**1.uzd**

def words(x):

sv = ""

y = input("Ievadiet vārdu ==> ")

if y == "XXX":

sv = sv + x + " "

return sv

else:

sv = words(y)

sv = sv + x + " "

return sv

x = ""

print(words(x))

**2.uzd**

n = int(input("Skaitļu daudzums ==> "))

sv = ""

for i in range(n):

sk = int(input("Skaitlis ==> "))

sk = str(sk)

sv += sk

print(sv) # skaitlis ievaditais

endsv = ""

for i in range(n - 3, -1, -3): # ņem katru trešu no gala

# print(sv[i])

endsv += sv[i] + " "

print(endsv)

**3.uzd (непонятно, что брать)**

**Burbulis sakartošana (Bubblesort NAV uzlabotais)**

**Sakartot augoši**

Salidzinam

**Mainam**

[2 5 3 6 1 4 8 7]

[2 5 3 6 1 4 8 7]

[2 3 **5 6** 1 4 8 7] mainam

[2 3 5 6 1 4 8 7]

[2 3 5 6 1 4 8 7]

[2 3 5 **1 6** 4 8 7] mainam

[2 3 5 1 6 4 8 7]

[2 3 5 1 **4 6** 8 7] mainam

[2 3 5 1 4 6 8 7]

[2 3 5 1 4 6 8 7]

[2 3 5 1 4 6 **7 8**] mainam

[2 3 5 1 4 6 7 8]

[2 3 5 1 4 6 7 8]

[2 3 5 1 4 6 7 8]

[2 3 **1 5** 4 6 7 8] mainam

[2 3 1 5 4 6 7 8]

[2 3 1 **4 5** 6 7 8] mainam

[2 3 1 4 5 6 7 8]

[2 3 1 4 5 6 7 8]

[2 3 1 4 5 6 7 8]

[2 3 1 4 5 6 7 8]

[2 1 3 4 5 6 7 8]

[2 1 3 4 5 6 7 8]

[2 1 3 4 5 6 7 8]

[2 1 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

[1 2 3 4 5 6 7 8]

**Burbuļa kartošana uzlabotais**

Salīdzina

**Samaina**

[2 5 3 6 1 4 8 7]

[2 5 3 6 1 4 8 7] vajag samainit

[2 **3 5** 6 1 4 8 7] samainam

[2 3 5 6 1 4 8 7] vajag samainit

[2 3 5 **1 6** 4 8 7] samainam

[2 3 5 1 4 6 8 7]

[2 3 5 1 4 6 8 7] vajag samainit

[2 3 5 1 4 6 **7 8**] samainam (8 ir sakartots)

[2 3 5 1 4 6 7 8]

[2 3 5 1 4 6 7 8] vajag samainit

[2 3 **1 5** 4 6 7 8] samainam

[2 3 1 4 5 6 7 8]

[2 3 1 4 5 6 7 8] (7 ir sakartots)

[2 3 1 4 5 6 7 8]

[2 3 1 4 5 6 7 8] vajag samainit

[2 **1 3** 4 5 6 7 8] samainam

[2 1 3 4 5 6 7 8]

[2 1 3 4 5 6 7 8] (6 ir sakarots)

[2 1 3 4 5 6 7 8] vajag samainit

[**1 2** 3 4 5 6 7 8] samainam

[1 2 3 4 5 6 7 8] (1 ir sakartots)

[1 2 3 4 5 6 7 8] (5 ir sakartots)

[1 2 3 4 5 6 7 8] (2 ir sakartots)

[1 2 3 4 5 6 7 8] (3 ir sakartots)

[1 2 3 4 5 6 7 8] (4 ir sakartots)

**Masīvs ir sakārtots**

[1 2 3 4 5 6 7 8]

**BURBUĻA KARTOŠANAS METODE PĒC PUSEM (varbūt quicksort)**

**Как выбирает непонятно**

Salīdzina

**samaina**

2, 5, 3, 6, 1, 4, 8, 7

2, 5, 3, 6, 1, 4, 8, 7

2, **3, 5**, 6, 1, 4, 8, 7

2, 3, 5, 6, 1, 4, 8, 7

2, 3, 5, 6, 1, 4, 8, 7

2, 3, 5, **1,6** , 4, 8, 7

2, 3, 5, 1, 6, 4, 8, 7

2, 3, **1, 5**, 6, 4, 8, 7

2, 3, 1, 5, 6, 4, 8, 7

2, **1, 3**, 5, 6, 4, 8, 7

2, 1, 3, 5, 6, 4, 8, 7

**1, 2**, 3, 5, 6, 4, 8, 7 (pirma puse tika atsortēta)

1, 2, 3, 5, 6, 4, 8, 7

1, 2, 3, 5, **4, 6**, 8, 7

1, 2, 3, 5, 4, 6, 8, 7

1, 2, 3, **4, 5**, 6, 8, 7

1, 2, 3, 4, 5, 6**,** 8, 7

1, 2, 3, 4, 5, 6, **7, 8**

**4.uzd**

import numpy

def is\_real\_check(n):

# Pārbauda vai simbolu virkne ir reāls skaitlis vai nav.

# Atgriež True, ja tas ir reāls skaitlis (float).

# Atgriež False, ja tas nav reāls skaitlis (float).

# n - pārbaudāma simbolu virkne.

try:

float(n)

except:

return False

else:

return True

def ievade\_matrica\_float(n, m):

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

# Ievādītas vērtības ir reālas vērtības (matricas elementi varētu būt float)

# Glīti izvada atkarīgi no tas, cik ir nepieciešams starpes likt.

# n - naturāls skaitlis, kurš nosaka matricas rindas skaitu

# m - naturāls skaitlis, kurš nosaka matricas kolonnas skaitu

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

for i in range(n):

for j in range(m):

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

while is\_real\_check(temp) == False:

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

a[i, j] = float(temp)

return a

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 transponet(a):

n = a.shape[0]

m = a.shape[1]

b = numpy.empty((m, n), "O")

for i in range(m):

for j in range(n):

b[i, j] = a[j, i]

return b

n = int(input("Ievadiet N ==> "))

m = int(input("Ievadiet M ==> "))

a = ievade\_matrica\_float(n, m)

print("Ievadītā matrica:")

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

print("\n")

print("Transponētā matrica:")

t = transponet(a)

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

**5.uzd**

import random

kopa = set() # sākotnēji tukša izvikto lodīšu kopa

for i in range(6):

ran = random.randint(0, 9)

while ran in kopa: # ja lodīte jau ir kopā kopa, tad ģenerē jaunu lodītes numuru

ran = random.randint(0, 9)

kopa.add(ran)

print(str(i + 1) + ". izvilktās lodītes numurs:", ran)

**6.uzd**

Ieraksti (struktūras) ir datu struktūras, kas ļauj grupēt dažāda veida atribūtus vienā objektā.

Ierakstā nav noteikts elementu skaits, bet masīva ir jāzina, cik elementu tajā būs.

Ieraksts piemērs:

cilveks = types.SimpleNamespace()

cilveks.name = "John"

cilveks.age = 25

tas ir ieraksts (klase, kur ir tikai \_\_init\_\_ un atribūti).

masīvi ir datu struktūras, kas ļauj glabāt un operēt ar vairākiem elementiem, kas var būt vienāda vai atšķirīga datu tipa.

Ieraksti ir nepieciešāmi lai organizēt un grupēt daudzus atribūtus vienā objektā, bet masīvi ir piemēroti, ja nepieciešams glabāt un operēt ar vairākiem elementiem secīgi.

Ierakstā katram objektam ir vairākas definētas īpašības.

Ierakstus var ievietot masīvā, tad katram elementam būs ne tikai tā nosaukums (un kāda vertība), bet arī citas īpašības, kas tika noteiktas ierakstā.

Steks - var dabūt tikai pirmo un pedēju elementu. Pēdējais elements, kas tiek pievienots stekam, ir pirmais elements, kas būs izņemts no steka.

Steku analoģija ir grāmatu kaudze (nevar dabūt vidējo grāmatu neizņēmot visus kas bijušas virs tas grāmatas).

Rinda - datu struktūra, pirmāis elements, kas tiek pievienots rindai, ir pirmais elements, kas būs izņemts no rindas.

Rindas analoģija ir rindas veikalā reāla dzīve. Kas pirmais atnāca, tas pirmais aiziet arā no rindas.

Galvenā atšķirība starp steku, rindu un sarakstu ir to secība, ar kādu tiek pievienoti un izņemti elementi.

Stekā pēdējais elements, kas tiek pievienots, ir pirmais, kas būs izņemts. Nevar piekļūt "vidējam" elementam uzreiz, vajadzēs secīgi no pirmā elementa vai pedēja iet.

Rindā pirmais elements, kas tiek pievienots, ir pirmais, kas būs izņemts. Bet rindā var piekļūt jebkūram elementam (iesprausties).

Saraksts glabā elementus pēc kārtas (pēc indeksācijas) un piekļuve notiek pēc konkrēta indeksa.