1MPR01_Simona_Bļinova sb24037

Gandrīz visos uzdevumos lietoju moduli ar funkcijām. (programmu mapē *funkcijas.py*)

1.uzdevums

Programma, kas nodrukā uz ekrāna formatētu pēc piemēra Paskāla trijstūri ar lietotāja noteiktu rindu skaitu.

Kods:

Funkcijas no *funckijas.py*:

```
# Funkcija faktoriāla aprēķinam
def faktorials(a):
    faktoriala_vertiba = 1
    for i in range(1, a+1):
        faktoriala_vertiba *= i
    return faktoriala_vertiba

# Funkcijas kombinācijas vērtības aprēķinam
def kombinacija(n, m):
    if n == 0 and m == 0:
        kombinacijas_vertiba = 1
    else:
        n_faktorials = faktorials(n)
        m_faktorials = faktorials(m)
        n_m_starpiba = n - m
        n_m_faktorials = faktorials(n_m_starpiba)
        kombinacijas_vertiba = n_faktorials / (m_faktorials * n_m_faktorials)
    return kombinacijas_vertiba

# Funkcija pārbaudei, lai skaitlis būtu naturāls skaitlis
def naturals_skaitlis(a):
    # meģinājumu skaita skaitītājs (3 meģinājumi ierakstam)
    meginajumi = 1
    while meginajumi <= 3:
        try:
            a = int(a)
            if a > 0:
                return int(a)
        except:
            if meginajumi < 3:
                meginajumi += 1
                a = input('Ievadiet naturālo skaitli --> ')
            else:
                print('Programma beidz darbību!')
                exit()
```

1MPR01_1_Simona_Blinova.py:

```
import funkcijas

'''Ievades un ievades pārbaudes bloks'''

rindu_skaits = input('Ievadiet Paskāla trijstūra rindu skaitu --> ')
rindu_skaits = funkcijas.naturals_skaitlis(rindu_skaits)


'''Bloks atstarpju skaita aprēķinam'''

# Atstarpes starp skaitliem
lielaka_skaitla_kartas_numurs = rindu_skaits // 2
lielakais_skaitlis = int(funkcijas.kombinacija(rindu_skaits, lielaka_skaitla_kartas_numurs))
lielaka_skaitla_garums = len(str(lielakais_skaitlis))

# Atstarpes pirms pirma rindas elementa
pedeja_rinda = ''
for kartas_numurs in range(rindu_skaits): # kartas numuru skaits ir vienāds ar rindas kārtas
numuru
    kombinacijas_vertiba = int(funkcijas.kombinacija(rindu_skaits-1, kartas_numurs))
    pedeja_rinda = pedeja_rinda + str(kombinacijas_vertiba)
    if rindu_skaits-1 != kartas_numurs:
        pedeja_rinda = pedeja_rinda + ' ' * lielaka_skaitla_garums

pedejas_rindas_garums = len(pedeja_rinda)


'''Bloks Paskāla trijstūra izvadei'''

rinda = ''
for rindas_numurs in range(rindu_skaits-1):
    for kartas_numurs in range(rindas_numurs+1):
        kombinacijas_vertiba = int(funkcijas.kombinacija(rindas_numurs, kartas_numurs))
        rinda = rinda + str(kombinacijas_vertiba)
        if rindas_numurs != kartas_numurs:
            rinda = rinda + ' ' * lielaka_skaitla_garums
    rindas_garums = len(rinda)
    atstarpju_skaits = pedejas_rindas_garums - rindas_garums
    sakuma_atsarpju_skaits = atstarpju_skaits // 2
    rinda = ' ' * sakuma_atsarpju_skaits + rinda
    print(rinda)
    rinda = ''

print(pedeja_rinda)
```

Testa piemērs(1)

```
Ievadiet Paskāla trijstūra rindu skaitu --> 9
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10   10   5   1
        1   6   15   20   15   6   1
      1   7   21   35   35   21   7   1
    1   8   28   56   70   56   28   8   1
```

Testa piemērs(2)

```
Ievadiet Paskāla trijstūra rindu skaitu --> 3
  1
 1 1
1 2 1
```

Testa piemērs(3)

```
Ievadiet Paskāla trijstūra rindu skaitu --> 14
                                        1
                                      1   1
                                    1   2   1
                                  1   3   3   1
                                1   4   6   4   1
                              1   5   10   10   5   1
                            1   6   15   20   15   6   1
                          1   7   21   35   35   21   7   1
                        1   8   28   56   70   56   28   8   1
                      1   9   36   84   126   126   84   36   9   1
                    1   10   45   120   210   252   210   120   45   10   1
                  1   11   55   165   330   462   462   330   165   55   11   1
                1   12   66   220   495   792   924   792   495   220   66   12   1
              1   13   78   286   715   1287   1716   1716   1287   715   286   78   13   1
```

2.uzdevums

Programma, kas atrod vienādojuma $ax^3+by^2+cz+d=0$ visus veselos atrisinājumus intervālā [-10, 10], kur koeficientus a, b, c un d, kas arī veseli skaitli, nosaka lietotājs.

Kods:

Funkcijas no *funckijas.py*:

```
# Funkcija pārbaudei, lai skaitlis būtu vesels skaitlis
def vesels_skaitlis(a):
    # meģinajumu skaitītājs (3 meģinājumi pareizi ievadīt)
    meginajumi = 1
    while meginajumi <= 3:
        try:
            a = int(a)
            return int(a)
        except:
```

```
        if meginajumi < 3:
            meginajumi += 1
            a = input('Ievadiet veselo skaitli vēlreiz --> ')
        else:
            print('Programma beidz darbību!')
            exit()
```

1MPR01_2_Simona_Blinova.py:

```
import funkcijas

'''Koeficientu ievades un pārbaudes bloks'''

koeficients_a = input('Ievadiet veselo koeficientu a --> ')
koeficients_a = funkcijas.vesels_skaitlis(koeficients_a)

koeficients_b = input('Ievadiet veselo koeficientu b --> ')
koeficients_b = funkcijas.vesels_skaitlis(koeficients_b)

koeficients_c = input('Ievadiet veselo koeficientu c --> ')
koeficients_c = funkcijas.vesels_skaitlis(koeficients_c)

koeficients_d = input('Ievadiet veselo koeficientu d --> ')
koeficients_d = funkcijas.vesels_skaitlis(koeficients_d)


'''Cikls atrisinājumu mēklēšanai'''

for x in range(-10, 11):
    for y in range(11):
        cz = koeficients_a*x*x*x + koeficients_b*y*y + koeficients_d
        if koeficients_c != 0:
            z = int(round(cz/koeficients_c))
        else:
            z = 0
        if cz + z*koeficients_c == 0:
            print(f'({x}, {y}, {z})')
```

Testa piemērs(1)

```
Ievadiet veselo koeficientu a --> 3e
Ievadiet veselo skaitli vēlreiz --> 3
Ievadiet veselo koeficientu b --> 1
Ievadiet veselo koeficientu c --> 0
Ievadiet veselo koeficientu d --> -3
(1, 0, 0)
```

Testa piemērs(2)

```
Ievadiet veselo koeficientu a --> 1
Ievadiet veselo koeficientu b --> 1
Ievadiet veselo koeficientu c --> 0
Ievadiet veselo koeficientu d --> 0
(-4, 8, 0)
(-1, 1, 0)
(0, 0, 0)
```

Testa piemērs(3)

```
Ievadiet veselo koeficientu a --> 0
Ievadiet veselo koeficientu b --> 1
Ievadiet veselo koeficientu c --> 1
Ievadiet veselo koeficientu d --> 0
(-10, 0, 0)
(-9, 0, 0)
(-8, 0, 0)
(-7, 0, 0)
(-6, 0, 0)
(-5, 0, 0)
(-4, 0, 0)
(-3, 0, 0)
(-2, 0, 0)
(-1, 0, 0)
(0, 0, 0)
(1, 0, 0)
(2, 0, 0)
(3, 0, 0)
(4, 0, 0)
(5, 0, 0)
(6, 0, 0)
(7, 0, 0)
(8, 0, 0)
(9, 0, 0)
(10, 0, 0)
```

3.uzdevums

Programma, kas aprēķina arcsin(x) izteiksmes vērtību pēc noradīta piemēra ar precizitāti 10^{-6}, ko nosaka pēdējais saskaitāmais, ja x nosaka lietotājs un |x| < 1.

Kods:

1MPR01_3_Simona_Blinova.py:

```
'''Funkciju bloks'''

# Funkcija, lai pārbaudītu ievades korektību
def vertibas_intervals(a):
    # meģinājumu skaita skaitītājs (3 meģinājumi ierakstam)
    meginajumi = 1
    while meginajumi <= 3:
        try:
```

```
            a = float(a)
            if a < 1 and a > -1:
                return float(a)
            else:
                raise Exception
        except:
            if meginajumi < 3:
                meginajumi += 1
                a = input('Ievadiet x vērtību (|x| < 1) vēlreiz --> ')
            else:
                print('Programma beidz darbību!')
                exit()


'''Lietotāja ievades un datu pārabaudes bloks'''
x = input('Ievadiet x vērtību (|x| < 1) --> ')
x = vertibas_intervals(x)


'''Izteiksmes aprēķina un precizitātes pārbaudes bloks'''
precizitate = 1e-6
saskaitamais = x
summa = saskaitamais

# Katra sskaitama sākuma sastāvdaļas
x_dala_saucejs = 1
x_dala_skaititajs = x
koeficientu_dala_saucejs = 1
koeficientu_dala_skaititajs = 1
pakape = 3

while abs(saskaitamais) >= precizitate:
    # Cikls, kas noskaidro un piereizina skaitļus skaititājam vai saucējam, pirms daļas ar x vērtību
kādā pakāpē
    for skaitlis in range(1, pakape):
        if skaitlis % 2 == 0:
            koeficientu_dala_saucejs *= skaitlis
        else:
            koeficientu_dala_skaititajs *= skaitlis

        x_dala_saucejs = pakape
        x_dala_skaititajs = x_dala_skaititajs * x * x
        saskaitamais = (koeficientu_dala_skaititajs/koeficientu_dala_saucejs) *
(x_dala_skaititajs/x_dala_saucejs)
        summa += saskaitamais
        pakape += 2
```

```
print(f'arcsin({x})={summa}')
```

Testa piemērs(1)

```
Ievadiet x vērtību (|x| < 1) --> 0.923
arcsin(0.923)=1.3697465553529333
```

Testa piemērs(2)

```
Ievadiet x vērtību (|x| < 1) --> 0.123456
arcsin(0.123456)=0.12408611272571543
```

Testa piemērs(3)

```
Ievadiet x vērtību (|x| < 1) --> -1
Ievadiet x vērtību (|x| < 1) vēlreiz --> 0.456
arcsin(0.456)=0.4899075247063777
```

4.uzdevums

Programma, kas nodrukā visus laimīgos pasta zīmogus (DD.MM.GGGG) kopš Kristus dzimšanas.

Kods:

Funkcijas no *funckijas.py*:

```
# Funkcija nuļļu pievienošanai
def nulles(a, p):
    datuma_vertiba = str(a)
    vertibas_garums = len(datuma_vertiba)
    if p == 'g':
        atlikums = 4 - vertibas_garums
    elif p == 'm' or p == 'd':
        atlikums = 2 - vertibas_garums
    datuma_vertiba = '0' * atlikums + datuma_vertiba
    return datuma_vertiba
```

1MPR01_4_Simona_Blinova.py:

```
import funkcijas

'''Cikls datumu pārbaudei un izvadei'''

for gads in range(1, 10000):
    datums_gads = funkcijas.nulles(gads, 'g')
    # print(datums_gads)
    for menesis in range(1, 13):
        datums_menesis = funkcijas.nulles(menesis, 'm')
        # print(datums_menesis)

        # match case, kas nosaka dienu skaitu menesī
        match menesis:
```

```
            case 1 | 3 | 5 | 7 | 8 | 10| 12:
                dienas = 31
            case 4 | 6 | 9 | 11:
                dienas = 30
            case 2:
                if gads % 4000:
                    dienas = 28
                elif gads % 400:
                    dienas = 29
                elif gads % 100:
                    dienas = 28
                elif gads % 4:
                    dienas = 29
                else:
                    dienas = 28

        for diena in range(1, dienas+1):
            datums_diena = funkcijas.nulles(diena, 'd')

            # Tiek ierakstīta simbolu virkne ar apgrieztu meneša un dienas vērtību
            parbaudes_dala = datums_menesis[::-1] + datums_diena[::-1]
            # print(parbaudes_dala)

            if parbaudes_dala == datums_gads:
                print(f'{datums_diena}.{datums_menesis}.{datums_gads}')
```

Testa piemēri šeit ir izgrieztas daļas no vienas izvades.

Testa piemērs(1)

```
05.09.9050
15.09.9051
25.09.9052
06.09.9060
16.09.9061
26.09.9062
07.09.9070
17.09.9071
27.09.9072
08.09.9080
```

Testa piemērs(2)

```
10.10.0101
20.10.0102
30.10.0103
01.10.0110
11.10.0111
21.10.0112
31.10.0113
02.10.0120
12.10.0121
22.10.0122
03.10.0130
```

Testa piemērs(3)

```
11.04.4011
21.04.4012
02.04.4020
12.04.4021
22.04.4022
03.04.4030
13.04.4031
23.04.4032
04.04.4040
14.04.4041
```

5.uzdevums

Programma, kas atrod un izvada Mersena skaitļus, kas nav lielāki par $2*10^{28}$.

Kods:

Funkcijas no *funckijas.py*:

```
# Funkcija pārbaudei vai skaitlis ir pirmskaitlis
def vai_pirmskaitlis(a):
    if a < 2:
        return False

    for i in range(2, a//2):
        if a % i == 0:
            return False
    else:
        return True
```

1MPR01_5_Simona_Blinova.py:

```
import funkcijas

'''Skaitļu aprēķina un pārbaudes bloks'''

mazakais_pirmskaitlis = 2

while mazakais_pirmskaitlis < 2*pow(10, 28):
    if funkcijas.vai_pirmskaitlis(mazakais_pirmskaitlis) == True:
        iespejamais_mersena_skaitlis = 2**mazakais_pirmskaitlis - 1
        if funkcijas.vai_pirmskaitlis(iespejamais_mersena_skaitlis) == True:
            print(iespejamais_mersena_skaitlis)

    mazakais_pirmskaitlis += 1
```

Testa piemērs(1)

```
3
7
31
127
8191
131071
524287
```

Šeit testa piemērs ir tikai viens, jo lielākie skaitli tālāk tiek meklēti ļoti ilgi un visātrāk ir atrasti pirmie 7.