

PANEVROPSKI UNIVERZITET "APEIRON"
U BANJOJ LUCI
FAKULTET INFORMACIONIH TEHNOLOGIJA

Seminarski rad iz predmeta Skriptni jezici i programiranje

NEKI ZADACI U PROGRAMSKOM JEZIKU "PYTHON"
SA PRIPADAJUĆIM RJEŠENJIMA I OBJAŠNJENJIMA

Profesor

Prof. dr Negovan Stamenković

Kandidat

Siniša Božić

192-20/RITP

Sadržaj

1	UVOD.....	3
2	ZADACI	4
2.1	RAČUNANJE PROSJEKA	4
2.2	PRORAČUN VIJEKA (STOLJEĆA)	5
2.3	IZRAČUN BMI (INDEKSA TJELESNE MASE)	6
2.4	BROJANJE IKS I OKS	7
2.5	PRIJATELJ ILI NEPRIJATELJ?	8
2.6	JEDINICE I NULE.....	9
2.7	BROJANJE PONAVLJANJA KARAKTERA	11
2.8	IZOGRAMI	12
2.9	RAST POPULACIJE	14
2.10	DETEKTOVANJE PROSTIH BROJEVA	16
2.11	PRONALAŽENJE ULJEZA	18
2.12	DIJELJENJE STRINGOVA	19
2.13	PREBROJAVANJE DUPLIKATA	20
2.14	KONVERZIJA IZ RGB U HEX	22
2.15	HANOJSKE KULE.....	24
3	ZAKLJUČAK	29
4	LITERATURA I DRUGI IZVORI	30

1 UVOD

Kako predmet Skriptni jezici i programiranje predviđa izradu seminarskog rada sa primjerima u programskom jeziku Python, u ovom radu sam obuhvatio 15 problema rangiranih od lakših ka težim. Svi problemi su jasno opisani, sa pratećim primjerima, diskusijom mogućih rješenja i objašnjenjima primjenjenog programskog koda, kao i sa kompletnim kodom.

Problemi su preuzeti sa stranice Codewars.com. U pitanju je platforma za kompetitivno kodiranje koja se koristi za praksu i vježbanje raznih problema u nekoliko desetaka programskih jezika. Korisnik bira programski jezik i probleme za vježbanje; po uspješnom rješavanju dobija mogućnost da vidi rješenja ostalih korisnika kao i pripadajući broj poena kojim se njegov nalog podiže u rangu. Rješenje problema je moguće vidjeti i bez korisnikovog ulaza ali u tom slučaju korisnički nalog ne dobija nikakve poene. Time se osigurava da onaj nalog koji je dobio poene, njegov korisnik je stvarno i riješio date probleme¹.

Problemi su prevedeni sa engleskog jezika. Svaki problem se sastoji od kreiranja funkcije koja treba da rješava problem. Rješenje korisnika se testira putem testnih slučajeva, koji obuhvataju desetine ili stotine testova koje programsko rješenje treba uspješno da prođe. U izabranih 15 problema, primjenjuju se osnovne tehnike u programskom jeziku Python (i inače u programiranju generalno), kao što su uslovna grananja (if naredba), iteracije (for, while petlje), od struktura podataka se uglavnom koriste liste i set. Neke naprednije tehnike obuhvataju korištenje regex, množenje lista, upotrebu rekurzije, kao i niz detalja kao što su konverzije iz jednog u drugi tip podataka te korištenje mnogobrojnih ugrađenih funkcija.

Navedene, kao i bilo koje druge probleme je moguće riješiti na više načina, a prezentovana rješenja su rezultat autorovog rada i istraživanja, te ih je moguće riješiti i na druge načine.

¹ Autorov profil: <https://www.codewars.com/users/sbozich>

2 ZADACI

2.1 RAČUNANJE PROSJEKA

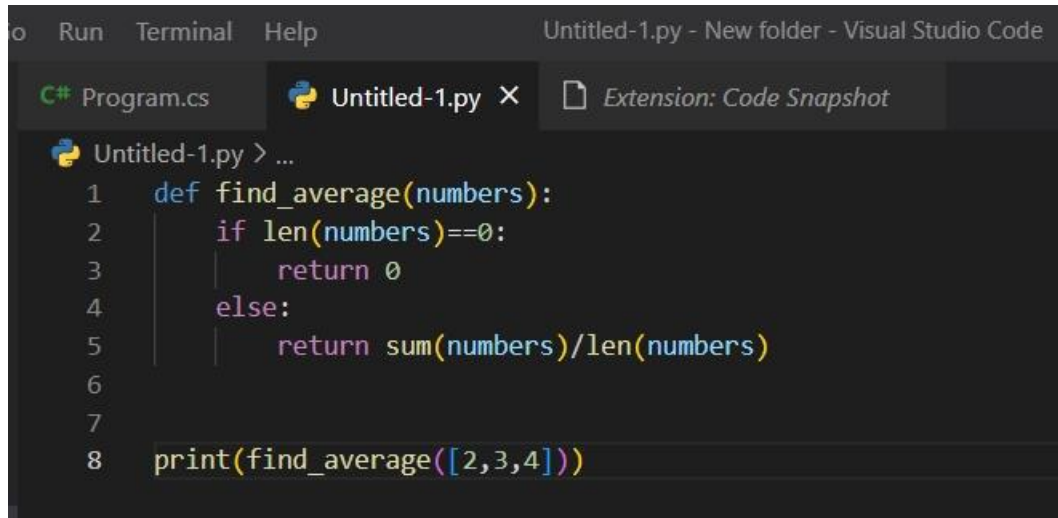
Napisati funkciju koja računa prosjek brojeva u datoj listi. Ako je data prazna lista, program treba da vraća 0.

Diskusija:

Korištenjem if uslovnog račvanja, odvojiti slučaj za praznu listu. Samo računanje prosjeka se može uraditi iteracijom elemenata u listi, njihovim zbrajanjem i dijeljenjem sa brojem elemenata, međutim Python ima funkcijom sum koji automatski sabira elemente liste pa je ta mogućnost u ovom primjeru iskorištena. Dobijena suma se potom dijeli sa dužinom liste koja je određena funkcijom len koja prebrojava elemente u listi. Kad se funkcija pozove sa odgovarajućim argumentima, računa prosjek dajući izlazni podatak u formatu broja sa decimalnim mjestima.

Rješenje:

```
def find_average(numbers):  
    if len(numbers)==0:  
        return 0  
    else:  
        return sum(numbers)/len(numbers)
```

A screenshot of the Visual Studio Code IDE interface. The top bar shows 'Run', 'Terminal', and 'Help' menus, along with the file name 'Untitled-1.py - New folder - Visual Studio Code'. The editor window displays a Python script with the following code:

```
1 def find_average(numbers):  
2     if len(numbers)==0:  
3         return 0  
4     else:  
5         return sum(numbers)/len(numbers)  
6  
7  
8 print(find_average([2,3,4]))
```

The code is color-coded: keywords like 'def', 'if', 'else', 'return', and 'print' are in blue, and string literals are in red. The list [2,3,4] is also highlighted in red.

Prilog br. 1: Ispis programa u IDE Visual Studio Code

2.2 PRORAČUN VIJEKA (STOLJEĆA)

Prvo stoljeće počinje od godine 1 i završava sa godinom 100; drugo stoljeće počinje od godine 101 i završava sa godinom 200, i tako dalje. Ako je data godina, vratiti kojem stoljeću ona pripada. Primjeri:

1705: 18

1900: 19

1601: 17

2000: 20

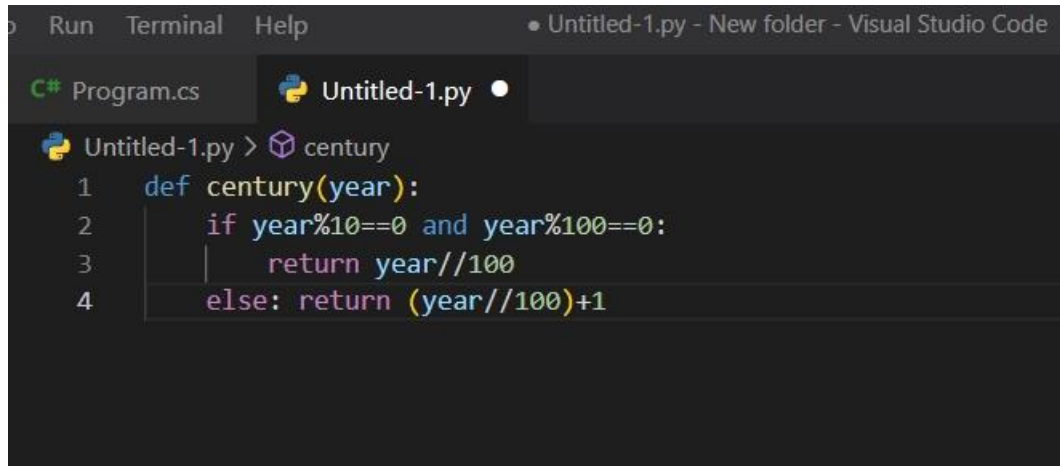
Diskusija:

Očigledno se razlikuju dva slučaja, odnosno kada je godina okrugla tipa 900, 1000, 1100, jer te godine pripadaju 9., 10. i 11. vijeku respektivno. Ostale godine, tipa 901, 1001, 1001 pripadaju 10., 11. i 12. vijeku respektivno, pa treba iznaći takvo rješenje koje će prepoznavati ova dva slučaja i primjenjivati drugačiju logiku.

U prvom slučaju treba nam mogućnost kako prepoznati okruglu godinu. Kao jednostavan izbor nameće se korištenje modula, odnosno prikazivanja ostatka dijeljenja nekim brojem. Ako je modulo dijeljenja sa 10 i 100 date godine jednak nuli (odnosno pokriva slučajeve do i preko godine 1000.), onda takve godine treba da podijelimo, koristeći // dijeljenje koje vraća cijeli broj, dok ostale godine treba također da podijelimo ali da dodamo jedan na povratni podatak jer te godine pripadaju idućem stoljeću.

Rješenje:

```
def century(year):  
    if year%10==0 and year%100==0:  
        return year//100  
    else: return (year//100)+1
```

A screenshot of the Visual Studio Code editor interface. The top bar shows 'Run', 'Terminal', and 'Help' menus, along with the file name 'Untitled-1.py - New folder - Visual Studio Code'. The editor window displays a Python file named 'Untitled-1.py' with a function 'century' defined. The function takes a 'year' parameter and returns the century based on the year. The code is as follows:

```
1 def century(year):
2     if year%100==0 and year%1000==0:
3         return year//1000
4     else: return (year//100)+1
```

Prilog br. 2: Ispis programa u IDE Visual Studio Code

2.3 IZRAČUN BMI (INDEKSA TJELESNE MASE)

Napisati funkciju koja računa BMI, odnosno indeks tjelesne mase, po formuli:

$$\text{BMI} = \text{težina} / \text{visina}^2$$

Neki primjeri vrijednosti:

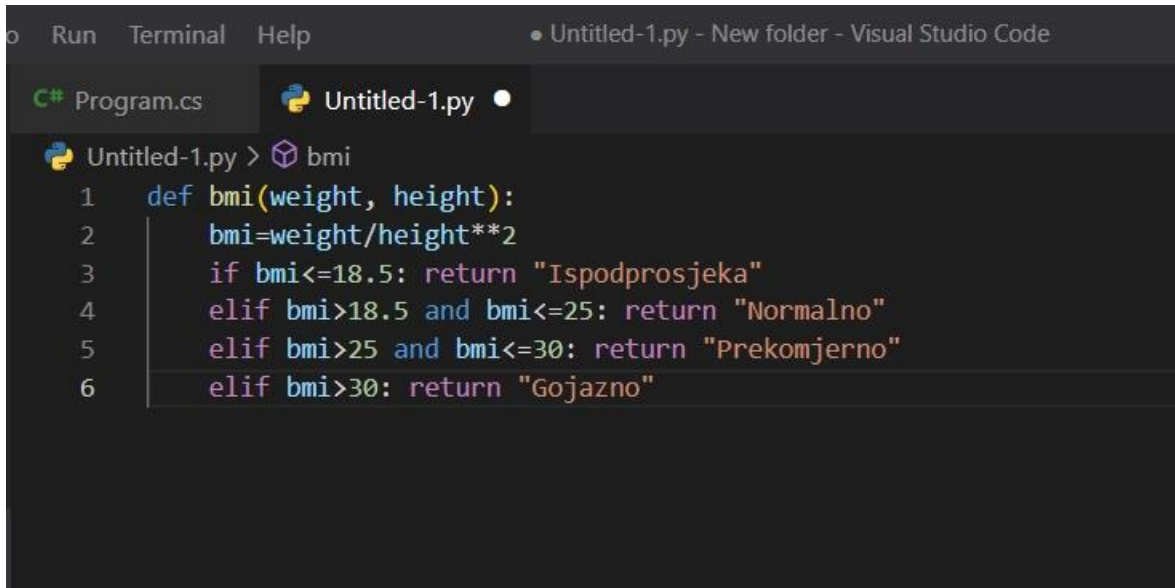
- ako $\text{bmi} \leq 18.5$ return "Ispodprosjeaka"
- ako $\text{bmi} \leq 25.0$ return "Normalno"
- ako $\text{bmi} \leq 30.0$ return "Prekomjerno"
- ako $\text{bmi} > 30$ return "Gojazno"

Diskusija:

Ovaj jednostavan problem se rješava korištenjem višestrukih grananja odnosno uslovnih if/elif petlji, shodno postavljenim vrijednostima.

Rješenje:

```
def bmi(weight, height):
    bmi=weight/height**2
    if bmi<=18.5: return "Ispodprosjeaka"
    elif bmi>18.5 and bmi<=25: return "Normalno"
    elif bmi>25 and bmi<=30: return "Prekomjerno"
    elif bmi>30: return "Gojazno"
```

The image shows a screenshot of the Visual Studio Code IDE. At the top, there's a menu bar with 'Run', 'Terminal', and 'Help'. Below it, the file explorer shows 'Untitled-1.py' as the active file. The editor area contains a Python script for calculating BMI. The script defines a function 'bmi(weight, height)' which calculates the BMI as 'weight/height**2'. It then uses a series of 'if' and 'elif' statements to return different strings based on the BMI value: 'Ispodprosjeaka' for BMI <= 18.5, 'Normalno' for 18.5 < BMI <= 25, 'Prekomjerno' for 25 < BMI <= 30, and 'Gojazno' for BMI > 30. The line numbers 1 through 6 are visible on the left side of the code editor.

```
1 def bmi(weight, height):
2     bmi=weight/height**2
3     if bmi<=18.5: return "Ispodprosjeaka"
4     elif bmi>18.5 and bmi<=25: return "Normalno"
5     elif bmi>25 and bmi<=30: return "Prekomjerno"
6     elif bmi>30: return "Gojazno"
```

Prilog br. 3: Ispis programa u IDE Visual Studio Code

2.4 BROJANJE IKS I OKS

Treba provjeriti da li string ima isti broj karaktera "x" i "o". Funkcija treba da vraća logičku vrijednost i da bude neosjetljiva na velika ili mala slova. String može sadržati bilo koji karakter.

Primjeri:

XO("ooxx") => true

XO("xooxx") => false

XO("ooxXm") => true

XO("zpzpzpp") => true // when no 'x' and 'o' is present should return true

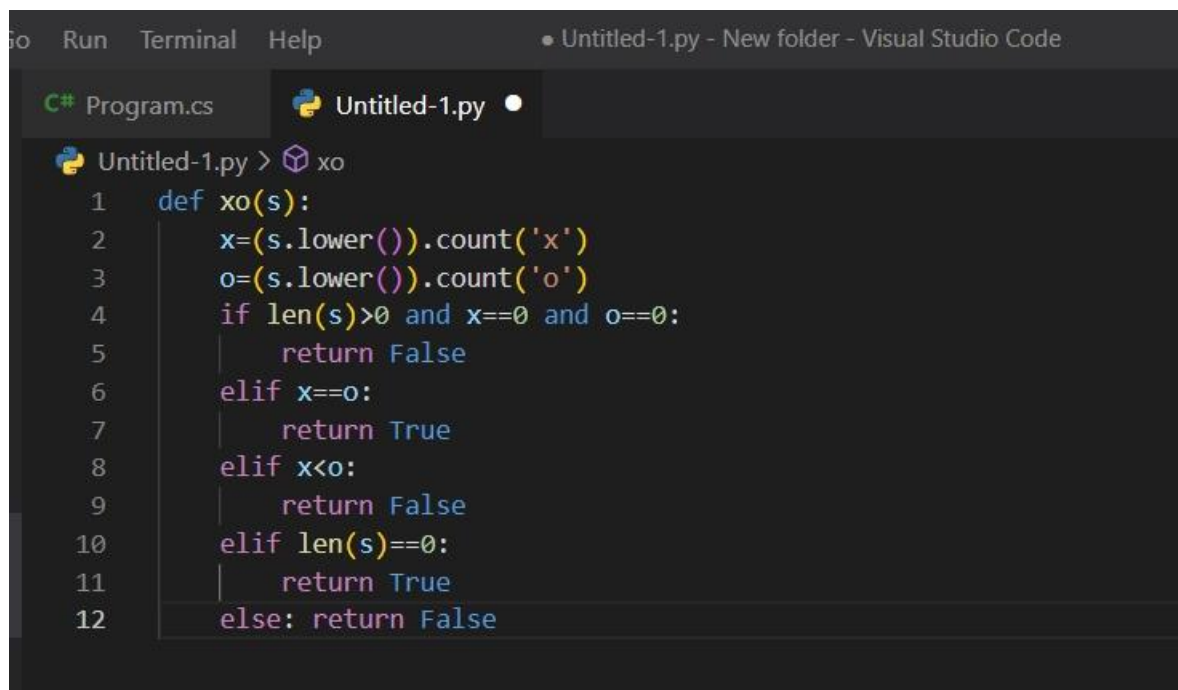
XO("zzoo") => false

Diskusija:

Shodno zahtjevima problema, jedan od mogućih pristupa rješavanju jeste kreiranje dvije varijable, koje će nositi podatke o broju ponavljanja karaktera "x" i "o", pritom koristeći lower funkciju koja konvertuje sva slova u mala. Potom koristeći uslovno grananje if/elif kreirati slučajeve u skladu sa zahtjevima.

Rješenje:

```
def xo(s):
    x=(s.lower()).count('x')
    o=(s.lower()).count('o')
    if len(s)>0 and x==0 and o==0:
        return False
    elif x==o:
        return True
    elif x<o:
        return False
    elif len(s)==0:
        return True
    else: return False
```

A screenshot of the Visual Studio Code IDE interface. The top bar shows 'Run', 'Terminal', and 'Help' menus, along with the file name 'Untitled-1.py - New folder - Visual Studio Code'. The editor window displays a Python file named 'Untitled-1.py' with the following code:

```
1 def xo(s):
2     x=(s.lower()).count('x')
3     o=(s.lower()).count('o')
4     if len(s)>0 and x==0 and o==0:
5         return False
6     elif x==o:
7         return True
8     elif x<o:
9         return False
10    elif len(s)==0:
11        return True
12    else: return False
```

Prilog br. 4: Ispis programa u IDE Visual Studio Code

2.5 PRIJATELJ ILI NEPRIJATELJ?

Kreirati program koji filtrira listu stringova i vraća listu sa stringovima okarakterisanim kao prijatelji. Kriterij za prijatelja je ako ime, odnosno string ima tačno 4 karaktera, u suprotnom je neprijatelj.

Primjer:

```
["Ryan", "Kieran", "Jason", "Yous"] = ["Ryan", "Yous"]
```



```
["Ryan", "Kieran", "Mark"] = ["Ryan", "Mark"]
```

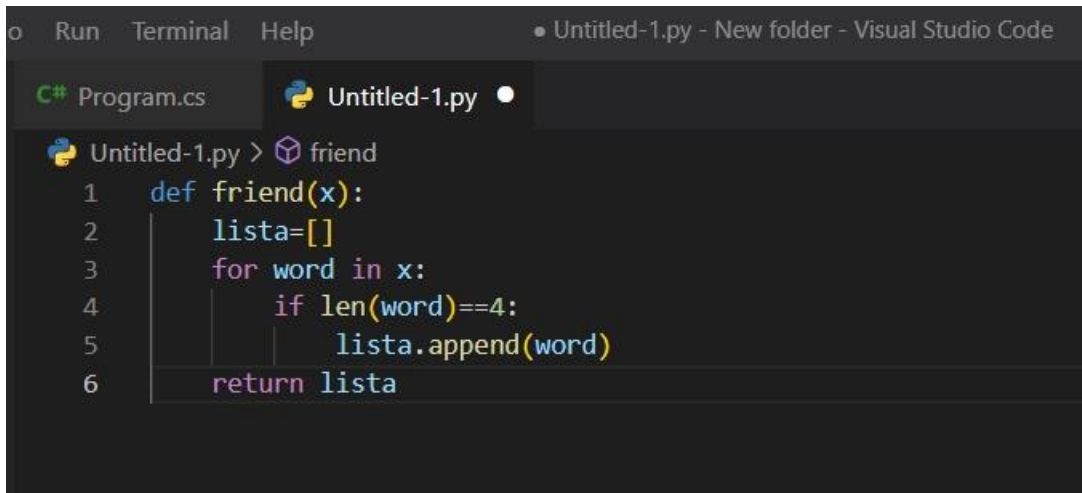
Zadržati originalni redoslijed stringova u listi.

Diskusija:

Problem se jednostavno rješava tako što se iteracijom karaktera u ulaznoj listi, filtriranjem prema njihovoj dužini, dodjeljuje element koji ispunjava zahtjev da je dužine od 4 karaktera. Potom se vraća data lista.

Rješenje:

```
def friend(x):  
    lista=[]  
    for word in x:  
        if len(word)==4:  
            lista.append(word)  
    return lista
```



Prilog br. 5: Ispis programa u IDE Visual Studio Code

2.6 JEDINICE I NULE

Za datu listu jedinica i nula, konvertovati ekvivalentnu binarnu vrijednost u cjelobrojnu varijablu.

Na primjer, [0,0,0,1] prikazano kao 0001 je binarna reprezentacija broja 1.

```
[0, 0, 0, 1] : 1
```

```
[0, 0, 1, 0] : 2
[0, 1, 0, 1] : 5
[1, 0, 0, 1] : 9
[0, 0, 1, 0] : 2
[0, 1, 1, 0] : 6
[1, 1, 1, 1] : 15
[1, 0, 1, 1] : 11
```

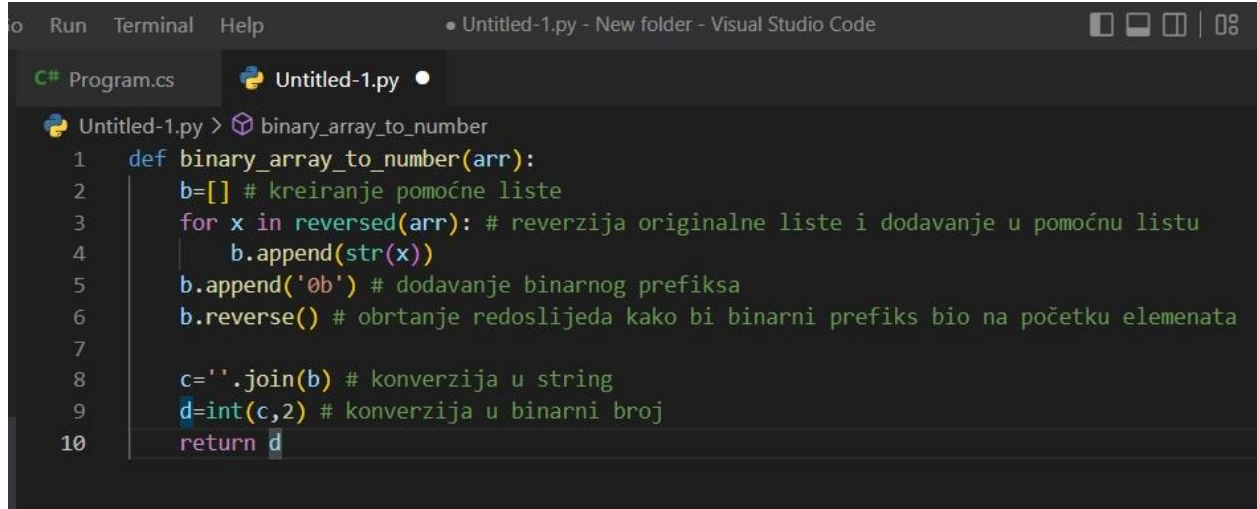
Diskusija:

Ulazni podatak je lista cijelih brojeva, odnosno nula i jedinica. Rješavanje komplikuje činjenica da Python prikazuje binarne brojeve sa prefiksom "0b", odnosno nije moguće jednostavno konvertovati ulazne podatke u binarni broj već im je potrebno dodati ovaj prefiks na početak svakog elementa i onda konvertovati u binarni broj. Ako se koristi lista, naredba `append` dodaje elemente na kraj liste, a ne na početak. Iz tog razloga, koristeći pomoćnu listu, izvršena je iteracija elemenata iz ulazne liste, brojevi konvertovani u string (jer prefiks "0b" se ne može prikazati brojevima) njihova reverzija, odnosno npr. 0,0,0,1 je postalo '1','0','0','0', potom je dodano na kraj "0b", tj. '1','0','0','0','0b'. Dobijeni rezultat treba potom obrnuti ('0b', '0', '0', '0', '1') izvršiti konverziju u string (0b0001) a potom konverziju u binarni broj.

Rješenje:

```
def binary_array_to_number(arr):
    b=[] # kreiranje pomoćne liste
    for x in reversed(arr): # reverzija originalne liste i
        dodavanje u pomoćnu listu
        b.append(str(x))
    b.append('0b') # dodavanje binarnog prefiksa
    b.reverse() # obrtanje redoslijeda kako bi binarni prefiks
    bio na početku elemenata

    c=''.join(b) # konverzija u string
    d=int(c,2) # konverzija u binarni broj
    return d
```



```

o Run Terminal Help • Untitled-1.py - New folder - Visual Studio Code
C# Program.cs • Untitled-1.py
Untitled-1.py > binary_array_to_number
1 def binary_array_to_number(arr):
2     b=[] # kreiranje pomoćne liste
3     for x in reversed(arr): # reverzija originalne liste i dodavanje u pomoćnu listu
4         b.append(str(x))
5     b.append('0b') # dodavanje binarnog prefiksa
6     b.reverse() # obrtanje redoslijeda kako bi binarni prefiks bio na početku elemenata
7
8     c=''.join(b) # konverzija u string
9     d=int(c,2) # konverzija u binarni broj
10    return d

```

Prilog br. 6: Ispis programa u IDE Visual Studio Code

2.7 BROJANJE PONAVLJANJA KARAKTERA

Napisati funkciju koja će brojati istovjetne karaktere po sljedećem pravilu:

```

accum("abcd") -> "A-Bb-Ccc-Dddd"
accum("RqaEzty") -> "R-Qq-Aaa-Eeee-Zzzzz-Tttttt-Yyyyyyy"
accum("cWAt") -> "C-Ww-Aaa-Tttt"

```

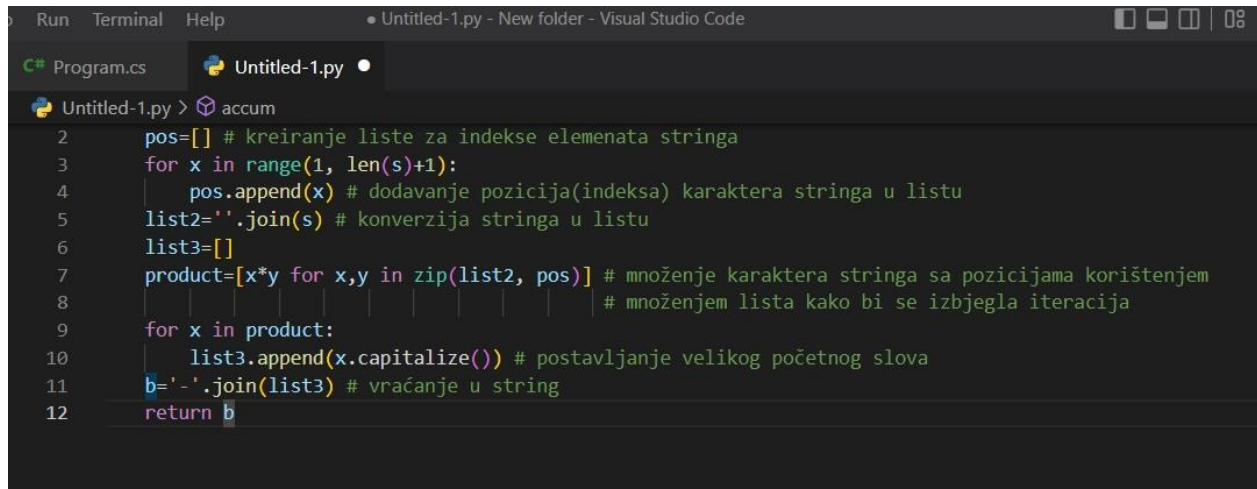
Parametar funkcije `accum` je string koji uključuje karaktere engleskog alfabeta ispisane velikim ili malim slovima.

Diskusija:

Funkcija treba da broji karaktere u stringu shodno poziciji, odnosno indeksnom elementu na kome se nalazi. Problem se svodi na detektovanje pozicije na kojem se dati karakter nalazi i množenje indeksa i karaktera. Odabrano rješenje uključuje kreiranje liste u koju će se iteracijom stringa dodati pozicije karaktera. Ako se ulazni string konvertuje u drugu listu, problem se svodi na množenje dvije liste. Na primjer, ako se `lista=['a ', 'b ', 'c ']` pomnoži sa `lista2=[1,2,3]` dobija se `['a ', 'b ', 'cc ']`, što je rezultat koji želimo dobiti. Množenje lista je moguće zip metodom i korištenjem list comprehension, što je dati izraz `[x*y for x,y in zip(lista2, pos)]`. Na kraju treba postaviti početno veliko slovo i konvertovati natrag listu u string koji funkcija vraća.

Rješenje:

```
def accum(s):
    pos=[] # kreiranje liste za indekse elemenata stringa
    for x in range(1, len(s)+1):
        pos.append(x) # dodavanje pozicija(indeksa) karaktera
stringa u listu
    list2=''.join(s) # konverzija stringa u listu
    list3=[]
    product=[x*y for x,y in zip(list2, pos)]
# množenje karaktera stringa sa pozicijama korištenjem
# množenja lista kako bi se izbjegla iteracija
    for x in product:
        list3.append(x.capitalize())
# postavljanje velikog početnog slova
    b='-'.join(list3) # vraćanje u string
    return b
```



```
Run Terminal Help • Untitled-1.py - New folder - Visual Studio Code
C# Program.cs Untitled-1.py
Untitled-1.py > accum
2 pos=[] # kreiranje liste za indekse elemenata stringa
3 for x in range(1, len(s)+1):
4     pos.append(x) # dodavanje pozicija(indeksa) karaktera stringa u listu
5 list2=''.join(s) # konverzija stringa u listu
6 list3=[]
7 product=[x*y for x,y in zip(list2, pos)] # množenje karaktera stringa sa pozicijama korištenjem
8                                     # množenjem lista kako bi se izbjegla iteracija
9 for x in product:
10     list3.append(x.capitalize()) # postavljanje velikog početnog slova
11 b='-'.join(list3) # vraćanje u string
12 return b
```

Prilog br. 7: Ispis programa u IDE Visual Studio Code

2.8 IZOGRAMI

Izogram je riječ koja nema ponavljajućih karaktera. Implementirati funkciju koja utvrđuje da li je string izogram. String će imati samo slova. Prazan string će se smatrati izogramom. Ignorirati velika i mala slova.

Primjer:

"Dermatoglyphics" --> true

"aba" --> false

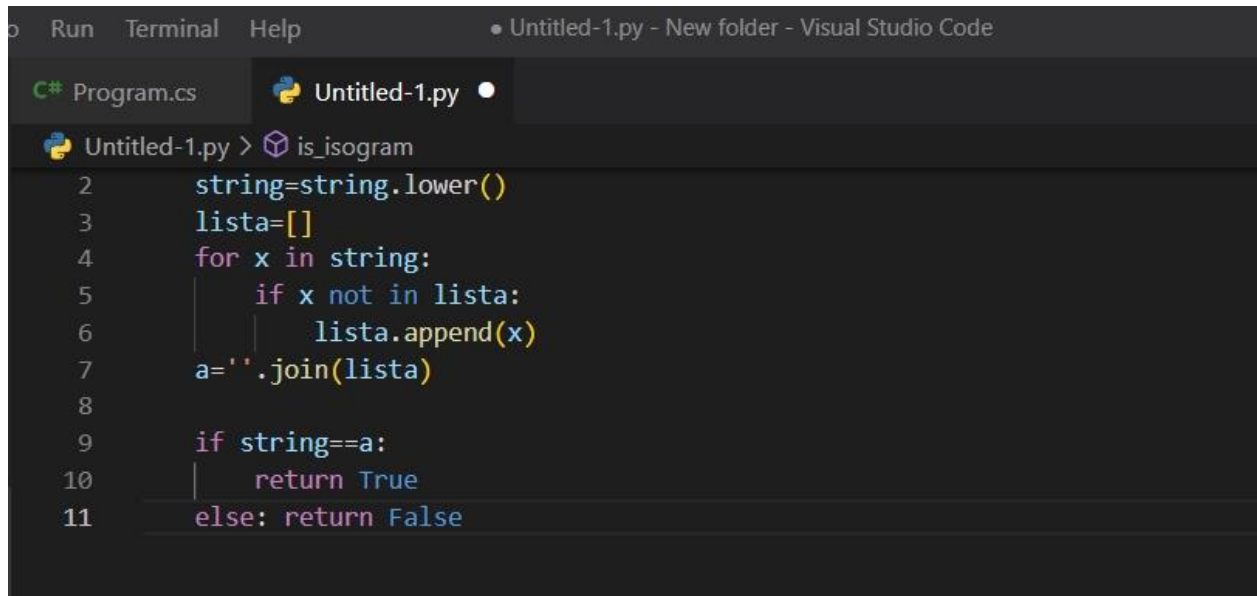
"moOse" --> false

Diskusija:

Na početku treba odmah koristiti lower metod odnosno konvertovati string u mala slova, shodno zahtjevu zadatka. Potom se kreira lista, u koju se ubacuju one vrijednosti stringa koje nisu prethodno u listi, čime se izbjegava dodavanje duplikata. Lista se potom konvertuje u string, i ako je tako dobijen string jednak ulaznom, to znači da u početnom stringu nije bilo duplikata pa se vraća True, u suprotnom False.

Rješenje:

```
def is_isogram(string):  
    string=string.lower()  
    lista=[]  
    for x in string:  
        if x not in lista:  
            lista.append(x)  
    a=''.join(lista)  
  
    if string==a:  
        return True  
    else: return False
```

A screenshot of the Visual Studio Code IDE interface. The top bar shows 'Run', 'Terminal', and 'Help' menus, along with the file name 'Untitled-1.py - New folder - Visual Studio Code'. The editor window displays a Python file named 'Untitled-1.py' with the following code:

```
1  def is_isogram(string):  
2      string=string.lower()  
3      lista=[]  
4      for x in string:  
5          if x not in lista:  
6              lista.append(x)  
7      a=''.join(lista)  
8  
9      if string==a:  
10         return True  
11     else: return False
```

The code is syntax-highlighted, and the line numbers 1 through 11 are visible on the left side of the editor.

Prilog br. 8: Ispis programa u IDE Visual Studio Code

2.9 RAST POPULACIJE

U nekom malom gradu populacija je $p_0=1000$ na početku godine. Populacija raste konstantno 2% godišnje i pored toga 50 novih stanovnika se godišnje doseljavaju. Koliko godina je potrebno da prođe da bi grad imao barem 1200 stanovnika?

Na kraju 1. godine:

$$1000 + 1000 * 0.02 + 50 \Rightarrow 1070 \text{ stanovnika}$$

Na kraju 2. godine:

$$1070 + 1070 * 0.02 + 50 \Rightarrow 1141 \text{ stanovnika}$$

Na kraju 3. godine:

$$1141 + 1141 * 0.02 + 50 \Rightarrow 1213 \text{ stanovnika}$$

Generalno uzevši, pri datim parametrima p_0 , `percent`, `aug` (doseljenici ili odseljenici tokom godine respektivno), p (ciljna populacija), funkcija treba da vraća broj godina koje su potrebne da prođu kako bi populacija dosegla parametar p .

`aug` je cijeli broj, `percent` je pozitivan ili null broj sa decimalnim mjestima, p_0 i p su pozitivni cijeli brojevi.

Primjer:

```
nb_year(1500, 5, 100, 5000) -> 15
```

```
nb_year(1500000, 2.5, 10000, 2000000) -> 10
```

Napomena: ne zaboraviti konvertovati argument `percent` kao procenat tokom pozivanja funkcije: ako je `percent=2`, konvertovati ga u 0,02.

Diskusija:

U ovom problemu je cilj izračunati broj godina za date ulazne podatke. U cilju boljeg razumijevanja problema, isti se može ubaciti u program za tablične proračune i dobiti slijedeće podatke za proračun dva naznačena slučaja:

nb_year(1000,2,20,1200)				
Godina	Početna populacija	Prirast %	Prirast abs	Populacija na kraju godine
1	1000	20	50	1070
2	1070	21	50	1141
3	1141	23	50	1214
nb_year(1500,5,100,5000)				
1	1500	75	100	1675
2	1675	84	100	1859
3	1859	93	100	2052
4	2052	103	100	2255
5	2255	113	100	2468
6	2468	123	100	2691
7	2691	135	100	2926
8	2926	146	100	3172
9	3172	159	100	3431
10	3431	172	100	3703
11	3703	185	100	3988
12	3988	199	100	4287
13	4287	214	100	4601
14	4601	230	100	4931
15	4931	247	100	5278

Prilog br. 9: Proračun godina za nb_year(1000,2,20,1200) i nb_year(1500,5,100,5000)

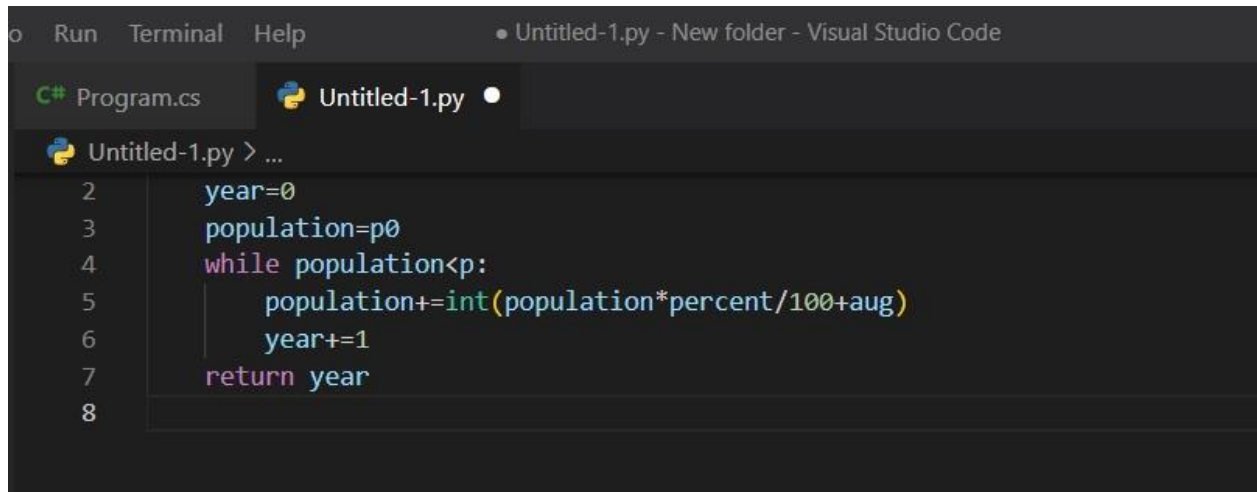
Sa matematičke strane, problem je lako riješiti računanjem podataka za svaku godinu, međutim u programerskom smislu je izazov kako postaviti postavku jer je nepoznata vrijednost godina, odnosno iterativna granica do koje vrijednosti iterisati. Iz tog razloga je nemoguće koristiti for petlju, ali je moguće koristiti while petlju. Program se rješava uz deklarisanje nekoliko pomoćnih varijabli, varijabla `year` će sadržavati broj godina dok varijabla `population` preuzima početnu vrijednost `p0` varijable, te se postavlja uslov:

Dok je vrijednost populacije manja od ciljne vrijednosti, sabrati u varijablu `populacija` prirast populacije u relativnom i apsolutnom iznosu; iterativnim koracima dok se izvršava while petlja će se u ovu varijablu sabirati vrijednosti rasta populacije sve dok ne premaše ciljnu vrijednost, u tom momentu petlja izlazi. Svaki iterativni korak povećava vrijednost varijable `year` za jedan. Kada petlja prestane, funkcija vraća vrijednost varijable `year`.

Rješenje:

```
def nb_year(p0, percent, aug, p):
```

```
year=0
population=p0
while population<p:
    population+=int (population*percent/100+aug)
    year+=1
return year
```



Prilog br. 10: Ispis programa u IDE Visual Studio Code

2.10 DETEKTOVANJE PROSTIH BROJEVA

Kreirati funkciju koja uzima kao argument cijeli broj i vraća logičku vrijednost zavisno od toga da li je broj prost ili ne.

Prost broj je prirodni pozitivni broj koji nema drugih djelioca osim 1 i samog sebe.

Zahtjevi:

Može se pretpostaviti da će ulazni broj biti cjelobrojni. Ne može se pretpostaviti da će broj uvijek biti pozitivan.

Brojevi će ići do 2^{31} , u zavisnosti od odabranog programskog jezika. Iteracija sve do n , ili $n/2$ biće prespora da bi se program izvršio.

Primjeri:

```
is_prime(1): False
is_prime(2): True
is_prime(-1): False
```


Diskusija:

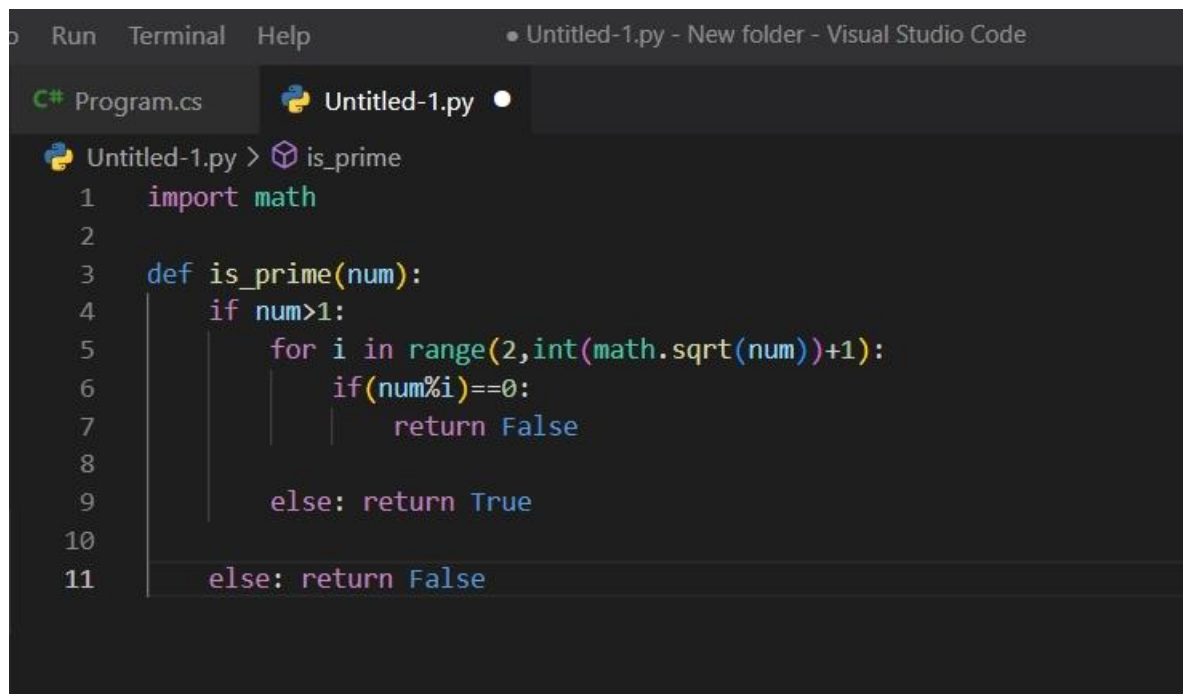
Rješenje ovog problema se može postići izučavanjem uslova za generisanje prostih brojeva. Najjednostavniji način je provjeravanje brojeva u rangu od broja 2 do njegovog kvadratnog korijena – taj metod se naziva prebrajanje djelilaca. Na primjer, ako utvrdjemo da li je broj 11 prost, provjerićemo da li je ostatak njegovog dijeljenja sa 2 nula za brojeve od 2 pa do $\sqrt{11}$. Mogli bismo provjeravati sve brojeve od 2 do n , međutim to bi bilo presporo, u skladu sa postavkama zadatka. U samom programskom kodu se importuje klasa `math` koja omogućava izračun kvadratnog korijena.

Rješenje:

```
import math

def is_prime(num):
    if num>1:
        for i in range(2,int(math.sqrt(num))+1):
            if(num%i)==0:
                return False
                break
        else: return True

    else: return False
```

The image is a screenshot of the Visual Studio Code IDE. At the top, there's a window title bar that says "Untitled-1.py - New folder - Visual Studio Code". Below that, there's a tab bar with two tabs: "C# Program.cs" and "Untitled-1.py". The "Untitled-1.py" tab is active. The main editor area shows the Python code for the `is_prime` function. The code is as follows:

```
1 import math
2
3 def is_prime(num):
4     if num>1:
5         for i in range(2,int(math.sqrt(num))+1):
6             if(num%i)==0:
7                 return False
8
9         else: return True
10
11     else: return False
```

Prilog br. 11: Ispis programa u IDE Visual Studio Code

2.11 PRONALAZENJE ULJEZA

Data je lista sa minimalnom dužinom od 3 elementa koja sačinjava cijele brojeve. Lista je ili potpuno sačinjena od neparnih ili od parnih brojeva, sa izuzetkom jednog elementa. Potrebno je pronaći taj element.

Primjeri:

```
[2, 4, 0, 100, 4, 11, 2602, 36]
```

Treba da vraća: 11 (jedini neparan broj)

```
[160, 3, 1719, 19, 11, 13, -21]
```

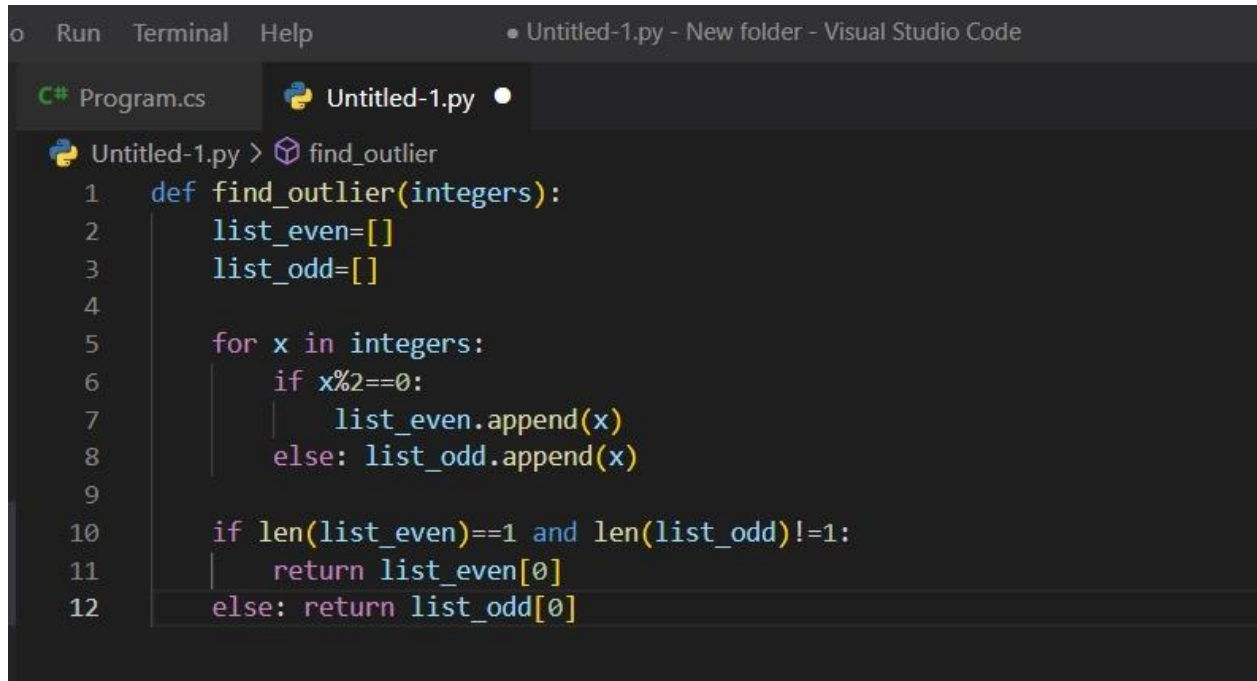
Treba da vraća: 160 (jedini paran broj)

Diskusija:

Problem se može riješiti sortiranjem parnih i neparnih brojeva unutar ulazne liste u dvije zasebne liste. Ako je broj elemenata u ovim listama različit, odnosno, ako je dužina jedne liste jedan a druge nije, onda funkcija vraća prvi i jedini element te manje liste, i obrnuto, jer je taj element uljez. Ako jedna lista ima jedan element a druga ima više elemenata, to znači da u drugoj listi ima više ili parnih ili neparnih brojeva, a u prvoj jedan ili neparan ili parni broj koji se traži.

Rješenje:

```
def find_outlier(integers):  
    list_even=[]  
    list_odd=[]  
  
    for x in integers:  
        if x%2==0:  
            list_even.append(x)  
        else: list_odd.append(x)  
  
    if len(list_even)==1 and len(list_odd)!=1:  
        return list_even[0]  
    else: return list_odd[0]
```



```
o Run Terminal Help • Untitled-1.py - New folder - Visual Studio Code

C# Program.cs Python Untitled-1.py •

Python Untitled-1.py > find_outlier
1 def find_outlier(integers):
2     list_even=[]
3     list_odd=[]
4
5     for x in integers:
6         if x%2==0:
7             list_even.append(x)
8         else: list_odd.append(x)
9
10    if len(list_even)==1 and len(list_odd)!=1:
11        return list_even[0]
12    else: return list_odd[0]
```

Prilog br. 11: Ispis programa u IDE Visual Studio Code

2.12 DIJELJENJE STRINGOVA

Podijeliti string u parove od dva karaktera. Ako string sadrži neparan broj karaktera onda kao nedostajući drugi karakter treba ubaciti donju crtu (_).

Primjeri:

```
'abc' => ['ab', 'c_']
```

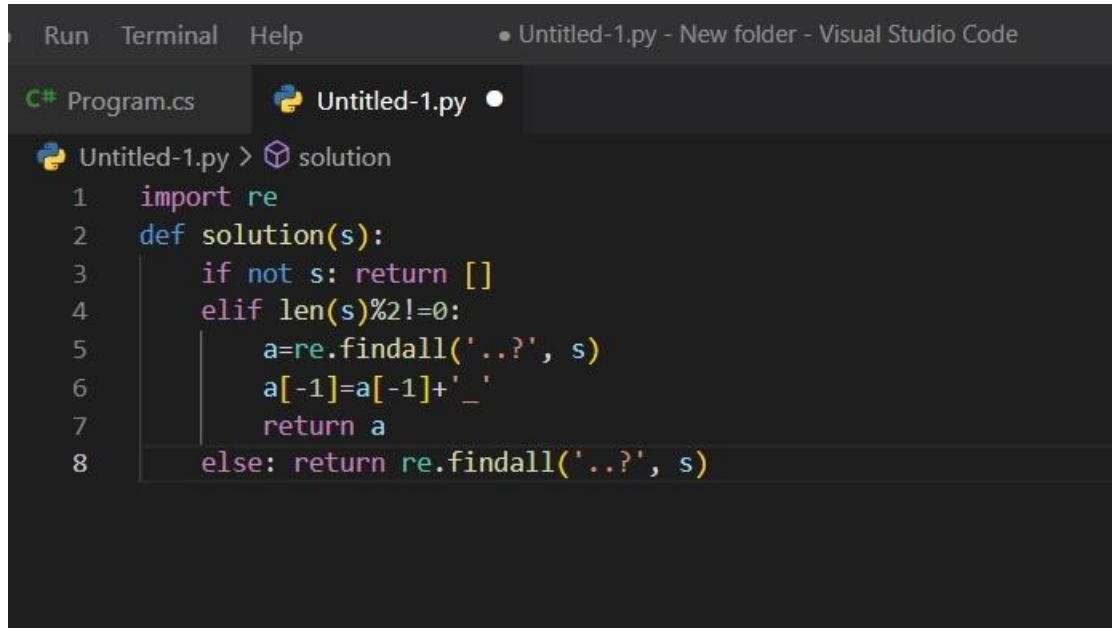
```
'abcdef' => ['ab', 'cd', 'ef']
```

Diskusija:

Problem se može riješiti korištenjem regex, odnosno regularnih izraza. Regex se importuje, potom se korištenjem funkcije findall definiše regex i string koji se koristi za pretragu. Regex '..?' znači: pronađi bilo koja dva karaktera, a znak ? određuje da se može pronaći i jedan karakter, što je potrebno u slučaju da se karakteri iz neparnog stringa tako grupišu da zadnji karakter nema svoj par. Potom se dodaje donja crta na takav karakter i vraća se string.

Rješenje:

```
import re
def solution(s):
    if not s: return []
    elif len(s)%2!=0:
        a=re.findall('..?', s)
        a[-1]=a[-1]+'_'
        return a
    else: return re.findall('..?', s)
```

A screenshot of the Visual Studio Code IDE interface. The top bar shows 'Run', 'Terminal', and 'Help' menus, along with the file name 'Untitled-1.py - New folder - Visual Studio Code'. The editor window displays a Python file named 'Untitled-1.py' with the following code:

```
1 import re
2 def solution(s):
3     if not s: return []
4     elif len(s)%2!=0:
5         a=re.findall('..?', s)
6         a[-1]=a[-1]+'_'
7         return a
8     else: return re.findall('..?', s)
```

Prilog br. 12: Ispis programa u IDE Visual Studio Code

2.13 PREBROJAVANJE DUPLIKATA

Kreirati funkciju koja će vraćati broj ponavljanja karaktera (uključujući slova i brojke).

Primjeri:

"abcde" -> 0 # nema nijednog karaktera koji se ponavlja više puta

"aabbcd" -> 2 # 'a' i 'b'

"aabBcd" -> 2 # 'a' se ponavlja 2 puta i 'b' također 2 puta ('b' i 'B')

"indivisibility" -> 1 # 'i' se ponavlja 6 puta, ali ne brojimo koliko se puta karakter ponavlja
već koliko se karaktera ponavlja dva ili više puta

"Indivisibilities" -> 2 # 'i' se ponavlja 7 puta i 's' se ponavlja 2 puta

"aA11" -> 2 # 'a' i '1'

"ABBA" -> 2 # 'A' i 'B'

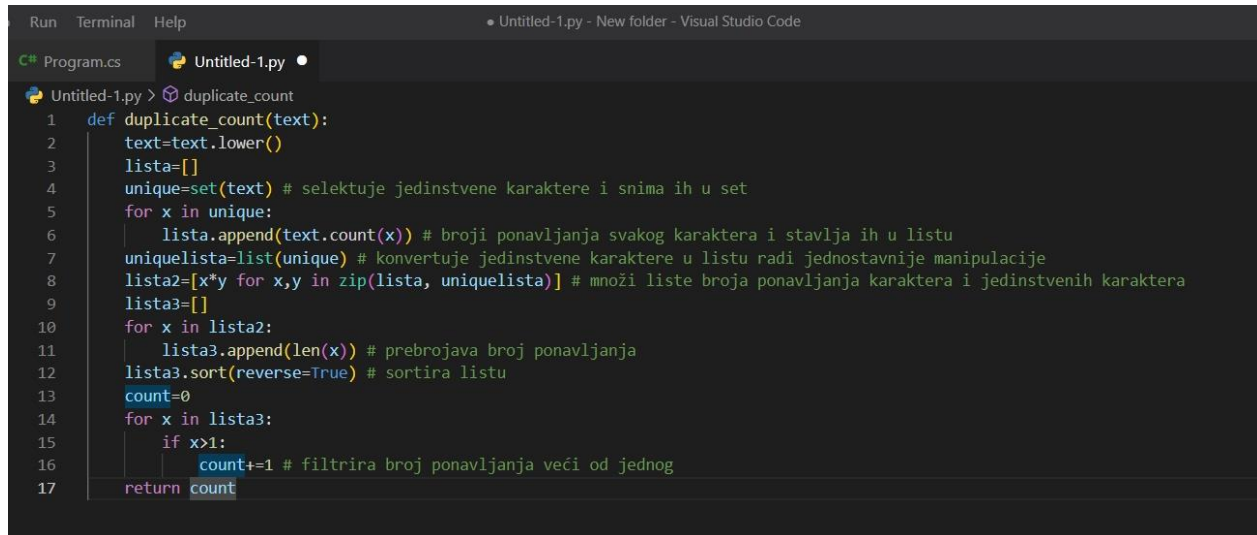
Diskusija:

Problem zahtjeva prebrojavanje broja karaktera koji se ponavljaju više od jednom u datom stringu. Dakle, problem ne zahtjeva prebrojavanje koliko puta se neki karakter ponavlja u stringu, već koliko postoji karaktera u stringu koji se ponavljaju više od jednom, što pravi bitnu razliku.

Rješenje se može sastojati od kreiranja jedinstvenog seta karaktera korištenjem set (što je upravo takva struktura u Pythonu). Potom se broji ponavljanje svakog takvog jedinstvenog karaktera u originalnom stringu i stavlja ih u listu. Potom se dvije liste množe, što je već opisivano u radu. Liste se množe jer se time izbjegava iteracija koja ne bi dovela do željenih rezultata. Konačno, treća lista prebrojava ponavljanja, potom se lista sortira i program vraća broj ponavljanja onih karaktera koji se pojavljuju više od jednog puta.

Rješenje:

```
def duplicate_count(text):
    text=text.lower()
    lista=[]
    unique=set(text)
# selektuje jedinstvene karaktere i snima ih u set
    for x in unique:
        lista.append(text.count(x))
# broji ponavljanja svakog karaktera i stavlja ih u listu
    uniquelista=list(unique)
# konvertuje jedinstvene karaktere u listu radi jednostavnije
# manipulacije
    lista2=[x*y for x,y in zip(lista, uniquelista)]
# množi liste broja ponavljanja karaktera i jedinstvenih
#karaktera
    lista3=[]
    for x in lista2:
        lista3.append(len(x)) # prebrojava broj ponavljanja
    lista3.sort(reverse=True) # sortira listu
    count=0
    for x in lista3:
        if x>1:
            count+=1 # filtrira broj ponavljanja veći od jednog
    return count
```



```

Run Terminal Help • Untitled-1.py - New folder - Visual Studio Code
C# Program.cs • Untitled-1.py
Untitled-1.py > duplicate_count
1 def duplicate_count(text):
2     text=text.lower()
3     lista=[]
4     unique=set(text) # selektuje jedinstvene karaktere i snima ih u set
5     for x in unique:
6         lista.append(text.count(x)) # broji ponavljanja svakog karaktera i stavlja ih u listu
7     uniquelista=list(unique) # konvertuje jedinstvene karaktere u listu radi jednostavnije manipulacije
8     lista2=[x*y for x,y in zip(lista, uniquelista)] # množi liste broja ponavljanja karaktera i jedinstvenih karaktera
9     lista3=[]
10    for x in lista2:
11        lista3.append(len(x)) # prebrojava broj ponavljanja
12    lista3.sort(reverse=True) # sortira listu
13    count=0
14    for x in lista3:
15        if x>1:
16            count+=1 # filtrira broj ponavljanja veći od jednog
17    return count

```

Prilog br. 13: Ispis programa u IDE Visual Studio Code

2.14 KONVERZIJA IZ RGB U HEX

RGB funkcija je nedovršena. Treba je kompletirati tako da ulazne RGB vrijednosti rezultuju u heksadecimalnim izlaznim vrijednostima. Validne decimalne vrijednosti za RGB su 0-255. Sve vrijednosti koje su izvan tog ranga moraju biti zaokružene na najbližu validnu vrijednost.

Napomena: izlaz treba uvijek da bude dugačak 6 karaktera.

Primjeri:

`rgb(255, 255, 255) # vraća FFFFFFFF`

`rgb(255, 255, 300) # vraća FFFFFFFF`

`rgb(0, 0, 0) # vraća 000000`

`rgb(148, 0, 211) # vraća 9400D3`

Diskusija:

Prvo treba riješiti problem zaokruživanja obzirom da vrijednosti mogu ići od 0-255 da bi se konvertovale u heksadecimalne brojeve. Heksadecimalni broj u Pythonu počinje sa 0x, što znači da te karaktere treba izbaciti jer se ne traže u problemu. Stoga se primjenjuje slicing stringa, odnosno ova linija:

```
rh= str((hex(r)[2:])).upper()
```

će konvertovati broj `r` u heksadecimalnu vrijednost, `[2:]` znači da će se uzeti vrijednosti od indeksa dva do kraja (heksadecimalna vrijednost broja 100 je `0x64`, slicing uklanja prva dva karaktera), te se cijeli izraz pretvara u string da bi ga na kraju mogli konkatenacijom spojiti (a ne sabrati), te se pretvara u velika slova što je uobičajeni i traženi način zapisa heksadecimalnih brojeva.

Rješenje:

```
def rgb(r,g,b):  
    if r<0: r=0  
    if g<0: g=0  
    if b<0: b=0  
    if r>255: r=255  
    if g>255: g=255  
    if b>255: b=255  
    rh= str((hex(r)[2:])).upper()  
    rg=str(hex(g)[2:]).upper()  
    rb=str(hex(b)[2:]).upper()  
    if len(rh)==1: rh=str(0)+rh  
    if len(rg)==1: rg=str(0)+rg  
    if len(rb)==1: rb=str(0)+rb  
    return rh+rg+rb
```



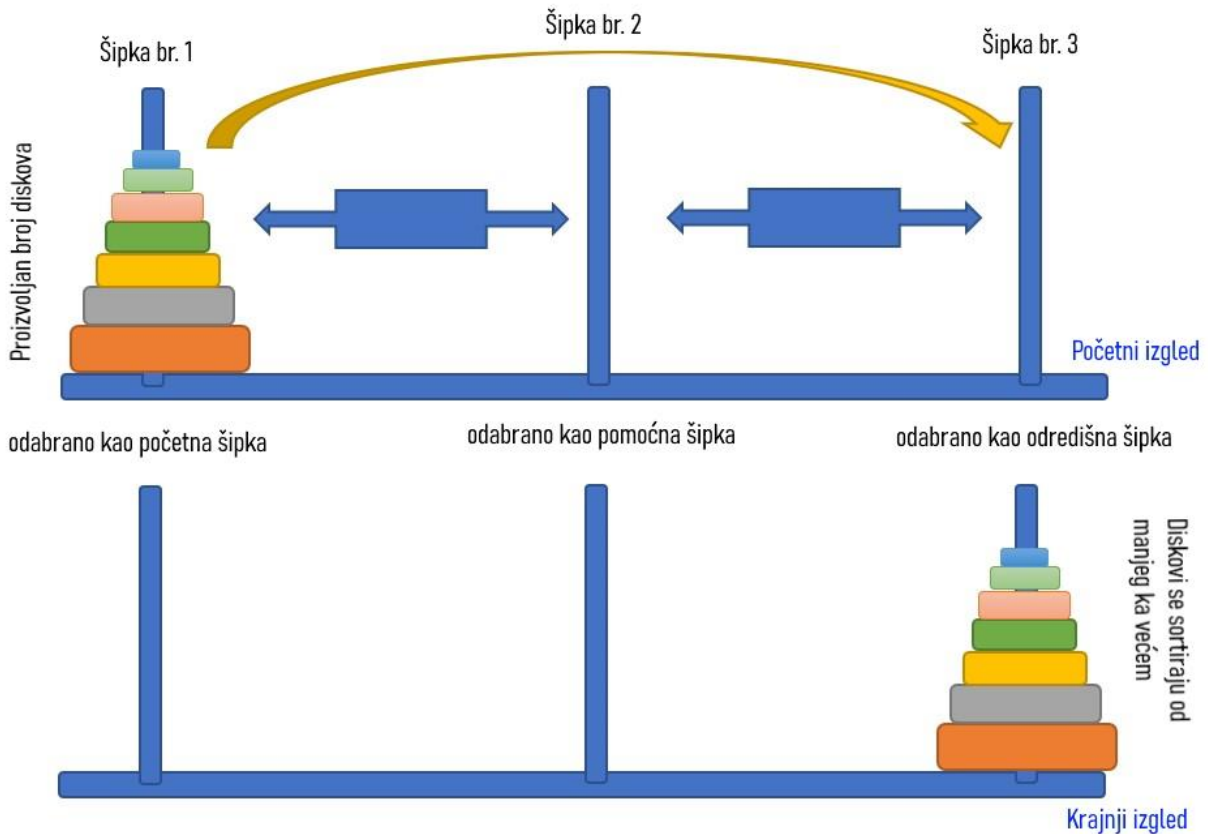
```
io Run Terminal Help • Untitled-1.py - New folder - Visual Studio Code
C# Program.cs • Untitled-1.py •
Untitled-1.py > rgb
1 def rgb(r,g,b):
2     if r<0: r=0
3     if g<0: g=0
4     if b<0: b=0
5     if r>255: r=255
6     if g>255: g=255
7     if b>255: b=255
8     rh= str((hex(r)[2:])).upper()
9     rg=str(hex(g)[2:]).upper()
10    rb=str(hex(b)[2:]).upper()
11    if len(rh)==1: rh=str(0)+rh
12    if len(rg)==1: rg=str(0)+rg
13    if len(rb)==1: rb=str(0)+rb
14    return rh+rg+rb
```

Prilog br. 14: Ispis programa u IDE Visual Studio Code

2.15 HANOJSKE KULE

Problem Hanojske kule sastoji se u sljedećem: na konstrukciju od tri šipke su, na jednoj šipki, nanizani diskovi. Diskovi su sortirani po prečniku od najvećeg, koji je na dnu, do najmanjeg na vrhu. Problem se sastoji od prebacivanja diskova sa šipke označene kao početna, ka ciljnoj šipki, uz respektovanje sljedećih pravila:

- Diskovi se pomjeraju u potezima; po potezu je dozvoljeno pomjeriti samo jedan disk,
- Pomjeranje se sastoji od micanja diska sa jedne šipke na drugu. Šipke su označene kao početna (source), odredišna (destination) i pomoćna (auxiliary) šipka. Redoslijed ovih šipki nije fiksna, odnosno moguće je različite šipke na početku rješavanja problema označiti različito.
- Diskovi se mogu pomicati ili na praznu šipku, ili na šipku na kojoj već ima diskova, ali uz pravilo da je samo moguće pomicati disk manjeg prečnika od onog na koji se disk stavlja.



Prilog br. 15: Ispis programa u IDE Visual Studio Code

Diskusija:

Označimo broj diskova kao n . Za $n=1$, problem je jednostavan i svodi se na jedno pomjeranje diska sa početne na krajnju šipku, odnosno broj ukupnih koraka za rješavanje problema je 1. Za $n=2$, potrebno je koristiti pomoćnu šipku jer zbog pravila pomjeranja diska, nije nikako drugačije moguće izvršiti pomjeranje diskova², odnosno broj koraka je tri:

1. Pomjeriti gornji, manji disk sa početne na pomoćnu šipku,
2. Pomjeriti donji, veći disk sa početne na krajnju šipku,
3. Pomjeriti gornji, manji disk sa pomoćne na krajnju šipku, na veći disk.

Za $n=3$, problem se usložnjava te je broj ukupnih koraka za rješavanje sada 7.

I tako dalje – na osnovu ovih posmatranja, možemo da donesemo iduće zaključke:

- U opštem slučaju, broj koraka potrebnih za rješavanje problema je 2^{n-1} ,
- Kad je broj diskova veći od jedan, problem se svodi na pomjeranje n -tog diska na odredišnu šipku, a $n-1$ diskova je prvo potrebno pomjeriti na

² Za detaljno objašnjenje ovog problema i analizu programskog koda pogledati autorov članak na: <https://medium.com/@sinisabozic/understanding-recursion-towers-of-hanoi-44a7c034c694>

pomoćnu šipku, pa onda jedan po jedan, poštujući pravila, na odredišnu šipku,

- Rekurzivna priroda problema se ponavlja u istovjetnom pomjeranju $n-1$ diskova, bez obzira na broj n , koji se dijeli u istovjetan niz manjih problema iste prirode.

Ovi zaključci pomažu u kreiranju algoritma u pseudo-kodu:

- Ako je $n=1$, u pitanju je bazni rekurzivni slučaj,
- Pomjeranje $n-1$ diskova sa početne na pomoćnu šipku,
- Pomjeranje n -tog diska sa početne na odredišnu šipku,
- Pomjeranje $n-1$ diskova sa pomoćne na odredišnu šipku.

Kada god se izvrši virtuelno pomjeranje diska, program treba da obavjesti korisnika, u konačnici dajući ispis konačnog rješenja problema ispisivanjem niza koraka. Ukupan broj koraka je minimalno 2^{n-1} , što se može koristiti za kontrolu ispravnosti algoritma. Rekurzivni pozivi se odvijaju pozivanjem koda sve dok se ne dosegne bazni slučaj, za koji je rješenje poznato.

Rješenje:

Programski kod se dobija prevođenjem pseudo-koda u programski jezik, u ovom slučaju Python. Definisana je funkcija sa četiri ulazna parametra, n =broj diskova, source, dest, aux su početna, odredišna i pomoćna šipka, respektivno.

```
def hanoi(n, source, dest, aux):  
    if n==1:  
        print(source, " do ", dest)  
    else:  
        hanoi(n-1, source, aux, dest)  
        print(source, " do ", dest)  
        hanoi(n-1, aux, dest, source)
```

Prilog br. 16: Ispis programa u IDE Visual Studio Code

Ukoliko izvršimo funkciju za broj diskova $n=3$, te uz obilježavanje šipki kao 1, 2, 3³, odnosno unošenjem:

```
hanoi(3, 1, 3, 2)
```

Dobijamo idući izlaz:

1 do 3

1 do 2

3 do 2

1 do 3

2 do 1

2 do 3

1 do 3

Za $n=4$, dobijamo izlaz:

1 do 2

³ Šipke se mogu proizvoljno obilježiti, odnosno početna može biti i npr. 3, pomoćna 1 a odredišna 2. Program će to uzeti u obzir i ispisati korake imajući u vidu takve ulazne podatke.

1 do 3

2 do 3

1 do 2

3 do 1

3 do 2

1 do 2

1 do 3

2 do 3

2 do 1

3 do 1

2 do 3

1 do 2

1 do 3

2 do 3

Vidimo da je za $n=3$ broj koraka 7, a za $n=4$, 15 koraka, što se uklapa u postavljeni okvir ukupnog broja koraka od 2^{n-1} .

3 ZAKLJUČAK

Kroz rješavanje 15 problema, nastojao sam demonstrirati neke osnovne programerske postulate kao što su iteracija, uslovni odabir vrijednosti, rekurzija, korištenje funkcionalnog programiranja. Upotrijebljeni su osnovni tipovi podataka i osnovne strukture u Pythonu.

Stranica Codewars.com, sa koje su problemi preuzeti, je vrlo korisna za sticanje znanja u algoritmima i strukturama podataka, kako za početnike tako i za iskusnije programere. Za uspješno rješavanje problema, kako na ovoj stranici tako i generalno uzevši, je potrebno prvo razumjeti postavku problema u potpunosti, potom razraditi logički okvir za rješavanje problema, a potom se pozabaviti implementacijom u datom programskom jeziku uzevši u obzir sintaksu samog jezika.

Što se tiče samog programskog jezika Python, isti je vrlo koncizan, konkretan i sa malo programskog koda se rješava mnogo. U zavisnosti od upotrebe, vrlo je pogodan za brzo rješavanje problema, stvaranje prototipa ili predložaka rješenja koja se potom mogu koristiti za izradu u nekim drugim programskim jezicima.

4 LITERATURA I DRUGI IZVORI

- [1] "Python Documentation," [Online]. <https://www.python.org/doc/>. [Pristupano 2022].
- [2] "Codewars," [Online]. <https://www.codewars.com>. [Pristupano 2022].
- [3] S. Bozic, "Personal Coding Web Blog," [Online]. <https://medium.com/@sinisabozic>. [Pristupano 2022].
- [4] M. Lutz, Learning Python, O'Reilly Media, 2013.