**7. praktiskais darbs. 2. semestris**

**1. uzdevums**

Lietotājs ievada divus dažāda garuma augošā secībā sakārtotus masīvus. Lineārā laikā apvienot abus masīvus vienā augoša secībā sakārtotā masīvā un izvadīt to uz ekrāna.

**Kods:**

# Programmas nosaukums: Apvienot divus masīvus augoša secībā lineārā laikā

# 1. uzdevums (1MPR07\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Lietotājs ievada divus dažāda garuma augošā secībā sakārtotus masīvus.

# Lineārā laikā apvienot abus masīvus vienā augoša secībā sakārtotā masīvā un izvadīt to uz ekrāna.

# Versija 1.0

import numpy

def izveidot\_masivu\_ar\_garumu(n):

# Izveido masīvu ar noradīto garumu n

# n - naturāls skaitlis

a = numpy.arange(n)

for i in range(n):

b = input("Ievadiet " + str(i) + ".elementu ===> ")

b = is\_whole(b, i)

a[i] = b

return a

def izvade(x):

# Izvada masīva elementus pēc kārtas līdz pedējam

# x - viendimensijas masīvs

n = len(x)

s = str(x[0])

for i in range(1, n):

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

print(s)

def is\_whole(x, i):

# Pārbauda vai simbolu virkne ir vesels skaitlis un ja nē, tad paprasa ievadīt to vēlreiz (bezgalīgi daudz ievades)

# Ja simbolu virkne ir vesels skaitlis, tad atgriež to kā int(x)

# x - pārbaudāma simbolu virkne

# i - i-tajs elements

while True:

try:

x = int(x)

except:

x = input("Kļūda! Ievadiet " + str(i) + ".elementu ===> ")

else:

return int(x)

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 apvieno(a, b):

# Apvieno divus sakartotus masīvus a un b, un sakārto tos

# Lineāra laiks o(n)

# a - viendimensijas masīvs

# b - viendimensijas masīvs

ga = len(a)

gb = len(b)

gc = ga + gb

c = numpy.arange(gc)

ia = 0

ib = 0

ic = 0

while (ia < ga) and (ib < gb):

if a[ia] < b[ib]:

c[ic] = a[ia]

ia = ia + 1

else:

c[ic] = b[ib]

ib = ib + 1

ic = ic + 1

if ia < ga:

for i in range(ia, ga):

c[ic] = a[i]

ic = ic + 1

else:

for i in range(ib, gb):

c[ic] = b[i]

ic = ic + 1

return c

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

# Galvenā programmas daļa

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

m = input("Ievadiet 1. masīva izmēru ===> ")

while is\_natural(m) == False:

m = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

m = int(m)

print("\nIevadiet sakārota augošā secība masīva skaitļus!")

a = izveidot\_masivu\_ar\_garumu(m)

n = input("\nIevadiet 2. masīva izmēru ===> ")

while is\_natural(n) == False:

n = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

n = int(n)

print("\nIevadiet sakārota augošā secība masīva skaitļus!")

b = izveidot\_masivu\_ar\_garumu(n)

print("\nPirmais augošā secība sakārtots masīvs:")

izvade(a)

print("\nOtrais augošā secība sakārtots masīvs:")

izvade(b)

c = apvieno(a, b)

print("\nApvienots augošā secība sakārtots masīvs:")

izvade(c)

**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

**2. uzdevums**

Lietotājs ievada trīs dažāda garuma masīvus, no kuriem pirmais ir sakārtots augošā secībā, otrais ir sakārtots dilstošā secībā un trešais sakārtots augošā secībā. Lineārā laikā apvienot visus trīs masīvus vienā dilstošā secībā sakārtotā masīvā un izvadīt to uz ekrāna.

**Kods:**

# Programmas nosaukums: Trīs masīvu apvienošana. 1.-augošs, 2.-dilstošs, 3.-augošs

# 2. uzdevums (1MPR07\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Lietotājs ievada trīs dažāda garuma masīvus, no kuriem pirmais ir sakārtots augošā secībā,

# otrais ir sakārtots dilstošā secībā un trešais sakārtots augošā secībā.

# Lineārā laikā apvienot visus trīs masīvus vienā dilstošā secībā sakārtotā masīvā un izvadīt to uz ekrāna.

# Versija 1.0

import numpy

def izveidot\_masivu\_ar\_garumu(n):

# Izveido masīvu ar noradīto garumu n

# n - naturāls skaitlis

a = numpy.arange(n)

for i in range(n):

b = input("Ievadiet " + str(i) + ".elementu ===> ")

b = is\_whole(b, i)

a[i] = b

return a

def izvade(x):

# Izvada masīva elementus pēc kārtas līdz pedējam

# x - viendimensijas masīvs

n = len(x)

s = str(x[0])

for i in range(1, n):

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

print(s)

def is\_whole(x, i):

# Pārbauda vai simbolu virkne ir vesels skaitlis un ja nē, tad paprasa ievadīt to vēlreiz (bezgalīgi daudz ievādes)

# Ja simbolu virkne ir vesels skaitlis, tad atgriež to kā int(x)

# x - pārbaudāma simbolu virkne

# i - i-tajs elements

while True:

try:

x = int(x)

except:

x = input("Kļūda! Ievadiet " + str(i) + ".elementu ===> ")

else:

return int(x)

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 apvieno(a, b):

# Apvieno divus sakartotus masīvus a un b, un sakarto tos

# Lineāra laiks o(n)

# a - viendimensijas masīvs

# b - viendimensijas masīvs

ga = len(a)

gb = len(b)

gc = ga + gb

c = numpy.arange(gc)

ia = 0

ib = 0

ic = 0

while (ia < ga) and (ib < gb):

if a[ia] < b[ib]:

c[ic] = a[ia]

ia = ia + 1

else:

c[ic] = b[ib]

ib = ib + 1

ic = ic + 1

if ia < ga:

for i in range(ia, ga):

c[ic] = a[i]

ic = ic + 1

else:

for i in range(ib, gb):

c[ic] = b[i]

ic = ic + 1

return c

def reverse(masivs):

# Pārkarto masīvā visus elementus pretēji

# masivs - viendimensijas masīvs

start\_index = 0

end\_index = len(masivs) - 1

while end\_index > start\_index:

temp = masivs[start\_index]

masivs[start\_index] = masivs[end\_index]

masivs[end\_index] = temp

start\_index = start\_index + 1

end\_index = end\_index - 1

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

# Galvenā programmas daļa

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

m = input("Ievadiet 1. masīva izmēru ===> ")

while is\_natural(m) == False:

m = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

m = int(m)

print("\nIevadiet sakārota augošā (nedilstoša) secība masīva skaitļus!")

a = izveidot\_masivu\_ar\_garumu(m)

n = input("\nIevadiet 2. masīva izmēru ===> ")

while is\_natural(n) == False:

n = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

n = int(n)

print("\nIevadiet sakārota dilstoša (neaugoša) secība masīva skaitļus!")

b = izveidot\_masivu\_ar\_garumu(n)

k = input("\nIevadiet 3. masīva izmēru ===> ")

while is\_natural(k) == False:

n = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

k = int(k)

print("\nIevadiet sakārota augošā (nedilstoša) secība masīva skaitļus!")

c = izveidot\_masivu\_ar\_garumu(k)

print("\nPirmais augošā (nedilstoša) secība sakārtots masīvs:")

izvade(a)

print("\nOtrais dilstoša (neaugoša) secība sakārtots masīvs:")

izvade(b)

print("\nTrešais augošā (nedilstoša) secība sakārtots masīvs:")

izvade(c)

reverse(b)

ab = apvieno(a, b)

abc = apvieno(ab, c)

reverse(abc)

print("\nApvienoti visi trīs sakārtoti masīvi dilstoša (neaugoša) secība:")

izvade(abc)

**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

**PU1. uzdevums**

Lietotājs ievada trīs sakārtotus masīvus, bet nav zināms, kurš no tiem ir sakārtots augošā vai dilstošā secībā. Lineārā laikā apvienot visus trīs masīvus vienā augošā secībā sakārtotā masīvā un izvadīt to uz ekrāna.

**Kods:**

# Programmas nosaukums: Trīs masīvu apvienošana

# Papilduzdevums 1 (1MPR07\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Lietotājs ievada trīs sakārtotus masīvus, bet nav zināms, kurš no tiem ir sakārtots augošā vai dilstošā secībā.

# Lineārā laikā apvienot visus trīs masīvus vienā augošā secībā sakārtotā masīvā un izvadīt to uz ekrāna.

# Versija 1.0

import numpy

def izveidot\_masivu\_ar\_garumu(n):

# Izveido masīvu ar noradīto garumu n

# n - naturāls skaitlis

a = numpy.arange(n)

for i in range(n):

b = input("Ievadiet " + str(i) + ".elementu ===> ")

b = is\_whole(b, i)

a[i] = b

return a

def izvade(x):

# Izvada masīva elementus pēc kārtas līdz pēdējam

# x - viendimensijas masīvs

n = len(x)

s = str(x[0])

for i in range(1, n):

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

print(s)

def is\_whole(x, i):

# Pārbauda vai simbolu virkne ir vesels skaitlis un ja nē, tad paprasa ievadīt to vēlreiz (bezgalīgi daudz ievades)

# Ja simbolu virkne ir vesels skaitlis, tad atgriež to kā int(x)

# x - pārbaudāma simbolu virkne

# i - i-tajs elements

while True:

try:

x = int(x)

except:

x = input("Kļūda! Ievadiet " + str(i) + ".elementu ===> ")

else:

return int(x)

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 apvieno(a, b):

# Apvieno divus sakārtotus masīvus a un b, un sakārto tos

# Ātrums - lineārs laiks o(n)

# a - viendimensijas masīvs

# b - viendimensijas masīvs

ga = len(a)

gb = len(b)

gc = ga + gb

c = numpy.arange(gc)

ia = 0

ib = 0

ic = 0

while (ia < ga) and (ib < gb):

if a[ia] < b[ib]:

c[ic] = a[ia]

ia = ia + 1

else:

c[ic] = b[ib]

ib = ib + 1

ic = ic + 1

if ia < ga:

for i in range(ia, ga):

c[ic] = a[i]

ic = ic + 1

else:

for i in range(ib, gb):

c[ic] = b[i]

ic = ic + 1

return c

def reverse(masivs):

# Pārkarto masīvā visus elementus pretēji

# masivs - viendimensijas masīvs

start\_index = 0

end\_index = len(masivs) - 1

while end\_index > start\_index:

temp = masivs[start\_index]

masivs[start\_index] = masivs[end\_index]

masivs[end\_index] = temp

start\_index = start\_index + 1

end\_index = end\_index - 1

return masivs

def is\_ascending(n): # vai nedilstoša?

# Pārbauda vai masīvs ir augošs (nedilstošs)

# Ja masīvs ir nedilstošs (nav augošs), tad return True

# Ja masīvs nav nedilstošs (nav augošs), tad return False

# n - viendimensijas masīvs

if len(n) == 1:

return True

for i in range(0, len(n)):

if i < len(n) - 1 and n[i] > n[i + 1]:

return False # Nē, nav augoša, nav nedilstoša, nav konstanta (varētu būt augoša, vai nekāda)

return True # Jā, ir augoša (vai nedilstoša)

def is\_descending(n): # vai neaugoša?

# Pārbauda vai masīvs ir dilstošs (neaugošs)

# Ja masīvs ir neaugošs (nav dilstošs), tad return True

# Ja masīvs nav neaugošs (nav dilstošs), tad return False

# n - viendimensijas masīvs

if len(n) == 1:

return True

for i in range(0, len(n)):

if i < len(n) - 1 and n[i] < n[i + 1]:

return False # Nē, nav dilstoša, nav neaugoša, nav konstanta (varētu būt augoša, vai nekāda)

return True # Jā, ir dilstoša (vai neaugoša)

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

# Galvenā programmas daļa

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

m = input("Ievadiet 1. masīva izmēru ===> ")

while is\_natural(m) == False:

m = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

m = int(m)

print("\nIevadiet sakārota augoša (nedilstoša) vai dilstoša (neaugoša) secība masīva skaitļus!")

a = izveidot\_masivu\_ar\_garumu(m)

if is\_ascending(a) == False and is\_descending(a) == False: # tad nekāda

print("\nKļūda!\nPirmais ievadītais masīvs:")

izvade(a)

print("Nav ne augoši sakārtots, nav ne dilstoši sakārtots!")

quit()

n = input("\nIevadiet 2. masīva izmēru ===> ")

while is\_natural(n) == False:

n = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

n = int(n)

print("Ievadiet sakārota augoša (nedilstoša) vai dilstoša (neaugoša) secība masīva skaitļus!")

b = izveidot\_masivu\_ar\_garumu(n)

if is\_ascending(b) == False and is\_descending(b) == False:

print("\nKļūda!\nOtrais ievadītais masīvs:")

izvade(b)

print("Nav ne augoši sakārtots, nav ne dilstoši sakārtots!")

quit()

k = input("\nIevadiet 3. masīva izmēru ===> ")

while is\_natural(k) == False:

n = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

k = int(k)

print("\nIevadiet sakārota augoša (nedilstoša) vai dilstoša (neaugoša) secība masīva skaitļus!")

c = izveidot\_masivu\_ar\_garumu(k)

if is\_ascending(c) == False and is\_descending(c) == False:

print("\nKļūda!\nTrešais ievadītais masīvs:")

izvade(c)

print("Nav ne augoši sakārtots, nav ne dilstoši sakārtots!")

quit()

print("\nPirmais sakārtots masīvs:")

izvade(a)

print("\nOtrais sakārtots masīvs:")

izvade(b)

print("\nTrešais sakārtots masīvs:")

izvade(c)

if is\_ascending(a) == False:

reverse(a)

if is\_ascending(b) == False:

reverse(b)

if is\_ascending(c) == False:

reverse(c)

ab = apvieno(a, b)

abc = apvieno(ab, c)

print("\nApvienots augošā (nedilstoša) secība sakārtots masīvs:")

izvade(abc)

**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

**3. uzdevums**

Sastādīt programmu, kas realizē saliešanas (izmantojot skaldi un valdi) kārtošanas metodi.

Norādījumi uzdevuma risinājuma izveidei:

* Nedrīkst rakstīt algoritmu, kas vispirms sakārto masīvu augošā secībā un pēc tam to nodrukā apgrieztā secībā!
* Katrā kārtošanas algoritma kods jāpapildina tā, lai tiktu saskaitīts masīva elementu salīdzināšanas reižu skaits.
* Lai salīdzinātu salīdzināšanas reižu skaitu katrai metodei, ar katru no metodēm jākārto viens un tas pats masīvs.
* Salīdzināt iegūtos rezultātus ar iepriekšējos praktiskajos darbos apskatītajiem kārtošanas algoritmiem.
* Programmas izpildes rezultātā uz ekrāna tiek parādīti abu kārtošanas metožu izpildes rezultāti, sakārtojot 10, 1 000, 10 000 skaitļus.

**Kods:**

# Programmas nosaukums: Saliešanas

# 3. uzdevums (1MPR07\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē saliešanas (izmantojot "Skaldi un valdi") kārtošanas metodi.

# Norādījumi uzdevuma risinājuma izveidei:

# Nedrīkst rakstīt algoritmu, kas vispirms sakārto masīvu augošā secībā un pēc tam to nodrukā apgrieztā secībā!

# Katrā kārtošanas algoritma kods jāpapildina tā, lai tiktu saskaitīts masīva elementu salīdzināšanas reižu skaits.

# Lai salīdzinātu salīdzināšanas reižu skaitu katrai metodei, ar katru no metodēm jākārto viens un tas pats masīvs.

# Salīdzināt iegūtos rezultātus ar iepriekšējos praktiskajos darbos apskatītajiem kārtošanas algoritmiem.

# Programmas izpildes rezultātā uz ekrāna tiek parādīti abu kārtošanas metožu izpildes rezultāti, sakārtojot 10, 1 000, 10 000 skaitļus.

# Versija 1.0

import numpy

import random

import math

import copy

def skaldi\_un\_valdi\_dilstosa(a, sv, bv):

# Sakarto masīvu dilstoša secība izmantojot "Skaldi un valdi" algoritmu un atgriež salīdzināšanas skaitu

# ātrums - o(nlog(n))

# a - viendimensijas masīvs

# sv - sākumvērtība (parasti 0)

# bv - beigu vērtība (parasti len(a))

p = 0

b = numpy.arange(len(a))

if sv < bv:

vv = (sv + bv) // 2

p1 = skaldi\_un\_valdi\_dilstosa(a, sv, vv)

p2 = skaldi\_un\_valdi\_dilstosa(a, vv + 1, bv)

for i in range(sv, bv + 1):

b[i] = a[i]

i = sv

j = vv + 1

k = sv

while (i <= vv) and (j <= bv):

p += 1

if b[i] > b[j]:

a[k] = b[i]

i = i + 1

else:

a[k] = b[j]

j = j + 1

k = k + 1

if j > bv:

while i <= vv:

a[k] = b[i]

i = i + 1

k = k + 1

p = p + p1 + p2

return p

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 izvadit\_masivu(x):

# Izvada masīva elementus ar komatiem

# x - viendimensijas masīvs

n = len(x)

s = str(x[0])

for i in range(1, n):

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

return s

def sort\_atspole\_dilstosa(a):

# Sakārto masīvu dilstoša secība un atgriež salīdzināšanas skaitu, lai sakārtotu masīvu

# Kārtošanas tiek izmantota atspoles metode

# a - viendimensijas masīvs

n = len(a)

count = 0

for i in range(1, n):

if a[i - 1] < a[i]:

count += 1

for j in range(i, 0, -1):

if a[j - 1] < a[j]:

count += 1

x = a[j]

a[j] = a[j - 1]

a[j - 1] = x

else:

count += 1

break

else:

count += 1

return count

def sort\_ievietosana\_dilstosa(a):

# Sakārto masīvu dilstoša secība un atgriež salīdzināšanas skaitu, lai sakārtotu masīvu

# Kārtošanas tiek izmantota ievietošanas metode (insertion sort)

# a - viendimensijas masīvs

n = len(a)

sk = 0

for i in range(1, n):

sk = sk + 1

if a[i - 1] < a[i]:

x = a[i]

j = i

sk = sk + 1

while a[j - 1] < x:

a[j] = a[j - 1]

j = j - 1

if j == 0:

break

sk = sk + 1

a[j] = x

return sk

def sort\_sella\_dilstosa(a):

# Sakārto masīvu dilstoša secība un atgriež salīdzināšanas skaitu, lai sakārtotu masīvu

# Kārtošanas tiek izmantota Šellas metode (Shell sort)

# a - viendimensijas masīvs

sk = 0

n = len(a)

solis = (3\*\*math.floor(math.log(2 \* n + 1, 3)) - 1) // 2

while solis >= 1:

for k in range(0, solis):

for i in range(solis + k, n, solis):

sk = sk + 1

if a[i - solis] < a[i]:

x = a[i]

j = i

sk = sk + 1

while a[j - solis] < x:

a[j] = a[j - solis]

j = j - solis

if j == k:

break

sk = sk + 1

a[j] = x

solis = (solis - 1) // 3

return sk

def sort\_atrais\_dilstosa(a, sv, bv):

# Sakārto masīvu dilstoša secība un atgriež salīdzināšanas skaitu, lai sakārtotu masīvu

# Kārtošanas tiek izmantota Hoara (ātrais) metode (quicksort)

# a - viendimensijas masīvs

# sv - sākuma vērtība

# bv - beigu vērtība

p = 0

if sv < bv:

i = sv

j = bv

solis = -1

lv = True

while i != j:

if lv == (a[i] < a[j]):

x = a[i]

a[i] = a[j]

a[j] = x

x = i

i = j

j = x

lv = not lv

solis = -solis

p += 1

j = j + solis

p1 = sort\_atrais\_dilstosa(a, sv, i - 1)

p2 = sort\_atrais\_dilstosa(a, i + 1, bv)

p = p + p1 + p2

return p

def izvadit\_masivu\_un\_salidzinasanas\_skaitu(x, sal):

# Izvada salīdzināšanas skaitu un izvada masīva elementus ar komatiem

# x - viendimensijas masīvs

# sal - int skaitlis

n = len(x)

s = str(x[0])

for i in range(1, n):

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

return str(sal) + " salīdzināšanas - " + s

def samaisit(masivs):

# Samaisa masīva elementus (funkcija tiek izmantota, lai no sakārtota masīva numpy.arange(n + 1) izveidot nesakārtotu (unsort)

# masivs - viendimensijas masīvs

i = 0

while i < len(masivs) // 2:

x = random.randint(1, len(masivs) - 1)

y = random.randint(1, len(masivs) - 1)

temp = masivs[x]

masivs[x] = masivs[y]

masivs[y] = temp

i = i + 1

return masivs

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

# Galvenā programmas daļa

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

n = input("Ievadiet masīva izmēru N ===> ")

while is\_natural(n) == False:

n = input("Masīva izmērs ir naturāls skaitlis!\nIevadiet masīva izmēru N ===> ")

n = int(n)

a = numpy.arange(n + 1)

samaisit(a)

print("\nSākotnējais masīvs:")

print(izvadit\_masivu(a))

y = copy.deepcopy(a)

z = copy.deepcopy(a)

k = copy.deepcopy(a)

p = copy.deepcopy(a)

u = copy.deepcopy(a)

sorted\_b = sort\_atspole\_dilstosa(y)

print("\nKārtošana ar atspoles metodi:")

print(izvadit\_masivu\_un\_salidzinasanas\_skaitu(y, sorted\_b))

sorted\_c = sort\_ievietosana\_dilstosa(z)

print("\nKārtošana ar ievietošanas metodi:")

print(izvadit\_masivu\_un\_salidzinasanas\_skaitu(z, sorted\_c))

sorted\_d = sort\_sella\_dilstosa(p)

print("\nKārtošana ar Šella metodi:")

print(izvadit\_masivu\_un\_salidzinasanas\_skaitu(p, sorted\_d))

sorted\_e = sort\_atrais\_dilstosa(k, 0, len(a) - 1)

print("\nKārtošana ar Hoara (ātrais) metodi:")

print(izvadit\_masivu\_un\_salidzinasanas\_skaitu(k, sorted\_e))

print("\nKārtošana ar 'Skaldi un valdi' metodi:")

sorted\_u = skaldi\_un\_valdi\_dilstosa(u, 0, len(a) - 1)

print(izvadit\_masivu\_un\_salidzinasanas\_skaitu(u, sorted\_u))

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

A picture containing calendar

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)

Calendar

Description automatically generated

**4. uzdevums**

Sastādīt programmu, kas doto naturālo skaitli no 1 līdz 3899 arābu pierakstā pārveido par skaitļi romiešu pierakstā. Cipari romiešu pierakstā jāuzglabā masīvā.

**Kods:**

# Programmas nosaukums: Trīs masīvu apvienošana

# 4. uzdevums (1MPR07\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas doto naturālo skaitli no 1 līdz 3899 arābu pierakstā pārveido par skaitļi romiešu pierakstā.

# Cipari romiešu pierakstā jāuzglabā masīvā.

# Versija 1.0

import numpy

def to\_roman(n):

# Pārveido veselo skaitli no 1 līdz 3899 no arābu pieraksta uz romiešu pierakstu

# n - naturāls skaitlis no 1 līdz 3899

romas = numpy.array(["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", 'I'])

vertibas = numpy.array([1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1])

sv = ""

for i in range(13):

while vertibas[i] <= n:

sv = sv + romas[i]

n = n - vertibas[i]

return sv

def is\_natural\_and\_interval(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):

if (int(n) < 1 or int(n) > 3899):

return False

else:

return True

else:

return False

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

# Galvenā programmas daļa

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

n = input("Ievadiet naturālo skaitli no 1 līdz 3899 ==> ")

while is\_natural\_and\_interval(n) == False:

n = input("Kļūda! \nIevadiet naturālo skaitli no 1 līdz 3899 ==> ")

n = int(n)

print("\nArābu -> Romiešu")

print(str(n) + ": " + str(to\_roman(n)))

**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

**PU2. uzdevums**

Sastādīt programmu, kas lietotāja ievadīto reālo skaitli ( < 10 000) noapaļo līdz diviem cipariem aiz komata un rezultātu nodrukā šādā formātā:

# Ievade: 523.6789

# Izvade: Jums ir jāmaksā EUR 523.68 (pieci simti divdesmit trīs euro un 68 euro centi)

# Nepieciešami vārdi skaitļu pieraksta izveidei uzglabājami masīvā.

**Kods:**

# Programmas nosaukums: Veikals

# Papilduzdevums 2 (1MPR07\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas lietotāja ievadīto reālo skaitli ( < 10 000) noapaļo līdz diviem cipariem aiz komata un rezultātu nodrukā šādā formātā:

# Ievade: 523.6789

# Izvade: Jums ir jāmaksā EUR 523.68 (pieci simti divdesmit trīs euro un 68 euro centi)

# Nepieciešami vārdi skaitļu pieraksta izveidei uzglabājami masīvā.

# Versija 1.0

import numpy

def to\_vards(n):

# Pārveido reālo skaitli no 0 līdz 9999 tā, ka tas ir izrūnāms latviešu valodā

# n - reāls skaitlis no 0 līdz 9999

lst\_word = numpy.array(["desmit tūkstoši ", "deviņi tūkstoši ", "astoņi tūkstoši ", "septiņi tūkstoši ", "seši tūkstoši ", "pieci tūkstoši ", "četri tūkstoši ", "trīs tūkstoši ", "divi tūkstoši ", "tūkstošs ",

"deviņi simti ", "astoņi simti ", "septiņi simti ", "seši simti ", "pieci simti ", "četri simti ", "trīs simti ", "divi simti ", "simts ",

"deviņdesmit ", "astoņdesmit ", "septiņdesmit ", "sešdesmit ", "piecdesmit ", "četrdesmit ", "trīsdesmit ", "divdesmit ",

"deviņpadsmit ", "astoņpadsmit ", "septiņpadsmit ", "sešpadsmit ", "piecpadsmit ", "četrpadsmit ", "trīspadsmit ", "divpadsmit ", "vienpadsmit ", "desmit ",

"deviņi ", "astoņi ", "septiņi ", "seši ", "pieci ", "četri ", "trīs ", "divi ", "viens "])

lst\_numbers = numpy.array([10000, 9000, 8000, 7000, 6000, 5000, 4000, 3000, 2000, 1000,

900, 800, 700, 600, 500, 400, 300, 200, 100,

90, 80, 70, 60, 50, 40, 30, 20,

19, 18, 17, 16, 15, 14, 13, 12, 11, 10,

9, 8, 7, 6, 5, 4, 3, 2, 1])

sv = ""

for i in range(46):

while lst\_numbers[i] <= n:

sv = sv + lst\_word[i]

n = n - lst\_numbers[i]

if sv == "":

return "nulle "

else:

return sv

def check():

# Bezgalīgi daudz ievades. Pārbauda vai skaitlis ir reāls pozitīvs un atrodas intervālā [0, 9999]

x = input("Ievadiet reālo pozitīvo skaitli no 0 līdz 9999 ===> ")

while True:

try:

x = float(x)

except:

x = input("Kļūda!\nIevadiet reālo pozitīvo skaitli no 0 līdz 9999 ===> ")

else:

if x < 0 or x > 9999:

x = input("Kļūda! Skaitlis nepieder intervālam [0, 9999]\nIevadiet reālo pozitīvo skaitli no 0 līdz 9999 ===> ")

else:

return float(x)

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

# Galvenā programmas daļa

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

input\_number = check()

input\_number = round(input\_number, 2)

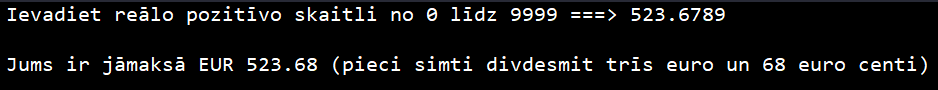
integer\_part = int(input\_number)

decimal\_part = int(round((input\_number - integer\_part) \* 100))

print("\nJums ir jāmaksā EUR " + str(input\_number) + " (" + str(to\_vards(integer\_part) + "euro un " + str(decimal\_part) + " euro centi)"))

**Testa piemēri:**

1)



2)

Text

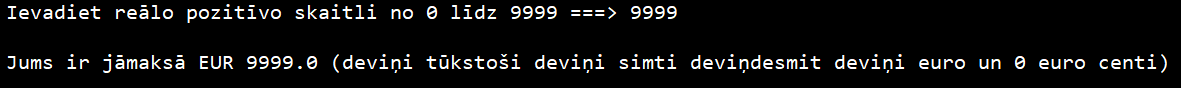
Description automatically generated

3)

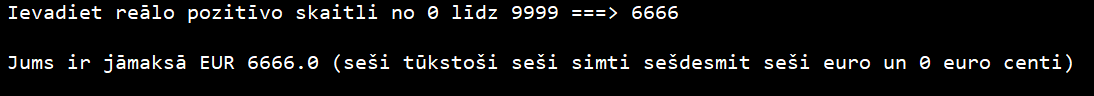
Graphical user interface, text

Description automatically generated

4)



5)



6)

Graphical user interface, text

Description automatically generated

7)

Text

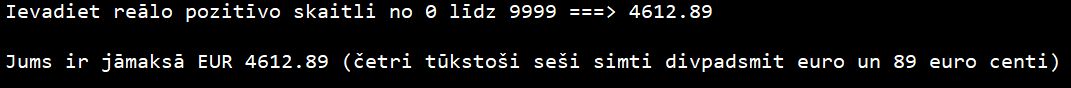
Description automatically generated

8)

Graphical user interface, text

Description automatically generated

9)

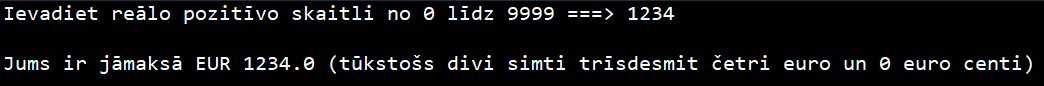


10)

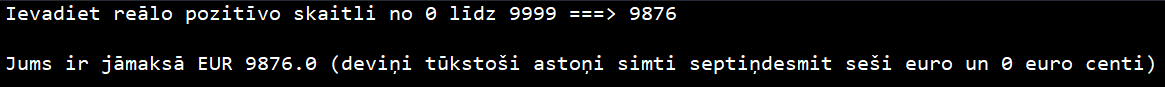
Text

Description automatically generated

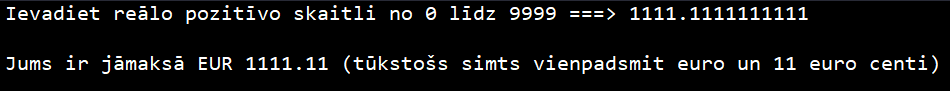
11)



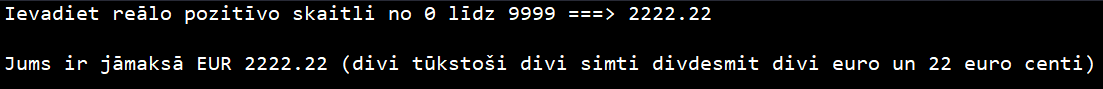
12)



13)



14)



15)

Text

Description automatically generated

**5. uzdevums**

Jums ir jāizveido viendimensijas masīvs ar 10 000 elementu. Šī masīva elementi ir uz labu laimi ģenerētas simbolu virknes (vārdi) garumā no 3 līdz 8 simboliem un sastāv tikai no lielajiem latīņu alfabēta burtiem. Pēc šī masīva izveides Jums jāveic visas nepieciešamās darbības, lai varētu ātri (ne sliktāk kā logaritmiskā laikā) pārbaudīt, vai šis masīvs satur vai nesatur lietotāja ievadīto frāzi.

**Kods:**

# Programmas nosaukums: Pārbauda vai masīvs satur frāzi

# 5. uzdevums (1MPR07\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Jums ir jāizveido viendimensijas masīvs ar 10 000 elementu.

# Šī masīva elementi ir uz labu laimi ģenerētas simbolu virknes (vārdi) garumā no 3 līdz 8 simboliem un sastāv tikai no

# lielajiem latīņu alfabēta burtiem. Pēc šī masīva izveides Jums jāveic visas nepieciešamās darbības, lai varētu ātri (ne sliktāk kā logaritmiskā laikā)

# pārbaudīt, vai šis masīvs satur vai nesatur lietotāja ievadīto frāzi.

# Versija 1.0

import numpy

import random

def atrais(a, sv, bv): # sv = 0, bv = len(a)

# Sakārto masīvu augoša secība

# Kārtošanas tiek izmantota Hoara (ātrais) metode (quicksort)

# a - viendimensijas masīvs

# sv - sākuma vērtība

# bv - beigu vērtība

if sv < bv:

i = sv

j = bv

solis = -1

lv = True

while i != j:

if lv == (a[i] > a[j]):

x = a[i]

a[i] = a[j]

a[j] = x

x = i

i = j

j = x

lv = not lv

solis = -solis

j = j + solis

atrais(a, sv, i - 1)

atrais(a, i + 1, bv)

def vards():

# Ģenerē vārdus ar garumu no 3 līdz 8 (garums - uz labu laimi) un ģenerē to vārdu ar lieliem latiņu alfabēta burtiem

# Atgriež vienu izveidotu vārdu

r = random.randint(3, 8)

v = ""

for i in range(r):

v += chr(random.randint(65, 90)) # ASCII 65;90

return v

def masivs(length):

# Aizpildā masīvu ar vārdiem atsaucoties uz "vards()". Atgriež aizpildīto masīvu.

# length - viendimensijas masīva garums

mas = numpy.empty(length, dtype=object)

for j in range(length):

mas[j] = vards()

# print(mas)

return mas

def meklet(a, b):

# Sameklē a masīva b skaitļi (vai vārdu). Atgriež to vērtību, kur viņa atrodas pēc index. Ja nav tādas vērtības masīva, tad atgriež -1.

# a - viendimensijas masīvs

# b - to ko mēs meklējam (skaitlis vai vārds (str))

l = 0

r = len(a) - 1

while (l <= r):

i = (l + r) // 2

if a[i] == b:

break

elif a[i] < b:

l = i + 1

else:

r = i - 1

if a[i] == b:

return i

else:

return -1

def izvade(x):

# Izvada masīva elementus pēc kārtas līdz pedējam

# x - viendimensijas masīvs

n = len(x)

s = str(x[0])

for i in range(1, n):

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

print(s)

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

# Galvenā programmas daļa

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

lenght = 10000

a = masivs(lenght)

atrais(a, 0, len(a) - 1)

print("Uz labu laimi izveidotie vārdi:")

izvade(a)

print("")

m = input("Ievadi meklējamo ===> ")

vieta = meklet(a, m)

if vieta == -1:

print("Meklējamais vārds " + str(m) + " nav atrasts.")

else:

print("Meklējamais ir " + str(vieta) + ". vietā.")

**Testa piemēri:**

1)

Text

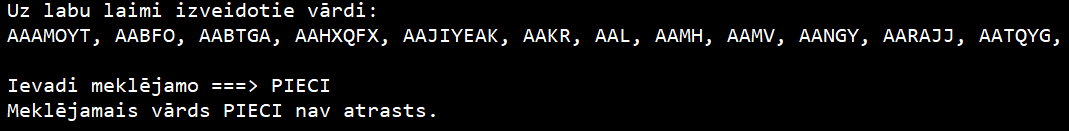
Description automatically generated

2)

Text

Description automatically generated

3)



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 with medium confidence