PANEVROPSKI UNIVERZITET APEIRON

FAKULTET INFORMACIONIH TEHNOLOGIJA

Redovne studije

Smjer „PROGRAMIRANJE I SOFTVERSKI INŽENJERING“

Predmet:

SKRIPT JEZICI I PROGRAMIRANJE

„Osnove programiranja u Pythonu sa praktičnim zadacima “

(seminarski rad)

Predmetni nastavnik

Doc. dr Tijana Talić

Student

Ivan Pavlović

Indeks br. 92-20 R-ITP-S

Banja Luka, Avg 2025

SADRŽAJ

Apstrakt:

Python kao višenamjenski programski jezik pruža širok spektar mogućnosti za rješavanje različitih programerskih problema, od jednostavnih algoritama do kompleksnih aplikacija. Ovaj rad istražuje praktičnu primjenu 15 ključnih Python koncepata, strukturiranih progresivno od osnovnih sintaksnih elemenata (varijable, petlje, funkcije) do naprednih tema (OOP, rad sa bibliotekama, web scraping). Cilj rada je da kroz konkretne primjere demonstrira razvoj vještina programiranja, prikazujući kako se teorijski koncepti implementiraju u stvarnim zadatcima.

Kroz analizu svakog zadatka, rad objašnjava logiku rješenja, sintaksu i očekivane izlaze, uz vizuelnu podršku isječaka koda i rezultata izvršavanja. Poseban naglasak stavljen je na hijerarhijski pristup učenju, gdje svaki sljedeći zadatak gradi na prethodno stečenom znanju. Dodatno, rad uključuje komentarisanu dokumentaciju koda i primjere testiranja, što ističe važnost dobre programerske prakse.

Rezultati rada pokazuju da sistematsko savladavanje Python-a, kroz praktične zadatke različitog nivoa složenosti, omogućuje brzo ovladavanje jezikom i njegovu efikasnu primjenu u realnim scenarijima. Rad pruža i korisne smjernice za dalji napredak, što ga čini vrijednim resursom kako za studente tako i za početnike u programiranju.

***Ključne riječi**: Python, algoritmi, programiranje, objektno orijentisano programiranje, praktični zadaci, sintaksa.*

UVOD

Python je moderan, interpretiran programski jezik visokog nivoa koji se široko koristi u različitim oblastima, od web razvoja i analize podataka do vještačke inteligencije i automatizacije. Zasnovan na jednostavnoj sintaksi i čitljivosti koda, Python je postao jedan od najpopularnijih jezika kako među početnicima tako i među iskusnim programerima. Njegova fleksibilnost i bogata standardna biblioteka omogućavaju brz razvoj aplikacija, dok podrška za više paradigmi (proceduralno, objektno-orijentisano i funkcionalno programiranje) čini ga pogodnim za različite projekte.

Cilj ovog seminarskog rada je da kroz seriju praktičnih zadataka prikaže razvoj vještina programiranja u Pythonu, počevši od osnovnih koncepata do naprednijih tema. Rad obuhvata 15 zadataka, strukturiranih progresivno, što omogućuje postepeno savladavanje sintakse, algoritama i rada sa bibliotekama. Prvi zadaci fokusiraju se na osnovne operacije i kontrolu toka, dok kasniji uključuju rad sa datotekama, objektno-orijentisano programiranje i primjenu eksternih modula.

Kroz ovaj rad, naglašava se važnost čistog i efikasnog koda, pravilne dokumentacije i praktične primjene teorijskih koncepata. Konačni cilj je da student stekne solidnu osnovu za dalji rad u Pythonu, uz razumijevanje ključnih principa koji se mogu prenijeti i na druge programske jezike.

1 Osnove programskog jezika "Python"

Python je programski jezik visokog nivoa sa jednostavnom i čitljivom sintaksom, što ga čini idealnim izborom i za početnike i za iskusne programere. U ovoj sekciji detaljno ćemo razmotriti ključne koncepte koji čine temelj Python programiranja.

Slika 1 - Jedan od zvaničnih Logotipa programskog jezika Python (Izvor: https://commons.wikimedia.org/wiki/File:Python_logo_and_wordmark.svg)

1.1 Varijable i tipovi podataka

U Python-u, varijable služe kao kontejneri za pohranjivanje podataka, a njihov tip se dinamički određuje prilikom dodjele vrijednosti. Python podržava više osnovnih tipova podataka:

- Cjelobrojni tip (int) – predstavlja cijele brojeve (npr. 5, -10, 2024)
- Brojevi sa pokretnim zarezom (float) – realni brojevi (npr. 3.14, -0.001)
- Stringovi (str) – nizovi znakova (npr. "Zdravo", 'Python')
- Logičke vrijednosti (bool) – True ili False
- Liste (list) – promjenjivi nizovi elemenata (npr. [1, 2, 3])
- Torke (tuple) – nepromjenjivi nizovi (npr. (1, 2, 3))
- Rječnici (dict) – strukture podataka u obliku ključ-vrijednost (npr. {"ime": "Marko", "godine": 25})

Primjer deklaracije varijabli:

```
1 x = 10                # int
2 y = 3.14              # float
3 ime = "Nikolina"      # str
4 lista = [1, 2, 3]     # list
```

Slika 2 - Primjer deklaracije int, float, str i list tipa podataka i njihovih imena varijabli

1.2 Osnovne funkcije

Python nudi bogat izbor ugrađenih funkcija koje značajno olakšavaju rad sa podacima i izvršavanje osnovnih operacija. Ove funkcije predstavljaju temelj Python programiranja i često se koriste u svakodnevnom kodiranju.

1.2.1 "print()" funkcija

Funkcija **print()** je jedna od najosnovnijih i najčešće korištenih funkcija u Pythonu. Služi za ispis vrijednosti na standardni izlaz (obično konzolu). Osim jednostavnog ispisa, print() podržava više korisnih parametara:

```
1   print("Pozdrav pope, direktore!")  # Osnovni ispis stringa
2   print(2 + 3)              # Ispis rezultata izračunava: 5
3   print("Vrijednost je:", 42)  # Ispis više vrijednosti odvojenih razmakom
```

Slika 3 - Primjer korištenja "print()" funkcije

Funkcija **print()** automatski dodaje novi red nakon svakog poziva, što možemo promijeniti parametrom **end**:

```
1   print("Hello", end=" ")
2   print("World!")  # Ispisuje "Hello World!" u istom redu
```

Slika 4 - Primjer korištenja "print()" funkcije sa "end" parametrom

1.2.2 "input()" funkcija

Funkcija input() omogućuje interakciju s korisnikom kroz unos podataka. Prikazuje opcionalni prompt i čeka dok korisnik ne unese podatke:

```
1   ime = input("Molimo unesite svoje ime: ")
2   print("Welcome to HELL,", ime)
```

Slika 5 - Primjer korištenja "input()" funkcije

Važno je napomenuti da input() uvijek vraća string, pa je često potrebno konvertovati rezultat:

```
1   godine = int(input("Unesite vaš broj potrošenih godina čekajući studentsku: "))
```

Slika 6 - Primjer korištenja "input()" funkcije sa konverzijom u Integer tip podatka

1.2.3 "len()" funkcija

Funkcija len() vraća dužinu (broj elemenata) različitih tipova sekvenci:

```
print(len("Python"))       # 6 (broj karaktera u stringu)
print(len([1, 2, 3]))      # 3 (broj elemenata u listi)
print(len({"a": 1, "b": 2}))  # 2 (broj parova ključ-vrijednost u rječniku)
```

Slika 7 - Primjer korištenja "len()" funkcije

1.2.4 "type()" funkcija

Funkcija type() je neophodna za provjeru tipa varijable, što je posebno korisno kod dinamičkog tipiziranja u Pythonu:

```
print(type(10))          # <class 'int'>
print(type(3.14))        # <class 'float'>
print(type("tekst"))     # <class 'str'>
print(type([1, 2]))      # <class 'list'>
```

Slika 8 - Primjer korištenja "type()" funkcije

1.2.5 "range()" funkcija

Koristi se za generisanje niza brojeva, često u kombinaciji s for petljama:

```
for i in range(5):       # Generiše 0, 1, 2, 3, 4
    print(i)
```

Slika 9 - Primjer korištenja "range()" funkcije

1.2.6 "str()", "int()" i "float()" funkcije

Koriste se za konverziju između različitih tipova podataka:

```
broj = str(42)          # Konvertuje u string "42"
string_broj = int("123")  # Konvertuje u integer 123
decimalni = float("3.14")  # Konvertuje u float 3.14
```

Slika 10 - Primjer korištenja str(), int() i float() funkcija

1.3 Operatori i osnovne varijacije

Python nudi širok spektar operatora koji omogućuju izvođenje različitih operacija nad podacima. Ti operatori se mogu podijeliti u kategorije aritmetičkih operatora, relacionih operatora i logičkih operatora.

1.3.1 Aritmetički operatori

Ovi operatori se koriste za osnovne matematičke operacije:

```
1   zbroj = 5 + 3        # Sabiranje (rezultat: 8)
2   razlika = 10 - 4      # Oduzimanje (6)
3   umnozak = 6 * 7       # Množenje (42)
4   kolicnik = 15 / 4     # Dijeljenje (3.75)
5   cjelobrojno = 15 // 4 # Cjelobrojno dijeljenje (3)
6   modulo = 10 % 3       # Ostatak pri dijeljenju (1)
7   potencija = 2 ** 3    # Potenciranje (8)
```

Slika 11 - Korištenje aritmetičkih operatora u Python-u

1.3.2 Relacioni operatori

Koriste se za uspoređivanje vrijednosti i uvijetno grananje:

```
1   jednako = (5 == 5)        # True
2   razlicito = (5 != 3)       # True
3   vece = (10 > 7)           # True
4   manje = (5 < 3)           # False
5   vece_jednako = (5 >= 5)    # True
6   manje_jednako = (5 <= 3)   # False
```

Slika 12 - Korištenje relacionih operatora u Python-u

1.3.3 Logički operatori

Omogućuju kombinovanje logičkih izraza:

```
1   i = (5 > 3) and (2 == 2)   # True (oba uslova su tačna)
2   ili = (5 > 3) or (2 != 2)   # True (prvi uslov je tačan)
3   ne = not (5 > 3)            # False (negacija tačnog uslova)
```

Slika 13 - Korištenje logičkih operatora u Python-u

1.4 Kontrola toka programa

Kontrola toka programa je temeljni koncept u programiranju koji određuje redoslijed izvršavanja instrukcija. U Pythonu se kontrola toka ostvaruje kroz mehanizme grananja i petlji, omogućavajući programerima implementaciju kompleksne logike u njihovim aplikacijama.

1.4.1 Grananje (if – elif – else)

Struktura grananja u Pythonu koristi if, elif i else iskaze za usmjeravanje toka programa na temelju evaluacije logičkih uslova. Osnovna sintaksa počinje s if iskazom koji provjerava primarni uslov. Ukoliko taj uslov nije ispunjen, program prelazi na elif iskaze (koji su opcionalni i može ih biti više) koji provjeravaju dodatne uvjete. Konačno, else iskaz (također opcionalan) pokriva sve preostale slučajeve kada nijedan prethodni uslov nije zadovoljen.

```
1   godine = 25
2   status = ""
3
4   if godine < 13:
5       status = "dijete"
6   elif godine < 18:
7       status = "tinejdžer"
8   elif godine < 65:
9       status = "odrasla osoba"
10  else:
11      status = "starija osoba"
12
13  print(f"Osoba ima {godine} godina i spada u kategoriju: {status}")
```

Slika 14 - Primjer korištenja grananja

Ključna obilježja grananja uključuju hijerarhijsku evaluaciju uslova gdje se čim pronađe ispunjen uslov, izvršava odgovarajući blok koda i prekida daljnja provjeravanja. Ova struktura omogućuje jasno i logično organiziranje kompleksnih uvjetnih scenarija.

1.4.2 Petlje

Petlje u Pythonu omogućuju ponavljanje blokova koda i postoje u dvije osnovne varijante for i while petlje.

1.4.2.1 "for" petlja

For petlja je idealna za iteraciju kroz sekvence poput lista, stringova ili range objekata.

```
1   for broj in range(1, 6):
2       kvadrat = broj ** 2
3       print(f"Kvadrat broja {broj} je {kvadrat}")
```

Slika 15 - Primjer korištenja "for" petlje

1.4.2.2 "while" petlja

While petlja izvršava blok koda sve dok je zadani uslov ispunjen.

```
1   brojac = 5
2   while brojac > 0:
3       print(f"Preostalo: {brojac} sekundi")
4       brojac -= 1
5   print("Vrijeme je isteklo!")
```

Slika 16 - Primjer korištenja "while" petlje

1.5 Rad sa funkcijama

Funkcije predstavljaju temeljnu strukturu u Python programiranju koja omogućava organizaciju koda u logičke, samostalne cjeline. One služe za grupisanje određenog niza operacija koje se često ponavljaju, čime se povećava čitljivost koda i smanjuje redundancija.

Osnovna sintaksa za definisanje funkcije u Pythonu koristi ključnu riječ def:

```
1   def pozdrav(ime):
2       """Ova funkcija ispisuje personalizirani pozdrav"""
3       print(f"Dobar dan, {ime}!")
```

Slika 17 - Primjer definisanja personalizovane funkcije

Poziv funkcije vrši se jednostavnim navođenjem njenog imena i prosleđivanjem potrebnih argumenata:

```
pozdrav("Marko")  # Ispisuje: Dobar dan, Marko!
pozdrav("Nikolina")    # Ispisuje: Dobar dan, Nikolina!
```

Slika 18 - Pozivanje prethodno navedene funkcije

Funkcije mogu primati različite tipove parametara:

- Obavezni parametri
- Podrazumijevane vrijednosti
- Promjenjiv broj argumenata

```
def izracunaj_kamatu(iznos, stopa=0.05, godine=1):
    return iznos * stopa * godine

print(izracunaj_kamatu(1000))         # Koristi podrazumijevane vrijednosti
print(izracunaj_kamatu(1000, 0.1, 2)) # Eksplicitno zadaje parametre
```

Slika 19 - Funkcija koja prima više tipova parametara

Funkcije mogu vratiti rezultat korištenjem ključne riječi **return**:

```
def kvadrat_broja(x):
    return x ** 2

rezultat = kvadrat_broja(4)  # 16
```

Slika 20 - Vraćanje rezultata funkcije korištenjem "return" ključne riječi

Python razlikuje tri nivoa vidljivosti varijabli:

- Lokalne - definisane unutar funkcije
- Enclosing - u slučaju ugniježđenih funkcija
- Globalne - definisane izvan svih funkcija

2 Zadaci u Python-u

2.1 Personalozovani pozdravni sistem sa uticajem vremena

Ovaj program predstavlja sofisticirano rješenje za generiranje kontekstualnih poruka koje se prilagođavaju ne samo unesenom korisničkom imenu već i trenutnom dobu dana. Kroz ovaj zadatak demonstrirana je integracija ključnih programerskih koncepata u stvarnoj primjeni, uključujući interakciju s korisnikom, validaciju unosa, dinamičko generiranje sadržaja i vremenski osjetljive poruke.

2.1.1 Funkcionalna logika personalizovanog pozdravnog sistema

Program koristi regularne izraze iz re modula za validaciju korisničkog unosa, osiguravajući da ime sadrži samo dopuštene znakove. Modul datetime omogućava dinamičko određivanje doba dana na temelju sistemskog vremena. Poruke su organizirane u rječniku koji se mapira prema vremenskim intervalima, što omogućuje brzo i efikasno dobivanje odgovarajuće poruke. Obrada izuzetaka osigurava robustnost programa u slučaju neispravnog unosa, dok modularni dizajn sa specijaliziranim funkcijama povećava čitljivost i održivost koda. Konačni ispis je formatiran korištenjem string operacija za vizualno privlačan prikaz.

Zadatak-1.py

```
1  import re
2  from datetime import datetime
```

Slika 21 - Importovanje modula "re" u zadatak br.1

Program koristi dva osnovna modula:

- re modul za rad s regularnim izrazima (regex) koji će služiti za validaciju unosa imena
- datetime modul iz kojeg se koristi datetime klasa za dobivanje trenutnog vremena

```
def validiraj_ime(ime):  1 usage
    """Provjerava ispravnost formata imena korištenjem regexa"""
    pattern = r'^[A-Za-zŠĐČĆŽšđčćž\s]+$'
    return bool(re.fullmatch(pattern, ime))
```

Slika 22 - Validacija imena u zadatku br.1

Definira regex pattern koji dopušta:

- Sva slova engleske abecede (velika i mala)
- Znakove Srpske azbuke (ŠĐČĆŽšđčćž)
- Razmake (\s)

re.fullmatch() provjerava da li cijeli unos odgovara patternu, a funkcija vraća True ako je unos validan, False ako nije.

Funkcija odredi_period analizira trenutno vrijeme kako bi odredila doba dana. Koristeći datetime.now().hour dobiva trenutni sat (0-23) i na osnovu toga vraća odgovarajući period: "jutro" (5-11 sati), "podne" (12-16 sati), "večer" (17-20 sati) ili "noć" (ostalo).

```
def odredi_period():  1 usage
    """Određuje vremenski period na osnovu trenutnog sata"""
    sat = datetime.now().hour
    if 5 <= sat < 12:
        return "jutro"
    elif 12 <= sat < 17:
        return "podne"
    elif 17 <= sat < 21:
        return "večer"
    else:
        return "noć"
```

Slika 23 - Funkcija za određivanje perioda dana u zadatku br.1

Za generisanje personalizirane poruke koristi se funkcija generisi_poruku koja sadrži rječnik s porukama mapiranim na pojedine periode. Poruke se formiraju koristeći f-stringove koji interpoliraju uneseno ime, a metoda get osigurava fallback opciju ako period nije pronađen.

```
def generisi_poruku(ime, period):  1 usage
    """Generise kontekstualnu poruku na osnovu vremenskog perioda"""
    poruke = {
        "jutro": f"Dobro jutro, {ime}! Kako ste spavali?",
        "podne": f"Prijatan dan, {ime}! Kako napredujete?",
        "večer": f"Prijatno veče, {ime}! Kako je prošao dan?",
        "noć": f"Laku noć, {ime}! Nadam se da ste imali produktivan dan."
    }
    return poruke.get(period, f"Zdravo, {ime}!")
```

Slika 24 - Funkcija za generisanje poruke u zadatku br.1

Prikaz rezultata se obavlja kroz funkciju prikazi_rezultat koja formatira ispis u vizualno privlačan okvir. Koristi se množenjem stringova za kreiranje graničnih linija i metoda center za centriranje teksta.

```
def prikazi_rezultat(poruka):  1 usage
    """Formatira i prikazuje konačni rezultat"""
    print("\n" + "=" * 50)
    print(poruka.center(50))
    print("=" * 50 + "\n")
```

Slika 25 - Funkcija za prikaz rezultata u zadatku br.1

Glavna funkcija main koordinira cijeli tok programa. Koristi beskonačnu petlju s try-except blokom za rukovanje greškama. Nakon unosa imena, poziva se strip za uklanjanje suvišnih razmaka, zatim se vrši validacija. Ako je unos validan, određuje se period dana, generiše poruka i prikazuje formatirani rezultat. U slučaju greške, korisnik se obavještava i traži ponovni unos.

```
def main():  1 usage
    """Glavna kontrolna funkcija programa"""
    while True:
        try:
            unos = input("Molimo unesite vaše ime: ").strip()

            if not unos:
                raise ValueError("Unos ne smije biti prazan!")

            if not validiraj_ime(unos):
                raise ValueError("Ime smije sadržavati samo slova!")

            period = odredi_period()
            poruka = generisi_poruku(unos.capitalize(), period)
            prikazi_rezultat(poruka)
            break

        except ValueError as e:
            print(f"\nGreška: {e}\nMolimo pokušajte ponovno.\n")
```

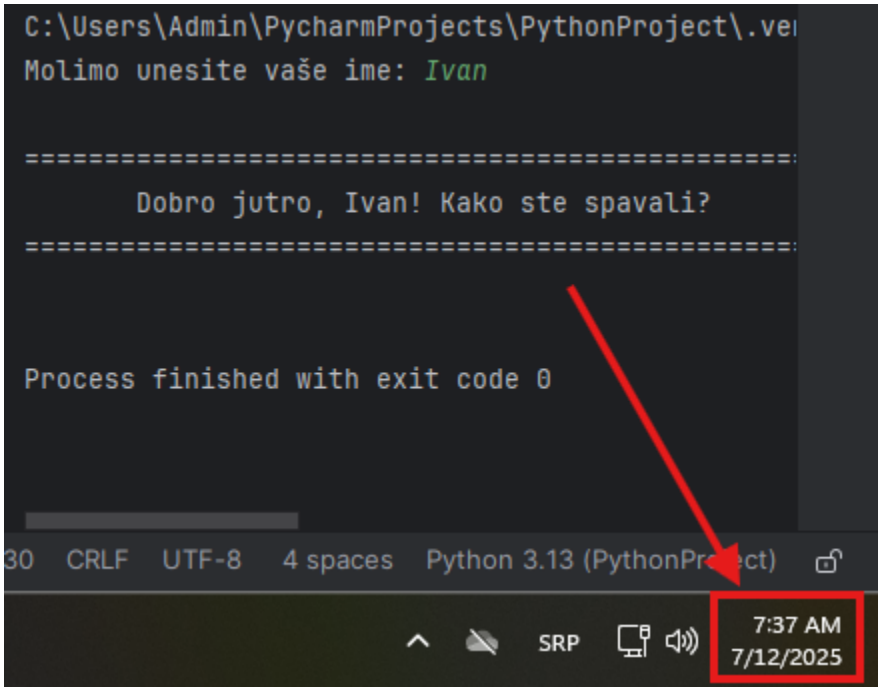
Slika 26 - Funkcija "main()" za koordinaciju toka programa u zadatku br.1

2.1.2 Kompletan Python kod za personalizovani pozdrav

```python
import re
from datetime import datetime


def validiraj_ime(ime):
    """Provjerava ispravnost formata imena korištenjem regexa"""
    pattern = r'^[A-Za-zŠĐČĆŽšđčćž\s]+$'
    return bool(re.fullmatch(pattern, ime))


def odredi_period():
    """Određuje vremenski period na osnovu trenutnog sata"""
    sat = datetime.now().hour
    if 5 <= sat < 12:
        return "jutro"
    elif 12 <= sat < 17:
        return "podne"
    elif 17 <= sat < 21:
        return "večer"
    else:
        return "noć"


def generisi_poruku(ime, period):
    """Generise kontekstualnu poruku na osnovu vremenskog perioda"""
    poruke = {
        "jutro": f"Dobro jutro, {ime}! Kako ste spavali?",
        "podne": f"Prijatan dan, {ime}! Kako napredujete?",
        "večer": f"Prijatno veče, {ime}! Kako je prošao dan?",
        "noć": f"Laku noć, {ime}! Nadam se da ste imali produktivan dan."
    }
    return poruke.get(period, f"Zdravo, {ime}!")


def prikazi_rezultat(poruka):
    """Formatira i prikazuje konačni rezultat"""
    print("\n" + "=" * 50)
    print(poruka.center(50))
    print("=" * 50 + "\n")


def main():
    """Glavna kontrolna funkcija programa"""
    while True:
        try:
            unos = input("Molimo unesite vaše ime: ").strip()

            if not unos:
                raise ValueError("Unos ne smije biti prazan!")

            if not validiraj_ime(unos):
                raise ValueError("Ime smije sadržavati samo slova!")

            period = odredi_period()
            poruka = generisi_poruku(unos.capitalize(), period)
            prikazi_rezultat(poruka)
            break

        except ValueError as e:
            print(f"\nGreška: {e}\nMolimo pokušajte ponovno.\n")


if __name__ == "__main__":
    main()
```

Slika 27 - Kompletan kod zadatka u Notepad++

2.1.3 Prikaz rezultata za personalizovani pozdravni sistem



Slika 28 - Rezultat zadatka br.1

2.2 Kalkulator

Ovaj jednostavan kalkulator izvodi osnovne matematičke operacije (sabiranje, oduzimanje, množenje i deljenje).

2.2.1 Objašnjenje logike kalkulatora

Program počinje definisanjem glavne funkcije kalkulator() koja sadrži sve potrebne logike. Na početku funkcije ispisujemo pozdravnu poruku i informacije o dostupnim operacijama koristeći obične print() naredbe.

```
def kalkulator():  1 usage
    print("\nDobrodošli u osnovni kalkulator")
    print("Dostupne operacije: + (sabiranje), - (oduzimanje), * (množenje), / (deljenje)")
```

Slika 29 - Definisanje glavne funkcije u zadatku br.2

Kalkulator radi u beskonačnoj petlji while True što omogućava korisniku da izvršava više operacija bez potrebe za ponovnim pokretanjem programa. Unutar petlje, koristimo try-except blok za hvatanje i obradu potencijalnih grešaka.

```
while True:
    try:
        # Unos podataka
        prvi_broj = float(input("Unesite prvi broj: "))
        operacija = input("Unesite operaciju (+, -, *, /): ")
        drugi_broj = float(input("Unesite drugi broj: "))

        # Izračunavanje rezultata
        if operacija == '+':
            rezultat = prvi_broj + drugi_broj
        elif operacija == '-':
            rezultat = prvi_broj - drugi_broj
        elif operacija == '*':
            rezultat = prvi_broj * drugi_broj
        elif operacija == '/':
            if drugi_broj == 0:
                print("Greška: Ne može se deliti nulom!")
                continue
            rezultat = prvi_broj / drugi_broj
        else:
            print("Greška: Nepoznata operacija!")
            continue

        # Prikaz rezultata
        print(f"Rezultat: {prvi_broj} {operacija} {drugi_broj} = {rezultat}")

        # Pitamo korisnika da li želi novi izračun
        nastavak = input("\nDa li želite novi izračun? (da/ne): ").lower()
        if nastavak != 'da':
            print("Hvala što ste koristili kalkulator!")
            break

    except ValueError:
        print("Greška: Morate uneti broj!")
    except Exception as e:
        print(f"Došlo je do greške: {e}")
```

Slika 30 - Kreiranje beskonačne petlje u zadatku br.2

Proces unosa podataka se sastoji iz tri koraka: prvo unosimo prvi broj konvertujući ga u float pomoću float(input()), zatim unosimo simbol operacije, i na kraju unosimo drugi broj. Svaki unos se odmah konvertuje u decimalni broj da bismo mogli da radimo sa decimalnim vrjednostima.

```
# Unos podataka
prvi_broj = float(input("Unesite prvi broj: "))
operacija = input("Unesite operaciju (+, -, *, /): ")
drugi_broj = float(input("Unesite drugi broj: "))
```

Slika 31 - Unos podataka u zadatak br.2

Logika za izračunavanje se nalazi u uslovnom bloku if-elif. Provjeravamo koju operaciju je korisnik unio i izvršavamo odgovarajuću matematičku operaciju. Posebno obrađujemo slučaj djeljenja sa nulom, gdje ispisujemo poruku o grešci i koristimo continue da preskočimo ostatak iteracije petlje.

```
# Izračunavanje rezultata
if operacija == '+':
    rezultat = prvi_broj + drugi_broj
elif operacija == '-':
    rezultat = prvi_broj - drugi_broj
elif operacija == '*':
    rezultat = prvi_broj * drugi_broj
elif operacija == '/':
    if drugi_broj == 0:
        print("Greška: Ne može se djeliti nulom!")
        continue
    rezultat = prvi_broj / drugi_broj
else:
    print("Greška: Nepoznata operacija!")
    continue
```

Slika 32 - Logika računanja prethodno unešenih vrijednosti u zadatku br.2

Kada dobijemo rezultat, ispisujemo ga u formatiranom obliku koristeći f-string koji prikazuje oba unijeta broja, operaciju i konačni rezultat.

Nakon svakog uspiješnog izračuna, pitamo korisnika da li želi da nastavi korištenje kalkulatora. Ako odgovor nije "da", ispisujemo zahvalnicu i prekidamo petlju koristeći break.

Program ima dve glavne obrade grešaka, ValueError za slučaj kada korisnik unese nešto što nije broj, i opštu obradu grešaka koja hvata sve ostale neočekivane greške. Svaka greška ispisuje odgovarajuću poruku o tome šta je pošlo naopako.

```
except ValueError:
    print("Greška: Morate unjeti broj!")
except Exception as e:
    print(f"Došlo je do greške: {e}")
```

Slika 33 - Obrada tj. kontrola grešaka u zadatku br.2

Na samom kraju, koristimo standardnu Python praksu if __name__ == "__main__": koja osigurava da će se kalkulator pokrenuti samo kada se skripta izvršava direktno, a ne kada se uvozi kao modul. Ovo je dobra programska praksa koja omogućava fleksibilniju upotrebu koda.

```
# Pokretanje kalkulatora
if __name__ == "__main__":
    kalkulator()
```

Slika 34 - Pokretanje programa (zadatak br.2)

2.2.2 Python kod kalkulatora

```
def kalkulator():
    print("\nDobrodošli u osnovni kalkulator")
    print("Dostupne operacije: + (sabiranje), - (oduzimanje), * (množenje), / (djeljenje)")

    while True:
        try:
            # Unos podataka
            prvi_broj = float(input("Unesite prvi broj: "))
            operacija = input("Unesite operaciju (+, -, *, /): ")
            drugi_broj = float(input("Unesite drugi broj: "))

            # Izračunavanje rezultata
            if operacija == '+':
                rezultat = prvi_broj + drugi_broj
            elif operacija == '-':
                rezultat = prvi_broj - drugi_broj
            elif operacija == '*':
                rezultat = prvi_broj * drugi_broj
            elif operacija == '/':
                if drugi_broj == 0:
                    print("Greška: Ne može se djeliti nulom!")
                    continue
                rezultat = prvi_broj / drugi_broj
            else:
                print("Greška: Nepoznata operacija!")
                continue

            # Prikaz rezultata
            print(f"Rezultat: {prvi_broj} {operacija} {drugi_broj} = {rezultat}")

            # Pitamo korisnika da li želi novi izračun
            nastavak = input("\nDa li želite novi izračun? (da/ne): ").lower()
            if nastavak != 'da':
                print("Hvala što ste koristili kalkulator!")
                break

        except ValueError:
            print("Greška: Morate unjeti broj!")
        except Exception as e:
            print(f"Došlo je do greške: {e}")


# Pokretanje kalkulatora
if __name__ == "__main__":
    kalkulator()
```

Slika 35 - Kompletan kod zadatka br.2 u Notepad++

2.2.3 Rezultat izvršavanja kalkulatora

```
Dobrodošli u osnovni kalkulator
Dostupne operacije: + (sabiranje), - (oduzimanje), * (množenje), / (djeljenje)
Unesite prvi broj: 22
Unesite operaciju (+, -, *, /): -
Unesite drugi broj: 11
Rezultat: 22.0 - 11.0 = 11.0

Da li želite novi izračun? (da/ne): ne
Hvala što ste koristili kalkulator!
```

Slika 36 - Rezultat zadatka br.2

2.3 Provjera parnosti broja

Ovaj jednostavan program provjerava da li je korisnički unesen broj paran ili neparan.

2.3.1 Analiza logike za provjeru parnosti

Program počinje ispisom poruke korisniku pomoću funkcije print() kako bi znao da treba unijeti broj. Koristi se funkcija input() za unos vrijednosti, koja se odmah konvertuje u cijeli broj pomoću int().

```
broj = int(input("Unesite jedan cijeli broj: "))
```

Slika 37 - Unos broja od strane korisnika u zadatku br.3

Glavna logika provjere da li je broj paran nalazi se u uslovnom bloku if-else. Koristi se operator modulo % koji vraća ostatak pri dijeljenju. Ako je ostatak pri dijeljenju broja sa 2 jednak nuli (broj % 2 == 0), broj je paran, u suprotnom je neparan.

```
if broj % 2 == 0:
    print(f"Broj {broj} je paran.")
else:
    print(f"Broj {broj} je neparan.")
```

Slika 38 - Logika provjere parnosti u zadatku br.3

2.3.2 Kod za provjeru parnosti

```
def provjeri_parnost():  1 usage
    broj = int(input("Unesite jedan cijeli broj: "))

    if broj % 2 == 0:
        print(f"Broj {broj} je paran.")
    else:
        print(f"Broj {broj} je neparan.")


if __name__ == "__main__":
    provjeri_parnost()
```

Slika 39 - Kompletan kod zadatka br.3 u PyCharm IDE-u

2.3.3 Prikaz rezultata provjere parnosti

```
C:\Users\Admin\PycharmProjects\Pyt
Unesite jedan cijeli broj: 2
Broj 2 je paran.
```

Slika 40 - Rezultat zadatka br.3

2.4 Pronalaženje najvećeg broja u listi

Ovaj program omogućava korisniku da unese niz brojeva, a zatim pronalazi i prikazuje najveći broj u unesenoj listi.

2.4.1 Logika pronalaska najveće vrijednosti u listi

Program počinje tako što korisniku nudi unos više brojeva odvojenih razmakom, koristeći input() funkciju. Uneseni string se pomoću split() razdvaja u listu stringova, a zatim se svaki element konvertuje u cijeli broj pomoću map(int, ...).

```
brojevi = list(map(int, input("Unesite brojeve odvojene razmakom: ").split()))
```

Slika 41 - Unos liste brojeva od strane korisnika u zadatku br.4

Nakon konverzije unosa u listu brojeva, koristi se ugrađena funkcija max() koja automatski pronalazi najveću vrijednost u listi. Ova funkcija radi brzo i efikasno čak i za veće liste.

```
najveci = max(brojevi)
```

Slika 42 - Pronalaženje najveće vrijednosti pomoću funkcije max() u zadatku br.4

2.4.2 Kod za pronalazak najveće vrijednosti u listi

```
def najveci_broj_u_listi():  1 usage
    brojevi = list(map(int, input("Unesite brojeve odvojene razmakom: ").split()))
    najveci = max(brojevi)
    print(f"Najveći broj u listi je: {najveci}")

if __name__ == "__main__":
    najveci_broj_u_listi()
```

Slika 43 - Kompletan kod zadatka br.4 u PyCharm IDE-u

2.4.3 Prikaz rezultata pronalaska najvećeg broja

```
Unesite brojeve odvojene razmakom: 3 1 2 3 41 24 99 102
Najveći broj u listi je: 102
```

Slika 44 - Rezultat programa u zadatku br.4

2.5 Faktorijel broja sa “for” petljom

Ovaj program izračunava faktorijel unesenog cijelog broja koristeći **for** petlju.

2.5.1 Logika izračunavanja faktorijela for petljom

Program počinje unosom cijelog broja pomoću input() funkcije, koji se odmah konvertuje u int. Nakon toga, postavlja se početna vrijednost varijable faktorijel na 1 jer je to neutralni element za množenje.

Korištenjem for petlje, program prolazi kroz sve brojeve od 1 do unesenog broja (uključujući ga) i množi ih sa trenutnom vrijednošću varijable faktorijel. Na taj način se korak po korak gradi konačni rezultat.

```
for i in range(1, broj + 1):
    faktorijel *= i
```

Slika 45 - For petlja za izračunavanje faktorijela u zadatku br.5

Na kraju, rezultat se ispisuje korisniku pomoću f-string izraza koji jasno prikazuje ulazni broj i izračunatu vrijednost njegovog faktorijela.

2.5.2 Kod za izračunavanje faktorijela for petljom

```
def faktorijel_broja():  1 usage
    broj = int(input("Unesite cijeli broj: "))
    faktorijel = 1

    for i in range(1, broj + 1):
        faktorijel *= i

    print(f"Faktorijel broja {broj} je: {faktorijel}")

if __name__ == "__main__":
    faktorijel_broja()
```

Slika 46 - Kompletan kod zadatka br.5 u PyCharm IDE-u

2.5.3 Prikaz rezultata izračunavanja faktorijela

```
Unesite cijeli broj: 5
Faktorijel broja 5 je: 120
```

Slika 47 - Rezultat programa u zadatku br.5

2.6 Provjera anagrama između dvije riječi

Ovaj program provjerava da li su dvije unesene riječi međusobni anagrami.

2.6.1 Logika provjere anagrama između riječi

Program od korisnika traži da unese dvije riječi. Svaka se pretvara u mala slova pomoću .lower() kako bi poređenje bilo neosjetljivo na velika/mala slova. Zatim se pomoću sorted() funkcije sortiraju slova u obje riječi.

Ako sortirani nizovi slova izgledaju isto, to znači da su riječi anagrami sadrže ista slova u istom broju, samo u drugačijem redoslijedu.

2.6.2 Kod za provjeru anagrama

```
def provjera_anagrama():  1 usage
    rijec1 = input("Unesite prvu riječ: ").lower()
    rijec2 = input("Unesite drugu riječ: ").lower()

    if sorted(rijec1) == sorted(rijec2):
        print(f"'{rijec1}' i '{rijec2}' su anagrami.")
    else:
        print(f"'{rijec1}' i '{rijec2}' nisu anagrami.")

if __name__ == "__main__":
    provjera_anagrama()
```

Slika 48 - Kompletan kod zadatka br.6 u PyCharm IDE-u

2.6.3 Prikaz rezultata provjere anagrama

```
Unesite prvu riječ: mora
Unesite drugu riječ: roma
'mora' i 'roma' su anagrami.
```

Slika 49 - Rezultat programa u zadatku br.6

2.7 Provjera prostog broja

Ovaj program provjerava da li je uneseni broj prost, tj. djeljiv samo sa 1 i samim sobom.

2.7.1 Logika provjere prostog broja

Korisnik unosi broj pomoću input() funkcije. Ako je broj manji od 2, odmah se označava kao složen broj jer prosti brojevi počinju od 2.

Zatim se for petljom prolazi kroz sve brojeve od 2 do kvadratnog korijena unesenog broja (jer nije potrebno provjeravati dalje).

Ako se pronađe djelilac bez ostatka, broj nije prost i prekida se provjera; u suprotnom, broj je prost.

```
import math

broj = int(input("Unesite broj: "))
prost = True

if broj < 2:
    prost = False
else:
    for i in range(2, int(math.sqrt(broj)) + 1):
        if broj % i == 0:
            prost = False
            break

if prost:
    print("Broj je prost.")
else:
    print("Broj nije prost.")
```

Slika 50 - Kompletan kod zadatka br.7 u PyCharm IDE-u

2.8 Rad sa rječnikom (telefonski imenik)

Ovaj program omogućava korisniku da kreira jednostavan telefonski imenik koristeći rječnik u Pythonu, sa opcijama dodavanja, pretrage i ispisivanja svih kontakata.

2.8.1 Logika rada sa rječnikom

Program koristi rječnik (dict) gdje je ključ ime kontakta, a vrijednost njegov broj telefona. Korisniku se prikazuje meni sa opcijama: dodavanje novog kontakta, pretraga broja po imenu i ispis svih kontakata. Dodavanje kontakta vrši se unošenjem imena i broja, koji se spremaju u rječnik. Pretraga se vrši provjerom da li uneseno ime postoji u rječniku. Ispis svih kontakata koristi for petlju koja prolazi kroz sve parove ključ–vrijednost.

2.8.2 Kod za rad sa telefonskim imenikom

```
imenik = {}

while True:
    print("\n--- Telefonski imenik ---")
    print("1. Dodaj kontakt")
    print("2. Pretraži kontakt")
    print("3. Prikaži sve kontakte")
    print("4. Izlaz")

    izbor = input("Odaberite opciju: ")

    if izbor == "1":
        ime = input("Unesite ime: ")
        broj = input("Unesite broj telefona: ")
        imenik[ime] = broj
        print("Kontakt dodat.")
    elif izbor == "2":
        ime = input("Unesite ime za pretragu: ")
        if ime in imenik:
            print(f"Broj telefona: {imenik[ime]}")
        else:
            print("Kontakt nije pronađen.")
    elif izbor == "3":
        for ime, broj in imenik.items():
            print(f"{ime}: {broj}")
    elif izbor == "4":
        print("Zatvaranje imenika...")
        break
    else:
        print("Nepoznata opcija.")
```

Slika 51 - Kompletan kod zadatka br.8 u PyCharm IDE-u

2.9 Rad sa datotekama (čitanje i pisanje)

Ovaj program omogućava korisniku unos teksta koji se čuva u datoteci, a zatim čitanje i prikaz njenog sadržaja.

2.9.1 Logika rada sa datotekama

Program koristi open() funkciju sa modovima "w" (write) za pisanje i "r" (read) za čitanje. Korisnik unosi tekst koji se pomoću metode .write() upisuje u novu ili postojeću datoteku. Nakon toga, datoteka se ponovo otvara u režimu čitanja, njen sadržaj se dobija metodom .read() i prikazuje korisniku. Korišćen je with blok kako bi se datoteka automatski zatvorila nakon završetka operacija.

```
with open("tekst.txt", "w", encoding="utf-8") as fajl:
    unos = input("Unesite tekst za upis u datoteku: ")
    fajl.write(unos)

with open("tekst.txt", "r", encoding="utf-8") as fajl:
    sadrzaj = fajl.read()
    print("\nSadržaj datoteke:")
    print(sadrzaj)
```

Slika 52 - Kompletan kod zadatka br.9 u PyCharm IDE-u

Kada korisnik unese tekst „Ovo je test poruka“, program će ga upisati u datoteku „tekst.txt“.

Nakon čitanja, program će ispisati:

```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Sc
Unesite tekst za upis u datoteku: Ovo je test poruka

Sadržaj datoteke:
Ovo je test poruka
```

Slika 53 – Rezultat programa u zadatku br.9

2.10 Lambda funkcije – filtriranje i mapiranje liste

Ovaj program koristi lambda funkcije za rad sa listama, tačnije za filtriranje parnih brojeva i množenje svih elemenata liste sa dva.

2.10.1 Logika filtriranja i mapiranja pomoću lambda funkcija

Korišćenjem ugrađenih funkcija filter() i map() zajedno sa lambda izrazima, program prvo izdvoji sve parne brojeve iz liste koristeći uslov x % 2 == 0. Zatim, svaki element iz originalne liste se množi sa 2 korišćenjem funkcije map() i lambda izraza lambda x: x * 2. Rezultati oba koraka se pretvaraju nazad u listu pomoću funkcije list() kako bi se mogli ispisati.

2.10.2 Kod za filtriranje i mapiranje liste

```
brojevi = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

parni = list(filter(lambda x: x % 2 == 0, brojevi))
pomnozeni = list(map(lambda x: x * 2, brojevi))

print("Parni brojevi:", parni)
print("Brojevi pomnoženi sa 2:", pomnozeni)
```

Slika 54 - Kompletan kod zadatka br.10 sa lambda funkcijama u PyCharm IDE-u

Program će ispisati sljedeće:

```
C:\Users\Admin\PycharmProjects\PythonProject\.venv\Scripts\py
Parni brojevi: [2, 4, 6, 8, 10]
Brojevi pomnoženi sa 2: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Slika 55 - Rezultat programa za filtriranje i mapiranje liste

2.11 Generisanje prostih brojeva do n (Sito Eratostenovo)

Ovaj program generiše sve proste brojeve manje ili jednake od unesenog pozitivnog cijelog broja koristeći Sito Eratostenovo, poznatu metodu za pronalaženje prostih brojeva.

2.11.1 Logika Sita Eratostenovog

Program započinje pravljenjem liste istinitih vrijednosti (True) za sve brojeve od 2 do n, pretpostavljajući da su svi prosti. Zatim iterira kroz listu i za svaki broj koji je još označen kao prost (True), označava sve njegove djelioce kao složene (False). Na kraju, svi indeksi koji su ostali True predstavljaju proste brojeve. Ovaj pristup izbjegava provjeru djeljivosti svakog broja pojedinačno, čime je efikasniji od osnovnih metoda.

```
def sito_eratostenovo(n):  1 usage
    if n < 2:
        return []

    prosti = [True] * (n + 1)
    prosti[0], prosti[1] = False, False

    for i in range(2, int(n ** 0.5) + 1):
        if prosti[i]:
            for j in range(i * i, n + 1, i):
                prosti[j] = False

    return [i for i, je_prost in enumerate(prosti) if je_prost]

granica = int(input("Unesite gornju granicu za generisanje prostih brojeva: "))

rezultat = sito_eratostenovo(granica)
print(f"Prosti brojevi do {granica} su:")
print(rezultat)
```

Slika 56 - Kompletan kod zadatka br.11 generisanja prostih brojeva

2.12 Rad sa modulima (math, random)

Ovaj program demonstrira korišćenje Python modula math i random za izvođenje matematičkih operacija i generisanje slučajnih brojeva.

2.12.1 Logika korišćenja modula math i random

Modul math se koristi za izvođenje funkcija poput korijena (sqrt), eksponenta (pow) i zaokruživanja. Modul random omogućava generisanje nasumičnih brojeva i izbora slučajnih elemenata iz liste. U programu se prvo generiše lista slučajnih cijelih brojeva unutar zadatog opsega. Zatim se za svaki broj računa njegova kvadratna vrijednost i korijen koristeći funkcije iz math modula. Na kraju se nasumično bira jedan od brojeva iz liste i ispisuje.

```
import math
import random

brojevi = [random.randint( a: 1,  b: 100) for _ in range(5)]
print("Generisani brojevi:", brojevi)

for broj in brojevi:
    kvadrat = math.pow(broj,  y: 2)
    korijen = math.sqrt(broj)
    print(f"Broj: {broj}, Kvadrat: {kvadrat:.2f}, Korijen: {korijen:.2f}")

nasumicni = random.choice(brojevi)
print(f"Nasumično izabrani broj iz liste je: {nasumicni}")
```

Slika 57 - Kompletan kod zadatka br.12 u PyCharm IDE-u

2.12.2 Prikaz rezultata programa

Program će ispisati pet nasumično generisanih brojeva, njihove kvadrate i korijene sa dvije decimale, te nasumično izabrani broj iz liste, na primjer:

```
Generisani brojevi: [20, 93, 27, 70, 67]
Broj: 20, Kvadrat: 400.00, Korijen: 4.47
Broj: 93, Kvadrat: 8649.00, Korijen: 9.64
Broj: 27, Kvadrat: 729.00, Korijen: 5.20
Broj: 70, Kvadrat: 4900.00, Korijen: 8.37
Broj: 67, Kvadrat: 4489.00, Korijen: 8.19
Nasumično izabrani broj iz liste je: 70
```

Slika 58 - Rezultat programa u zadatku br.12

2.13 OOP – klasa Osoba (ime, godine, metoda za ispis)

Ovaj program prikazuje osnovnu implementaciju klase Osoba u Pythonu, sa atributima ime i godine, te metodom za ispis informacija o osobi.

2.13.1 Logika implementacije klase Osoba

Klasa Osoba definiše dva atributa: ime i godine, koji se postavljaju prilikom kreiranja objekta kroz konstruktor __init__. Metoda prikazi_podatke ispisuje ime i godine osobe na jasan i čitljiv način. Program kreira primjer objekta Osoba, poziva metodu za ispis i tako demonstrira osnove objektno-orijentisanog programiranja u Pythonu.

2.13.2 Kod klase Osoba

```
class Osoba:  1 usage
    def __init__(self, ime, godine):
        self.ime = ime
        self.godine = godine

    def prikazi_podatke(self):  1 usage
        print(f"Ime: {self.ime}, Godine: {self.godine}")

osoba1 = Osoba( ime: "Marko",  godine: 25)
osoba1.prikazi_podatke()
```

Slika 59 – Kompletan kod zadatka br.13 u PyCharm IDE-u

2.14 Rad sa pandas (osnovna analiza podataka)

Ovaj program koristi biblioteku pandas za učitavanje i analizu jednostavnog skupa podataka, prikazujući osnovne statističke informacije.

2.14.1 Logika rada sa pandas bibliotekom

Program učitava podatke iz CSV datoteke koristeći pandas.read_csv(). Nakon učitavanja, koristi se funkcija head() za prikaz prvih nekoliko redova podataka. Zatim se prikazuju osnovne statistike skupa podataka pomoću metode describe(), koja daje informacije poput srednje vrijednosti, standardne devijacije, minimuma i maksimuma. Ovaj zadatak uvodi rad sa tabelarnim podacima i osnovnu analizu.

2.14.2 Kod za osnovnu analizu podataka sa pandas

```
import pandas as pd

df = pd.read_csv('podaci.csv')

print("Prvih 5 redova podataka:")
print(df.head())

print("\nOsnovne statistike:")
print(df.describe())
```

Slika 60 - Kompletan kod zadatka br.14 u PyCharm IDE-u

2.14.3 Prikaz rezultata programa

Ako CSV sadrži podatke o prodaji proizvoda, program će prikazati prvih 5 redova i statistički pregled kolona kao što su količina, cijena i prihodi.

Na primjer:

Prvih 5 redova podataka:

	Proizvod	Količina	Cijena
0	Jabuka	10	1.20
1	Kruška	5	0.80
2	Banana	15	1.10
3	Narandža	12	1.30
4	Limun	7	1.00

2.15 Web scraping (dohvaćanje naslova sa vijesti)

Ovaj program koristi biblioteku requests za preuzimanje web stranice i BeautifulSoup za parsiranje HTML sadržaja, te dohvaća naslove vijesti sa odabrane stranice.

2.15.1 Logika web scrapinga

Program prvo šalje HTTP zahtjev prema web stranici koristeći requests.get(). Nakon što dobije HTML sadržaj, koristi BeautifulSoup za parsiranje i pronalaženje HTML elemenata koji sadrže naslove vijesti (npr. <h2>, <h3> ili elementi sa određenom klasom). Izvučeni naslovi se potom ispisuju korisniku. Ovaj zadatak uvodi rad sa vanjskim podacima i obradu HTML-a.

2.15.2 Kod za dohvaćanje naslova sa vijesti

```
import requests
from bs4 import BeautifulSoup

url = 'https://atvbl.rs/'

response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

naslovi = soup.find_all('h2')

print("Naslovi vijesti:")
for naslov in naslovi:
    print(naslov.get_text(strip=True))
```

Slika 61 - Kompletan kod zadatka br.15 u PyCharm IDE-u

Za pokretanje ovog programa potrebna je instalacija biblioteke BeautifulSoup (pip install beautifulsoup4) i requests (pip install requests).

Program će ispisati listu naslova vijesti pronađenih na stranici, na primjer:

Naslovi vijesti:

Nova tehnologija u razvoju

Ekonomski izvještaj za ovaj kvartal

Sport: Rezultati utakmica

ZAKLJUČAK

U ovom seminarskom radu kroz niz praktičnih zadataka prikazan je postupni razvoj programerskih vještina u Pythonu, od osnovnih koncepata do složenijih tema poput objektno-orijentisanog programiranja i rada sa eksternim modulima. Rad je pokazao kako jednostavna i čitljiva sintaksa Pythona omogućava efikasno rješavanje različitih problema, kao i važnost razumijevanja osnovnih algoritamskih pristupa i struktura podataka.

Takođe, naglašena je primjena standardnih i dodatnih biblioteka koje značajno proširuju mogućnosti jezika, što predstavlja ključni element u modernom razvoju softvera. Kroz realizaciju zadataka, stečena su praktična znanja koja su osnova za dalje usavršavanje i primjenu Pythona u različitim oblastima, uključujući web razvoj, analizu podataka i automatizaciju.

Ovaj rad potvrđuje da je Python zbog svoje jednostavnosti, fleksibilnosti i široke upotrebljivosti idealan izbor kako za početnike, tako i za iskusne programere koji žele brzo i efektivno razvijati softverska rješenja.

POPIS SLIKA

CITATNI IZVORI

[1] M. Lutz, Learning Python, 5th ed., Sebastopol: O'Reilly Media, 2013.

[2] E. Martelli, Python in a Nutshell, 2nd ed., Sebastopol: O'Reilly Media, 2006.