Automatizacija zadataka u informacijskom sustavu pomoću programskog jezika Python

Horvatić, Patrik

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:996808

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Horvatić

AUTOMATIZACIJA ZADATAKA U INFORMACIJSKOM SUSTAVU POMOĆU PROGRAMSKOG JEZIKA PYTHON

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Horvatić

Matični broj: 0016135559

Studij: Poslovni sustavi

Automatizacija zadataka u informacijskom sustavu pomoću programskog jezika Python

ZAVRŠNI RAD

Mentor/Mentorica:

Aleksandra Sobodić, mag. inf.

Varaždin, rujan 2021.

Patrik Horvatić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U ovom završnom radu objašnjava se što je Python i primjene Pythona. Rad sadrži osnovne informacije o programskom jeziku, načinu na koji radi i njegovu uporabu u određenim granama programiranja sa statističkim podacima. Detaljno je objašnjena osnovna sintaksa verzije 3.8.7., a to su operatori, tipovi podataka i njihove prikladne funkcije, tokovi izvođenja, funkcije, klase, iznimke i rad s datotekama i sve od toga sa prikladnim primjera radi lakšeg razumijevanja. Rad također sadrži skripte koje služe za automatizaciju zadataka slanja elektronične pošte i dohvaćanja podataka dionica pomoću vanjskih servisa, njihova obrada i izvoz u PDF datoteku. Svrha tih skripti je prikazati lakoću i smislenost automatizacije takvih zadataka u poslovnim sustavima.

Ključne riječi: Python; sintaksa; automatizacija; trendovi; elektronična pošta; dionice; API

Sadržaj

1. Uvod

Ovaj završni rad predstavit će koliko je jednostavno u Pythonu automatizirati zadatke informacijskog sustava koji su ključni za organizaciju i njene zaposlenike. Današnje vrijeme predstavlja obradu velikih količina podataka i brzo prosljeđivanje i dohvaćanje najnovijih i najrelevantnijih informacija. Implementacija informacijskog sustava ključna je za opstanak organizacija. Dohvaćanje vanjskih podataka, automatsko slanje mailova, brza obrada slika i generiranje izvješća samo su neke od aktivnosti koje velika većina organizacija mora obaviti. Automatizacija zadataka u informacijskom sustavu snažan je alat koji omogućuje smanjenje troškova, podiže efikasnost i rasterećuje zaposlenike. Osobna motivacija za pisanje ovog završnog rada jest rast popularnosti i potražnje Python razvojnih programera, a ispred svega veliki interes o znanosti i statističkoj obradi podataka.

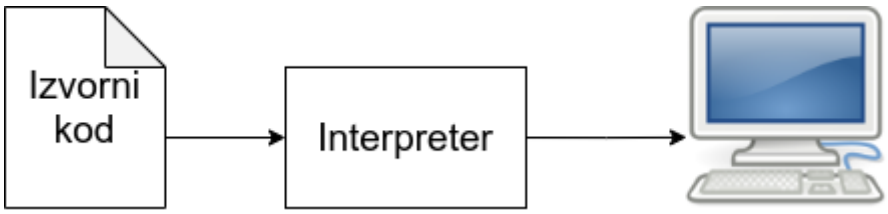
2. Metode i tehnike rada

Pri razradi teme literatura koju ću koristiti primarno su službena dokumentacija jezika Python i ugledni web portali koji sadrže valjane i provjerene informacije. Nakon potrošene veće količine vremena na traženje dobrih izvora literature, odlučio sam najprije izraditi skripte za automatizaciju zadataka. Na njih sam potrošio najviše vremena radi detaljnog proučavanja dokumentacije koja je prilično opsežna. Skripte su rađene u **Ubuntu 20.04** operacijskom sustavu. Korišten alat za pisanje skripti jest **Visual Studio Code** koji se dokazano pokazao kao najbolji alat za razvoj u Pythonu. Ovaj alat omogućuje instalaciju korisnih dodataka kao bojilo zagrada, formater koda, automatsko predlaganje riječi i sl. Za pisanje Word dokumenta korišten je **VirtualBox** sa pripremljenom Windows slikom sustava koji sadrži sve **Office** alate.

3. Uvod u Python

Po svojoj koncepciji, Python je objektno orijentirani jezik. Međutim, Python pored toga podržava i ostale stilove programiranja. Posebno je zanimljiv funkcijski stil koji je u Pythonu znatno bolje podržan u odnosu na druge objektno orijentirane jezike (Kalafić i sur., 2016). Python je viši programski jezik i nudi veliki broj prednosti. Mnogo je lakše programirati u višim jezicima jer se brže pišu, kraći su, lakše se čitaju i veća je vjerojatnost da će ispravno raditi. Programi pisani u višim jezicima mogu se prenositi, što znači da će se moći izvršavati na različitim vrstama računala uz gotovo nikakve ili male izmjene. Zbog ovih prednosti skoro svi programi pišu se pomoću viših programskih jezika. (Downey, 2014).

Prema Christennsonu (2010.), Python koristi interpreter za izvršavanje programa. Interpreter je program koji čita i izvršava kod i to uključuje izvorni kod i skripte. Poznati interpreteri su Perl, Python i Ruby interpreteri. Za razliku od kompajlera, interpreteri čitaju i izvršavaju kod liniju po liniju, stoga su korisni za pokretanje skripti i ostalih manjih programa. Česta je praksa instalacije interpretera kod servera gdje je potrebno pokrenuti skripte za obavljanje radnji. Ograničenja interpreterskih jezika jest da je za pokretanje programa potrebno instalirati interpreter na uređaju na kojem se program treba izvoditi. Inače je napisani program samo običan tekst.



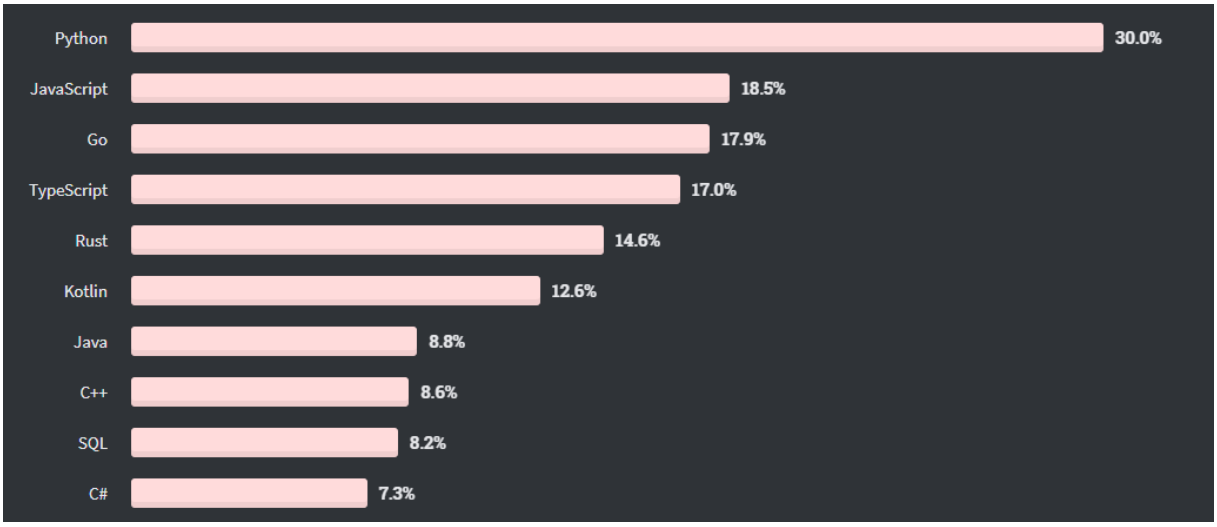
Slika 1: Interpreter obrađuje program dio po dio te naizmjenice čita redove i izvršava naredbe (Downey, 2014., str. 2)

Prema Chrisu (2019.), neke značajke programskog jezika Python su:

- Velik raspon tipova podataka
- Podrška objektno orijentiranom programiranju s klasama i višestrukim nasljedstvom
- Kod se može grupirati u module i pakete
- Jezik podržava podizanje i hvatanje iznimaka, što rezultira čišćim rukovanjem pogreškama
- Tipovi podataka su snažno i dinamički upisani
- Automatsko upravljanje memorijom oslobađa programera ručne dodjele i oslobađanja memorije u kodu

Python je jezik čija velika većina programa može izvesti na više operacijskih sustava kao Windows, Linux, Mac OS X i ostalim manje poznatim platformama. Ovo je moguće zbog opsežnih standardnih biblioteka koje pružaju platformski specifične funkcionalnosti iza transparentnog platformski neovisnog sučelja. Standardne biblioteke omogućuju izvođenje kompleksnih zadataka i zahtjeva u nekoliko linija koda (Kalafatić i sur., 2016).

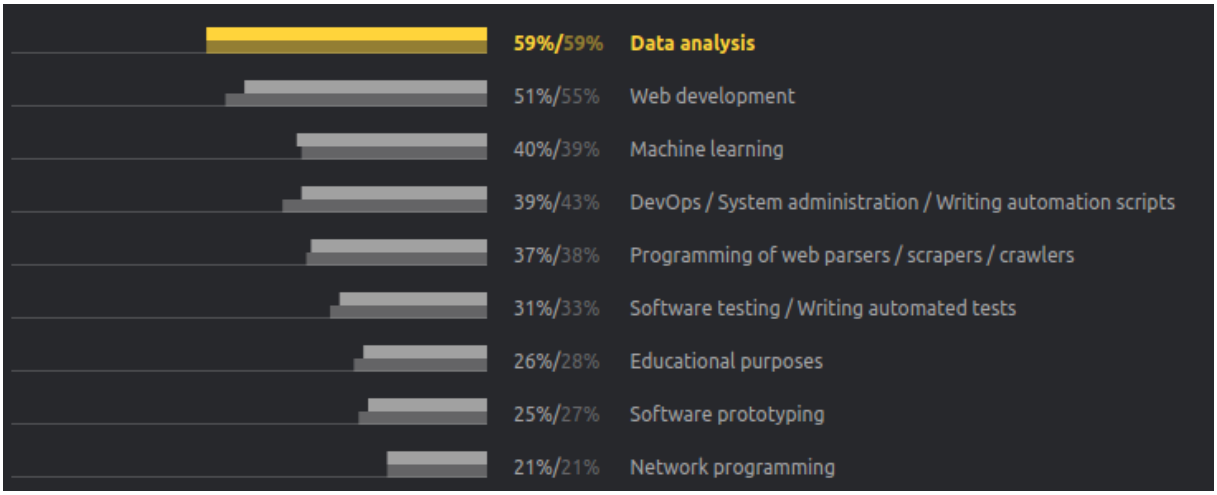
Programski jezik Python danas je najpopularniji programski jezik. Velika zajednica dijeli svoje projekte i programska rješenja otvorenog koda zbog čega je i Python zadnjih nekoliko godina dobio na popularnosti. Isto tako, Python je dobio na popularnosti pojavom umjetne inteligencije, znanosti podataka i financijskih analiza. Python je jednostavan jezik za naučiti zbog jednostavne sintakse, velike biblioteke i integracije drugih popularnih programskih jezika kao C i C++ (Eastwood, 2020). Pogledamo li statističke podatke sa Stackoverflow-a, vidimo da je Python 2020. godine bio na prvoj poziciji jezika koje bi razvojni programeri htjeli naučiti. Iz odabranog izvora nije objašnjen razlog razvojnih programera, ali pojavu i popularizaciju znanosti podataka, umjetne inteligencije i financijske obrade mogli bi smo usko povezati uz njihove odgovore. To je dio informacijskih znanosti koji raste i razvija se velikom brzinom.



Slika 2: Rezultati ankete najpoželjnijeg jezika.
Preuzeto 22.1.2021. s https://insights.stackoverflow.com/survey/2020#technology-mostloved-dreaded-and-wanted-languages-wanted

4. Trendovi korištenja Pythona

Jetbrains je organizacija koja svake godine provodi anketu sa razvojnim programerima. Sljedeća anketa prikazuje rezultate ankete na pitanje zašto programeri koriste Python. Anketa je provedena na oko 27 000 ispitanika iz više od 150 različitih zemalja.

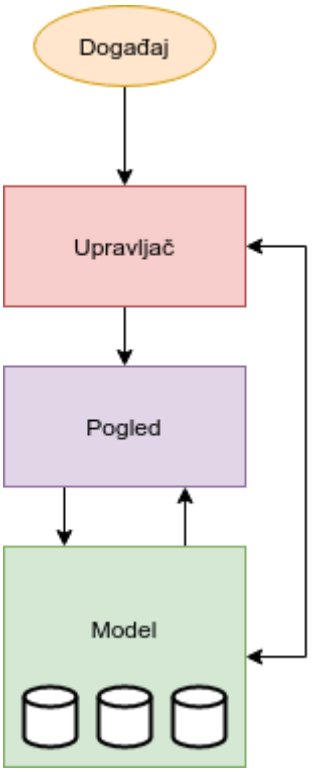


Slika 3: Anketa Python razvojnih programera (Jetbrains, 2019.)

Python se sve više koristi u **web razvoju** i izvrstan je izbor za brz razvoj web aplikacija. Sa Django, Flask, TurboGears i Pyramid okvirima (eng. frameworks) moguće je izraditi mnogo različitih web aplikacija velikom brzinom (Zestminds, 2020).

Django je full-stack okvir savršen za izradu web aplikacija. Opremljen je značajkama poput provjera autentičnosti, sučelja administratora, upravljanja sesijama, mehanizma uzoraka, objektno-relacijskog mapiranja i mnogo drugih. Uz sve to, širok raspon dodatnih modula i njihov razvoj još više podižu Djangovu popularnost i značaj (Azorin, 2019).

TurboGears izrađen je da ubrza i izgladi proceduru razvoja web aplikacija. Cijeli okvir ovisi o Ruby on Rails i stvoren je po uzorku „model-pogled-upravljač“ (engl. Model View Controller, MVC) (Zestminds, 2020). Model predstavlja najniži dio uzorka i odgovoran je za održavanje podataka. Odgovara na zahtjeve pogleda. Pogled je sloj odgovoran za prikaz dijelova podataka korisniku u određenom formatu. Upravljač je sloj odgovoran za odgovor unosa korisnika i izvođenje interakcija na objektima podatkovnog modela. Upravljač prima ulaz, provjerava ga i izvodi operaciju koja mijenja stanje podataka (Tutorialspoint, bez dat.).



Slika 4: Slojevi MVC uzorka. (Tutorialspoint, bez dat.)

Umjetna inteligencija danas je postao revolucionaran trend i mnoge organizacije žele ga primijeniti u svojim svakodnevnim poslovnim procesima. Za to je potreban programski jezik koji će pojednostaviti razvoj projekata temeljenih na umjetnoj inteligenciji. Python je najpovjerljivija opcija zbog prilično jednostavne sintakse i dobrog upravljanja ključnim procesima. Umjetna inteligencija je tehnologija koja pomaže znanstvenicima u rješavanju problema i donošenju rješenja u stvarnome vremenu. Python je stoga učvrstio svoje mjesto kod razvoja umjetne inteligencije (Zestminds, 2020). Glavni razlozi zašto je Python najviše primijenjen kod umjetne inteligencije jest što je potrebna 1/5 koda za razvoj umjetne inteligencije u odnosu na druge objektno orijentirane programske jezike i ima izrađene module za znanstveno i napredno računanje i strojno učenje, razvijenu zajednicu razvojnih programera koji dijele svoja rješenja i uče druge programere (Cuelogic, bez dat.).

Znanost podataka obuhvaća razvoj metoda snimanja, pohrane i analize podataka za učinkovito izdvajanje korisnih informacija. Cilj je steći uvid i znanje iz strukturiranih i nestrukturiranih vrsta podataka. Ovo je zasebno, ali usko vezano područje s računalstvom. Računalstvo u znanost podataka uključuje stvaranje programa i algoritama za obradu podataka, dok znanost podataka sama po sebi usko vezana s matematičkim područjem statistike koje uključuje prikupljanje, organizaciju, analizu i prezentaciju podataka. Isto tako može, ali ne mora uključiti računala (Christensson, 2017). Prema Marutitech.com (bez dat.) razlozi zbog kojih se Python koristi u znanosti podataka su njegova skalabilnost i izbor dodatnih

modula. Za razliku od drugih jezika kao R, Matlab i Stata, Pythonova skalabilnost se pokazala kao velika prednost. Izbor modula, odnosno biblioteka, značajan je čimbenik koji je dao Pythonu prednost. Moduli kao Pandas, StatsModels, NumPy, SciPy i Scikit-Learn dobro su poznati u Pythonovoj zajednici i njihova primjena našla se i u znanosti podataka.

Što se tiče **automatizacije** programski jezici kao Python, Java, C# i Ruby imaju dobru programsku podršku za automatizaciju zadataka (Sheth, 2020). Automatizacija je stvaranje i primjena tehnologija za proizvodnju i isporuku robe i usluga uz minimalnu ljudsku intervenciju. Implementacija automatizacije dokazano povećava učinkovitost, pouzdanost i brzinu zadataka koje su prije izvršavali ljudi. U informatičkoj domeni, najbolji primjer automatizacije je skripta za testiranje rada softvera i kreiranje izvještaja rada. Jedini zadatak ljudi je definirati proces i konfigurirati skriptu. Jedan od naprednijih primjera automatizacije su samo vozeći automobili koji nude mogućnost auto pilota (Techopedia, 2021). Automatizacija pomoću jezika Python u čestoj je uporabi najprije zbog jednostavne sintakse. Najčešća primjene automatizacije pomoću Pythona su sljedeće (Abi-Nakhoul, bez dat.):

- Slanje, sortiranje i odgovor na elektroničnu poštu
- Ispunjavanje PDF i Excel datoteka
- Slanje HTTP zahtjeva
- Rad sa slikama
- Izvođenje matematičkih jednadžbi
- Izračuni financijskih pokazatelja
- Filtriranje podataka s web stranica i njihovo spremanje

Najčešće korišteni alati, odnosno moduli za automatizaciju su *PyAutoGUI*, *Selenium, BeautifulSoup, Pandas* (Nimkar, 2021).

PyAutoGUI omogućuje pomoću Python skripte kontrolu miša i tipkovnice za automatizaciju interakcije s drugim aplikacijama. Ovaj modul radi na operacijskim sustavima Windows i Linux. Značajke ovog modula su mogućnost pomicanje i klik miša, slanje pritiska tipki tipkovnice aplikacijama, npr. za ispunjavanje obrazaca i izrada snimke zaslona (Sweigart, 2020).

Selenium je alat otvorenog koda koji se koristi kod automatizacije rada u web pregledniku. Ima jedno sučelje koje omogućuje izradu skripti ne samo u Pythonu, već u jezicima Java, NodeJS, PHP, Perl i C#. Vrste testiranja koja se mogu izvesti pomoću ovog alata su testiranje kompatibilnosti, performansi, integracije, sistema i regresije. Ovaj alat koriste svi kori razvijaju web stranice najprije zbog njegove fleksibilnosti. Selenium je korišten za testiranje stranice bilo koje veličine. U kompanijama razvojni programeri pomoću ovog alata izrađuju detaljno razrađene i vrlo učinkovite automatizirane skripte za testiranje rada svoje stranice najprije zbog velikih mogućnosti i pouzdanosti u ovaj alat (Browserstack, bez dat.).

BeautifulSoup koristi se za raščlanjivanje strukturiranih podataka. Omogućuje interakciju s HTML-om na sličan način na koji komuniciramo s web stranicom. Sadrži nekoliko intuitivnih funkcija za analiziranje HTML koda koji preuzimamo. Ovaj alat potrebno je preuzeti i instalirati jer ne dolazi u sklopu Python instalacije (Breuss, 2021).

Pandas je brz, snažan, fleksibilan i jednostavan alat otvorenog koda korišten za analizu i manipulaciju podataka. Izrada alata počela je 2008. godine i bio je zatvorenog koda. 2009. godine postao je otvorenog koda s ciljem što većeg razvoja i koristi za sve pojedince koji se bave znanošću podataka. Koristi se u raznim akademskim i komercijalnim domenama, uključujući financije, neuroznanost, ekonomiju, statistiku, oglašavanje, web analitiku itd. (Pandas.org, bez dat.). Izdvojene prednosti ovog alata su sljedeće:

- Brz i učinkovit objekt *DataFrame* za manipulaciju podacima s integriranim indeksiranjem
- Sadrži alate za čitanje i pisanje podataka između struktura podataka u memoriji i različitih formata kao što su CSV, tekstualne datoteke, Microsoft Excel, SQL baze podataka i brzi format HDF5
- Inteligentno poravnavanje podataka i integrirano rukovanje podacima koji nedostaju. Uređuje neuredne podatke.
- Fleksibilno preoblikovanje i izmjena skupova podataka
- Vrlo optimiziran za performanse s kritičnim putanjama koda napisanim na Cythonu ili C.

5. Uvod u sintaksu Pythona 3.8.7

U skriptama će se koristiti Python 3.8.7, stoga ću i ovdje objasniti sintaksu za korištenu verziju Pythona. Python je dizajniran za čitljivost i ima mnogo sličnosti sa engleskim jezikom. Koristi novu liniju za završetak naredbe za razliku od drugih jezika koji koriste točku i zarez ili zagrade. Oslanja se na uvlake ili razmake da definira opsege petlji, funkcija i klasa, dok ostali jezici u tu svrhu koriste vitičaste zagrade. (w3schools.com, bez dat.). Sve vrijednosti spremaju se u varijable. Varijable su vrijednosti koje tokom izvođenja programa mogu mijenjati svoje vrijednosti. Varijabli se može pridodati osnovni i složeni tip podatka. Pridruživanje se izvršava pomoću operatora pridruživanja (=).

5.1. Operatori i uspoređivanje vrijednosti

Operatori su specifični simboli koji predstavljaju računske operacije. Prioriteti operacija kod izračuna vrijedi kao u matematici. Operatori koji se javljaju u Pythonu, njihov opis i jednostavan vlastito izrađen primjer prikazani su u sljedećoj tablici.

Tablica 1: Popis operatora (vlastita izrada)

Operator	Opis	Primjer
+	Operator zbrajanja. Koristi se kod zbrajanja brojeva i spajanja stringova.	A = 5 + 3 string = „Python je” + „ super”
-	Operator oduzimanja. Koristi se kod oduzimanja brojeva.	B = 4 - 1
*	Operator množenja. Koristi se kod množenja bojeva.	C = 5*12
/	Operator dijeljenja. Koristi se kod dijeljenja brojeva. Neovisno o tome da li su brojevi cijeli ili decimalni, vraćeni rezultat dijeljenja je realan broj.	D = 20/3 E = 20/4 D ima vrijednost 6.6666... E ima vrijednost 5.0
**	Operator potenciranja. Broj na lijevoj strani operatora je baza, a na desnoj eksponent.	F = 2**5 G = 5**7
//	Operator koji vraća cijeli dio broja.	H = 17 // 3 H ima vrijednost 5
%	Operator koji vraća ostatak dijeljenja	I = 17%3 I ima vrijednost 2

Za uspoređivanje vrijednosti u Pythonu može se koristiti osamoperacija. Operacije „is“ i „is not“ testiraju identitet objekta. Identitet objekta definira se pomoću funkcije id() (Python Software Foundation, 2021).

Tablica 2: Tablica operacija

Operacija	Opis	Primjer
<	Strogo manje od	A < B
<=	Manje ili jednako od	A <= B
>	Strogo veće od	A > B
>=	Veće ili jednako od	A >= B
==	Jednako	A == B
!=	Nije jednako	A != B
is	Testira identitet objekta	A is B
is not	Testira identitet objekta	A is not B

5.2. Ugrađeni tipovi podataka i prikladne funkcije

U ovom poglavlju će se opisati tipovi podataka i funkcije koje se najčešće koriste s tim tipom podatka.

5.2.1. Brojevi

Brojevi mogu biti cijeli i realni te se spremaju u varijable na sljedeće načine:

```
a = 5                   # a=5
b,c,d = 6,7,8           # b=6, c=7, d=8
e = int(43234)          # e=43234
f = int(133432.45)      # f=133432, decimalni dio se odbacuje.
zbroj = a + b + c       # zbroj=18
razlika = a - b - c     # razlika = -8
#Decimalni brojevi
dec1 = 4.5              # dec1 = 4.5
dec2 = float(5)         # dec2 = 5.0
dec3 = zbroj/b          # dec3 = 3.0
```

```
pot1 = 2**4             # pot1 = 8
cijeli = 17//3          # ost1 = 5
decimalni = 17%3        # decimalni = 2
```

Prikladne ugrađene funkcije koje je najbitnije spomenuti vrijede i za cijele i za realne brojeve, a to su sljedeće:

```
abs(-10.1)              # vraća apsolutnu vrijednost
pow(3, 9)               # vraća potenciju, radi isto kao i 3**9
max(2,6,4,8,5,3)        # vraća najveći broj
min(2,6,4,8,5,3)        # vraća najmanji broj
```

5.2.2. Znakovni niz

Znakovni niz (*eng. string*) je tip podatka koji sadrži niz znakova. Postoji dva načina na koji se može deklarirati znakovni niz, a to su pomoću jednostrukih navodnika, dvostrukih navodnika. (w3schools.com, bez dat.)

```
znakovni_niz = 'Bok svima'
znakovni_niz = „Bok svima”
znakovni_niz = str(„Bok, ja sam string koji sadrzi brojeve 35 i 22”)
```

Znakovni niz za koji želimo da se nalazi u više redaka možemo deklarirati na sljedeće načine (w3schools.com, bez dat.):

```
string = „””Lijep pozdrav,
ovaj string nalazi se u više redova.
To se može dobiti dodavanjem triju dvostrukih navodnika,
ili na način da dodamo delimiter.”””
string = „Ovaj string će se nalaziti\n u više redova \n jer smo
dodali delimiter.”
```

Znakovni nizovi mogu se spajati, odnosno spojiti pomoću operatora zbrajanja (+). Ako izostavimo operator, znakovni nizovi će se i dalje spojiti. Isto tako možemo pristupiti pojedinom slovu u znakovnom nizu (Python Software Foundation, 2021).

```
Spajanje = „Python automatski spaja” „ u jedan string”
Spajanje = „Python automatski spaja” + „ u jedan string”
Slovo = Spajanje[1] #Slovo ima vrijednost 'y'
```

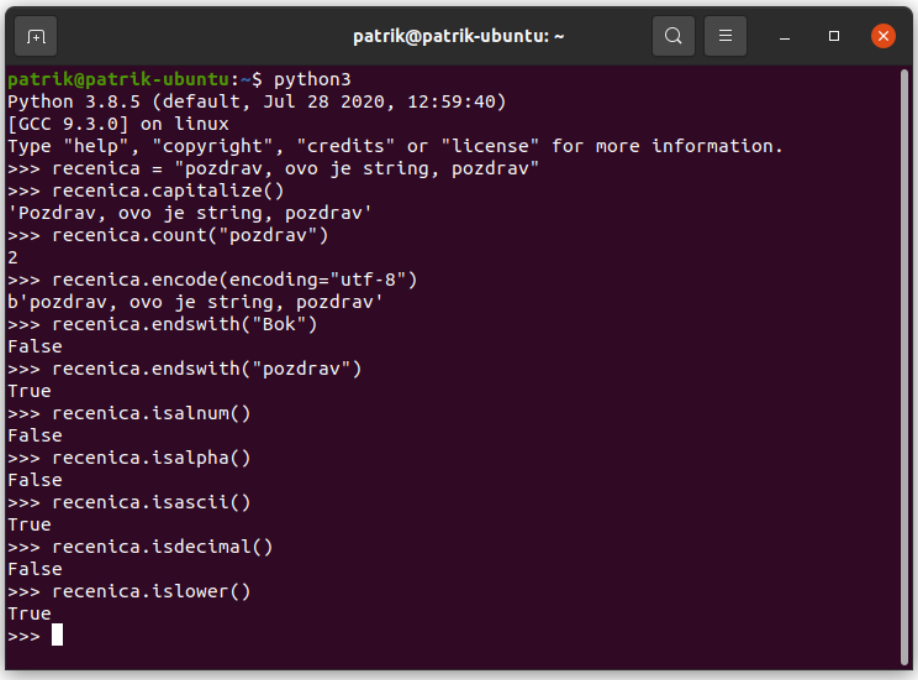
Postoji veliki broj korisnih funkcija za manipulaciju znakovnim nizovima. Ove funkcije uvelike olakšavaju rad sa znakovnim nizovima zbog njihove vrlo jednostavne uporabe i jasno definiranim povratnim vrijednostima. Pošto postoji velik broj funkcija, navest ćemo one koje se najčešće koriste. Funkcije se nalaze u sljedećoj vlastito izrađenoj tablici. Imena metoda i njihov opis preuzete su iz službene dokumentacije (Python Software Foundation, 2021):

Tablica 3: Metode za manipulaciju znakovnih nizova (vlastita izrada)

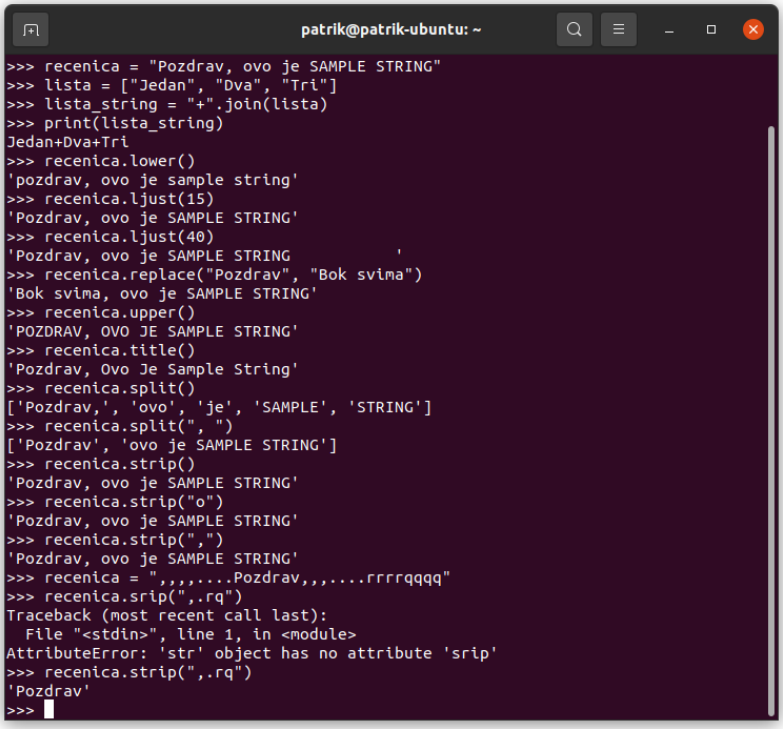
Naziv funkcije	**Opis**
capitalize()	Vraća kopiju znakovnog niza u kojem je prvo slovo veliko, a ostala slova mala.
count(sub[, start[, end]])	Vraća broj pojava koja se ne poklapaju u podnizu *sub* u rasponu *[start, end]*.
encode(encoding="utf-8", errors="strict")	Vraća kodiranu verziju kao objekt bajtova. Zadano kodiranje je utf-8. Može se i dati vrsta greške radi postavljanje drugačijeg načina rukovanja greškama. Zadana vrijednost za pogreške je *„strict”*, što znači da greške u kodiranju uzrokuju UnicodeError
endswith(suffix[, start[, end]])	Vraća *True* ako znakovni niz završava sa zadanim sufiksom, inače vraća *False*.
isalnum()	Vraća *True* ako su svi znakovi u nizu alfanumerički znakovi i mora biti barem jedan znak, inače vraća *False*.
isalpha()	Vraća *True* ako su svi znakovi u nizu abecedni i ako postoji barem jedan znak, inače vraća *False*.
isascii()	Vraća *True* ako su svi znakovi u nizu ASCII znakovi, inače vraća *False*.
isdecimal()	Vraća *True* ako su svi znakovi u nizu decimalni znakovi i ako postoji barem jedan znak, inače vraća *False.*
islower()	Vraća *True* ako su svi znakovi u nizu mala slova i postoji barem jedan znak, inače vraća *False*.
join(iterable)	Vraća znakovni niz koji je spoj riječi. Iterable je argument kojim spajamo riječi, to može biti vlastito definiran znak.
lower()	Vraća znakovni niz koji sadrži samo mala slova.
ljust(width[, fillchar])	Vraća niz poravnat lijevo u znakovnom nizu duljine širine. Padding se vrši pomoću navedenog *fillchar* (zadani je ASCII razmak).

replace(old, new[, count])	Vraća znakovni niz sa svim pojavama podniza *old* zamijenjen s *new*. Ako je dan neobavezni broj argumenata, zamjenjuju se samo prva pojavljivanja broja.
split(sep=None, maxsplit=-1)	Vraća listu riječi iz znakovnog niza, koristeći *sep* kao graničnik (*eng. delimiter)*. Ako je zadan *maxsplit*, izvodi se najviše *maxsplit* podjela (dakle, popis će imati najviše *maxsplit* + 1 elemenata). Ako *maxsplit* nije naveden ili -1, tada nema ograničenja broja dijeljenja (napravljena su sva moguća dijeljenja).
strip([chars])	Vraća znakovni niz s uklonjenim vodećim i završnim znakovima. Argument *chars* je niz koji specificira skup znakova koje treba ukloniti. Ako je izostavljeno ili *None*, argument *chars* zadani je za uklanjanje razmaka. Argument chars nije prefiks ili sufiks; nego su uklonjene sve kombinacije njegovih vrijednosti.
upper()	Vraća znakovni niz u kojem su svi znakovi pretvoreni u velika slova.
title()	Vraća znakovni niz u kojem je svaka riječ ima veliko početno slovo.

Na sljedećim slikama vidi se rad metoda iz tablice.



Slika 5: Rad metoda za manipulaciju znakovnih nizova (vlastita izrada)



Slika 6: Rad metoda za manipulaciju znakovnih nizova 2 (vlastita izrada)

5.2.3. Sekvencijalni (slijedni) tipovi

Sekvencijalni tipovi su liste, N-torke i rasponi (*eng. list, tuple, range*.) Zovemo ih slijednim tipovima zato što njihovim elementima možemo pristupiti rednim brojem ili indeksom, a prvi element počinje brojem 0. Prednosti ovakvih tipova podataka su jednostavnost i brzina dohvaćanja podataka. Pomoću petlji lako se može doći do određenog podatka (Kalafatić i sur., 2016).

Rasponi (*eng. range*) su nepromjenjivi i brojivi tip podatka. U rasponima se najčešće predstavljaju nizovi cijelih brojeva i njihova primjena vrlo je česta kod uporaba petlji. Kod raspona unaprijed znamo sve elemente (Kalafatić i sur., 2016). Prilikom stvaranja raspona možemo dodati najviše 3 argumenta, to su početak, kraj i korak. Početak je vrijednost od kojeg raspon počinje, a standardna vrijednost je 0. Kraj je vrijednost do koje raspon dolazi i to je jedini obavezan argument. Korak predstavlja svaku n-tu vrijednost od početka i ne smije biti 0. Bitno je naglasiti da je interval vrijednosti [Početak, Kraj> (Python Software Foundation, 2021).

```
range(5) # brojevi od 0 do 4
range(10,3) # kriva deklaracija
range(10,3,-1) # dobra deklaracija
```

Rasponi olakšavaju primjerice stvaranje liste. Bez raspona morali bi stvoriti novu petlju i nekoliko puta pozvati metodu *append(),* dok s rasponima možemo to obaviti u jednoj liniji s jednim pozivom.

```
lista = list(range(1,11,1))   # u listi se nalaze brojevi od 1 do 10
```

N-torke su nepromjenjiv tip sekvence i obično se koriste za pohranu zbirki heterogenih podataka. Također se koriste u slučajevima kada je potreban nepromjenjivi slijed homogenih podataka kao što je omogućavanje pohrane u skupu ili instanci rječnika (Python Software Foundation, 2021). N-torke imaju raspoređene vrijednosti, što znači da je definiran redoslijed zapisa i on se ne može mijenjati. Sadržaj n-torki ne može se mijenjati nakon stvaranja i dozvoljeni su duplikati vrijednosti (w3schools.com, bez dat.). N-torke se mogu konstruirati na sljedeće načine (Python Software Foundation, 2021):

- Upotrebom zagrada za označavanje prazne n-torke: ()
- Uporabom jednog zareza nakon n-torke samo s jednom vrijednošću: a, ili (a,)
- Podjelom elemenata sa zarezom: a, b, c ili (a, b, c)
- Korištenjem tuple() konstruktora: tuple((„Jedan”, „Dva”, „Tri”))

Liste su tip podatka koje ima raspoređene vrijednosti, što znači da se raspored vrijednosti ne mijenja, osim s određenim funkcijama. Vrijednosti liste mogu se mijenjati i dopuštaju duplicirane vrijednosti, što znači da se na različitim indeksima, odnosno mjestima liste može pojaviti ista vrijednost (w3schools.com, bez dat.). Vrijednosti u listi odvajaju se zarezom. Vrijednosti u listi ne moraju biti isti tip podatka, ali je praksa da se u liste stavlja isti tip (Python Software Foundation, 2021). Liste se mogu stvoriti na sljedeće načine:

```
lista = [„Prvi element”, „Drugi element”, 3, True, 25.4]
lista = list((„Prvi element”, „Drugi element”, Treci element”))
```

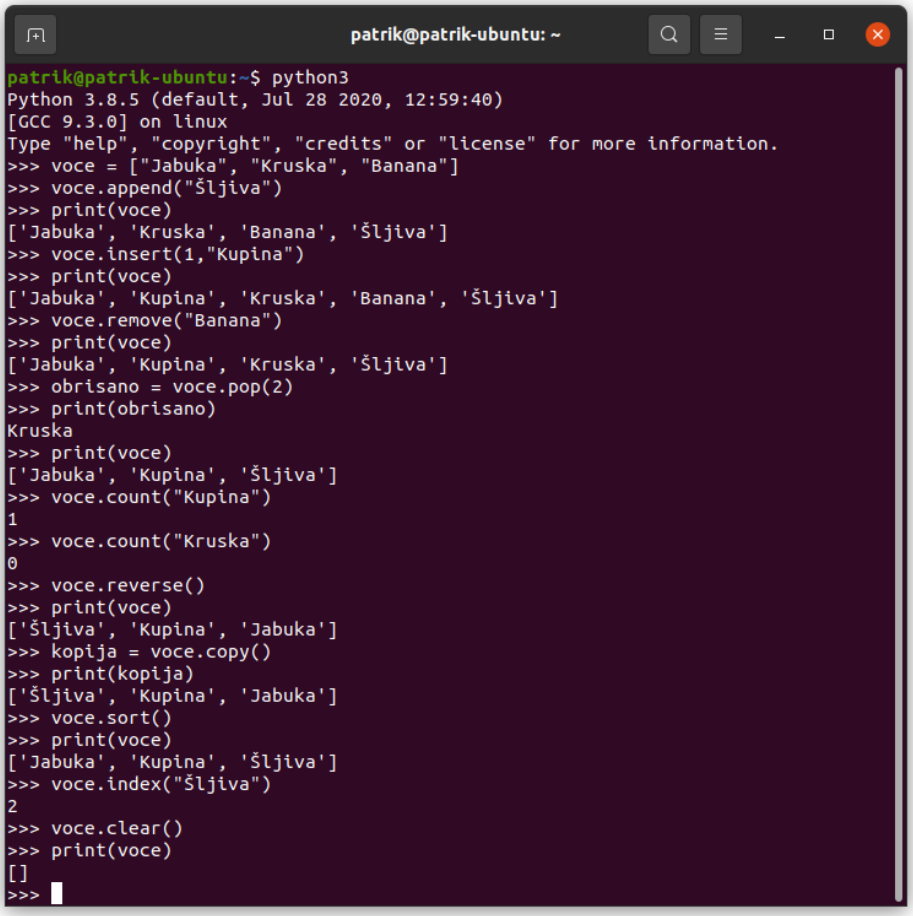
Listama možemo manipulirati sa metodama definiranim u sljedećoj tablici (Python Software Foundation, 2021)

Tablica 4: Metode za rad s listama (vlastita izrada)

Metoda	Opis
append(x)	Dodaj jednu vrijednost na kraj liste
extend(iterable)	Dodaj više vrijednosti na kraj liste

insert(i, x)	Dodaj element na specifično mjesto. Sve vrijednosti ispred tog elementa se pomiću za jedan.
Remove(x)	Izbriši prvu pojavu vrijednosti x iz liste.
Pop(i)	Izbriši vrijednost na poziciji *i* i vrati ju. Ako vrijednost *i* nije zadana, briše se zadnji element liste.
Clear()	Briše sve vrijednosti iz liste.
Indeks(x[, start[, end]])	Vraća poziciju vrijednosti x u listi.
Count(x)	Vraća broj ponavljanja vrijednosti x u listi.
Sort()	Uzlazno sortira listu
reverse()	Preokreće elemente u listi.
Copy()	Vraća kopiju liste

Sljedeća slika prikazuje rad navedenih metoda.



Slika 7: Rad metoda vezanih uz liste. (vlastita izrada)

5.2.4. Boolean

Boolean je tip podatka koji predstavlja dvije vrijednosti *True* i *Flase.* Ove dvije vrijednosti mogu se prikazati i kao brojevi 1 i 0, respektabilno. Ovaj tip podatka ključan je kod tokova izvođenja programa. Pomoću boolean-a može se evaluirati bilo koji izraz u Pythonu. Boolean tip podatka vidljiv je kod metoda za s*tring* podatke. Metode kao *isalnum()* i *isalpha()* vraćaju boolean tip podatka.

Vrijednost *True* imat će svaki objekt koji ima spremljenu bilo kakvu vrijednost ili neku vrstu sadržaja. Na sljedećoj slici je to i demonstrirano.

```
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> bool("abc")
True
>>> bool(123)
True
>>> bool(["jabuka", "kruška"])
True
>>> bool(15>11)
True
>>>
>>> bool(("vrijednost1", 2, 3))
True
>>> bool({"kljuc": "vrijednost"})
True
>>>
```

Slika 8: Bool vrijednosti 1 (vlastita izrada)

Vrijednost *False* imat će svaki objekt koji nema ništa spremljeno. Na sljedećoj slici je to i demonstrirano.

```
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> bool(False)
False
>>> bool(None)
False
>>> bool(0)
False
>>> bool("")
False
>>> bool(())
False
>>> bool([])
False
>>> bool({})
False
>>>
```

Slika 9: Bool vrijednosti 2 (vlastita izrada)

Korist booleana kod funkcija i tokova izvođenja programa će biti objašnjeni u tim poglavljima.

5.2.5. Skup

Python također uključuje tip podatka **skup.** Skup je neraspoređena kolekcija bez dupliciranih elemenata. Osnovne namjene uključuju testiranje članstva i uklanjanje dvostrukih unosa. Postavljeni objekti također podržavaju matematičke operacije poput spajanja, presijecanja, razlike i simetrične razlike (Python Software Foundation, 2021).

Skupovi se mogu deklarirati na 3 načina:

```
noviSkup = {"vrijednost1", "vrijednost2", "Banana"}
noviSkup = set("jabuka")
noviSkup = {c for c in "banana" if c noti n 'n'}
```

Sljedeća tablica prikazuje metode za rad s skupovima.

Tablica 5: Metode za rad s skupovima

Metoda	Opis
add()	Dodaje element u skup
clear()	Briše sve elemente iz skupa
copy()	Vraća kopiju skupa
difference()	Vraća set koji sadrži razliku dvaju ili više setova
difference_update()	Briše elemente u odabranom skupu koji se nalaze u drugom specificiranom skupu
discard()	Briše specificiran element
intersection()	Vraća set koji je presjek dvaju ili više setova
intersection_update()	Briše elemente u skupu koji nisu prisutni u drugim specificiranim skupovima
pop()	Briše nasumičan element iz skupa. Ova metoda može i izmijeniti raspored elemenata u skupu.
remove()	Briše specificiran element iz skupa
union()	Vraća skup koji je unija skupova.
update()	Dopunjava skup unijom sa drugim skupom

(Izvor: w3schools.com, bez dat.)

5.2.6.Rječnici

Rječnici se u drugim programskim jezicima kao asocijativni nizovi. Za razliku od sekvenci koje se indeksiraju brojevima, rječnici se indeksiraju ključevima koji mogu biti bilo koja nepromjenjiva (eng. *immutable*) vrsta podatka. Rječnik je najbolje zamisliti kao skup

vrijednosti u odnosu *ključ:vrijednost*. (Python Software Foundation, 2021). Rječnici su raspoređeni što znači da se raspored ključeva ne može mijenjati, vrijednosti ključeva mogu se mijenjati i ne dozvoljavaju duplicirane vrijednosti. Najčešća praksa imenovanja ključa je pomoću znakovnog niza ili broja. Vrijednosti dodane ključu mogu biti bilo koji tip podatka (w3schools.com, bez dat.). Rječnici se mogu deklarirati na sljedeće načine (Python Software Foundation, 2021):

```
Rjecnik = {"Brand": "Ford", "Model": "Mustang", "Godina": 1964}
Rjecnik = dict(a=100, b=200)        #a i b su ključevi tipa string
Rjecnik = {x: x**2 for x in range(100)}
```

Sljedeća tablica prikazuje metode vezane uz rad s rječnicima.

Tablica 6: Metode vezane uz rječnike.

Metoda	**Opis**	**Primjer**
clear()	Briše sve elemente iz rječnika. Rječnik postaje prazan.	Rjecnik.clear()
copy()	Vraća kopiju rječnika.	Kopija = Rjecnik.copy()
fromkeys()	Vraća rječnik sa specificiranim ključevima i njihovim pridodanim vrijednostima.	Kopija = Rjecnik.fromkeys(“kljuc1“, “kljuc2“)
get()	Vraća vrijednost koja je pridodana ključu.	Model = Auto.get(“model“)
items()	Vraća listu čiji su elementi n-torke u obliku ključ vrijednost. Kako bi mogli dohvatiti te podatke potrebno ih je spremiti u listu.	Elementi = list(Auto.items())
keys()	Vraća listu koja sadrži sve ključeve rječnika. Kako bi mogli dohvatiti te podatke potrebno ih je spremiti u listu.	Kljucevi = list(Rjecnik.keys())
pop()	Briše element sa specificiranim ključem.	Rjecnik.pop(“Model“)
popitem()	Briše zadnje unesen element.	Rjecnik.popitem()
setdefault()	Vraća vrijednost pridodanu specificiranom ključu. Ako ključ ne postoji, dodaje se ključ s prikladnom vrijednošću. Ako vrijednost nije specificirana, ključu se pridodaje None	Rjecnik.setdefault(“Model“, *vrijednost*)

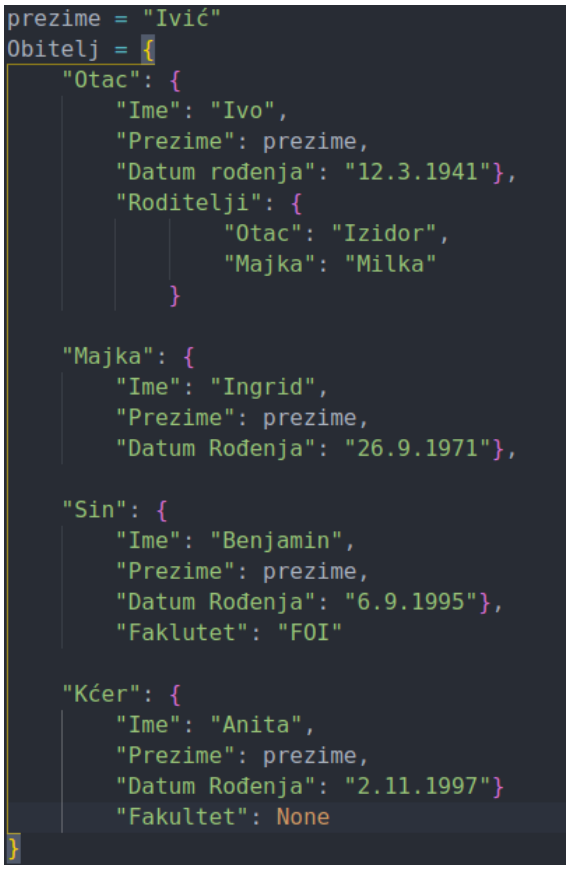
update()	Ažurira rječnik sa specificiranim ključem i vrijednošću. Ako ključ već postoji u rječniku, ažurirat će njegovu vrijednost. Ako ključ ne postoji, dodat će ga s vrijednošću	Rjecnik.update({“boja“: “bijela“})
values()	Vraća listu svih vrijednosti koje su pridodane ključevima.	Vrijednosti = list(Rjecnik.values())

Sljedeća slika prikazuje rad navedenih metoda.

```
kopija = rjecnik.copy()
{'Brand': 'Ford', 'Model': 'Mustang', 'Godina': 1964, 'Boja': 'Bijela'}

kljuc = rjecnik.get("Godina")
1964

Sadrzaj = list(rjecnik.items())
[('Brand', 'Ford'), ('Model', 'Mustang'), ('Godina', 1964), ('Boja', 'Bijela')]

Kljucevi = Rjecnik.keys()
['Brand', 'Model', 'Godina', 'Boja']

rjecnik.pop("Model")
{'Brand': 'Ford', 'Model': 'Mustang', 'Boja': 'Bijela'}

rjecnik.popitem()
{'Brand': 'Ford', 'Model': 'Mustang'}

rjecnik.setdefault("Vrijednost")
{'Brand': 'Ford', 'Model': 'Mustang', 'Vrijednost': None}
rjecnik.setdefault("Vrijednost", 342145)
None
rjecnik.setdefault("Boja", "Crna")
Crna

rjecnik.update({"Model": "Focus"})
{'Brand': 'Ford', 'Model': 'Focus', 'Vrijednost': None, 'Boja': 'Crna'}
rjecnik.update(Broj sjedala: 5)
{'Brand': 'Ford', 'Model': 'Focus', 'Vrijednost': None, 'Boja': 'Crna', 'Broj sjedala': 5}

vrijednosti = list(rjecnik.values())
['Ford', 'Focus', None, 'Crna', 5]

rjecnik.clear()
{}
```

Slika 10: Rad metoda rječnika (vlastita izrada).

Rječnici mogu biti ugniježđeni, što znači da ključu može biti pridodan vrijednost tipa rječnik. Primjer ugniježđenog rječnika nalazi se na sljedećoj slici.



Slika 11: Primjer ugniježđenih rječnika (vlastita izrada)

Kako bi ispisali općenite podatke o ocu, moramo unijeti sljedeće:

```
print( Obitelj[“Otac“] )
```

Ispisat će se sljedeće:

```
{'Ime': 'Ivo', 'Prezime': 'Ivić', 'Datum rođenja': '12.3.1941',
'Roditelji': {'Otac': 'Izidor', 'Majka': 'Milka'}}
```

Ako želimo doznati neki konkretni podatak o ocu, primjerice datum rođenja ili ime njegovog oca moramo unijeti sljedeće:

```
print(Obitelj["Otac"]["Datum rođenja"])      #ispis: 12.3.1941
print(Obitelj["Otac"]["Roditelji"]["Otac"])  #ispis: Izidor
```

5.3. Tokovi izvođenja programa (grananja, iteracije)

Tok izvođenja programa je slijed kojim se neki program izvodi. Taj tok može biti definiran pomoću grananja i iteracija. U Pythonu se za grananje koristi *if-else* zraz. Za iteracije koriste se *while* i *for* petlje.

If-else izraz koristi se za grananje. Postoji više varijacija koje je moguće implementirati. Njegova sintaksa je sljedeća (Python Software Foundation, 2021):

```
if logički_izraz_1:
      Blok_naredbi_1
elif logički_izraz_2:
      Blok_naredbi_2
elif logički_izraz_3:
      Blok_naredbi_3
else:
      Blok_naredbi_4
```

Grananje mora početi izrazom *if.* Svaki izraz uz sebe ima logički izraz koji može biti bilo što. Logički izraz je zapravo provjera istinitosti tog izraza. U logički izraz možemo pozvati funkciju koja će vratiti neku vrijednost i na temelju te vrijednosti odrediti tok izvođenja, dok isto tako možemo staviti bilo kakvu jednostavnu ili složenu usporedbu. Izrazi *elif i else* nisu obavezni. Broj *elif* izraza nema ograničenja Pravila za logičke izraze kod *elif*a su ista kao i za *if* izraz. Blok naredbi *else* izvodi se jedino ako svi prethodni logički izrazi nisu istiniti. Grananja mogu biti i ugnježđena, što znači da imamo grananje u grananju.

Iteracije koje se pojavljuju su *while* i *for* iteracije. ***While*** petlja vrlo je jednostavna. Prije početka nove iteracije provjerava zadani uvjet. Ako je uvjet, odnosno logički izraz istinit izvršava se blok naredbi unutar te petlje, a ako je lažan blok naredbi unutar petlje se preskače i izvodi se prva linija koda iza tog bloka. U logičkom izrazu može biti bilo što. Sintaksa je vrlo jednostavna i glasi ovako:

```
while logički_izraz:
      Blok naredbi
```

For petlja najčešće se koristi za iteracije u sekvencijalnim tipovima podataka. Ona je specifična u odnosu na druge programske jezike zato što nema iterator i uvjet izvođenja kao kod C++-a. Sintaksa ove petlje je sljedeća:

```
for vrijednost in sekvenca:
      blok naredbi
```

Za primjer uzmimo ispis parnih brojeva od 1 do 1000. Funkcija *range()* vraća tražen niz brojeva.

```
for broj in range(0, 1001):
      print(broj)
```

Svaku od petlji možemo prisilno prekinuti ili prisilno započeti novi ciklus. Za prekid petlje koristi se ključna riječ ***break***. Njezina praktična primjena je traženje neke vrijednosti u sekvencijalnom tipu podatka. Kada je nađena vrijednost nema smisla nastaviti iteraciju petlje zato što je tražena vrijednost nađena. Daljnje odvijanje petlje samo usporava rad programa. Ključna riječ *continue* koristi se kada znamo da za neki slučaj u sekvenci nećemo naći potrebnu vrijednost. Primjerice ako u listi brojeva tražimo proste brojeve znamo da brojevi djeljivi s 2 neće biti prosti. Kada u petlji dođe broj djeljiv s 2 jednostavno započinjemo sljedeću iteraciju pomoću *continue*.

Postoji i **for-else** petlja. Kod *for-else* petlje postoji blok naredbi *else* koji se izvede ako je rad petlje završen bez nasilnog prekida, U sljedećem primjeru ako bi u programskom kodu maknuli komentare sa uvjeta i naredbe *break*, ispis u bloku naredbi *else* ne bi se dogodio.

```
for broj in range(0, 1001):
      print(broj)
      #if broj == 500:
      #     break
else:
      print("Gotov!")
```

5.4. Funkcije

Funkcija je programski konstrukt čiji se blok naredbi izvršava njenim pozivom. Svrha funkcija je obavljanje određenog posla u sklopu kompleksnog algoritma, odnosno programskog rješenja. Funkcije mogu, ali ne moraju vratiti vrijednost. U velikoj većini ostalih objektno orijentiranih programskih jezika kao C#, potrebno je definirati koji tip podatka će funkcija vratiti.

```
public int funkcija(){
      return 5*5;
}
```

U Pythonu to nije slučaj. Za definiranje funkcija u Pythonu koristi se ključna riječ ***def***. Nakon ključne riječi stavlja se ***ime funkcije***. Nakon imena funkcije pišu se ***uglate zagrade*** u koje se upisuju ***argumenti funkcije*** koji nisu obavezni. Argumenti funkcije postat će lokalna varijabla i moći će se koristiti samo u funkciji. Nakon uglatih zagrada slijedi dvotočka (:) . Sljedeća linija koda mora biti uvučena. Ako želimo da funkcija vrati vrijednost potrebno je upisati ključnu riječ ***return*** i pokraj nje izraz. Kada izvođenje funkcije dođe do riječi *return*, funkcija završava svoj rad (Python Software Foundation, 2021). Programer mora pripaziti da se u funkciji ne bi nalazio kod koji se nikad ne izvršava. Primjer deklaracije funkcije i njezin poziv vidljivi su na sljedećoj slici.

```
1
2   def Usporedi(argument_1, argument_2):
3       print("Funkcija pozvana!")
4
5       if argument_1 == argument_2:
6           return "Brojevi su jednaki"
7       elif argument_1 > argument_2:
8           return f"{argument_1} je veći od {argument_2}"
9       else:
10          return f"{argument_1} je manji od {argument_2}"
11
12      print("Ova poruka se nikada neće ispisati")
13
14  x = int(input("Unesi prvi broj: "))
15  y = int(input("Unesi drugi broj: "))
16  stanje = Usporedi(x, y)
17  print(stanje)
```

Slika 12: Deklaracija funkcije (vlastita izrada)

Naziv funkcije je *Usporedi* i argumenti funkcije su *argument_1, argument_2*. Sadržaj funkcije prostire se od linije broj 3 do linije broj 12, zbog razine uvlaka. U ovoj funkciji naredba *print()* koja se nalazi na liniji 12 nikada se neće izvršiti iako se nalazi u opsegu funkcije. Razlog tome su ključne riječi *return*. Kod uspoređivanja brojeva obuhvaćena su sva 3 moguća uvjeta. Sve tri ključne riječi *return* nalaze se prije ispisa poruke, stoga se ispis nikada neće dogoditi. Ako bi smo tu poruku izbacili iz funkcije, odnosno maknuli jednu razinu uvlake, poruka bi se ispisala prije unosa brojeva od strane korisnika. Isto tako, opseg funkcije *Usporedi* se smanji od linije 3 do linije 10. Funkcija se poziva na liniji 16. Vidljivo je da su proslijeđena 2 argumenta *x* i *y*. Argumentu *Argument*_1 dodijeljena je vrijednost *x,* dok je argumentu *Argument_2* dodijeljena vrijednost *y.* Vraćena vrijednost funkcije sprema se u varijablu *stanje*. Bitno je napomenuti da za je **za rad ove funkcije potrebno unijeti sve argumente. Raspored argumenata prilikom poziva funkcije može utjecati na rad i rezultat funkcije, stoga treba paziti kako se funkcija deklarira i poziva**.

Što se tiče argumenata funkcija postoje i **zadani argumenti funkcije**. Zadani argumenti funkcije omogućuju da se funkcija pozove s manje argumenata nego što je dopušteno. Oni su ujedno i **Argumenti ključnih riječi** (eng. *keyword argument*) (Python Software Foundation, 2021). U prošlom primjeru vidjeli smo da je bilo potrebno unijeti sve argumente, odnosno maksimalan broj argumenata. Uzmimo za primjer igru pogađanja brojeva. Sljedeća slika prikazuje programsko rješenje igre pogađanja brojeva.

```
1  import random
2
3  def PogodiBroj(broj, brojPokušaja=3, poruka="Netočno!"):
4      #nasumicni broj dobivamo pomoću funkcije randint iz modula random
5      nasumicanBroj = random.randint(1, broj)
6
7      #sve dok ima pokusaja
8      while brojPokušaja > 0:
9          unos = int(input("Unesi broj: "))
10
11         #slucaj krivog unosa
12         if unos != nasumicanBroj:
13             brojPokušaja -= 1
14             print(poruka)
15         #slucaj tocnog unosa
16         else:
17             print("Broj pogođen")
18             break
19
20     if brojPokušaja == 0:
21         print(f"Ostali ste bez pokušaja. Nasumičan broj bio je {nasumicanBroj}")
22
23
24 #funkcija se može pozvati na sljedeće načine:
25 PogodiBroj(10)
26 PogodiBroj(20,5)
27 PogodiBroj(5,1,"Krivo ti je!")
28
29 PogodiBroj(brojPokušaja=5, poruka="Krivo!", broj=15)
30 PogodiBroj(poruka="Ne!", broj=10, brojPokušaja=2)
31 PogodiBroj(5, poruka="Krivo ti je")
32 PogodiBroj(broj=5, poruka="Krivo ti je!")
33 PogodiBroj(broj=15)
34 PogodiBroj(brojPokušaja=4, poruka="Krivo!")     #GREŠKA!
35
```

Slika 13: Igra pogađanja brojeva (vlastita izrada)

Za razliku od funkcije iz prošlog primjera, ovdje imamo zadane argumente funkcije. To su konkretno argumenti *brojPokušaja* i *poruka*. To je vidljivo zato što u liniji 3 gdje se nalazi deklaracija funkcije navedeni argumenti pokraj sebe imaju dodijeljene vrijednosti. Bitno je uočiti da **prvi argument *broj* pokraj sebe nema dodijeljenu vrijednost**. To znači da će biti potrebno unijeti barem jedan argument kod pozivanja funkcije, što je vidljivo u liniji 25. Njega ćemo nazvati **obavezan argument**. Vidljivo je da su sve deklaracije pravilne osim zadnje na liniji 34. Razlog tome je nedostatak obaveznog argumenta *broj*. Ako postoje argumenti ključnih riječi i njima želimo dodijeliti vrijednost različitu od zadane, možemo to učiniti na bilo kojoj poziciji. Bitno je napomenuti da ako se argument ključne riječi nalazi ispred obaveznog parametra, u ovom slučaju su to linije 29,30,31 potrebno je upisati i obavezan parametar.

Proizvoljni argumenti (eng. *arbitraty arguments, *args)* se koriste kada ne znamo maksimalan broj argumenata koji će se proslijediti funkciji. Prilikom deklaracije funkcije proizvoljnom argumentu ispred imena dodajemo znak * (w3schools.com, bez dat.). Najjednostavniji primjer je suma N brojeva.

```
1  def Zbroj(operacija, *brojevi):
2
3      if operacija == "zbrajanje":
4          suma = 0
5          for i in brojevi:
6              suma += i
7          return suma
8
9      else:
10         print("Prvi argument je broj!")
11
12
13 print(Zbroj("zbrajanje", 5,2,3,8,6))
14 print(Zbroj(3,5,2,1))
15
```

Slika 14: Sumiranje nepoznatog broja brojeva (vlastita izrada)

U ovom primjeru imamo jedan obavezan argument *operacija* i proizvoljan argument *brojevi,* što je vidljivo po znaku * ispred njegovog imena. **Tip podatka argumenta *brojevi* je n-torka,** što znači da se može pristupiti vrijednostima n-torke pomoću indeksa. Za uspješno pozivanje funkcije biti će potrebno unijeti barem jedan argument, a to je *operacija*. Ishod poziva funkcije na liniji 13 rezultirat će ispisom sume brojeva 5,2,3,8,6. Ishod poziva funkcije na liniji 13 rezultirat će ispisom poruke „Prvi argument je broj!“ i ispisom *None* zato što funkcija nije vratila bilo kakvu vrijednost. Argument *operacija* ima vrijednost 3, dok je argument *brojevi* n-torka s

vrijednostima 5,2,1. U praksi se ovakva implementacija mora izbjegavati i radi se na potpuno drugačiji način, ali za demonstraciju proizvoljnih argumenata ovaj je primjer dobar.

Rekurzivna funkcija je funkcija koja poziva samu sebe tijekom izvođenja. Omogućuje programerima pisanje učinkovitih algoritama koristeći manju količinu programskog koda. Kod rekurzije postovi mnogo veća šansa da se dogodi beskonačna petlja, odnosno da funkcija nikad ne prestane izvršavati samu sebe, stoga programer mora paziti kako barata s rekurzivnim funkcijama (Christensson, 2020). Klasičan i najjednostavniji primjer rekurzije je izračun faktorijela. Faktorijela je pozitivan cijeli broj nastao množenjem nekog broja sa svim njegovim pozitivnim cjelobrojnim prethodnicima.

```
1  def faktorijela(x):
2
3      if x == 1:
4          return 1
5      else:
6          return (x * faktorijela(x-1))
7
8
9  broj = int(input("Unesite broj: "))
10 print(f"Faktorijela broja {broj} je {faktorijela(broj)}")
```

Slika 15: Rekurzija funkcija (vlastita izrada)

Korisnik mora unijeti cijeli broj. Poziva se funkcija faktorijela. Prvi uvjet provjerava ako je *x* jednak 1, što znači da smo došli do kraja računskog postupka. Ako *x* nije 1, funkcija množi *x* sa svojom rekurzivnom vrijednošću.

5.5. Klase

Klase programskog jeziku Pythonu su njegova srž. Python je objektno-orijentirani jezik što znači da se velika većina programskih rješenja temelji na klasama.

Klase pružaju mogućnost povezivanja podataka i funkcionalnosti zajedno. Stvaranjem nove klase stvara se nova vrsta objekta, dopuštajući izradu novih instanci te vrste. Svaka instanca klase može imati pridružene atribute za održavanje svog stanja. Instance klase također mogu imati metode (definirane svojom klasom) za mijenjanje svog stanja (Python Software Foundation, 2021).

Opća sintaksa definiranja klasa je sljedeća:

```
class MojaKlasa():
```

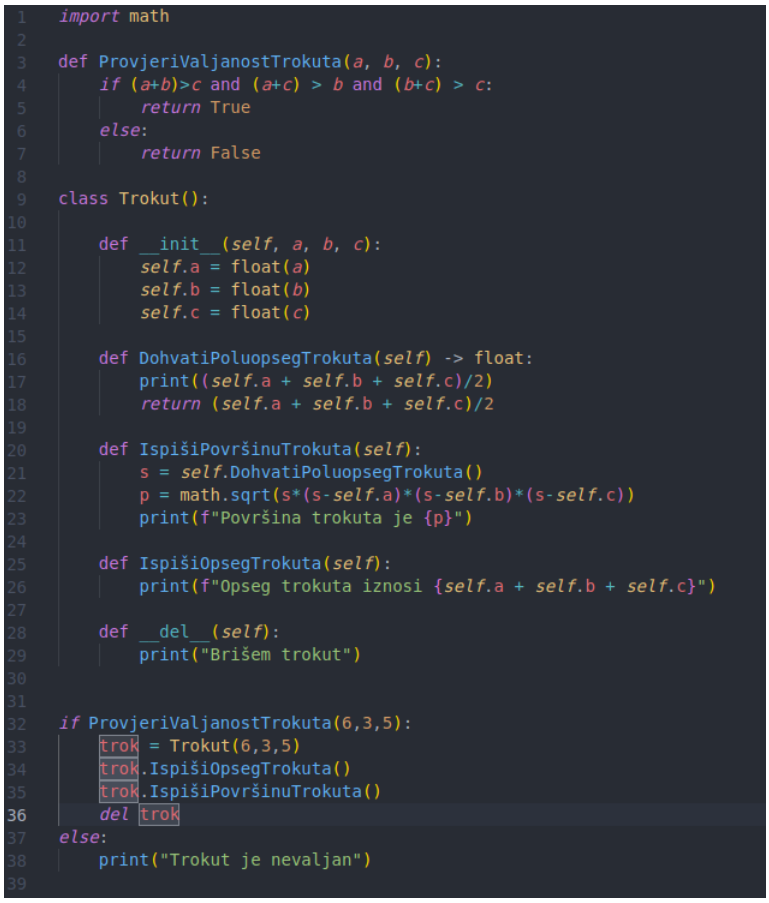
Uočimo da je bitna ključna riječ *class*. U zagrade možemo upisati imena klasa koje želimo da klasa *MojaKlasa* naslijedi. Ako klasa ne nasljeđuje drugu klasu nije potrebno staviti zagrade. Svi atributi i metode moraju biti pravilno uvučeni. Klasa ima svoje **atribute i metode**. U suštini, atributi su podaci specifični za klasu. I mogu biti bilo koji tip podatka, dok su metode funkcije specifične za neku klasu. U sljedećem primjeru, klasi *Osoba* želimo dodati atribute I*me, Prezime, OIB, EMail, Mobitel*. Uočimo uvlake atributa klase. Zbog krive uvlake varijabla *Mobitel* neće biti dio klase *Osoba* i prilikom ispisa javit će se greška.

```
1  #------Definiranje klase i njezinih atributa---------
2  class Osoba:
3      Ime = "Patrik"
4      Prezime = "Horvatić"
5      OIB = "123123123"
6      EMail = "patrikmail@domena.hr"
7
8  Mobitel = "0981234567"
9
10 #------Instanciranje i ispis atributa---------------
11 novaOsoba = Osoba()
12 print(novaOsoba.Ime)
13 print(novaOsoba.Prezime)
14 print(novaOsoba.OIB)
15 print(novaOsoba.EMail)
16 print(novaOsoba.Mobitel)
17
```

Slika 16: Primjer uvlačenja atributa (vlastita izrada)

Da bi se funkcija mogla pozvati potrebno je kao argument upisati ključnu riječ *self*. Parametar *self* referenca je na trenutnu instancu klase i koristi se za pristup varijablama ili metodama koje pripadaju klasi (w3schools.com, bez dat.).

Sada slijedi primjer klase *Trokut*. U našem primjeru ćemo stvoriti klasu trokut koja sadrži atribute *a, b, c* i metode *DohvatiPoluopsegTrokuta, IspišiPovršinuTrokuta* i *IspišiOpsegTrokuta.* Imena metoda ukazuju za što je funkcija odgovorna. Prije definiranja klase kreirana je funkcija *ProvjeriValjanostTrokuta*. Ako je trokut valjan tada kreiramo instancu klase *Trokut*. Imamo funkciju *__init__(self, a, b ,c)* koja se izvodi prilikom instanciranja klase. Argumenti funkcije *a, b* i *c* su stranice trokuta koje su zadane prilikom instanciranja u liniji 33. Na kraju imamo metodu *__del__(self)* koja se poziva prilikom brisanja instance objekta, a to je u liniji 36.



```python
import math

def ProvjeriValjanostTrokuta(a, b, c):
    if (a+b)>c and (a+c) > b and (b+c) > c:
        return True
    else:
        return False

class Trokut():

    def __init__(self, a, b, c):
        self.a = float(a)
        self.b = float(b)
        self.c = float(c)

    def DohvatiPoluopsegTrokuta(self) -> float:
        print((self.a + self.b + self.c)/2)
        return (self.a + self.b + self.c)/2

    def IspišiPovršinuTrokuta(self):
        s = self.DohvatiPoluopsegTrokuta()
        p = math.sqrt(s*(s-self.a)*(s-self.b)*(s-self.c))
        print(f"Površina trokuta je {p}")

    def IspišiOpsegTrokuta(self):
        print(f"Opseg trokuta iznosi {self.a + self.b + self.c}")

    def __del__(self):
        print("Brišem trokut")


if ProvjeriValjanostTrokuta(6,3,5):
    trok = Trokut(6,3,5)
    trok.IspišiOpsegTrokuta()
    trok.IspišiPovršinuTrokuta()
    del trok
else:
    print("Trokut je nevaljan")
```

Slika 17: Primjer implementacije klase Trokut (vlastita izrada)

5.6. Iznimke

Prilikom razvoja veće aplikacije ili bilo kakvog programskog rješenja mogućnost da se ne dogodi greška je nemoguća. Python ima implementiranu funkcionalnost dohvaćanja grešaka. Greške u Pythonu možemo podijeliti na **sintaksne greške i iznimke. Sintaksne greške** se pojavljuju prilikom pisanja programskog koda i velik broj alata ima implementirano prepoznavanje sintaksnih grešaka. Prije izvođenja programskog rješenja programer lako otkloni te greške. **Iznimke** se pojavljuju iako je programsko rješenje sintaktički točno napisano. Python u sebi ima ugrađen velik broj iznimki i svaka od njih potiče iz klase *BaseException*. Kada se dogodi greška prilikom izvođenja diže se greška. Ako greška nije rukovana taka se program ruši. Kako bi se spriječilo potencijalno rušenje programa koristi se *try..except* blok naredbi. Opća sintaksa bloka naredbi *try..except* je sljedeća (Python Software Foundation, 2021):

```
try:
      blok naredbi
except ValueError:
```

```
        blok naredbi
except NameError as e:
        blok naredbi
except:
        blok naredbi
else:
        blok naredbi
finally:
        blok naredbi
```

Bitno je napomenuti sljedeće:

1. Potrebno je imati barem jedno rukovanje iznimkom, odnosno barem jedan *except,* inače se javlja sintaksna greška.
2. Kod rasporeda pisanja *except* blokova, općeniti *except* blok mora biti na zadnjem mjestu. Specifični *except* blokovi dolaze prije općenitog.
3. Ključne riječi *else* i *finally* nisu obavezne**.**
4. Blok naredbi *finally* **uvijek se izvodi**, neovisno o tome da li se dogodila greška.
5. Blok naredbi *else* izvodi se samo onda kada se greška **nije** dogodila.

Način rada rukovanja iznimki (Python Software Foundation, 2021):

1. Počinje se izvoditi blok naredbi između ključnih riječi *try* i *except.*
2. **Ako se ne dogodi greška** tada se blokovi naredbi kod ključnih riječi *except* preskaču. Ako postoji blok naredbi kod ključne riječi *else*, tada se i taj blok naredbi izvodi. Bitno je napomenuti i da se u tom bloku naredbi može dogoditi greška koja neće biti rukovana i rezultat neće biti povoljan.
3. **Ako se dogodi greška** prilikom izvođenja *try* bloka naredbi, ostatak bloka naredbi se preskače, što znači da izvođenje *try* bloka naredbi prestaje u onom trenutku i na onom mjestu kada se dogodi greška. U gornjem primjeru imamo dvije vrste dohvaćanja grešaka. Općenito dohvaćanje i specifično dohvaćanje. Specifično dohvaćanje korisno je jer možemo dohvatiti određen problem za koji možemo naći konkretno rješenje. Ako se dogodi greška koja nije specifično rukovana, izvodi se blok naredbi kod *except.*
4. **Neovisno o tome da li se dogodila greška** izvodi se blok naredbi *finally*.

Sljedeća tablica prikazuje najpoznatije iznimke koje se mogu dogoditi

Tablica 7: Najpoznatije iznimke

Naziv	Opis
ArithmeticError	Greška prilikom numeričkih operacija
AttributeError	Javlja se kada klasa nema tražen atribut
Exception	Bazna klasa svih iznimaka
EOFError	Javlja se kada metoda unosa dođe do kraja
ImportError	Javlja se kada tražen modul ne postoji
IndentationError	Greška uvlaka linija
IndexError	Javlja se kada traženi indeks sekvencijalnog tipa podatka ne postoji
KeyError	Javlja se kada tražen ključ u riječniku ne postoji
KeyboardInterrupt	Javlja se kada korisnik želi tipkovnicom zaustaviti rad programa (Ctrl +C, Ctrl + z)
NameError	Javlja se kada tražena varijabla ne postoji
OSError	Javlja se kada se operacija vezana uz operacijski sustav ne izvede pravilno
RuntimeError	Javlja se kada greška ne pripada nijednoj od specifičnih grešaka
UnicodeError	Greška unikoda
ValueError	Javlja se kada je kriva vrijednost dodana određenom tipu podatka
SystemError	Javlja se kada se dogodi greška u sustavu
ZeroDivisionError	Greška dijeljenja s nulom.

(Izvor: w3schools.com, bez dat.)

Uzmimo za primjer unos od strane korisnika. Imamo jednostavan unos brojeva koji će se prilikom unosa broja 0 zbrojiti. Stvorili smo dvije klase. Prva klasa je *Greške* koja nasljeđuje klasu *Exception*. Zatim klasa *GreškaPrirodnogBroja* sadrži atribut *Poruka* koja sadrži opis greške i nasljeđuje klasu *Greške*. Prilikom svakog unosa potrebno je provjeriti da li je unijeta vrijednost prirodni broj. Možemo rukovati sljedećim greškama:

1. Ako je broj manji od nule tada dižemo pozivamo grešku *GreškaPrirodnogBroja* i za nju smo pripremili poseban blok naredbi.
2. Ako unesena vrijednost nije broj nego primjerice riječ, tada se izvodi *ValueError* blok naredbi.
3. Ako korisnik želi nasilno prekinuti program, poziva se općenita greška, prekida se petlja i ispisuje suma do tad unesenih brojeva.

Blok naredbi za grešku *ArithmeticError* nikada se neće izvesti. Ako je unesen broj nula, petlja se prekida i ispisuje suma. Ako se nije dogodila greška unesen broj se dodaje u sumu. Nakon svakog unosa ispisuje se poruka iz *finally* bloka. Rješenje je prikazano na sljedećoj slici.

```python
suma = 0

#Klasa Greške nasljeđuje klasu Exception
class Greške(Exception):
    pass

#Klasa GreškaPrirodnogBroja nasljeđuje baznu klasu Greške
class GreškaPrirodnogBroja(Greške):
    def __init__(self):
        self.Poruka = "Uneseni broj ne smije biti negativan!"

#--------------Program------------------
while True:
    try:
        broj = int(input("Unesite prirodan broj: "))
        if broj < 0:
            raise GreškaPrirodnogBroja
        if broj == 0:
            break

    except GreškaPrirodnogBroja as e:
        print(e.Poruka)

    except ArithmeticError:
        print("Ova poruka se nikada neće ispisati jer se neće dogoditi ovakva vrsta greške.")

    except ValueError:
        print("Morate unijeti broj!")

    except:
        print("\nDogodila se općenita greška. Ova poruka će se ispisati kada tipkovnicom pokušamo prekinuti rad programa (Ctrl + C)")
        print("Ispisujem trenutnu sumu...")
        break

    else:
        suma += broj

    finally:
        print("Ja sam spam poruka.\n")

print(f"\nSuma: {suma}")
```

Slika 18: Primjer grešaka (vlastita izrada)

Ovako rad programa izgleda u terminalu:

```
Unesite prirodan broj: 8
Ja sam spam poruka.

Unesite prirodan broj: 9
Ja sam spam poruka.

Unesite prirodan broj: 7
Ja sam spam poruka.

Unesite prirodan broj: -5
Uneseni broj ne smije biti negativan!
Ja sam spam poruka.

Unesite prirodan broj: slova a
Morate unijeti broj!
Ja sam spam poruka.

Unesite prirodan broj: 56
Ja sam spam poruka.

Unesite prirodan broj: 1
Ja sam spam poruka.

Unesite prirodan broj: ^C
Dogodila se općenita greška. Ova poruka će se ispisati kada tipkovnicom pokušamo prekinuti rad programa (Ctrl + C)
Ispisujem trenutnu sumu...
Ja sam spam poruka.


Suma: 81
patrik@patrik-Aspire-A517-51G:~/Desktop$
```

Slika 19: Rad grešaka u terminalu(vlastita izrada)

5.7. Rad s datotekama

Rad s datotekama neophodan je u svakom jeziku. U jeziku Python datoteke možemo stvarati, otvarati, čitati, zapisivati u njih, mijenjati itd. Glavna funkcija za rad s datotekama je *open().* Argumenti funkcije *open* su prikazani u sljedećoj tablici.

Tablica 8: Argumenti funkcije open

Argument	Opis
file	Naziv, odnosno putanja do datoteke.
mode	Način otvaranja datoteke (čitanje, pisanje…)
buffering	Ovisno o postavljenom broju definira se način *bufferinga*.
encoding	Način kodiranja (UTF-8, latin-1 i sl.)
errors	Definira kako će se rukovati greškama kodiranja i dekodiranja
newline	Interpretacija znaka za novi red

(Izvor: Python Software Foundation, 2021)

Sljedeća tablica prikazuje načine na koje se datoteka može otvoriti.

Tablica 9: Načini otvaranja datoteke.

Metoda	Opis
„r“	Otvara datoteku za čitanje. Ako datoteka ne postoji javlja se greška.
„a“	Otvara datoteku za nadopunjavanje. Ako datoteka ne postoji kreira ju.
„w“	Otvara datoteku za pisanje. Ako datoteka ima neki sadržaj, briše ga.
„x“	Stvara datoteku.
„t“	Tekstualni način rada
„b“	Binarni način rada

(Izvor: w3schools.com, bez dat.)

Zadani način je način čitanja. Način otvaranja *'w+'* i '*w+b*' otvaraju i brišu sadržaj datoteke. Način otvaranja '*r+*' i '*r+b*' otvara datoteku bez brisanja.

Za primjer imat ćemo datoteku zvanu *demodatoteka.txt* i njezin sadržaj je sljedeći:

```
Pozdrav! Ja sam demodatoteka.txt
```

```
Ova datoteka služi za testiranje
Sretno!
```

Primjer programskog koda za čitanje:

```
F = open("demodatoteka.txt", "r", encoding="utf-8")
print(f.read())
f.close()
```

Primjer programskog koda za dodavanje sadržaja:

```
F = open("demodatoteka.txt", "a")
f.write("Ova rečenica je dodan sadržaj datoteke i nalazi se na kraju.")
f.close()
```

6. Korišteni moduli za izradu skripti

Većina modula korištenih za izradu skripti integrirani su u Pythonu, odnosno može ih se koristiti odmah nakon instalacije Pythona. Najprije ću objasniti module korištene za izrađene skripte koje će biti objašnjenje u sljedećem poglavlju. Za izradu skripti korišteni su moduli *smptlib, datetime, os, email, time, requests, fpdf* i *matplotlib*.

smptlib je modul koji definira objekt sesije SMTP klijenta koji se koristi za slanje elektronične pošte na bilo koji internetski sloj s *SMTP* ili *ESMTP daemonom*. Pojedinosti o radu tih protokola potrebno je konzultirati *RFC 821 i RFC 1869* (Python Software Foundation, 2021).

datetime modul koristi se za rad s datumima i vremenom. Moguće je koristiti sve datume u intervalu od 1 do 9999 godine. U sebi sadrži sljedeće klase podataka (Python Software Foundation, 2021):

- ***Date*** koja je idealiziran gregorijanski kalendar i u sebi sadrži atribute *year, month, day (hrv. godina, mjesec, dan)*
- ***Time*** koja je idealizirano vrijeme, što znači da se pretpostavlja da svaki dan ima točno 86 400 sekundi i u sebi sadrži atribute *hour, minute, second, microsecont i tzinfo (hrv. sat, minuta, sekunda, mikrosekunda i tzinfo)*
- ***Datetime*** koja je kombinacija *date* i *time* klasa
- ***Timedelta*** koja je zadužena za prikaz razlika između dvije *date, time* ili *datetime* instance objekta.
- ***Tzinfo*** koja je apstraktna bazna klasa i sadrži objekte s informacijama o određenim vremenskim zonama
- ***Timezone*** koja je dio implementacije klase *tzinfo*

Modul ***os*** sadrži metode vezane uz rad sa operacijskim sustavom i radnjama koje operacijski sustav može učiniti Sadržaj ovog modula ovisi o operacijskom sustavu, što primjerice znači da neke značajke koje sadrži Linux nisu dostupne na Windowsu. Pomoću ovog modula može se (Python Software Foundation, 2021):

- manipulirati datotekama i direktorijima
- manipulirati s argumentima naredbenog retka, varijablama okruženja, parametrima procesa
- kreirati objekt datoteka
- dohvatiti nasumičan broj
- koristiti sučelje rasporeda (eng. *Interface scheduler*)

Modul ***email*** sadrži metode za rad s elektroničnom poštom i teži u pratnji RFC standarda. Ovaj modul je podijeljen na tri glavne komponente i četvrtu komponentu koja je zadužena za praćenje rada ostalih komponenti (Python Software Foundation, 2021). Ovaj modul je korišten zato što prati RFC standarde i zbog načina na koji je ovaj modul implementiran lako ga je koristiti.

Modul ***time*** sadrži metode relativne za rad s vremenom. Sve funkcije nisu dostupne na svim platformama. Većina funkcija dolazi iz biblioteke jezika C (Python Software Foundation, 2021).

Modul ***requests*** nudi elegantan i jednostavan način korištenja HTTP protokola u Pythonu. Omogućuje jednostavno slanje HTTP/1.1 zahtjeva, nema potrebe ručno dodavati nizove upita ili kodirati obrasce POST podataka i održavanje HTTP veze je automatizirano. Ovaj modul uvelike se koristi za dohvaćanje i obradu stranih podataka i prati standarde i zahtjeve korisnika. Ono što nudi su održavanje veze, međunarodne domene i veze, sesije s postojanošću kolačića, automatsko dekodiranje sadržaja, provjera autentičnosti, automatska dekompresija, prijenos datoteka i ostalo (Reitz, 2021).

Modul ***fpdf*** vrlo je jednostavan i sadrži osnovne funkcionalnosti za izradu datoteka PDF formata. Glavne osobine ovog modula su jednostavnost korištenja, sadrži gotove skripte za korištenje u više programskih jezika i nije potrebno preuzeti vanjske pakete. Podržan je za 3.4+ verzije Pythona. Postoji mogućnost dodavanja vlastitih fontova, mogu se koristiti PNG, GIF i JPG formati slika, korisnik može odabrati dimenzije PDF datoteke, mijenjati zaglavlje i podnožje stranice, implementirano je automatsko dodavanje nove stranice i poravnanje teksta i na kraju stvaranja PDF datoteke vrši se kompresija (readthedocs.io, 2021).

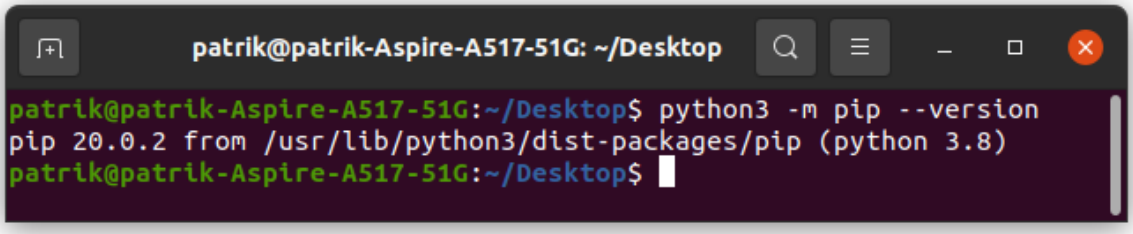
Modul ***matplotlib*** primarno se koristi za izradu statičkih, animiranih i interaktivnih vizualizacija u Pythonu (Hunter et al*.,* 2021). Proučavajući opsežnu i detaljnu dokumentaciju ovog modula možemo zaključiti da sigurno ima veliku primjenu kod velike obrade podataka i znanosti podataka. Zbog svoje jednostavnosti i velikog broja mogućnosti nije iznenađujuće što ga velik broj ljudi razvija i koristi.

7. Izrada skripti

Prije početka izrade skripti potrebno je preuzeti sve potrebne module. Moduli koje ćemo preuzeti su *fpdf, matplotlib* i *requests.* Za preuzimanje modula otvorit ćemo naredbeni redak. Naredba *pip3* koristi se za preuzimanje i instalaciju vanjskih modula. Prvo treba provjeriti ako nakon instalacije Pythona *pip* postoji. Može se dogoditi da nakon instalacije nemamo *pip*. U naredbeni redak potrebno je unijeti sljedeće (pypa.io, 2021):

```
python3 -m pip –version
```

U naredbenom retku bi se nakon unosa trebao dogoditi ispis verzije kao što je vidljivo na slici.



Slika 20: Provjera postojanosti *pip-a*

Ako *pip* ne postoji potrebno ga je preuzeti i izvršiti instalaciju. Python u svojem paketu instalacije sadrži modul *ensurepip* koji vrši konfiguraciju *pip* naredbe. To se može učiniti na sljedeći način:

```
python –m ensurepip –upgrade
```

Nakon što smo osigurali nesmetano preuzimanje potrebnih modula u naredbeni redak potrebno je unijeti sljedeće naredbe:

```
pip3 install matplotlib
pip3 install fpdf
pip3 install requests
```

Potrebno je napomenuti da svaki modul ima jedinstveno ime. Ako nam za izradu skripte treba vanjski modul potrebno je u dokumentaciji pronaći naredbu za instalaciju, odnosno naziv paketa koji je jedinstveno prepoznat od strane *pip-a*. Nakon preuzimanja i instalacije navedenih modula možemo krenuti s izradom skripti.

7.1. Automatizirano slanje elektronične pošte

Automatizirano slanje elektronične pošte danas je uobičajeno kod velikih kompanija i organizacija. Uzmimo primjerice Facebook. Prilikom registracije na Facebook koristite jednu od e-mail adresa koje imate. Kada jedan od vaših prijatelja ima rođendan dobit ćete poruku u sandučić pošte da vaš prijatelj ima rođendan. Ovaj zadatak sigurno ne obavlja zaposlenik Facebooka, već skripta koja radi u pozadini. Zaposlenik Facebooka je izradio tu skriptu kako ljudi ne bi trebali raditi taj zadatak. Skripta prepoznaje tko su vaši prijatelji i koji od njih ima rođendan i na taj dan dobivate poruku u sandučić. Isto tako možemo uzeti za primjer *newsletter*. Određene web stranice nude mogućnost slanja *newslettera* na račun vaše pošte. Kada se obavi prijava na *newsletter,* server u bazu sprema vaš mail kao mail na koji se mogu slati *newsletteri*. Skripta koja radi u pozadini uzima ne samo vašu, već sve e-mail adrese koje su se prijavile na *newsletter* i njima šalje poruku. Zamislite da zaposlenik mora poslati *newsletter* poruku 50 000 različitih osoba. To je izvedivo, ali bi dugo trajalo. Automatizacijom se ovaj zadatak obavi u nekoliko minuta. U sljedećoj skripti pokazat ću koliko je pomoću Pythona jednostavno automatizirati slanje elektronične pošte.

Kako bi smo mogli poslati elektroničnu poštu potrebno je imati račun. U ovoj skripti koristit ću *Gmail*. Izradio sam testni račun *pacpython123@gmail.com*. Nakon izrade računa potrebno je u postavkama računa smanjiti razinu sigurnosti na minimum zato što Gmail sadrži zaštitu od spam pošte.

Za početak potrebni su nam moduli *smptlib, datetime, os* i *email.* Oni sadrže sve što nam treba za izradu ove skripte. Na početak skripte upisujemo sljedeće:

```
import smtplib
import datetime as dt
from os.path import basename
from email.message import EmailMessage
from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email import errors
```

Nakon toga potreban je unos naših podataka, adresu primatelja i poruku primatelju. Kada bi se radilo o aplikaciji sa implementiranim grafičkim sučeljem, iz forme bi se povukli podaci koje je korisnik unio, ali to kod nas nije slučaj, stoga ćemo koristiti funkciju *input()* i unesene podatke spremiti u prikladno imenovane varijable.

```
mojmail = input("Unesite svoju Gmail adresu: ")  #Unesite Gmail adresu
mojpass = input("Unesite lozinku: ")      #Unesite lozinku računa
primatelj = input("Unesite adresu primatelja: ") #unesite primatelja
```

```
poruka = input("Unesite sadržaj poruke: ")       #unos sadržaja poruke
```

Kako bi mogli poslati elektroničnu poštu potrebno je povezati se sa SMTP serverom domene našeg računa, a to je u našem slučaju Google. Google ima otvoreni port broj 587 na koji se je potrebno povezati kako bi mogli pravilno izvesti slanje elektronične pošte. Mogu se dogoditi greške i iznimke tijekom slanja elektronične pošte, stoga ćemo blok naredbi zadužen za povezivanje i provjeru povezanosti sa serverom staviti u *try...except* naredbu.

```
try:
    veza = smtplib.SMTP("smtp.gmail.com","587") #stvori SMTP instancu
    veza.starttls() #stvori tls vezu
    veza.login(user=mojmail, password=mojpass) #prijava korisnika

except smtplib.SMTPAuthenticationError:
    print("Krivo korisničko ime ili lozinka!")
except smtplib.SMTPConnectError:
      print("Povezivanje neuspješno.")
else:
      blok naredbi za slanje koji će biti objašnjen u nastavku
finally:
    veza.quit()
```

Slijedi objašnjenje svakog bloka naredbi. U ***try*** bloku naredbu stvaramo vezu, odnosno SMTP protokol. Klasa SMTP za svoje instanciranje zahtjeva dva argumenta, a to su adresa servera i port. To su u našem slučaju adresa *„smtp.gmail.com“* i port 587. Zatim SMTP vezu treba staviti u određenu razinu sigurnosti, a to se čini pomoću metode *starttls().* Nakon uspostave kriptirane veze potrebno se prijaviti kao korisnik, za što služi metoda *login()* koja prima dva argumenta, a to su korisničko ime i lozinka (Python Software Foundation, 2021.).

Imamo dva ***except*** bloka naredbi. Prva iznimka koja se može dogoditi jest *SMTPAuthenticationError,* što znači da uneseni podaci o korisniku nisu točni. Druga iznimka jest *SMTPConnectError* što znači da povezivanje sa serverom nije uspjelo (Python Software Foundation, 2021.). To može biti do korisnika ili servera. Ako se nije dogodila greška, odnosno ako je povezivanje i prijava korisnika uspješna izvodi se ***else*** blok naredbi kojeg ćemo sada objasniti. U bilo kojem slučaju, potrebno je zatvoriti vezu sa serverom, a to se izvodi u ***finally*** bloku pomoću metode *quit()* koja ne traži argumente.

U ***else*** bloku stvorit ćemo ugniježđeni *try…except* blok naredbi. Potrebno je stvoriti podnožje poruke. Dohvatiti ćemo trenutni datum i vrijeme pomoću metode *now()* i spremiti ga

u varijablu *vrijeme*. Varijabla *podnozje* na početku postavlja dva prazna retka i ima prikladan tekst. Sadržaj poruke kojeg je korisnik unio spojit ćemo sa podnožjem i spremiti ga u varijablu *tekst.* Ovako to izgleda.

```
vrijeme = dt.datetime.now()
podnozje = f"\n\nPoslano s Python3 skripte:
\n{vrijeme.day}.{vrijeme.month}.{vrijeme.year}\n{vrijeme.hour}:{vrijeme.min
ute}"
tekst = poruka + podnozje
```

Nakon toga potrebno je stvoriti objekt poruke. Stvaramo instancu *poruka* klase *MIMEMultipart.* Ova klasa omogućuje da u poruku dodamo i privitke, odnosno datoteke koje želimo poslati. Kako bi mogli poslati poruku potrebno je definirati pošiljatelja, primatelja i temu (Python Software Foundation, 2021).

```
poruka = MIMEMultipart()
poruka["From"] = mojmail #pošiljatelj
poruka["To"] = primatelj #primatelj
poruka["Subject"] = "Testiranje s attachmentom" #tema
poruka.attach(MIMEText(tekst)) #dodajemo tekst
```

Pošto naša skripta nema implementirano grafičko sučelje preko kojeg bi korisnik mogao odabrati datoteke koje želi poslati, mi ćemo u listu ručno unijeti datoteke koje želimo poslati u privitku. Prethodno je stvorena datoteka *tekstualniformat.txt* koja se nalazi u istom direktoriju kao i datoteka skripte i čiji je sadržaj sljedeći:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut et purus eu sapien
interdum viverra ut at velit.
Vestibulum libero justo, rhoncus ut tincidunt a, pellentesque sit amet ipsum.
Ut at quam vel mi convallis vestibulum id at erat. Phasellus quis massa
mollis, ullamcorper sapien sed, interdum est.
Sed augue urna, euismod posuere turpis id, volutpat vestibulum risus.
Orci varius natoque penatibus et magnis dis parturient montes, nascetur
ridiculus mus.
Vestibulum sagittis sem luctus faucibus aliquam. Nulla facilisi.
```

Potrebno je svaku datoteku iz liste datoteka otvoriti i učitati u binarnom načinu, za što ćemo koristiti *for* petlju. Stvorit ćemo instancu *prilog* klase *MIMEApplication* i kojoj ćemo za instanciranje priložiti čitanu datoteku s određenim imenom. Prilog ćemo priložiti poruci pomoću metode *attach*.

```
lista_datoteka = ["tekstualniformat.txt"]
```

```
for element in lista_datoteka:
      with open(element, "rb") as datoteka:
            prilog = MIMEApplication(datoteka.read(),
            Name=basename(element))
      prilog["Content-Disposition"] = f"attachment;
      filename={basename(element)}"
      poruka.attach(prilog)
```

Nakon toga potrebno je poslati poruku za što se koristi metoda *sendmail,* čiji su argumenti primatelj, pošiljatelj i poruka.

```
veza.sendmail(from_addr=mojmail, to_addrs=primatelj, msg=poruka.as_string())
```

Slijedi cijeli blok naredbi nakon uspješne prijave korisnika.

```
try:
      vrijeme = dt.datetime.now() #dohvati trenutni datum i vrijeme
      podnozje = f"\n\nPoslano s Python3
skripte:\n{vrijeme.day}.{vrijeme.month}.{vrijeme.year}\n{vrijeme.hour}:{vri
jeme.minute}"
      tekst = poruka + podnozje

      poruka = MIMEMultipart()
      poruka["From"] = mojmail #pošiljatelj
      poruka["To"] = primatelj #primatelj
      poruka["Subject"] = "Testiranje s attachmentom" #tema
      poruka.attach(MIMEText(tekst))

      lista_datoteka = ["tekstualniformat.txt"]
      for element in lista_datoteka:
            #otvori datoteku iz liste
            with open(element, "rb") as datoteka:
                  #stvori objekt prilog
                  prilog = MIMEApplication(datoteka.read(),
Name=basename(element))

            prilog["Content-Disposition"] = f"attachment;
      filename={basename(element)}"
            poruka.attach(prilog)

      veza.sendmail(from_addr=mojmail, to_addrs=primatelj,
msg=poruka.as_string())
```

```
except smtplib.SMTPRecipientsRefused:
      print("Mail nije poslan. Ovaj primatelj ne postoji.")
except smtplib.SMTPDataError:
      print("Neočekivani kod greške od maila. Sadržaj maila nije dobar.")
except errors.MessageError:
      print("Greška kod poruke")
except:
      print("Nepoznata greška")
else:
      print("Mail uspješno poslan.", end="\n\n")
finally:
      pass
```

Može se dogoditi nekoliko grešaka. Možemo unijeti adresu koja se ne koristi, odnosno da ne postoji korisnik. Takva vrsta iznimke jest *SMTPRecipientsRefused.* Datoteke koje želimo priložiti mogu biti nevaljane, stoga će se javiti iznimka *SMTPDataError* . Malo općenitija greška kod slanja poruke jest *MessageError,* koja obuhvaća općenite greške vezane uz slanje poruke. Ako se nije dogodila greška prilikom slanja ispisat će se prikladna poruka.

7.2. Vrijednosti dionica na burzi

Ova skripta predstavit će se kao dobar primjer analize podataka i automatizacije. Promotrimo sljedeći primjer. Brokeri koji se bave statističkom analizom dionica i stvaranja prognoze kod sebe nemaju spremljene podatke o tim dionicama već ih dobivaju na zahtjev kojem daju izvoru podataka. Oni ne mogu raditi svoj posao bez podataka o vrijednostima dionica. Ti izvori podataka obično su kompanije sa serverima i ostalom potrebnom infrastrukturom i imaju spremljene velike količine relevantnih podataka. Takvih kompanija je malo i stoga one mogu prodavati te podatke. Broker nema pristup tim podacima, stoga ih mora kupiti, a to će obaviti kupnjom API ključa.

Sučelje za programiranje aplikacija ili API omogućuje tvrtkama da otvore podatke i funkcionalnost svojih aplikacija vanjskim programerima trećih strana, poslovnim partnerima i unutarnjim odjelima unutar svojih tvrtki. To omogućuje uslugama i proizvodima međusobnu komunikaciju i međusobno iskorištavanje podataka i funkcionalnosti putem dokumentiranog sučelja. Programeri ne moraju znati kako je API implementiran, jednostavno koriste sučelje za komunikaciju s drugim proizvodima i uslugama. Upotreba API-ja porasla je u posljednjem desetljeću, do stupnja da mnoge od najpopularnijih web aplikacija danas ne bi bile moguće bez API-ja (IBM, 2021). Broker ima svoju aplikaciju preko koje obavlja svoj posao, a ta aplikacija koristi API kako bi dohvaćala podatke.

Zadatak ove skripte jest pomoću vanjskog izvora podataka dohvatiti podatke o određenoj dionici koju korisnik unese. Za unesene dionice korisnik će dobiti vrijednosti dionica u vremenskom intervalu kojeg također on definira. Pomoću API-ja će se dohvatiti podaci o tvrtki i podaci o dionicama. Dohvaćene podatke će spremiti u PDF datoteku kako bi korisnik nakon toga imao uvid u dobivene podatke.

Za početak je potrebno naći API koji će nam moći dostaviti potrebne podatke. **Alpha Vantage** pruža podatke o financijskom tržištu putem skupa moćnih API-ja prilagođenih razvojnim programerima. Od tradicionalnih klasa imovine (npr. Dionica i ETF-ova) do forex-a i kriptovaluta, od temeljnih podataka do tehničkih pokazatelja, Alpha Vantage nudi sve na jednom mjestu za podatke o globalnom tržištu koji se isporučuju putem API-ja u oblaku, Excela i Google tablica (Alpha Vantage, 2021). Ovaj API također je besplatan i nudi upravo ono što trebamo.

Kako bi koristili ovaj API potrebno je stvoriti račun. Ovdje neću objasniti postupak registracije zato što je prilično intuitivan i jednostavan. Dobivene podatke potrebno je spremiti kako bi mogli koristiti API. Imamo ključ potreban za dohvaćanje podataka, stoga možemo započeti izradu skripte. Za početak trebamo uvesti module *time, requests, datetime, fpdf* i *matplotlib*.

```
import time
import requests
import datetime as dt
from fpdf import FPDF
import matplotlib.pyplot as plt
```

Nakon toga definirat ćemo klasu **PDF** koja *nasljeđuje* klasu FPDF, što znači da naša klasa sadrži sva svojstva klase FPDF. Definirat ćemo veličinu PDF datoteke koja će biti krajnji rezultat rada skripte. Bit će formata A4, što znači da će širina biti 210 milimetara, a visina 297 milimetara. Definirat ćemo širinu i visinu grafova.

```
class PDF(FPDF):
      #velicina a4 papira u mm
      SIRINA = 210
      VISINA = 297
      SLIKA_SIRINA = 160
      SLIKA_VISINA = 120

      PRORED = 8
      SIRINA_CELIJA = 40
```

Dodat ćemo vlastito zaglavlje na svakoj stranici. Zapisat ćemo datum i vrijeme kada je izvještaj generiran i osobu koja ga je generirala.

```
      def header(self):
            self.set_font("Arial","B",12)
            self.cell(0,10,datumivrijeme(),0,0,'L')
            self.cell(0,10,"Generirao: Patrik Horvatic",0,0,'R')
            self.ln(20)
```

Dodat ćemo vlastito podnožje i ono će sadržavati broj stranice.

```
      def footer(self):
            self.set_y(-15)
            self.set_font("Arial", "I", 10)
            self.cell(0,10,f"Strana {self.page_no()}",0,0,'C')
```

Dodat ćemo metodu koja će stvoriti naslov na stranicu. Funkcija prima argument *simbol*, a on predstavlja simbol dionice za koju je korisnik zatražio podatke.

```
      def naslov(self, simbol):
            """ Dodaje naslov na stranicu."""
            self.set_font("Arial", 'B', 16)
            self.cell(0,16,txt=f"{simbol} - stanje dionica", align='C')
```

```
        self.ln(12)
```

Dodajemo metodu ***dodajtekst*** koju ne treba mnogo objašnjavati. Funkcija prima jedan argument *tekst*, a to je ono što želimo zapisati u PDF.

```
    def dodajtekst(self, tekst):
        #self.set_xy(10.0,80.0)
        self.set_text_color(0,0,0)
        self.set_font("Arial", '', 12)
        self.multi_cell(w=0,h=10,txt=tekst,border=0,
        align='J',fill=False)
```

Dodat ćemo metodu ***dodajsliku*** koja kao argument prima sliku stvorenu pomoću modula *matplotlib*.

```
    def dodajsliku(self, slika):
        self.image(name=slika,
        w=self.SLIKA_SIRINA,
        h = self.SLIKA_VISINA,
        type="PNG")
```

Zadnja metoda koju dodajemo je metoda *dodajtablicu*. Kao argumente prima dvije liste, datume i vrijednosti za određeni datum.

```
    def dodajtablicu(self, datumi, vrijednosti):
        #SPREMI VRIJEDNOSTI
        self.datumi = datumi
        self.vrijednosti = vrijednosti
        self.set_font("Arial", '', 10)
        self.cell(w=self.SIRINA_CELIJA,h=self.PRORED,txt="",
        align='C',border=0)
        self.ln(self.PRORED)

        #STVORI ZAGLAVLJE TABLICE
        self.set_font("Arial", '', 10)
        self.cell(w=self.SIRINA_CELIJA,h=self.PRORED,txt="Datum",
        align='C',border=1)
        self.cell(w=self.SIRINA_CELIJA,h=self.PRORED,
        txt="Vrijednost dionice", align='C',border=1)
        self.ln(self.PRORED)

        #ISPIŠI PODATKE
        for i in range(0,len(self.datumi)):
            self.cell(self.SIRINA_CELIJA,self.PRORED,
```

```
                txt=self.datumi[i],align='C',border=1)
                self.cell(self.SIRINA_CELIJA,self.PRORED,
                txt=str(self.vrijednosti[i]),align='C',border=1)
                self.ln(self.PRORED)

        #OSTAVI PRAZNOG PROSTORA
        self.ln(self.PRORED*2)
        self.set_font("Arial", '', 12)
```

Završena je implementacija klase PDF.

Klasa **Dionica** u sebi će sadržavat sve potrebne podatke i metode za dohvaćanje i manipulaciju dohvaćenih podataka. Najprije ćemo spremiti stranicu i API ključ koji smo dobili prilikom registracije.

```
class Dionica:
      ALPHA_STRANA = "https://www.alphavantage.co/query"
      KLJUC_ALPHA = "AAAAAAAAAAAAAAAA"
```

Prilikom instanciranja ove klase spremit ćemo simbol i definirani opseg. Naša skripta dohvaćat će općenite podatke o poduzeću i stanje dionica određenog broja prošlih mjeseci. Za svako od toga potrebno je kreirati posebno zaglavlje parametara. Za dohvaćanje općenitih podataka o dionici, odnosno kompaniji, potrebna su tri parametra u obliku rječnika, dok su za dohvaćanje podataka vrijednosti dionica potrebna četiri. Vrijednosti ključa *function* eksplicitno su određeni u dokumentaciji, stoga mijenjanjem te vrijednosti ili unos drugačije ne će dati željene podatke (Alpha Vantage, 2021.). To izgleda ovako:

```
      def __init__(self, simbol, opseg):
            self.simbol = simbol
            self.opseg = opseg

            #parametri za dohvaćanja podataka vrijednosti dionica
            self.parametriBurza = {
                  "function": "TIME_SERIES_MONTHLY_ADJUSTED",
                  "symbol": self.simbol,
                  "datatype": "json",
                  "apikey": self.KLJUC_ALPHA
            }

            #parametri za dohvaćanja podataka o kompaniji
            self.paramteriKompanija = {
```

```
            "function": "OVERVIEW",
            "symbol": self.simbol,
            "apikey": self.KLJUC_ALPHA
        }
```

Metoda *dohvatipodatke* uspostavlja vezu i dohvaća podatke na temelju danih parametara Ovdje se koristi modul *requests* i njegova metoda *get* koja prima dva argumenta. To su poveznica s koje ćemo pokušati dohvatiti podatke i parametri koje želimo, odnosno moramo proslijediti. Nakon toga pozivamo metodu *raise_for_status* koja definira statusni kod ove radnje. Ako je povratni kod 200, tada je povezivanje uspjelo i možemo zatražiti podatke koje trebamo. Na kraju radimo ispis tog koda kako bi imali povratnu informaciju. To izgleda ovako:

```
def dohvatipodatke(self):
      #dohvati podatke s burze
      self.odgovorBurza = requests.get(url=self.ALPHA_STRANA,
       params=self.parametriBurza)
      self.odgovorBurza.raise_for_status()
      print(f"[{self.simbol}] - odgovor servera za burzu:
      {self.odgovorBurza.status_code}")

      #dohvati podatke o poduzeću
      self.odgovorKompanija = requests.get(url=self.ALPHA_STRANA,
      params=self.paramteriKompanija)
      self.odgovorBurza.raise_for_status()
      print(f"[{self.simbol}] - odgovor servera za kompaniju:
      {self.odgovorKompanija.status_code}")
```

Sljedeće stvaramo **metodu *spremipodatke***. Ona je odgovorna za spremanje podataka u datoteku koja će se nalaziti na našem računalu. Nakon uspješnog odgovora API-a možemo koristiti metodi *json* koja će zatražene podatke spremiti u JSON formatu.

```
def spremipodatke(self):
      self.tekst = ""

      #spremi podatke kao json i spremi ih u datoteku
      self.podaciBurza = self.odgovorBurza.json()
      self.podaciKompanija = self.odgovorKompanija.json()

      #spremi sadržaj JSON-a kao tekst, potrebno za pisanje u PDF
      for kljuc,vrijednost in self.podaciKompanija.items():
            self.tekst += f"{kljuc}: {vrijednost}\n"
```

```
#spremi podatke u datoteke
with open("podaci.json", "w", encoding="utf8") as datoteka:
      datoteka.write(str(self.podaciBurza))
      datoteka.close()

with open("podaci-kompanija.json", "w", encoding="utf8") as
datoteka:
      datoteka.write(str(self.podaciKompanija))
      datoteka.close()
```

Metoda *obradipodatke* iz spremljenih podataka dohvaća specifičan podatak *„4. close“.* Ovaj podatak predstavlja vrijednost dionice prilikom zatvaranja burze. Sljedeća slika prikazuje strukturu JSON datoteke koju smo spremili.

Slika 21: Struktura JSON datoteke s vrijednostima dionice

Analizirajmo dobivene podatke. Vidljivo je da ključ „*Monthly Adjusted Time Series*“ ima niz rječnika kojima je naziv ključa datum vrijednosti dionice. Taj ključ u sebi sadrži rječnik s podacima i onaj podatak koji trebamo. Kada dođemo do traženog podatka spremit ćemo ga u listu, kao i odgovarajući datum. Pošto trebamo kronološki prikazati vrijednosti, invertirati ćemo vrijednosti lista pomoću metode *reverse*.

```
def obradipodatke(self):
      self.datumi = []
      self.vrijednosti_dionice = []
```

```
            for k1,v1 in self.podaciBurza["Monthly Adjusted Time
Series"].items():
                  self.datumi.append(k1)
                  self.vrijednosti_dionice.append(float(v1.get("4. close")))

            self.datumi.reverse()
            self.vrijednosti_dionice.reverse()
```

U metodi **nacrtajispremigraf** crtamo graf na temelju dobivenih podataka i spremamo ga u datoteku. Ovdje koristimo modul *matplotlib*. Prvo je potrebno definirati figuru, odnosno površinu po kojoj ćemo crtati. Zatim određujemo naziv grafa i nazive osi, odnosno dodajemo tekst pokraj svake osi kako bi znali kontekst podataka koji prikazujemo Na kraju se graf sprema u datoteku i koristi kod kreiranja PDF-a.

```
      def nacrtajispremigraf(self):
            fig = plt.gcf()         #stvaramo figuru
            plt.xlabel("Datum")     #dodaj naziv x osi
            plt.ylabel("Vrijednost dionice")   #dodaj naziv y osi
            plt.title(f"Vrijednosti dionica {self.simbol}") #ime grafa
```

Slijedi prosljeđivanje podataka koji će se nacrtati na graf i određuju se dekoracije na grafu. Za to ćemo koristiti metoda *plot_date* kojoj je prvi argument je lista podataka koji ide na x-os, drugi argument je lista podatka koja ide na y-os i zadnji argument tip linije na grafu. Ova metoda ima i više argumenata, ali njih nema smisla obrađivati jer su izvan naših potreba.

```
            plt.plot_date(self.datumi[-self.opseg:-1:1],
            self.vrijednosti_dionice[-self.opseg:-1:1],
            linestyle="solid")       #nacrtaj graf
```

Nakon crtanja dodajemo mrežu u pozadini, rotiramo datume kako ne bi došlo do njihovih preklapanja, dodajemo margine i prostor jer bi inače datumi i ime x-osi bili u presjeku.

```
            plt.grid(True)          #dodaj mrežu
            plt.xticks(rotation=90) #rotiraj datume za 90 stupnjeva
            plt.margins(0.03)       #dodaj margine grafu,
            plt.subplots_adjust(bottom=0.25) #dodaj više prostora
```

Definiramo veličinu slike radi izvoza. Spremamo sliku pomoću metode *savefig*. Argumenti funkcije su putanja, format slike, orijentacija i dpi. Na kraju je potrebno očistiti graf.

```
            fig.set_size_inches(8,6)      #definiraj veličinu
            plt.savefig(f"Grafovi/{self.simbol} vrijednosti
            dionica.png",format="png",
            orientation="landscape", dpi=300) #spremi sliku u PNG
```

```
        self.slika = f"Grafovi/{self.simbol} vrijednosti dionica.png"
        plt.clf() #ocisti graf
```

Zadnja je metoda **generirajizvjesce**. Ovdje ćemo instancirati klasu PDF i pomoću ove metode ćemo stvoriti PDF datoteku za svaku traženu dionicu. Na početku instanciramo klasu PDF i određujemo orijentaciju, mjernu jedinicu i format PDF-a.

```
    def generirajizvjesce(self):
        pdf = PDF(orientation="P", unit="mm", format="A4")
```

Određujemo margine pomoću metode *set_margins* i prosljeđujemo tri argumenta, a to su broj milimetara od ruba stranice. Unesene brojke za margine su standard kod A4 formata. Postavljamo autora pomoću metode *set_author*, započinjemo brojanje stranica pomoću metode *alias_nb_pages* radi ispisa broja stranice u podnožju i dodajemo jednu stranicu pomoću metode *add_page* kako bi mogli započeti s radom. Jednom kada smo pozvali ovu metodu dalje će se stranice dodavati automatski.

```
        pdf.set_margins(25,25,25)
        pdf.set_author("Patrik Horvatic")
        pdf.alias_nb_pages()
        pdf.add_page()
```

Dodajemo naslov i spremljeni tekst, odnosno podatke o poduzeću.

```
        pdf.naslov(self.simbol)
        pdf.dodajtekst(self.tekst)
```

Dodajemo tablicu povijesnih cijena dionice. Dodajemo sliku grafa u PDF i radimo izvoz PDF u željeni direktorij.

```
        pdf.dodajtablicu(self.datumi[-self.opseg:-1:1],
        self.vrijednosti_dionice[-self.opseg:-1:1])
        pdf.dodajsliku(self.slika)
        pdf.output(f"Izvjestaji/{self.simbol} - izvjestaj.pdf","F")
```

Kako smo završili s definiranjem klasa i njihovim objašnjenjem, vrijeme je da stvorimo smislen programski kod kojim ćemo efikasno izvršiti zadatak. Na početku potrebno je zatražiti od korisnika unos željenih dionica. Dionice imaju svoj definiran simbol i uneseni simboli od strane korisnika moraju biti usklađeni s NASDAQ-ovim popisom. Taj popis može se naći na njihovoj službenoj stranici. Simbol dionice mora biti isključivo velikim slovima što ćemo osigurati s pozivom metode *upper*. Sve unesene simbole spremit ćemo u listu. Za prekid unosa unijet će

se nula. Nakon prekida unosa simbola dionica, tražit ćemo od korisnika da unese opseg podataka, odnosno broj prošlih mjeseci od trenutka unosa.

```
simboli = []
while True:
	inp = input("Unesi simbol koji zelis dodati: ").upper()
	if inp == "0":
		break
	else:
		simboli.append(inp)
```

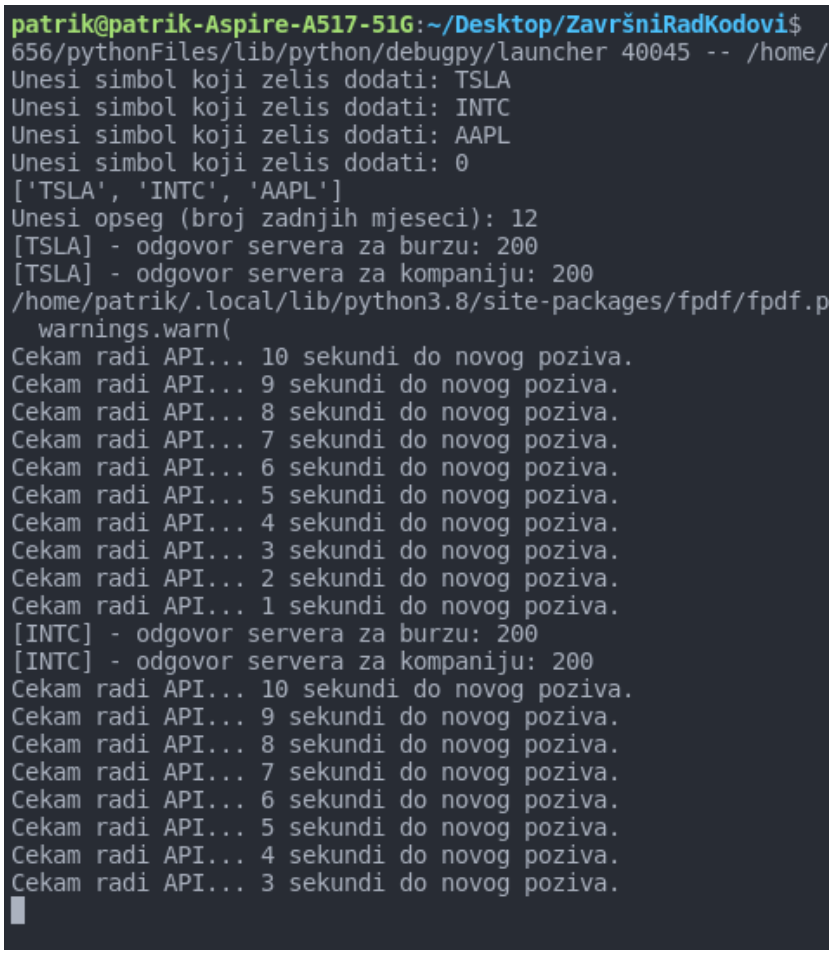
Nakon toga stvaramo listu instanci objekata *Dionica*. Pomoću petlje ćemo instancirati dionice koje je korisnik zatražio i spremiti u listu dionica.

```
dionice = []
for simbol in simboli:
	dionice.append(Dionica(simbol,opseg))
```

Pomoću petlje ćemo za svaku dionicu izvršiti definirane metode. Pred kraj izvršenja jedne iteracije petlje potrebno je pričekati 10 sekundi. Zbog besplatnog API ključa postoje i ograničenje broja zahtjeva u minuti. U minuti možemo šest puta napraviti zahtjev.

```
for i in range(0,len(dionice)):
	dionice[i].dohvatipodatke()
	dionice[i].spremipodatke()
	dionice[i].obradipodatke()
	dionice[i].nacrtajispremigraf()
	dionice[i].generirajizvjesce()
	sekundi = 10
	for _ in range(0,sekundi):
		print(f"Čekam radi API... {sekundi} sekundi do novog poziva.")
		sekundi -= 1
		time.sleep(1)
```

Sljedeće slike prikazuju rad skripte i krajnji rezultat. Sve skripte i sav potreban materijal za njihovo izvođenje može se naći i na mojem GitHub repozitoriju.



Slika 22: Demonstracija rada skripte dionica u naredbenom retku.

28.8.2021 - 21:28:31 **Generirao: Patrik Horvatic**

TSLA - stanje dionica

Symbol: TSLA

AssetType: Common Stock

Name: Tesla Inc

Description: Tesla, Inc. is an American electric vehicle and clean energy company based in Palo Alto, California. Tesla's current products include electric cars, battery energy storage from home to grid-scale, solar panels and solar roof tiles, as well as other related products and services. In 2020, Tesla had the highest sales in the plug-in and battery electric passenger car segments, capturing 16% of the plug-in market (which includes plug-in hybrids) and 23% of the battery-electric (purely electric) market. Through its subsidiary Tesla Energy, the company develops and is a major installer of solar photovoltaic energy generation systems in the United States. Tesla Energy is also one of the largest global suppliers of battery energy storage systems, with 3 GWh of battery storage supplied in 2020.

CIK: 1318605

Exchange: NASDAQ

Currency: USD

Country: USA

Sector: MANUFACTURING

Industry: MOTOR VEHICLES & PASSENGER CAR BODIES

Address: 3500 DEER CREEK RD, PALO ALTO, CA, US

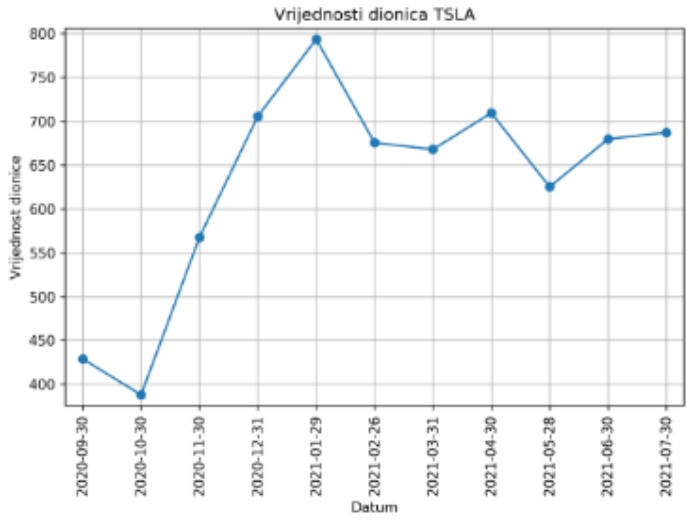
FiscalYearEnd: December

LatestQuarter: 2021-06-30

Slika 23: Primjer teksta u PDF datoteci

28.8.2021 - 21:28:31 **Generirao: Patrik Horvatic**

Datum	Vrijednost dionice
2020-09-30	429.01
2020-10-30	388.04
2020-11-30	567.6
2020-12-31	705.67
2021-01-29	793.53
2021-02-26	675.5
2021-03-31	667.93
2021-04-30	709.44
2021-05-28	625.22
2021-06-30	679.7
2021-07-30	687.2



Strana 4

Slika 24: Primjer tablice i grafa u PDF datoteci.

8. Zaključak

Python je interpreterski jezik i zbog svoje jednostavne sintakse i široke primjene drži prvo mjesto najtraženijeg programskog jezika. Njegova primjena može se naći u razvoju web stranica, automatizaciji, znanosti podataka, umjetnoj inteligenciji i ostalim trendovima programiranja. Sintaksa ovog jezika vrlo je jednostavna i sa vrlo malo programskog koda možemo postići velike i korisne rezultate. Za razliku od drugih jezika, Python ima ugrađene tipove podataka kao što su rječnici i skupovi koji su bez problema našli svoju primjenu kod analize podataka, kao što je vidljivo u izrađenim skriptama. Dodano tome, Python je odličan jezik za one koji žele naučiti programirati. Izrađene skripte pokazuju koliko je jednostavno automatizirati osnovne radnje koje bi inače radili ljudi. Automatsko slanje elektronične pošte dobar je primjer i koriste ga sve kompanije. Izrađena skripta prikazuje osnovu slanja elektronične pošte pomoću Pythona, ali kada bi ju prilagodili ili skalirali radi obavljanja većeg broja zadataka, primjerice, povezivanje s bazom podataka korisnika, tada samo dokazujemo da automatizacija takvog zadatka ima svoj smisao. Automatizacija dohvaćanja vanjskih podataka također se pokazala vrlo jednostavnom i u nekim dijelovima poslovanja jednostavno je neizbježna. U nekoliko linija koda uspješno su dohvaćeni iznimno vrijedni podaci s kojima možemo raditi što je potrebno. Naša skripta dohvatila je podatke o kompaniji i njezine vrijednosti dionica i na temelju toga stvorila jednostavan izvještaj. Za to nije bila potrebna velika i složena skripta. Ovo je još jedan dokaz da se pomoću Pythona na jednostavan način mogu odraditi velike i važne stvari.

Popis literature

Kalafatić, Z., Pošćić, A., Šegvić, S., Šribar, J. (2016). PYTHON za znatiželjne : sasvim drukčiji pogled na programiranje. Zagreb: Element

Downey, A. B. (2014). Naučite Python. Zagreb: Dobar plan.

Christensson, P. (2010). *Interpreter Definition.* Preuzeto 27.1.2021. s https://techterms.com/definition/interpreter

Chris, M. (2019.) *Python Begginers Guide*. Preuzeto 27.1.2021. s https://wiki.python.org/moin/BeginnersGuide/Overview

Eastwood, B. (2020.). *The 10 Most Popular Programming Languages To Learn In 2020*. Preuzeto 22.1.2021. s https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/

Jetbrains.com (2019.) *Python Developers Survey 2019 Results*. Preuzeto 28.1.2021. s https://www.jetbrains.com/lp/python-developers-survey-2019/

Zestminds (2020.) *Top 10 Python Development Trends For 2020*. Preuzeto 27.1.2021. s https://www.zestminds.com/blog/python-development-trends-for-2020/

Zestminds (2020.) *7 Important Python Development Trends You Cannot Ignore in the Year 2021*. Preuzeto 27.1.2021. s https://www.zestminds.com/blog/top-python-development-trends-in-2021/

Azorin, P. (2019.) *Python Development Trends for 2019 and 2020.* Preuzeto 27.1.2021. s https://medium.com/@guestposts_92864/python-development-trends-for-2019-and-2020-70d39d93b5d9

Tutorialspoint (bez dat.) *TurboGears – Overview.* Preuzeto 27.1.2021. s https://www.tutorialspoint.com/turbogears/turbogears_overview.htm

Cuelogic (bez dat.) Role of Python in Artificial Intelligence. Preuzeto 27.1.2021. s https://www.cuelogic.com/blog/role-of-python-in-artificial-intelligence

Christensson, P. (2017). *Data Science Definition.* Preuzeto 28.1.2021. s https://techterms.com

Marutitech.com (bez dat.) *Is Python the most popular language for dana science?.* Preuzeto 28.1.2021. s https://marutitech.com/python-data-science/

Sheth, H. (2020.) *Why Python Is My Favourite For Test Automation?* Preuzeto 28.1.2021. s https://www.lambdatest.com/blog/python-automation-testing/

Techopedia (2021.) *Automation.* Preuzeto 30.8.2021. s https://www.techopedia.com/definition/32099/automation

Abi-Nakhoul, N. (bez dat.) *How is Python used in automation?* Preuzeto 30.8.2021. s https://www.lighthouselabs.ca/en/blog/how-python-is-used-in-automation

Nimkar, A. (2021.) *Best Python Modules for Automation.* Preuzeto 30.8.2021. s https://www.geeksforgeeks.org/best-python-modules-for-automation/

Sweigart, A. (2020.) *PyAutoGUI documentation.* Preuzeto 30.8.2021. s https://pyautogui.readthedocs.io/en/latest/

Browserstack.com (bez dat.) *Selenium.* Preuzeto 30.8.2021. s https://www.browserstack.com/selenium#what-testing-can-be-automated-with-selenium

Breuss, M. (2021.) *Beautiful Soup: Build a Web Scraper With Python.* Preuzeto 30.8.2021. s https://realpython.com/beautiful-soup-web-scraper-python/#reasons-for-web-scraping

Pandas.org (bez dat.) *About pandas.* Preuzeto 30.8.2021. s https://pandas.pydata.org/about/index.html

w3schools.com (bez dat.) Python tutorial. Preuzeto 28.1.2021. s https://www.w3schools.com/python/default.asp

Python Software Foundation (2021.) Python službena dokumentacija. Preuzeto 29.1.2021. s https://docs.python.org/3.8/tutorial/index.html

Christensson, P. (2020, September 21). *Recursive Function Definition*. Retrieved 2021, May 7, from https://techterms.com

Reitz, K. (2021.) Službena dokumentacija modula *requests.* Preuzeto 28.8.2021. s https://docs.python-requests.org/en/master/

readthedocs.io (2021.) Dokumentacija za modul FPDF. Preuzeto 28.8.2021. s https://pyfpdf.readthedocs.io/en/latest/index.html

Hunter, J., Dale, D., Firing, E., Droettboom, M. (2021.) *Matplotlib.* Preuzeto 28.8.2021. s https://matplotlib.org/Matplotlib.pdf

pypa.io (2021.) Dokumentacija za naredbu *pip.* Preuzeto 28.8.2021. s https://pip.pypa.io/en/stable/installation/

IBM (2021.) *Application Programming Interface (API).* Preuzeto 28.8.2021. s https://www.ibm.com/cloud/learn/api

Alpha Vantage (2021.) *Alpha Vantage*. Preuzeto 28.8.2021. s https://www.alphavantage.co/#about

Popis slika

Popis tablica