**8. praktiskais darbs. 2. semestris**

**1. uzdevums**

Sastādīt programmu, kas izveido visu pirmskaitļu, kas nepārsniedz 10000, augošā secībā sakārtotu masīvu un izvada šo masīvu uz ekrāna. Atrast un pielietot formulu, kas nosaka iespējamo pirmskaitļu, kas nepārsniedz patvaļīgu skaitli N, skaitu, definējot pirmskaitļu masīva garumu.

**Kods:**

# Programmas nosaukums: Izveido masīvu ar visiem pirmskaitļiem, kuri ir ne lielāki 10000 un izvada to augošā secībā

# 1. uzdevums (1MPR08\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido visu pirmskaitļu, kas nepārsniedz 10000, augošā secībā sakārtotu masīvu

# un izvada šo masīvu uz ekrāna. Atrast un pielietot formulu, kas nosaka iespējamo pirmskaitļu, kas nepārsniedz patvaļīgu skaitli N,

# skaitu, definējot pirmskaitļu masīva garumu.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

import math

def izvade(x):

# Izvada masīva elementus pēc kārtas līdz pedējam

# x - viendimensijas masīvs

try:

n = len(x)

s = str(x[0])

for i in range(1, n):

s = s + ", " + str(x[i])

print(s)

except IndexError:

print("Nav pirmskaitļu šajā intervālā.") # Ja parādās index error, tad tas nozīme, ka šajā intervālā nav pirmskaitļu.

except:

print("Kļūda!") # Citām kļūdam

def pirmskaitlu\_masivs(n, drosibas\_koeficients):

# Atgriež tuple ar pirmskaitļu masīvu līdz skaitlim n bez liekām 0 un ar nodzēsto nulles skaitu. (Ja ir lieki elementi, tad funkcija to nodzes)

# n - naturāls skaitlis (int), līdz kurām meklēsīm pirmskaitļus

# drosibas\_koeficients - drošības koeficients. Tiek reizināts ar n/log(n) un tiek izmantots masīva izmēra definēšanai.

# Parasti, jo lielāks, jo vairāk liekas nulles būs. Pēc noklusējuma labāk, lai tās būtu aptuvēni 1.2

if n <= 1: # Funkcija nedarbotos jā ievadīs 0, 1 vai 2, tāpēc tas tiek atsevišķi definēts.

return numpy.array([])

if n == 2:

return numpy.array([2])

# drosibas\_koeficients = 1.2 # izmantots, lai palidzētu nodefinētu masīva izmēru (size). Masīva izmēra definēšanai tiek izmantota formula n/log(n) https://en.wikipedia.org/wiki/Prime\_number\_theorem un https://en.wikipedia.org/wiki/Prime-counting\_function

size = math.ceil((n / math.log(n)) \* drosibas\_koeficients) # aprēķina masīva garumu, izmantojot formulu n/log(n) un reizinot ar drošības koeficientu (bez viņa nevienmēr pietiks vietas)

p = numpy.zeros(size, dtype=numpy.int32) # izveido masīvu ar 0 vērtībām tādu garumu, kuru aprēķinaja ar formulu (n/log(n))\*drosibas\_koeficientu

p[0] = 2 # pirmais pirmskaitlis

p[1] = 3 # otrais pirmskaitlis

j = 2

k = 5

try:

while k <= n:

i = 0

s = round(math.sqrt(k))

while (k % p[i]) != 0:

i = i + 1

if p[i] > s:

p[j] = k

j = j + 1

break

k = k + 2

deleted\_zeros\_count = len(p) - len(p[:j])

return (p[:j], deleted\_zeros\_count) # [:j], lai izvadītu bez nullem

except:

print("Palieliniet drošības koeficientu! Nepietika vietas masīva aizpildīšanai.")

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

n = 10000 # Līdz kuram skaitlim meklējam pirmskaitļus

drosibas\_koeficients = 1.2 # izmantots, lai palidzētu nodefinētu masīva izmēru (size). Masīva izmēra definēšanai tiek izmantota formula n/log(n) https://en.wikipedia.org/wiki/Prime\_number\_theorem un https://en.wikipedia.org/wiki/Prime-counting\_function

print("Pimskaitļi līdz " + str(n) + ":") # Var mainīt drošības koeficientu, lai mainītu definētā masīva izmēru

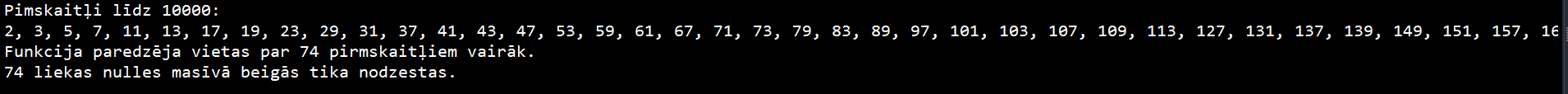
pirmskaitli = pirmskaitlu\_masivs(n, drosibas\_koeficients)

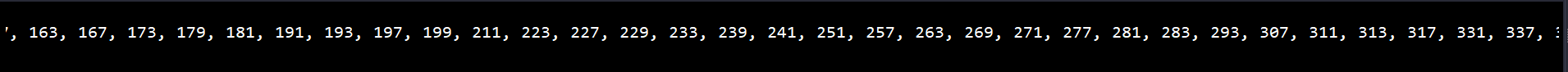
izvade(pirmskaitli[0])

print("Funkcija paredzēja vietas par " + str(pirmskaitli[1]) + " pirmskaitļiem vairāk.")

print(f"{pirmskaitli[1]} liekas nulles masīvā beigās tika nodzestas.")

**Testa piemēri:**

1)

2)

3)

4)

**2. uzdevums**

Sastādīt programmu, kas realizē lielo naturālo skaitļu (vismaz ar 50 cipariem) saskaitīšanu. Jāveic ievaddatu korektuma pārbaude!

**Kods:**

# Programmas nosaukums: Lielo skaitļu saskaitīšana

# 2. uzdevums (1MPR08\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē lielo naturālo skaitļu (vismaz ar 50 cipariem) saskaitīšanu.

# Jāveic ievaddatu korektuma pārbaude!

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def is\_natural\_or\_zero\_long(s):

# Pārbauda vai simbolu virkne reprezentē naturālo skaitli vai 0, vai nē.

# Atgriež True, ja virkne reprezentē naturālo skaitli.

# Atgriež False, ja nereprezentē naturālo skaitli.

# s - pārbaudama simbolu virkne

# Noņema no virknes visas sākuma vai beigu atstarpes

s = s.strip()

# Pārbauda, vai virkne ir tukša

if len(s) == 0:

return False

# Iet cikliski cauri katrām simbolam simbolu virknē (string'ā)

for c in s:

# Ja kāda rakstzīme nav cipars, virkne neatspoguļo naturālu skaitli. return False

if not c.isdigit():

return False

# Virkne atspoguļo naturālu skaitli, ja ietu cauri ciklas netika pamanīts not .isdigit()

return True

def izvade\_bez\_komatiem\_kopigi(x):

# Izvada masīva elementus pēc kārtas līdz pedējam kopīgi bez komatiem (izmanto lai izvadītu masīvu kā vienu str skaitli)

# x - viendimensijas masīvs

n = len(x)

s = str(x[0])

for i in range(1, n):

s = s + "" + str(x[i])

print(s[::-1])

def parveide\_sv\_to\_mas(sv, m):

# Transformē simbolu virkni par masīvu ar noradīto garumu. Ja garums ir lielāks nekā simbolu virkne, tad beigās tiek pieliktas 0

# sv - simbolu virkne, kura sastāv no cipariem

# m - garums, līdz kurām vajag transformēt simbolu virkni sarakstā. Ja garums ir lielāks nekā simbolu virkne, tad beigās tiek pieliktas 0

n = len(sv)

a = numpy.arange(m)

for i in range(n):

a[i] = int(sv[-i - 1])

for i in range(n, m):

a[i] = 0

return a

def parveide\_mas\_to\_sv(a):

# Transformē masīvu par simbolu virkni. Ja masīvā priekšā ir 0, tad tas tiek noņemtas

# a - viendimensijas masīvs

n = len(a)

sv = ""

for i in range(n):

sv = str(a[i]) + sv

try: # try/except gadījumam ja 0 + 0

while sv[0] == "0":

sv = sv[1:]

return sv

except IndexError: # Tas ir nepieciešams gadījumam ja lietotājs ievadīs 0 + 0 = 0.

return "0" # bez šim rindām būtu kļūda, kad 0 + 0

def saskaitisana(a, b):

# Saskaita masīvu a un b izmantojot str. Jāizmanto lielo skaitļu (vismaz ar 50 cipariem) saskaitīšanu

# a - viendimensijas masīvs

# b - viendimensijas masīvs

a = a[::-1]

b = b[::-1]

n1 = len(a)

n2 = len(b)

if n1 > n2:

n = n1

else:

n = n2

m1 = parveide\_sv\_to\_mas(a, n)

m2 = parveide\_sv\_to\_mas(b, n)

m3 = numpy.zeros(n + 1, dtype=numpy.int\_)

s = 0

for i in range(n):

s = s + m1[i] + m2[i]

m3[i] = s % 10

s = s // 10

m3[n] = s

return parveide\_mas\_to\_sv(m3)

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

a = input("Ievadiet 1.saskaitāmo ===> ")

while is\_natural\_or\_zero\_long(a) == False:

a = input("Kļūda! Ievadītam skaitlim jābūt naturālam!\nIevadiet 1.saskaitāmo ===> ")

a = parveide\_sv\_to\_mas(a, len(a))

b = input("Ievadiet 2.saskaitāmo ===> ")

while is\_natural\_or\_zero\_long(b) == False:

b = input("Kļūda! Ievadītam skaitlim jābūt naturālam!\nIevadiet 1.saskaitāmo ===> ")

b = parveide\_sv\_to\_mas(b, len(b))

print("")

izvade\_bez\_komatiem\_kopigi(a)

print("+")

izvade\_bez\_komatiem\_kopigi(b)

print("=")

print(saskaitisana(a, b))

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

7)

Text

Description automatically generated

8)

Text

Description automatically generated

9)

Text

Description automatically generated

10)

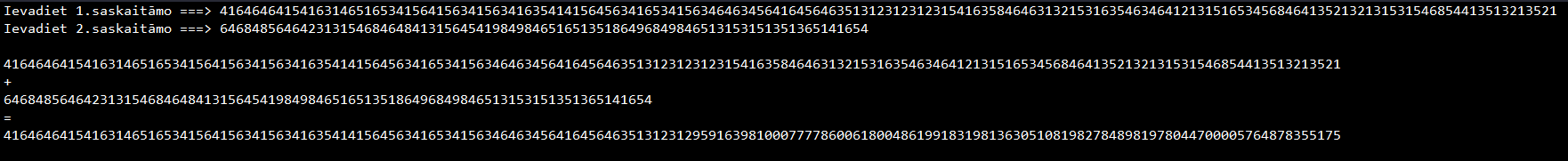
Text

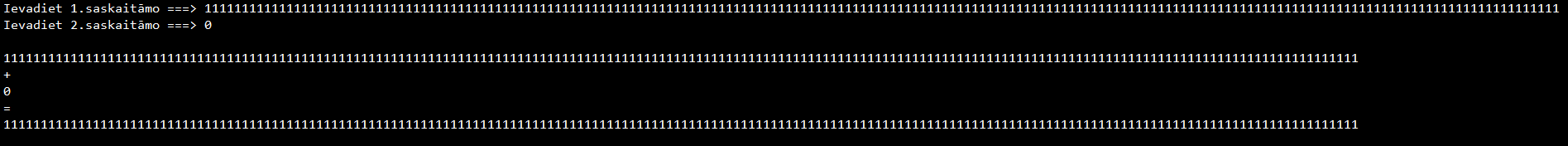
Description automatically generated

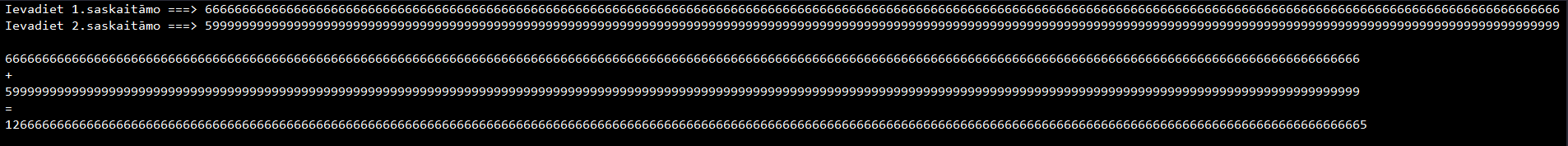
11)

Graphical user interface, text, application

Description automatically generated

12)

13)

14)

15)

Text

Description automatically generated

16)

Graphical user interface, text

Description automatically generated

**3. uzdevums**

Sastādīt programmu, kas realizē lielo naturālo skaitļu (vismaz ar 50 cipariem) atņemšanu. Pirms darbības veikšanas jāpārliecinās, ka mazināmai ir lielāks nekā mazinātājs. Jāveic ievaddatu korektuma pārbaude!

**Kods:**

# Programmas nosaukums: Lielo skaitļu atņemšana

# 3. uzdevums (1MPR08\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē lielo naturālo skaitļu (vismaz ar 50 cipariem) atņemšanu.

# Pirms darbības veikšanas jāpārliecinās, ka mazināmai ir lielāks nekā mazinātājs.

# Jāveic ievaddatu korektuma pārbaude!

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def izvade\_bez\_komatiem\_kopigi(x):

# Izvada masīva elementus pēc kārtas līdz pedējam kopīgi bez komatiem (izmanto lai izvadītu masīvu kā vienu str skaitli)

# x - viendimensijas masīvs

n = len(x)

s = str(x[0])

for i in range(1, n):

s = s + "" + str(x[i])

print(s)

def parveide\_sv\_to\_mas(sv, m):

# Transformē simbolu virkni masīvā ar noradīto garumu. Ja garums ir lielāks nekā simbolu virkne, tad beigās tiek pieliktas 0

# sv - simbolu virkne, kura sastāv no cipariem

# m - garums, līdz kurām vajag transformēt simbolu virkni sarakstā. Ja garums ir lielāks nekā simbolu virkne, tad beigās tiek pieliktas 0

n = len(sv)

a = numpy.arange(m)

for i in range(n):

a[i] = int(sv[-i - 1])

for i in range(n, m):

a[i] = 0

return a[::-1]

def parveide\_mas\_to\_sv(a):

# Transformē masīvu simbolu virknē. Ja masīvā priekšā ir 0, tad tas tiek noņemtas

# a - viendimensijas masīvs

n = len(a)

sv = ""

for i in range(n):

sv = str(a[i]) + sv

return sv

def parveide\_mas\_to\_sv\_reverse(a):

# Transformē masīvu simbolu virknē un apgriež to

# a - viendimensijas masīvs

n = len(a)

sv = ""

for i in range(n):

sv = str(a[i]) + sv

return sv[::-1]

def remove\_front\_zeros(s):

# Atgriež simbolu virkni bez nullem priekšā

# s - simbolu virkne, kura sastav no cipariem, potenciāli ar nullēm priekšā

i = 0

while i < len(s) - 1 and s[i] == '0':

i += 1

s = s[i:]

return s

def atnemsana(a, b):

# Atņem masīvu a no b izmantojot str. Jāizmanto lielo skaitļu (vismaz ar 50 cipariem) atņemšanai. Atgriež simbolu virkni bez nullem priekšā

# a - viendimensijas masīvs

# b - viendimensijas masīvs

if str(a) == str(b):

return "0"

n1 = len(a)

n2 = len(b)

if n1 > n2:

n = n1

else:

n = n2

m1 = parveide\_sv\_to\_mas(a, n)

m2 = parveide\_sv\_to\_mas(b, n)

m3 = numpy.zeros(n, dtype=numpy.int\_)

for i in range(-1, -n1, -1):

if m1[i] < m2[i]:

m1[i - 1] = m1[i - 1] - 1

m1[i] = m1[i] + 10

for i in range(n):

m3[i] = m1[i] - m2[i]

a = parveide\_mas\_to\_sv(m3)

a = a[::-1]

a = remove\_front\_zeros(a)

return a

def is\_natural\_or\_zero\_long(s):

# Pārbauda vai simbolu virkne reprezentē naturālo skaitli vai 0, vai nē.

# Atgriež True, ja virkne reprezentē naturālo skaitli.

# Atgriež False, ja nereprezentē naturālo skaitli.

# s - pārbaudāma simbolu virkne

# Noņema no virknes visas sākuma vai beigu atstarpes

s = s.strip()

# Pārbauda, vai virkne ir tukša

if len(s) == 0:

return False

# Iet cikliski cauri katrām simbolam simbolu virknē (string'ā)

for c in s:

# Ja kāda rakstzīme nav cipars, virkne neatspoguļo naturālu skaitli. return False

if not c.isdigit():

return False

# Virkne atspoguļo naturālu skaitli, ja ietu cauri ciklas netika pamanīts not .isdigit()

return True

def is\_a\_bigger\_b(a, b):

# Vai masīvs a ir lielāks vai vienāds nekā masīvs b, ja jā, tad return True. Ja nē, tad return False

# a - pirmais viendimensijas masīvs

# b - otrais viendimensijas masīvs

a = parveide\_mas\_to\_sv\_reverse(a)

b = parveide\_mas\_to\_sv\_reverse(b)

if int(a) < int(b):

return False

else:

return True

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

a = input("Ievadiet mazināmo ===> ")

while is\_natural\_or\_zero\_long(a) == False:

a = input("Kļūda! Skaitļa izmēram jābūt naturālam skaitlim vai nulle!\nIevadiet mazināmo ===> ")

a = parveide\_sv\_to\_mas(a, len(a))

b = input("Ievadiet mazinātāju ===> ")

while is\_natural\_or\_zero\_long(b) == False:

b = input("Kļūda! Skaitļa izmēram jābūt naturālam skaitlim vai nulle!\nIevadiet mazinātāju ===> ")

b = parveide\_sv\_to\_mas(b, len(b))

if is\_a\_bigger\_b(a, b) == False:

print("Kļūda! Mazināmam jābūt lielākam par mazinātāju!")

else:

print("")

izvade\_bez\_komatiem\_kopigi(a)

print("-")

izvade\_bez\_komatiem\_kopigi(b)

print("=")

print(atnemsana(a, b))

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

7)

Text

Description automatically generated

8)

Text

Description automatically generated

9)

Text

Description automatically generated

10)

Text

Description automatically generated

11)

Text

Description automatically generated

12)

Text

Description automatically generated

13)

Text

Description automatically generated

14)

Text

Description automatically generated

15)

Text

Description automatically generated

**4. uzdevums**

Sastādīt programmu, kas realizē lielo naturālo skaitļu (vismaz ar 50 cipariem) reizināšanu, izmantojot naivo skolas reizināšanas algoritmu. Jāveic ievaddatu korektuma pārbaude!

**Kods:**

# Programmas nosaukums: Lielo skaitļu reizināšana

# 4. uzdevums (1MPR08\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē lielo naturālo skaitļu (vismaz ar 50 cipariem) reizināšanu,

# izmantojot naivo skolas reizināšanas algoritmu.

# Jāveic ievaddatu korektuma pārbaude!

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def is\_natural\_or\_zero\_long(s):

# Pārbauda vai simbolu virkne reprezentē naturālo skaitli vai 0, vai nē.

# Atgriež True, ja virkne reprezentē naturālo skaitli.

# Atgriež False, ja nereprezentē naturālo skaitli.

# s - pārbaudama simbolu virkne

# Noņema no virknes visas sākuma vai beigu atstarpes

s = s.strip()

# Pārbauda, vai virkne ir tukša

if len(s) == 0:

return False

# Iet cikliski cauri katrām simbolam simbolu virknē (string'ā)

for c in s:

# Ja kāda rakstzīme nav cipars, virkne neatspoguļo naturālu skaitli. return False

if not c.isdigit():

return False

# Virkne atspoguļo naturālu skaitli, ja ietu cauri ciklas netika pamanīts not .isdigit()

return True

def izvade\_bez\_komatiem\_kopigi(x):

# Izvada masīva elementus pēc kārtas līdz pedējam kopīgi bez komatiem (izmanto, lai izvadītu masīvu kā vienu str skaitli)

# x - viendimensijas masīvs

n = len(x)

s = str(x[0])

for i in range(1, n):

s = s + "" + str(x[i])

s = s[::-1]

print(s)

def parveide\_sv\_to\_mas(sv, m):

# Transformē simbolu virkni masīvā ar noradīto garumu. Ja garums ir lielāks nekā simbolu virkne, tad beigās tiek pieliktas 0

# sv - simbolu virkne, kura sastāv no cipariem

# m - (vēlamais masīva garums) garums, līdz kurām vajag transformēt simbolu virkni sarakstā. Ja garums ir lielāks nekā simbolu virkne, tad beigās tiek pieliktas 0

n = len(sv)

a = numpy.arange(m)

for i in range(n):

a[i] = int(sv[-i - 1])

for i in range(n, m):

a[i] = 0

return a

def parveide\_mas\_to\_sv(a):

# Transformē masīvu simbolu virknē pretējā secībā. Ja masīvā priekšā ir 0, tad tas tiek noņemtas

# a - viendimensijas masīvs

n = len(a)

sv = ""

for i in range(n):

sv = str(a[i]) + sv

try: # try/except gadījumam ja a \* 0, a - jebkurš skaitlis

while sv[0] == "0":

sv = sv[1:]

return sv

except IndexError: # Tas ir nepieciešams gadījumam ja lietotājs ievadīs a \* 0 = 0, a - jebkurš skaitlis

return "0" # bez šim rindām būtu kļūda, kad a \* 0, a - jebkurš skaitlis

def reizinasana(a, b):

# Sareizina masīvu a ar b izmantojot str. Jāizmanto lielo skaitļu (vismaz ar 50 cipariem) reizināšanai

# a - viendimensijas masīvs

# b - viendimensijas masīvs

a = a[::-1]

b = b[::-1]

n1 = len(a)

n2 = len(b)

if n1 > n2:

n = n1

else:

n = n2

m1 = parveide\_sv\_to\_mas(a, n)

m2 = parveide\_sv\_to\_mas(b, n)

m3 = numpy.zeros(2 \* n, dtype=numpy.int\_)

s = 0

for j in range(2 \* n - 1):

for i in range(n):

if (0 <= j - i) and (j - i <= n - 1):

s = s + m1[i] \* m2[j - i]

m3[j] = s % 10

s = s // 10

m3[n \* 2 - 1] = s

return parveide\_mas\_to\_sv(m3)

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

a = input("Ievadiet 1.reizinātāju ===> ")

while is\_natural\_or\_zero\_long(a) == False:

a = input("Kļūda! Ievadītam skaitlim jābūt naturālam vai 0.\nIevadiet 1.reizinātāju ===> ")

a = parveide\_sv\_to\_mas(a, len(a))

b = input("Ievadiet 2.reizinātāju ===> ")

while is\_natural\_or\_zero\_long(b) == False:

b = input("Kļūda! Ievadītam skaitlim jābūt naturālam vai 0.\nIevadiet 1.reizinātāju ===> ")

b = parveide\_sv\_to\_mas(b, len(b))

print("")

izvade\_bez\_komatiem\_kopigi(a)

print("\*")

izvade\_bez\_komatiem\_kopigi(b)

print("=")

print(reizinasana(a, b))

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

7)

Text

Description automatically generated

8)

Text

Description automatically generated

9)

Text

Description automatically generated

10)

Text

Description automatically generated

11)

Text

Description automatically generated

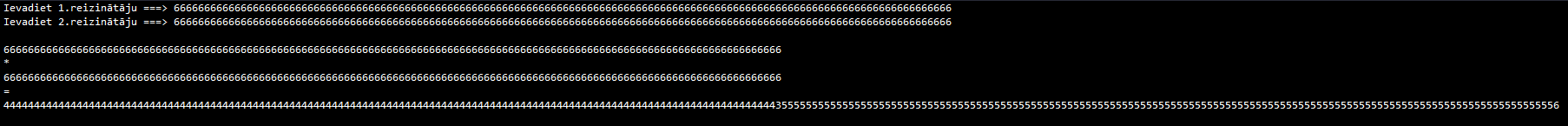
12)

Text

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence13)

14)

15)

Graphical user interface, text

Description automatically generated

16)

A picture containing text

Description automatically generated

17)

Text

Description automatically generated

18)

Text

Description automatically generated

19)

Graphical user interface, text

Description automatically generated

**5. uzdevums**

Sastādīt programmu, kas nodrošina divdimensiju matricas elementu ievadi, kas ir veseli skaitļi. Matricas rindu un kolonnu skaitu ievada lietotājs. Pēc matricas elementu ievades tiek nodrošināta matricas elementu izvade tabulas veidā, pieņemot, ka neviens no matricas elementiem nesastāv no ne vairāk kā 4 cipariem, kā arī lielāko un mazāko matricas elementa vērtību un tā atrašanās vietu, noradot atbilstošo rindu un kolonnu.

Izvades piemērs (\* - tukšums):

\*\*\*\*1\*\*137\*\*\*23\*8765

\*1234\*\*\*53\*\*\*\*1\*\*876

\*\*\*13\*\*\*\*2\*\*\*34\*\*\*\*5

Mazākais elements ir 1, un tas atrodas 1.rindas un 1.kolonnas krustpunktā.

Lielākais elements ir 8765, un tas atrodas 1.rindas un 4.kolonnas krustpunktā.

**Kods:**

# Programmas nosaukums: Matricas max un min elements

# 5. uzdevums (1MPR08\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas nodrošina divdimensiju matricas elementu ievadi,

# kas ir veseli skaitļi. Matricas rindu un kolonnu skaitu ievada lietotājs.

# Pēc matricas elementu ievades tiek nodrošināta matricas elementu izvade\_matrica\_int tabulas veidā,

# pieņemot, ka neviens no matricas elementiem nesastāv no ne vairāk kā 4 cipariem, kā arī

# lielāko un mazāko matricas elementa vērtību un tā atrašanās vietu, noradot atbilstošo rindu un kolonnu.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def is\_natural(n):

# Pārbauda vai simbolu virkne ir naturāls skaitlis vai nav

# Ja ir naturāls skaitlis, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

if str(n).isdigit() and float(n) == int(n) and int(n) > 0:

return True

else:

return False

def is\_whole(n):

# Pārbauda vai simbolu virkne ir vesels skaitlis vai nav

# Ja ir vesels skaitlis, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

try:

n = int(n)

except:

return False

else:

return True

def ievade\_matrica(n, m):

# Lietotājs var ievādīt nXm matricas elementus un funkcija atgriež divdimensijas masīvu ar n rindam un m kolonnam ar ievadītām vērtībam

# n - naturāls skaitlis, kurš nosaka matricas rindas skaitu

# m - naturāls skaitlis, kurš nosaka matricas kolonnas skaitu

a = numpy.empty((n, m), dtype=int)

for i in range(n):

for j in range(m):

temp = input("Ievadiet matricas elememtu a(" + str(i) + "," + str(j) + ") ===> ")

while is\_whole(temp) == False:

temp = input("Kļūda! Ievadītais elements nav vesels skaitlis!\nIevadiet matricas elememtu a(" + str(i) + "," + str(j) + ") ===> ")

a[i, j] = int(temp)

return a

def izvade\_matrica\_int(a):

# Atgriež divdimensiju masīvu (matricu), tabulas veidā, str formāta, kur katra rinda ir atdalīta ar jauno rindkopu

# a - divdimensijas masīvs (matrica)

n = a.shape[0] # "x axis" masīva izmēri

m = a.shape[1] # "y axis" masīva izmēri

s = ""

for i in range(n):

for j in range(m):

s = s + "{:8.2f}".format(int(a[i, j]))

s = s + "\n"

return s

def matricas\_izvade\_4\_cipari(m):

# Izvada divdimensijas masīvu tabulas veidā tā, ka neviens no matricas elementiem nesastāv no ne vairāk kā 4 cipariem

# Tiek nodrošināta matricas elementu izvade tabulas veidā, pieņemot, ka neviens no matricas elementiem nesastāv no ne vairāk kā 4 cipariem

# Atgriež matricas elementu izvade tabulas veidā

# m - divdimensijas masīvs

nm = numpy.shape(m)

sv = ""

for x in range(nm[0]):

for y in range(nm[1]):

for i in range(5 - len(str(m[x, y]))):

sv = sv + " "

sv = sv + str(m[x, y])

sv = sv + "\n"

nn = len(sv)

return sv[:nn - 1]

def max\_value\_in\_2\_dimensional\_array\_and\_its\_coords(a):

# Atgriež tuple (max\_value, coords) ar divdimensijas masīva (matricas) maksimālo elementu vērtību un tas koordinātu [i,j] pēc rindām un kolonnam

# Atgriež maksimālo vērtību un kur tā vērtība atrodas pēc koordinātam

# a - divdimensijas masīvs

max\_value = a[0][0]

coords = [0, 0]

for i in range(len(a)):

for j in range(len(a[0])):

if a[i][j] > max\_value:

max\_value = a[i][j]

coords = [i, j]

return (max\_value, coords)

def min\_value\_in\_2\_dimensional\_array\_and\_its\_coords(a):

# Atgriež kortežu (tuple) (min\_value, coords) ar divdimensijas masīva (matricas) minimālo elementu vērtību un tas koordinātu [i,j] pēc rindām un kolonnam

# Atgriež minimālo vērtību un kur tā vērtība atrodas pēc koordinātam

# a - divdimensijas masīvs

min\_value = a[0][0]

coords = [0, 0]

for i in range(len(a)):

for j in range(len(a[0])):

if a[i][j] < min\_value:

min\_value = a[i][j]

coords = [i, j]

return (min\_value, coords)

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

n = input("Ievadiet matricas rindu skaitu ===> ")

while is\_natural(n) == False:

n = input("Kļūda! Matricas rindu skaitam jābūt naturālam skaitlim!\nIevadiet matricas rindu skaitu ===> ")

n = int(n)

m = input("Ievadiet matricas kolonnu skaitu ===> ")

while is\_natural(m) == False:

m = input("Kļūda! Matricas kolonnu skaitam jābūt naturālam skaitlim!\nIevadiet matricas kolonnu skaitu ===> ")

m = int(m)

a = ievade\_matrica(n, m)

print(matricas\_izvade\_4\_cipari(a))

max\_value\_and\_coord = max\_value\_in\_2\_dimensional\_array\_and\_its\_coords(a)

min\_value\_and\_coord = min\_value\_in\_2\_dimensional\_array\_and\_its\_coords(a)

print("Mazākais elements ir " + str(min\_value\_and\_coord[0]) + ", un tas atrodas " + str(min\_value\_and\_coord[1][0] + 1) + ".rindas un " + str(min\_value\_and\_coord[1][1] + 1) + ".kolonnas krustpunktā.")

print("Lielākais elements ir " + str(max\_value\_and\_coord[0]) + ", un tas atrodas " + str(max\_value\_and\_coord[1][0] + 1) + ".rindas un " + str(max\_value\_and\_coord[1][1] + 1) + ".kolonnas krustpunktā.")

**Testa piemēri:**

1)

Text

Description automatically generated

2)

A screenshot of a computer

Description automatically generated with medium confidence

3)

A picture containing graphical user interface

Description automatically generated

4)

Text

Description automatically generated with medium confidence

5)

A screenshot of a computer

Description automatically generated with medium confidence

6)

Text

Description automatically generated

7)

A picture containing text

Description automatically generated

8)

Text

Description automatically generated

9)

Text

Description automatically generated with medium confidence

10)

A picture containing text

Description automatically generated

**6. uzdevums**

Sastādīt programmu, kas nodrošina divu matricu elementu ievadi, kuri ir veseli skaitļi (matricu izmēri tiek izvēlēti tā), lai varētu veikt kādu no zemāk minētajām darbībām.

1. – sareizina abu matricu atbilstošās elementu vērtības.
2. – veic divu matricu reizināšanu.

Programmu var veidot tā, ka katrs no punktiem tiek pildīts atsevišķi vai arī vispirms ievada, kuru darbību veiks un tad izpilda tikai izvēlēto darbību.

**Kods:**

# Programmas nosaukums: Matricas reizinājums un matricu atbilstošo elementu reizinājums

# 6. uzdevums (1MPR08\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas nodrošina divdimensiju matricas elementu ievadi,

# kas ir veseli skaitļi. Matricas rindu un kolonnu skaitu ievada lietotājs.

# Pēc matricas elementu ievades tiek nodrošināta matricas elementu izvade\_matrica\_int tabulas veidā,

# pieņemot, ka neviens no matricas elementiem nesastāv no ne vairāk kā 4 cipariem, kā arī

# lielāko un mazāko matricas elementa vērtību un tā atrašanās vietu, noradot atbilstošo rindu un kolonnu.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def is\_natural(n):

# Pārbauda vai simbolu virkne ir naturāls skaitlis vai nav

# Ja ir naturāls skaitlis, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

if str(n).isdigit() and float(n) == int(n) and int(n) > 0:

return True

else:

return False

def is\_whole(n):

# Pārbauda vai simbolu virkne ir vesels skaitlis vai nav

# Ja ir vesels skaitlis, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

try:

n = int(n)

except:

return False

else:

return True

def ievade\_matrica(n, m):

# Lietotājs var ievādīt nXm matricas elementus un funkcija atgriež divdimensijas masīvu ar n rindam un m kolonnam ar ievadītām vērtībam

# n - naturāls skaitlis, kurš nosaka matricas rindas skaitu

# m - naturāls skaitlis, kurš nosaka matricas kolonnas skaitu

a = numpy.empty((n, m), dtype=int)

for i in range(n):

for j in range(m):

temp = input("Ievadiet matricas elememtu a(" + str(i) + "," + str(j) + ") ===> ")

while is\_whole(temp) == False:

temp = input("Kļūda! Ievadītais elements nav vesels skaitlis!\nIevadiet matricas elememtu a(" + str(i) + "," + str(j) + ") ===> ")

a[i, j] = int(temp)

return a

def izvade\_matrica\_int(a):

# Atgriež divdimensiju masīvu (matricu), tabulas veidā, str formāta, kur katra rinda ir atdalīta ar jauno rindkopu

# a - divdimensijas masīvs (matrica)

n = a.shape[0] # "x axis" masīva izmēri

m = a.shape[1] # "y axis" masīva izmēri

s = ""

for i in range(n):

for j in range(m):

s = s + "{:8.2f}".format(int(a[i, j]))

# s = s + " " + str(int(a[i,j])) #"{:8.2f}".format(a[i,j], dtype=int)

s = s + "\n"

return s

def matricas\_izvade\_4\_cipari(m):

# Izvada divdimensijas masīvu tabulas veidā tā, ka neviens no matricas elementiem nesastāv no ne vairāk kā 4 cipariem

# Tiek nodrošināta matricas elementu izvade\_matrica\_int tabulas veidā, pieņemot, ka neviens no matricas elementiem nesastāv no ne vairāk kā 4 cipariem

# Atgriež matricas elementu izvade\_matrica\_int tabulas veidā

# m - divdimensijas masīvs

nm = (numpy.shape(m))

sv = ""

for x in range(nm[0]):

for y in range(nm[1]):

for i in range(5 - len(str(m[x, y]))):

sv = sv + " "

sv = sv + str(m[x, y])

sv = sv + "\n"

nn = len(sv)

return sv[:nn - 1]

def check\_fake\_matrix\_multiplication(n1, m1, n2, m2):

# Pārbauda vai divas matricas (divdimensiju masīvi) ir ar vienādu izmēru. Pārbauda vai sakrīt 1.matricas rindas skaits ar 2.matricas rindas skaitu.

# un pārbauda vai 1.matricas kolonnu skaits sakrīt ar 2.matricas kolonnu skaitu. Ja abas prasības izpildās, tad return True. Ja kaut viena neizpildās, tad return False.

# Tiek izmantota, lai pārbaudītu vai var sareizināt matricas atbilstošus elementus vai nē.

# n1 - 1.matricas rindu skaits

# m1 - 1.matricas kolonnu skaits

# n2 - 2.matricas rindu skaits

# m2 - 2.matricas kolonnu skaits

if n1 == n2 and m1 == m2:

return True

else:

return False

def check\_matrix\_multiplication(n1, m1, n2, m2):

# Pārbauda vai 1.matricas kolonnu skaits sakrīt ar 2.matricas rindu skaitu. Ja sakrīt, tad return True. Ja nē, tad return False.

# Tiek izmantots, lai pārbaudītu vai ir iespējams sareizināt divas matricas.

# n1 - 1.matricas rindu skaits (tā ne uz ko neietekme)

# m1 - 1.matricas kolonnu skaits

# n2 - 2.matricas rindu skaits

# m2 - 2.matricas kolonnu skaits (tā ne uz ko neietekme)

if m1 == n2:

return True

else:

return False

def matrix\_multiplication\_integer(a, b):

# Sareizina divu divdimensijas masīvus (divas matricas) a ar b.

# Atgriež divdimensiju masīvu, vai ja nevar sareizināt atgriež "Kļūda"

# a - divdimensijas masīvs

# b - divdimensijas masīvs

n1 = a.shape[0]

m1 = a.shape[1]

n2 = b.shape[0]

m2 = b.shape[1]

if m1 == n2:

c = numpy.zeros((n1, m2), numpy.int\_)

for i in range(n1):

for j in range(m2):

for k in range(m1):

c[i, j] = c[i, j] + a[i, k] \* b[k, j]

return c

else:

return "Kļūda"

def fake\_matrix\_multiplication(a, b):

# Sareizina divas matricas atbilstošus elementus un atgriež atbilstošu divdimensiju masīvu

# a - divdimensiju masīvs

# b - divdimensiju masīvs

c = numpy.empty((n1, m1), dtype=int)

for i in range(n1):

for j in range(m2):

c[i, j] = a[i][j] \* b[i][j]

return c

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

n1 = input("Ievadiet 1.matricas rindu skaitu ===> ")

while is\_natural(n1) == False:

n1 = input("Kļūda! Matricas rindu skaitam jābut naturālam skaitlim!\nIevadiet 1.matricas rindu skaitu ===> ")

n1 = int(n1)

m1 = input("Ievadiet 1.matricas kolonnu skaitu ===> ")

while is\_natural(m1) == False:

m1 = input("Kļūda! Matricas kolonnu skaitam jābut naturālam skaitlim!\nIevadiet 1.matricas kolonnu skaitu ===> ")

m1 = int(m1)

a = ievade\_matrica(n1, m1)

n2 = input("\nIevadiet 2.matricas rindu skaitu ===> ")

while is\_natural(n2) == False:

n2 = input("Kļūda! Matricas rindu skaitam jābut naturālam skaitlim!\nIevadiet 2.matricas rindu skaitu ===> ")

n2 = int(n2)

m2 = input("Ievadiet 2.matricas kolonnu skaitu ===> ")

while is\_natural(m2) == False:

m2 = input("Kļūda! Matricas kolonnu skaitam jābut naturālam skaitlim!\nIevadiet 2.matricas kolonnu skaitu ===> ")

m2 = int(m2)

b = ievade\_matrica(n2, m2)

print("\nPirmā ievadīta matrica:")

print(matricas\_izvade\_4\_cipari(a))

print("\nOtrā ievadīta matrica:")

print(matricas\_izvade\_4\_cipari(b))

if check\_fake\_matrix\_multiplication(n1, m1, n2, m2):

print("\nAbu matricu atbilstošās elementu reizinājums:")

c = fake\_matrix\_multiplication(a, b)

print(matricas\_izvade\_4\_cipari(c))

else:

print("\nKļūda! Šādas matricas nevar sareizināt atbilstošus elementus.")

if check\_matrix\_multiplication(n1, m1, n2, m2):

print("\nDivu matricu reizinājums:")

d = matrix\_multiplication\_integer(a, b)

print(matricas\_izvade\_4\_cipari(d))

else:

print("\nKļūda! Šādas matricas nevar sareizināt, jo 1.matricas kolonnu skaits nav vienāds ar 2.matricas rindas skaitu.")

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated with medium confidence

6)

Text

Description automatically generated

7)

Text

Description automatically generated

8)

Text

Description automatically generated

9)

Text

Description automatically generated

10)

Text

Description automatically generated

11)

Calendar

Description automatically generated

12)

Text

Description automatically generated

13)

Text

Description automatically generated

14)

Graphical user interface

Description automatically generated with medium confidence

15)

Text

Description automatically generated

**PU1. uzdevums**

Sastādīt programmu, kas realizē lielo naturālo skaitļu (vismaz ar 50 cipariem) dalīšanas ar atlikumu algoritmu. Jāveic ievaddatu korektuma pārbaude!

**Kods:**

# Programmas nosaukums: Lielo skaitļu dalīšana ar atlikumu

# 1.Papilduzdevums (1MPR08\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē lielo naturālo skaitļu (vismaz ar 50 cipariem) dalīšanas ar atlikumu algoritmu.

# Jāveic ievaddatu korektuma pārbaude!

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import math

def dalit\_lielus\_skaitlus(s, n):

# Funkcija atgriež tuple (dalījums, atlikums) no s / n. Dalījums ir int un atlikums ir int.

# Jāizmanto lielo skaitļu dalīšanai.

# Dalā pēc "cipariem".

# s - str, dalāmais

# n - str, dalītājs

n = int(n)

galva = 0

atlikums = 0

dalijums\_sv = ""

for i in range(len(s)):

cipars = int(s[i])

tmp = cipars + atlikums \* galva

dalijums\_sv = dalijums\_sv + str(tmp // n)

if tmp % n == 0:

galva = 0

atlikums = 0

else:

atlikums = tmp % n

galva = 10

dalijums = nodzest\_liekas\_nulles(dalijums\_sv)

dalijums = int(dalijums)

return (dalijums, atlikums)

def nodzest\_liekas\_nulles(s):

# Nodzes liekas nulles sākumā

# s - simbolu virkne

for i in range(len(s)):

if s[i] != '0':

return s[i:]

return '0'

def is\_natural\_long(s):

# Pārbauda vai simbolu virkne reprezentē naturālo skaitli, vai nē.

# Atgriež True, ja virkne reprezentē naturālo skaitli.

# Atgriež False, ja nereprezentē naturālo skaitli.

# s - pārbaudama simbolu virkne

# Noņema no virknes visas sākuma vai beigu atstarpes

s = s.strip()

# Pārbauda, vai virkne ir tukša

if len(s) == 0 or (len(s) == 1 and s == "0"):

return False

# Iet cikliski cauri katrām simbolam simbolu virknē (string'ā)

for c in s:

# Ja kāda rakstzīme nav cipars, virkne neatspoguļo naturālu skaitli. return False

if not c.isdigit():

return False

# Virkne atspoguļo naturālu skaitli, ja ietu cauri ciklas netika pamanīts not .isdigit()

return True

def is\_natural\_or\_zero\_long(s):

# Pārbauda vai simbolu virkne reprezentē naturālo skaitli vai 0, vai nē.

# Atgriež True, ja virkne reprezentē naturālo skaitli.

# Atgriež False, ja nereprezentē naturālo skaitli.

# s - pārbaudama simbolu virkne

# Noņema no virknes visas sākuma vai beigu atstarpes

s = s.strip()

if not s.isdigit():

return False

if len(s) == 1 and s == "0":

return True

# Iet cikliski cauri katrām simbolam simbolu virknē (string'ā)

for c in s:

# Ja kāda rakstzīme nav cipars, virkne neatspoguļo naturālu skaitli. return False

if not c.isdigit():

return False

# Virkne atspoguļo naturālu skaitli, ja ietu cauri ciklas netika pamanīts not .isdigit()

return True

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

dalamais = input("Ievadiet dalāmo ==> ")

while is\_natural\_or\_zero\_long(dalamais) == False:

dalamais = input("Kļūda! Ievadiet naturālo dalāmo vai nulle ==> ")

dalitajs = input("Ievadiet dalītāju ==> ")

while is\_natural\_long(dalitajs) == False:

dalitajs = input("Kļūda! Ievadiet naturālo dalītāju ==> ")

rezultats = dalit\_lielus\_skaitlus(dalamais, dalitajs)

print("\nNepilnais dalījums:")

print(rezultats[0])

print("\nAtlikums:")

print(rezultats[1])

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

7)

Text

Description automatically generated

8)

Text

Description automatically generated

9)

Text

Description automatically generated

10)

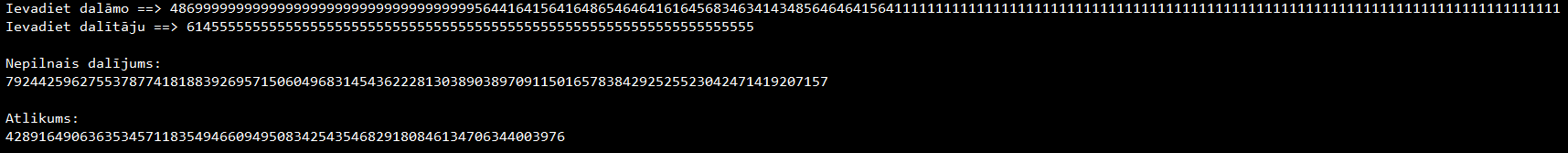
Text

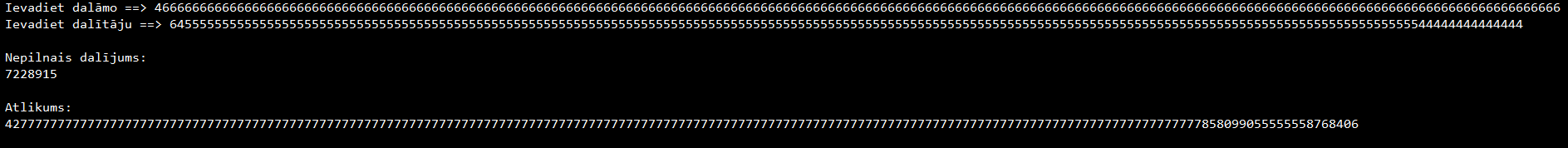
Description automatically generated

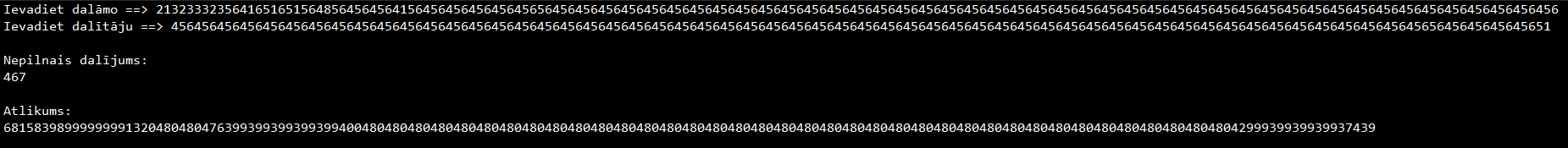
11)

Text

Description automatically generated

12)

13)

14)

**PU2. uzdevums**

Sastādīt programmu, kas realizē kvadrātsaknes izvilkšanas algoritmu lielajiem naturālajiem skaitļiem (vismaz ar 50 cipariem) ar precizitāti līdz veselam skaitlim. Jāveic ievaddatu korektuma pārbaude!

**Kods:**

# Programmas nosaukums: Lielo skaitļu kvadrātsaknes izvilkšanas algoritms

# 2.Papilduzdevums (1MPR08\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē kvadrātsaknes izvilkšanas algoritmu lielajiem naturālajiem skaitļiem (vismaz ar 50 cipariem) ar precizitāti līdz veselam skaitlim.

# Jāveic ievaddatu korektuma pārbaude!

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

# import math # Testēšanai

def long\_sqrt(skaitlis):

# Funkciju, kas kā ievadi izmanto nenegatīvu veselu skaitli un atgriež tā veselā skaitļa kvadrātsakni

# skaitlis - int, skaitlis, no kura vajag atrast kvadrātsakni

# Atgriež sqrt(skaitlis) kā int

# Konvertējat ievadīto veselo nenegatīvo skaitli par simbolu virkni un saglabājam to mainīgajā

sk\_sv = str(skaitlis) # sk\_sv - skaitlis\_simbolu virkne

# Saglabājam ievadītā veselā skaitļa simbolu virknes garumu

len\_sk = len(sk\_sv) # slen\_skaitlis

# Ja ievadītā veselā skaitļa simbolu virknes garums ir nepāra skaitlis, tad pievienojam sākuma nulli, lai padarītu to par pāru simbolu virkni (garums ir pāra skaitlis)

if len\_sk % 2 == 1:

sk\_sv = '0' + sk\_sv # Ievadītā veselā skaitļa simbolu virknei sākumam pievienojam '0'

len\_sk = len\_sk + 1 # Palielinām ievadītā veselā skaitļa simbolu virknes garumu par 1

result = '' # Tukša simbolu virkne, lai saglabātu kvadrātsaknes aprēķina rezultātu

atlikums = 0

# Sadalām ievadītā veselā skaitļa virknes pirmos divus ciparus [:2]

a = int(sk\_sv[:2])

j = 1 # Atrodam pirmo "tuvinājumu"

while j \* j <= a: # Tas ir lai atrastu pirmo ciparu kvadrātsaknei

j = j + 1 # Var arī atrast tā:

sakne = j - 1 # sakne = int(a\*\*0.5) (lai nebūtu cikls)

result = result + str(sakne)

atlikums = a - sakne \* sakne # Atlikums

# Cikls pār katru otro "ciparu pāri"

for i in range(2, len\_sk, 2): # Stabiņveidā atrodam nākamo un nākamo un ... un pēdējo ciparu

# Reizinām atlikumu ar 100 un pievienojiet nākamos divus veselā skaitļa ciparus, lai iegūtu dividendi

a = atlikums \* 100 + int(sk\_sv[i:i + 2])

cipars = 0

temp = sakne \* 20

while (temp + cipars) \* cipars <= a:

cipars += 1

cipars -= 1

result += str(cipars) # Rezultātam pievienojam iegūto ciparu

atlikums = a - (temp + cipars) \* cipars

sakne = sakne \* 10 + cipars

return int(result)

def is\_natural\_or\_zero\_long(s):

# Pārbauda vai simbolu virkne reprezentē naturālo skaitli vai 0, vai nē.

# Atgriež True, ja virkne reprezentē naturālo skaitli.

# Atgriež False, ja nereprezentē naturālo skaitli.

# s - pārbaudāma simbolu virkne

# Noņema no virknes visas sākuma vai beigu atstarpes

s = s.strip()

if not s.isdigit():

return False

if len(s) == 1 and s == "0":

return True

# Iet cikliski cauri katrām simbolam simbolu virknē (string'ā)

for c in s:

# Ja kāda rakstzīme nav cipars, virkne neatspoguļo naturālu skaitli. return False

if not c.isdigit():

return False

# Virkne atspoguļo naturālu skaitli, ja ietu cauri ciklas netika pamanīts not .isdigit()

return True

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

n = input("Ievadiet naturālo skaitli vai 0 kvadrātsaknes vilkšanai ==> ")

while is\_natural\_or\_zero\_long(n) == False:

n = input("Kļūda! Ievadiet naturālo skaitli vai 0 kvadrātsaknes vilkšanai ==> ")

n = int(n)

print("\nsqrt(" + str(n) + ")")

print("≈")

print(long\_sqrt(n))

# math.sqrt(skaitlis) # Testēšanai

# Testēšanai

'''

for i in range(1, 10000000, 1): # for i in range(1, 1000000, 1):

print("sqrt(" + str(i) + ") =", math.floor(math.sqrt(i)))

print(str(long\_sqrt(i)) + " == " + str(math.floor(math.sqrt(i))))

if long\_sqrt(i) == math.floor(math.sqrt(i)):

print("TRUE\n")

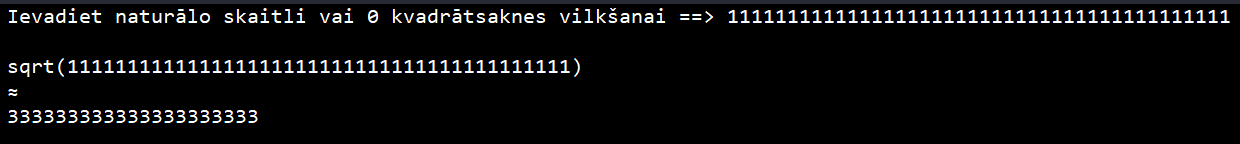
else:

print("FALSE\n")

'''

**Testa piemēri:**

1)



2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

7)

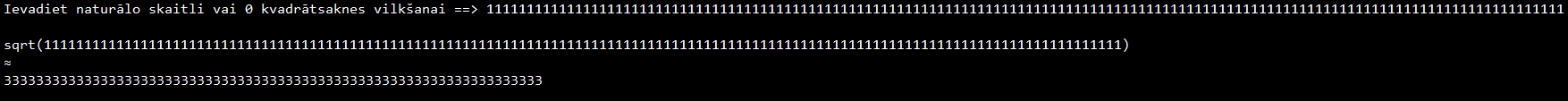
Text

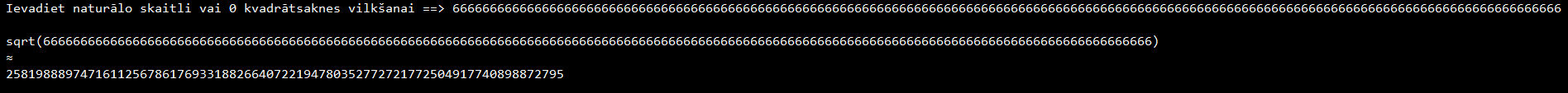
Description automatically generated

8)

Text

Description automatically generated

9)

10)

11)

Text

Description automatically generated

12)

Text

Description automatically generated

13)

Text

Description automatically generated

14)

Text

Description automatically generated with medium confidence

15)

Text

Description automatically generated

16)

Text

Description automatically generated with medium confidence

17)

Text

Description automatically generated

18)

Text

Description automatically generated

19)

Text

Description automatically generated

20)

A screenshot of a computer

Description automatically generated with medium confidence

21)

Text

Description automatically generated with medium confidence