: `SyntaxError: unexpected EOF while parsing`
   Razlaga: Oklepaji (zaokroženi, oglati ali zaviti) morajo biti vedno pravilno zaprti.

4. Manjkajoči narekovaji pri znakovnih spremenljivkah
   Primer:
   ```
   print(Hello, world!")
   ```
   Opozorilo: `SyntaxError: EOL while scanning string literal`
   Razlaga: Vsaka znakovna spremenljivka (*string*) mora biti ustrezno zaključena z enojnim (`'`) ali dvojnim (`"`) narekovajem.

5. Nepravilna uporaba ključnih besed
   Primer:
   ```
   True = 5
   ```
   Opozorilo: `SyntaxError: cannot assign to True`
   Razlaga: Ključnih besed, kot so `True`, `False`, `None`, `and`, `or`, `if`, `for`, `while` itd., ni mogoče uporabiti kot imen spremenljivk.

6. Uporaba nedefinirane spremenljivke
Primer:
```
x=5
print(y)
```
Opozorilo: `NameError: name 'y' is not defined`
Razlaga: Spremenljivka mora biti definirana, preden jo lahko uporabimo v programu.

7. Nepravilen vrstni red priglasitve spremenljivk
Primer:
```
y = x + 2
x = 5
```
Opozorilo: `NameError: name 'x' is not defined`
Razlaga: Spremenljivka `x` mora biti priglašena in dodeljena pred uporabo v drugih izrazih.

8. Nepravilna uporaba primerjalnih operaterjev
Primer:
```
if x = 5:
    print("X je 5")
```
Opozorilo: `SyntaxError: invalid syntax`
Razlaga: Za primerjavo vrednosti se v *Pythonu* uporablja `==`, medtem ko se enačaj (`=`) uporablja za dodelitev vrednosti spremenljivki.

9. Preseganje obsega indeksa seznama
Primer:
```
my_list = [1, 2, 3]
print(my_list[5])
```
Opozorilo: `IndexError: list index out of range`
Razlaga: Poskušali ste dostopati do elementa z indeksom, ki ne obstaja v seznamu.

10. Deljenje z nič
Primer:
```
a,b=3,0
c=a/b
```
Opozorilo: `ZeroDivisionError: division by zero`
Razlaga: Deljenje z nič ni dovoljeno, saj matematično ni definirano.

11. Napaka pri klicu funkcije brez oklepajev
Primer:
```
def pozdrav():
    print("Pozdrav!")

pozdrav
```
Opozorilo: `TypeError: 'function' object is not callable`
Razlaga: Funkcijo je treba klicati z oklepaji `()`.

14.2 Rešitve problemov

V tem poglavju so zbrane rešitve problemov. Pri vsakem problemu je praviloma prikazana ena rešitev. Možnih rešitev pa je veliko. Če ste ustvarili drugačno rešitev, kot je zapisana v gradivu, jo preverite s programom. Če dobite enak rezultat, je tudi vaša rešitev najverjetneje pravilna.

14.2.1 Rešitve problemov 1. poglavja

Problem 1-1:

$134_{(10)} = 10000110_{(2)}$

$86{,}09375_{(10)} = 1010110{,}00011_{(2)}$

$112{,}1_{(10)} = 1110000{,}000\overline{1100}_{(2)}$

Problem 1-2:

$11001101_{(2)} = 205_{(10)}$

$10101010{,}10_{(2)} = 170{,}5_{(10)}$

$11110111110{,}1111_{(2)} = 1982.9375_{(10)}$

Problem 1-3:

```
ZAČETEK

VNOS PODATKOV
    V = 0.1  # volumen raztopine v litrih
    c = 0.1  # množinska koncentracija v mol/L
    M = 40   # molekulska masa NaOH v g/mol

IZRAČUNI
    n = V * c     # množina NaOH v mol
    m = n * M     # masa NaOH v g
    GAMA = m / V  # masna koncentracija raztopine v g/L

IZPIS
    Izpiši m
    Izpiši GAMA

KONEC
```

Problem 1-4:

```
ZAČETEK

VNOS PODATKOV
    Preberi m_NaCl    # masa NaCl (g)
    Preberi V_H2O     # volumen vode (mL)
    Preberi ro_razt   # gostota raztopine (kg/L)
    Preberi M_NaCl    # molska masa NaCl (g/mol)
```

```
IZRAČUNI
    Predpostavka volumen raztopine je enak volumnu vode
    Pretvori volumen iz mL v L: V_razt = V_H2O/1000    #(L)
    Izračunaj množino:       n_NaCl = m_NaCl/M_NaCL    #(mol)
    Izračunaj množin. konc.: c_NaCl = n_NaCl/V_razt    #(mol/L)
    Izračunaj masno konc.:   gama_NaCl = m_NaCl/V_razt #(g/L)

IZPIS
    Izpiši c_NaCl       # množinska koncentracija v mol/L
    Izpiši gama_NaCl    # masna koncentracija v g/L
KONEC
# Razmisli, ali je uporabljeno poimenovanje spremenljivk potencialno
# problematično.
```

Problem 1-5:

```
ZAČETEK

VNOS PODATKOV
  Preberi H_r2mol # reakcijska entalpija za sežig 2 mol CH4 (kJ)
  Preberi n_CH4     # množina CH4 za sežig (mol)

IZRAČUNI
  Izračunaj entalpijsko spremembo: H_r = H_r2mol* n_CH4/2 #(kJ)

IZPIS
  Izpiši H_r  # entalpijska sprememba (kJ)

KONEC
```

14.2.2 Rešitve problemov 3. poglavja

Problem 3-1:

`answeR`	pravilno
`J+329`	nepravilno, + v imenu
`Lambda_stop`	pravilno
`X-ray`	nepravilno, vezaj
`2145`	nepravilno, številka na prvem mestu
`Try_1`	pravilno

Problem 3-2:

`a = b + c`	pravilno
`b + c = A`	nepravilno, ker je aritmetični izraz na levi
`a = a + a`	pravilno
`A+B = C+D`	nepravilno, ker je aritmetični izraz na levi ali + v imenu spremenljivk
`Ab = 5x`	nepravilno, ker na desni manjka aritmetični operator ali ker je številka na prvem mestu v imenu spremenljivke
`oseba_ime = "Ana"+"Marko"`	pravilno
`oseba_ime= Ana + Marko`	nepravilno, če sta Ana in Marko znakovni spremenljivki, ker nista v narekovajih; pravilno, če sta numerični spremenljivki

Problem 3-3:

```
in kemijsko tehnologijo
```

Izpis vseh treh nizov:

```
var_ime = "Fakulteta"
print (var_ime)
var_ime = "za kemijo"
print (var_ime)
var_ime = "in kemijsko tehnologijo"
print (var_ime)
```

ali (s poseganjem v imena spremenljivk)

```
var_ime1 = "Fakulteta"
var_ime2 = "za kemijo"
var_ime3 = "in kemijsko tehnologijo"
var_ime = var_ime1 +" "+ var_ime2 +" "+ var_ime3
print (var_ime)
```

Problem 3-4:

V prvi vrstici ni definirana spremenljivka *y*. Zamenjaj vrstni red vrstic:

```
y = 10
x = y + 5
```

Problem 3-5:

a) `10 = x + 5` → `x = 10 - 5`

b) `y + 5 = 12` → `y = 12 - 5`

Problem 3-6:
Prikazana je ena možna rešitev za vsak primer. Možnih rešitev je več.

a)
```
first_variable = 10
my_variable = 5
total_sum = first_variable + my_variable
print(total_sum)
```

b)
```
ifx = 20
x = ifx + 10
print(x)
```

c)
```
age = 25
message = "Star sem " + str(age) + " let."
print(message)
```

ali

```
age = "25"
message = "Star sem " + age + " let."
print(message)
```

Problem 3-7:
```
x=33.9
y=3.0
N=x+y
Z=x-y
print(x,y,N,Z)
```

14.2.3 Rešitve problemov 4. poglavja

Problem 4-1:

$$X = A + J*(L/3)**2 + B/4*(L/A)$$
$$X = 15.3 + 8*(32/3)**2 + (-7.4)/4*(32/15.3)$$
$$X = 15.3 + 8*(10.6667)**2 + (-7.4)/4*(2.0915)$$
$$X = 15.3 + 8*113.7785 + (-7.4)/4*2.0915$$
$$X = 15.3 + 910.2280 + (-7.4)/4*2.0915$$
$$X = 15.3 + 910.2280 + (-1.85)*2.0915$$
$$X = 15.3 + 910.2280 + (-3.8693)$$
$$X = 925.5280 + (-3.8693)$$
$$X = 921.6587$$

Problem 4-2:

a) `sqrt(5*x**2+8*y**2)`

b) `sin(x-2*y) + exp(x*y) - abs(x**2-y**2)`

Problem 4-3:

a) `p=x*y/(z*1000)`

b) `z = 2*x+3/y+exp(-2*x)`

c) `z = sqrt(2*x+1)+(2*x+1)**(-3)`

d) `a = (c**2-b**2)**(1/2)` ali `a = sqrt(c**2-b**2)`

Problem 4-4:

```
4.711687595755898
8.160882305241266
```

Problem 4-5:

```
6.907755278982137
3.0
```

Problem 4-6:

a)

$$p = \frac{RT}{V_m - b} - \frac{a(T)}{V_m(V_m + b)} \quad \Rightarrow \quad ulomek_1 = \frac{RT}{V_m - b}$$
$$ulomek_2 = \frac{a(T)}{V_m(V_m + b)}$$
$$p = ulomek_1 - ulomek_2$$

$$p = \frac{RT}{V_m - b} - \frac{a(T)}{V_m(V_m + b)} \quad \Rightarrow$$
$$stev_1 = RT$$
$$imen_1 = V_m - b$$
$$stev_2 = a(T)$$
$$imen_2 = V_m(V_m + b)$$
$$p = \frac{stev_1}{imen_1} - \frac{stev_2}{imen_2}$$

b)

$$p = \frac{RT}{V_m - b} - \frac{a(T)}{V_m(V_m + \check{n}b) + \zeta b(V_m - b)} \quad \Rightarrow$$

$$ulomek_1 = \frac{RT}{V_m - b}$$

$$ulomek_2 = \frac{a(T)}{V_m(V_m + \check{n}b) + \zeta b(V_m - b)}$$

$$p = ulomek_1 - ulomek_2$$

$$p = \frac{RT}{V_m - b} - \frac{a(T)}{V_m(V_m + \check{n}b) + \zeta b(V_m - b)} \quad \Rightarrow$$

$$stev_1 = RT$$

$$imen_1 = V_m - b$$

$$stev_2 = a(T)$$

$$imen_2 = V_m(V_m + \check{n}b) + \zeta b(V_m - b)$$

$$p = \frac{stev_1}{imen_1} - \frac{stev_2}{imen_2}$$

Problem 4-7:

```
from math import *
a=3
b=4
c=sqrt(a**2+b**2)
print(c)
```

Problem 4-8:

```
from math import *
kot1=60
kot2=30
kot1_rad=kot1*pi/180
kot2_rad=kot2*pi/180
a=cos(kot1_rad)
b=sin(kot2_rad)
print(a,b)
```

ali

```
from math import *
a=cos(60*pi/180)
b=sin(30*pi/180)
print(a,b)
```

ali še bolj zgoščeno

```
from math import *
print(cos(60*pi/180), sin(30*pi/180))
```

Vsi trije načini so pravilni – izberite svojega.

14.2.4 Rešitve problemov 5. poglavja

Problem 5-1:

```
 3 4
 48
 mesto Maribor
mesto
Ljubljana
```

Problem 5-2:

```
a=3
b=4
print('c='+str(a)+'**'+str(b))
```

Problem 5-3:

```
a=2
b=3
c=4
```

a) `print(a,b,c)`

b) `print(c,b,a)`

c) `print(str(a)+str(b)+str(c))`
v šestem poglavju tudi: `print(f"{a}{b}{c}")`

d) `print(str(a)+', '+str(b)+', '+str(c))`
v šestem poglavju tudi: `print(f"{a}, {b}, {c}")`

e)
```
print(a)
print(b)
print(c)
```
ali
```
print(a, b, c, sep='\n')
```

Problem 5-4:

Datoteka input.txt:

```
Vrstica0
Vrstica1
Vrstica2
Vrstica3
Vrstica4
Vrstica5
Vrstica6
Vrstica7
Vrstica8
Vrstica9
```

Program:

```
infile= open('input.txt', 'r')
lines = infile.read()
```

```
infile.close()

outfile=open('output.txt', 'w')
print(lines, file=outfile)
outfile.close()
```

ali:

```
# Preberi input.txt in zapisi vsebino v output.txt
# Odpri input.txt za branje
with open('input.txt', 'r') as infile:
    lines = infile.read()  # Preberi vse vrstice v vhodni datoteki

# Odpri output.txt za pisanje
with open('output.txt', 'w') as outfile:
    outfile.write(lines)  # Zapisi vse podatke v izhodno datoteko

print("Podatki iz input.txt so bili zapisani v output.txt.")
```

Problem 5-5: (po 9. poglavju – vsebuje pogojni stavek, zanko, seznam)

```
infile= open('input.txt', 'r')
lines = infile.readlines()
lines = [line.strip() for line in lines]
infile.close()

outfile=open('output.txt', 'w')
for index in range(len(lines)):
    if index % 2 == 0:
        print(lines[index], file=outfile)
outfile.close()
```

ali

```
# Preberi input.txt in zapisi vsako drugo vrstico v output.txt
# Odpri input.txt za branje
with open('input.txt', 'r') as infile:
    lines = infile.readlines()  # Preberi vse vrstice v seznam

# Odpri output.txt za pisanje
with open('output.txt', 'w') as outfile:
    for index in range(len(lines)):
        if index % 2 == 0:  # Zapisi vsako drugo vrstico (0, 2, ...)
            outfile.write(lines[index])

print("Vsaka druga vrstica iz input.txt je bila zapisana v
output.txt.")
```

14.2.5 Rešitve problemov 6. poglavja

Problem 6-1:

```
from math import*
```

a) Samo celi del števila

```
a=4.56744
a=int(a)
print(a)
#kompaktni zapis
a=4.56744
print(int(a))
```

b) Samo decimalni del števila

```
a=4.56744
a=a-int(a)
print(a)
#kompaktni zapis
a=4.56744
print(a-int(a))
```

c) Navzgor zaokroženo celo število

```
a=4.56744
a=ceil(a)
print(a)
#kompaktni zapis
a=4.56744
print(ceil(a))
#ali
print(f"{a:.0f}")
```

d) Navzdol zaokroženo celo število

```
a=4.56744
a=floor(a)
print(a)
#kompaktni zapis
a=4.56744
print(floor(a))
```

e) Število z dvema decimalnima mestoma, zaokroženo navzdol (4.56)

```
a=4.56744
a=100*a
a=int(a)
a=a/100
print(a)
#kompaktni zapis
a=4.56744
print(int(100*a)/100)
```

f) Število z dvema decimalnima mestoma, zaokroženo navzgor (4.57)

```
a=4.56744
a=100*a
a=int(ceil(a))
a=a/100
print(a)
#kompaktni zapis
a=4.56744
print(int(ceil(100*a))/100)
#ali
print(f"{a:.2f}")
```

g) Kot 4,56744

```
a=4.56744
a=str(a) # spremenljivka a postane znakovna spremenljivka!!!
a=a.replace('.',',')
print(a)
#kompaktni zapis
print(str(a).replace('.',','))
```

Problem 6-2:

```
a=12.34567
b=152.1
c=-1115.3
print(f"{a:<10.3f}{b:<10.0f}{c:<10.2f}")
```

Izpis:

```
12.346    152       -1115.30
```

Problem 6-3:

Navedene so le nekatere možnosti, ki lahko predstavljajo težavo. Kot primer navajamo število 7.02:

a) `Vnos: 7,02  # decimalna vejica namesto pike`

b) `Vnos: 7. 02 # presledek za decimalno piko (problematičen je lahko`
`# tudi pred prvo številko ali za zadnjo)`

c) `Vnos: z.O2  # 7 pogosto postane z in 0 pogosto O`
`# (bližina tipk na tipkovnici)`

d) `Vnos: abc   # popolnoma napačen tip spremenljivke`

e) `Vnos:       # brez kakršnegakoli vnosa`

14.2.6 Rešitve problemov 7. poglavja

Problem 7-1:

a)

```
n=0
if n==0:
    x=10
    print(x)
```

b)

```
a=8
b=6
if a>b:
    print(a)
```

c)

```
a=8
b=12
x=3
y=2
z=22
if x>y and a<b:
    c=z
    print(c)
```

d)

```
from math import *
X=24
if X > 0:
    Y=sqrt(X)
    print(X, Y)
```

Problem 7-2:

a)

```
a=3
b=4
c=3
if a == b & b == c:
    print("a je enak c")
else:
    print("a je morda enak c")
```

b)

```
a=2
b=6
if b/a == 2:           #ali if b == 2*a:
    print("b = 2a")
else:
    print("b != 2a")
```

c)

```
a=8
b=-2
if a%b == 0:
    c = a/b
    print("b je delitelj a")
    print("kvocient je " + str(c))
else:
    print("b ni delitelj a")
```

Problem 7-3:

```
a="CO2"
b="H2O"
if (a == "Na" or a== "K") and b == "H2O":
    print("Pazi, eksplozija! ")
else:
    print("Vrei je… verjetno")
```

Problem 7-4:

```
a = "H2O"
b = "K"
if a != "Na" and a != "K" and a != "H2O" or \
   b != "Na" and b != "K" and b != "H2O":
    print("Ne prepoznam spojine! ")
else:
    if (a == "Na" or a== "K") and b == "H2O" or \
        a == "H2O" and (b=="Na" or b=="K"):
        print("Pazi, eksplozija! ")
    else:
        print("Vrei je… verjetno")

#Opomba: \ je znak za nadaljevanje vrstice
```

14.2.7 Rešitve problemov 8. poglavja

Problem 8-1:

Datoteka TLAK.DAT (se mora nahajati na istem direktoriju kot *Python* koda):

```
0.08314
273.15
22.4
```

Program:

```
podatki = open("TLAK.DAT", "r")
a=podatki.readlines()
podatki.close()
R=float(a[0])
temperatura=float(a[1])
volumen=float(a[2])
for n in range(1,6):
    tlak=R*temperatura*n/volumen
    print(f"n= {n} mol   tlak= {tlak:.5f} bar")
```

Po zagonu se izpiše:

```
n= 1 mol   tlak= 1.01383 bar
n= 2 mol   tlak= 2.02765 bar
n= 3 mol   tlak= 3.04148 bar
n= 4 mol   tlak= 4.05530 bar
n= 5 mol   tlak= 5.06913 bar
```

Problem 8-2:

```
for faren in range(0,213,2):
    celzij=5/9*(faren-32)
    print(faren, celzij)
```

Po zagonu se izpiše:

```
0   -17.77777777777778
2   -16.666666666666668
4   -15.555555555555557
6   -14.444444444444445
8   -13.333333333333334
...
208   97.77777777777779
210   98.8888888888889
212   100.0
```

Če želimo, da so stopinje Celzija zapisane na 3 decimalna mesta, uporabimo formatiranje v stavku `print`:

```
print(faren, f"{celzij:.3f}")
```

Problem 8-3:

```
for i in range(10):
    print("Ime")
```

ali z zanko *while*:

```
i=1
while i<=10:
    print(i,"Ime")
    i=i+1
```

Problem 8-4:

```
for stev in range(1,100,2):
    print(stev)
```

ali

```
for stev in range(1,51):
    print(2*stev-1)
```

ali z zanko *while*:

```
stev=1
while stev<=50:
    print(2*stev-1)
    stev=stev+1
```

Problem 8-5:

```
for celo_stev in range(1,16):
    kvadrat=celo_stev**2
    kub=celo_stev**3
    print(celo_stev,kvadrat,kub)
```

Problem 8-6:

```
for milje in range(10,101,10):
    km=milje*1.6
    print(milje, km)
```

Problem 8-7:

```
3 0
4 1
5 2
6 3
7 4
8 5
```

Problem 8-8:

```
Leto:  100  Stoletje:   1
Leto:  200  Stoletje:   2
Leto:  300  Stoletje:   3
Leto:  400  Stoletje:   4
Leto:  500  Stoletje:   5
Leto:  600  Stoletje:   6
Leto:  700  Stoletje:   7
Leto:  800  Stoletje:   8
Leto:  900  Stoletje:   9
Leto:  1000  Stoletje:   10
Leto:  1100  Stoletje:   11
```

```
Leto:  1200  Stoletje:   12
Leto:  1300  Stoletje:   13
Leto:  1400  Stoletje:   14
Leto:  1500  Stoletje:   15
Leto:  1600  Stoletje:   16
Leto:  1700  Stoletje:   17
Leto:  1800  Stoletje:   18
Leto:  1900  Stoletje:   19
Leto:  2000  Stoletje:   20
```

Problem 8-9:

```
8
9
10
```

Problem 8-10:

Vhodna datoteka s podatki DELAVCI.DAT:

```
Ime1 Priimek1 185 10.44
Ime2 Priimek2 145 13.58
Ime3 Priimek3 174 18.22
```

Program:

```
file = open('DELAVCI.DAT', 'r')

for i in range(3):
    data = file.readline().split()
    ime = data[0]
    priimek = data[1]
    ure = float(data[2])
    vrednost = float(data[3])

    if ure <= 174:
        placa = ure * vrednost
        nadure = 0
    else:
        placa = 174 * vrednost
        nadure = (ure - 174) * vrednost * 1.3
    print(ime, priimek, placa, nadure)
```

Če želimo izpis vrednosti plače in nadur na dve decimalni mesti, zamenjamo `print` stavek z naslednjim:

```
print(ime, priimek, f"{placa:.2f}", f"{nadure:.2f}")
```

Brez branja iz datoteke bi program lahko zapisali npr. tako:

```
ime=["Ime1", "Ime2", "Ime3"]
priimek=["Priimek1", "Priimek2", "Priimek3"]
ure=[185, 145, 174]
vrednost=[10.44,13.58,18.22]
placa=[None]*3
nadure=[None]*3
```

```
        for i in range(3):
             if ure[i] <= 174:
                placa[i]=ure[i]*vrednost[i]
                nadure[i]=0
             else:
                placa[i]=174*vrednost[i]
                nadure[i]=(ure[i]-174)*vrednost[i]*1.3
             print(ime[i], priimek[i], placa[i], nadure[i])
```

Problem 8-11:

```
skrito_stevilo="4"
i=1
konec=False

while i<=3 and konec==False:
    ugibam=input("Ugani celo stevilo med 1 in 5 v treh poskusih: ")
    if ugibam == skrito_stevilo:
        konec = True
    else:
        i=i+1

if konec==True:
    print("Cestitke")
else:
    print("Vec srece prihodnjic")
```

ali

```
skrito_stevilo = 4
ugibam = ""
i=1
limit=3
konec=False

while ugibam != skrito_stevilo and not(konec):
    if i <= limit:
        ugibam = input("Ugani celo stevilo med 1 in 5 v treh poskusih: ")
        ugibam = int(ugibam)
        i=i+1
    else:
        konec=True

if konec==True:
     print("Vec srece prihodnjic!")
else:
     print("Cestitke!")
```

Opomba: spremenljivka `konec` je logična spremenljivka, ki označuje, ali je uporabnik že izkoristil tri poskuse (`true`) ali še ne (`false`).

14.2.8 Rešitve problemov 9. poglavja

Problem 9-1:

```
voda 100000
etanol 64000
metanol 60840
```

Problem 9-2:

```
['voda', 'etanol', 'metanol'] [100000, 64000, 60840]
```

Problem 9-3:

```
voda 12.471
etanol 35.63142857142857
metanol 5.19625
[12.471, 35.63142857142857, 5.19625]
```

Problem 9-4:

Datoteka Trikemikalije.dat

```
Voda 2 1.000 18
Etanol 3 0.789 46
Glicerol 1 1.260 92.1
```

Program brez seznamov:

```
file=open('Trikemikalije.dat', 'r')
for i in range(3):
    line=file.readline()
    parts=line.split()
    tekocine=parts[0]
    volumen=float(parts[1])
    gostota=float(parts[2])
    mmasa=float(parts[3])
    masa=volumen*gostota
    mnozina=masa/mmasa*1000
    print(f"{tekocine:<8}{masa:>8.3f}{mnozina:>10.3f}")
```

Program s seznami:

```
file=open('Trikemikalije.dat', 'r')
tekocine=[None]*3
volumen=[None]*3
gostota=[None]*3
mmasa=[None]*3
masa=[None]*3
mnozina=[None]*3
for k in range(3):
    line=file.readline()
    parts=line.split()
    tekocine[k]=parts[0]
    volumen[k]=float(parts[1])
    gostota[k]=float(parts[2])
```

```
        mmasa[k]=float(parts[3])
        masa[k]=volumen[k]*gostota[k]
        mnozina[k] = masa[k]/volumen[k]*1000
        print(f"{tekocine[k]:<8}{masa[k]:>8.3f}{mnozina[k]:>10.3f}")
```

Problem 9-5:

```
111.0
```

Problem 9-6:

```
1 1
2 4
3 9
4 16
5 25
6 36
```

Problem 9-7:

```
1 50 70 60
2 60 80 70
3 70 90 80
```

S formatiranim print stavkom:

```
1    50   70   60
2    60   80   70
3    70   90   80
```

Problem 9-8:

Podatkovna datoteka Podatki.dat

```
10 20 30
40 50 60
70 80 90
```

Varianta 1: Prebere vse vrstice iz datoteke in jih shrani v spremenljivko `vsebina`. Vsak element predstavlja eno vrstico iz datoteke. Ko iteriramo prek seznama `vsebina`, se vsaka vrstica po vrsti shrani v spremenljivko `line`.

```
10 20 30

40 50 60

70 80 90
```

Varianta 2: Ta varianta iterira prek datoteke vrstico po vrstico neposredno v zanki `for line in file`. S funkcijo `split` loči vsako vrstico na seznam znakovnih spremenljivk, ločenih s presledki, to je seznam `x`. Nato se znakovni elementi seznama `x` z ukazom `float` spremenijo v numerične elemente in s tem se tvori seznam `y`.

```
[[10.0, 20.0, 30.0], [40.0, 50.0, 60.0], [70.0, 80.0, 90.0]]
```

Opomba: Z ukazom `for line in file` program bere po eno vrstico naenkrat, zato ukaz `read` ali `readline` ni potreben.

Kaj se izpiše, če programu dodamo še naslednje:

```
for line in vrstice:
    a=sum(line)
    print(a,line)
```

Izpiše se:

```
[[10.0, 20.0, 30.0], [40.0, 50.0, 60.0], [70.0, 80.0, 90.0]]
60.0 [10.0, 20.0, 30.0]
150.0 [40.0, 50.0, 60.0]
240.0 [70.0, 80.0, 90.0]
```

Problem 9-9:

Datoteka Podatki.txt

```
10 20 30
60 70 80
90 100 110
```

Program:

```
file = open("Podatki.txt", "r")
a=file.readlines()
vsa=[]
for line in a:
    stevila=list(map(int,line.split()))
    vsa.append(stevila)

N1, N2, N3 = vsa
print(N1)
print(N2)
print(N3)
Povpr=[(x+y+z)/3 for x,y,z in zip(N1,N2,N3)]
print(Povpr)
```

Izpis:

```
[10, 20, 30]
[60, 70, 80]
[90, 100, 110]
[53.333333333333336, 63.333333333333336, 73.33333333333333]
```

14.2.9 Rešitve problemov 10. poglavja

Problem 10-1:

Program z aritmetičnim vnosom podatkov

```
from math import *
def izracun(temperatura,a,b,c):
    parni_tlak=exp(a-b/(c+temperatura))/750
    return parni_tlak

spojine=["etanol","voda","metanol"]
aji=[18.5242, 18.3036, 18.5875]
bji=[3578.91, 3816.44, 3626.55]
cji=[-50.5, -46.13, -34.29]

temperatura=303
for i in range(3):
    a=aji[i]
    b=bji[i]
    c=cji[i]
    ptlak=izracun(temperatura,a,b,c)
    print(spojine[i],"  ", f"{ptlak:.4f}")
```

Program z branjem podatkov iz datoteke

Datoteka s podatki ParniTlak.dat:

```
etanol  18.5242 3578.91 -50.5
voda    18.3036 3816.44 -46.13
metanol 18.5875 3626.55 -34.29
```

Program:

```
from math import *
def izracun(temperatura,x,y,z):
    parni_tlak=exp(x-y/(z+temperatura))/750
    return parni_tlak

file=open('ParniTlak.dat', 'r')
temperatura=303
for line in file:
    parts=line.split()
    spojina=parts[0]
    a=float(parts[1])
    b=float(parts[2])
    c=float(parts[3])
    ptlak=izracun(temperatura,a,b,c)
    print(spojina,"  ", f"{ptlak:.4f}")
```

Problem 10-2:

Datoteka s podatki Povpr.dat:

```
100 200 300
6 8 9
13 15 17
```

Program:

```
def povprfun(x):
    z = sum(x) / len(x)
    return z

file = open('Povpr.dat', 'r')
for line in file:
    a = line.split()
    x = [float(i) for i in a]
    povprecje = povprfun(x)
    print(f"{povprecje:.2f}")
```

Ko uporabimo ukaz »`for line in file:`«, ni treba ponovno klicati »`file.readline()`«. Namesto tega lahko neposredno uporabimo ukaz »`line`« znotraj zanke.

Problem 10-3:

```
def potenciranje(osnova, eksponent):
    rezultat = 1
    for index in range(eksponent):
        rezultat = rezultat * osnova
    return rezultat
print(potenciranje(3, 4))
```

Program preuredimo tako, da bo uporabnik interaktivno vstavil osnovo in eksponent. Funkcija `potenciranje` ostane enaka kot prej, preostali del programa se spremeni tako:

```
osnova = float(input("Osnova: ") )
eksponent = int(input("Eksponent: ") )
rezultat = potenciranje(osnova, eksponent)
print(rezultat)
```

Problem 10-4:

```
def sprememba_g(besedilo):
    novo_besedilo = ""
    for i in besedilo:
        if i in "AEIOUaeiou":
            novo_besedilo = novo_besedilo + "g"
        else:
            novo_besedilo = novo_besedilo + i
    return novo_besedilo

print(sprememba_g(input("Vnesi besedilo: ")))
```

14.3 Podatki za primer strojnega učenja

Podatke prenesite v *Excel* in jih shranite v datoteko `spojine.csv`.

mmasa	temp	agrst
18	–20	trdno
18	55	tekoce
17	–100	trdno
17	25	plinasto
32	–120	trdno
32	50	tekoce
44	20	plinasto
44	–100	tekoce
58	–55	tekoce
58	15	plinasto
71	5	plinasto
71	–50	tekoce
98.91	–15	tekoce
98.91	35	plinasto
58.08	28	tekoce
58.08	77	plinasto
58.08	–100	trdno
28	–250	tekoce
28	28	plinasto
78	45	tekoce
78	102	plinasto
78	–5	trdno
65	–115	trdno
65	45	tekoce
65	120	plinasto

Datoteka `.csv` je tekstovna datoteka, v kateri so podatki ločeni z vejico (angl. *comma separated values*). Datoteko odprite s katerim od tekstovnih urejevalnikov, npr. z *notepadom*, in preverite, ali so res uporabljene vejice. Če so zaradi nastavitev računalnika podatki ločeni s katerim drugim ločilom, npr. podpičjem, jih je treba spremeniti v vejico ali spremeniti nastavitve za decimalno ločilo v *Excelu* (*File*, *Options*, *Advanced*, *Decimal separator*).

15 Dodatna študijska literatura

Gradivo je bilo pripravljeno na osnovi številnih video učnih pripomočkov, od katerih izpostavljamo:

freeCodeCamp.org, Learn Python – Full Course for Beginners (Tutorial), https://www.youtube.com/watch?v=rfscVS0vtbw&t=12285s. Dostop 11. 2. 2025.

Programming with Mosh, Python Machine Learning Tutorial (Data Science), https://www.youtube.com/watch?v=7eh4d6sabA0&list=LL&index=28&t=32s. Dostop 11. 2. 2025.

Ideje za nekatere primere in probleme izhajajo iz učbenika:

Majda Krajnc, Računalništvo v kemiji, zbrano gradivo, 2. popravljena izdaja. Maribor, FKKT Univerze v Mariboru, 2013.

Pri pripravi učbenika so bili uporabljeni tudi primeri in razlage, pridobljeni s pomočjo umetne inteligence (ChatGPT, OpenAI, 2025).

Seznam tabel

Seznam slik

Python v kemiji in kemijskem inženirstvu: Učbenik za predmet Računalništvo v kemiji

Zorka Novak Pintarič, Sanja Potrč, Miloš Bogataj
Univerza v Mariboru, Fakulteta za kemijo in kemijsko tehnologijo, Maribor, Slovenija
zorka.novak@um.si, sanja.potrc@um.si, milos.bogataj@um.si

Učbenik Python v kemiji in kemijskem inženirstvu je namenjen študentom prvega letnika, ki se prvič srečujejo z računalniškim programiranjem. Na preprost in sistematičen način uvaja bralca v osnovne pojme programskega okolja Python ter postopno prikazuje njegovo uporabo pri reševanju izbranih problemov iz kemije in kemijskega inženirstva. Poleg osnov programiranja prinaša uvodne primere uporabe sodobnih digitalnih pristopov, kot sta podatkovna analiza in umetna inteligenca, s čimer študentom odpira vpogled v sodobne trende kemijsko-inženirske stroke. Publikacija je zasnovana tako, da razvija praktične računalniške spretnosti ter spodbuja samostojno učenje in nadaljnje raziskovanje. Učbenik je objavljen kot odprto učno gradivo, kar študentom, učiteljem in strokovnjakom, ki jih zanima uporaba Pythona v kemiji in kemijskem inženirstvu, omogoča dolgoročno podporo pri študiju in delu.

DOI
https://doi.org/10.18690/um.fkkt.7.2025

ISBN
978-961-299-075-6

Ključne besede:
Python,
kemija,
kemijsko inženirstvo,
programiranje,
podatkovna analiza,
učbenik

Univerzitetna založba Univerze v Mariboru

DOI
https://doi.org/
10.18690/um.fkkt.7.2025

ISBN
978-961-299-075-6

PYTHON IN CHEMISTRY AND CHEMICAL ENGINEERING: TEXTBOOK FOR THE COURSE COMPUTER SCIENCE IN CHEMISTRY

ZORKA NOVAK PINTARIČ, SANJA POTRČ, MILOŠ BOGATAJ
University of Maribor, Faculty of Chemistry and Chemical Engineering, Maribor, Slovenia
zorka.novak@um.si, sanja.potrc@um.si, milos.bogataj@um.si

Keywords:
Python,
chemistry, chemical engineering,
programming,
data analysis,
textbook

The textbook Python in Chemistry and Chemical Engineering is intended for first-year students who are new to computer programming. It provides a clear and systematic introduction to the fundamentals of the Python programming environment and gradually demonstrates its application in solving selected problems from chemistry and chemical engineering. In addition to the basics, it introduces students to modern digital approaches such as data analysis and artificial intelligence, offering insight into current trends in the chemical engineering profession. The textbook is organized to build practical computational skills, promote independent learning, and encourage further exploration. Published as open educational material, it offers lasting support for students, educators, and professionals interested in applying Python in chemistry and chemical engineering.

Univerza v Mariboru

Fakulteta za kemijo
in kemijsko tehnologijo