1. uzdevums

Sastādīt programmu, kas aprēķina cos(x) ar lietotāja norādīto precizitāti, ja pieņemam, ka un precizitāti nosaka pēdējais saskaitāmais. Skaitli X un precizitāti ievada lietotājs.

Kods:

```
# Programmas nosaukums: 1. uzd MPR12

# 1. uzdevums MPR12

# Uzdevuma formulējums: Sastādit programmu, kas aprēķina cos(x) ar lietotāja norādīto precizitāti, ja pieņemam, ka un precizitāti nosaka pēdējais saskaitāmais. Skaitli X un precizitāti ievada lietotājs.

# Versija 1.0


x = float(input("Ievadi funkcijas argumentu ===> "))

pr = float(input("Ievadi precizitāti ===> "))


z = 1

s = 1

n = 0

y = 1


while abs(y) > pr:

    z = -z

    n = n + 1

    y = y * x * x / 2 / n / (2 * n - 1)

    s = s + z * y


print("cos(" + str(x) + ") = " + str(s) + " ar precizitāti " + str(pr))
```

Testa piemēri:

1)

```
Ievadi funkcijas argumentu ===> 1.57
Ievadi precizitāti ===> 0.0001
cos(1.57) = 0.0007958647579494565 ar precizitāti 0.0001
```

2)

```
Ievadi funkcijas argumentu ===> 3.14
Ievadi precizitāti ===> 0.0001
cos(3.14) = -0.9999985976926036 ar precizitāti 0.0001
```

3)

```
Ievadi funkcijas argumentu ===> 6.28
Ievadi precizitāti ===> 0.000001
cos(6.28) = 0.9999949300620153 ar precizitāti 1e-06
```

2. uzdevums

Sastādīt programmu, kura zīmē "Fibonači eglīti". Iespējami garākā zara garumu ievada lietotājs.

Kods:

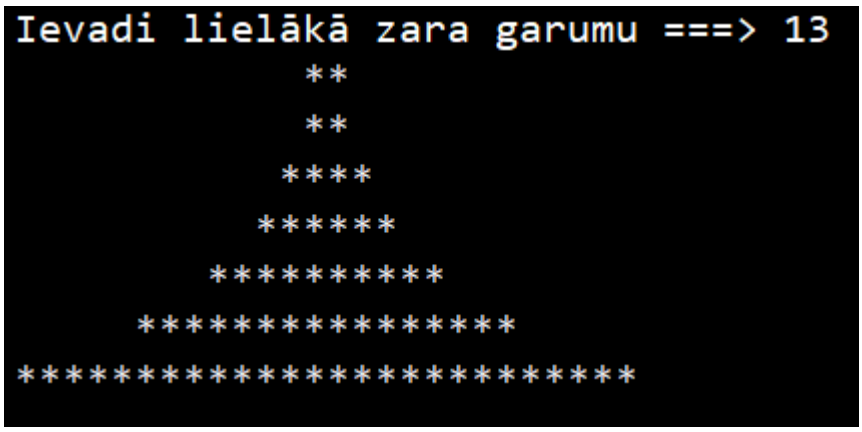
```
# Programmas nosaukums: 2. uzd MPR12

# 2. uzdevums MPR12

# Uzdevuma formulējums: Sastādit programmu, kura zīmē "Fibonači eglīti". Iespējami garākā zara garumu ievada lietotājs.

# Versija 1.0


d = int(input("Ievadi lielākā zara garumu ===> "))


a = 1

b = 1

c = a + b

garums = 1


# ------ Maksimālo garumu noteikšana


k=1
```

```
while k != 0:

    if c>d:
        garums=b*2 # *2, jo eglītei ir divas puses. Tāpēc, lai reķinātu tukšumus mums
        k=0 # cikla apstāšanai

    a = b
    b = c
    c = a + b # Fibonači virkne

# ------

#--------------Pirmais zars (augšējais - pirmā rinda)

tuksumi = " " * int((garums-len("**"))//2)    # Tukšumu skaitīšana. Dalām ar 2 jo tukšumi ir no divām pusēm. Nakāmajā rindā tas ir parādīts
egles_pirmais_zars = tuksumi + "**" + tuksumi
print(egles_pirmais_zars)

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

# ----- Parējo zaru noteikšana

a = 1
b = 1
c = 1 # mainīgie Fibonači virknei

egle = ""

while c <= d :
```

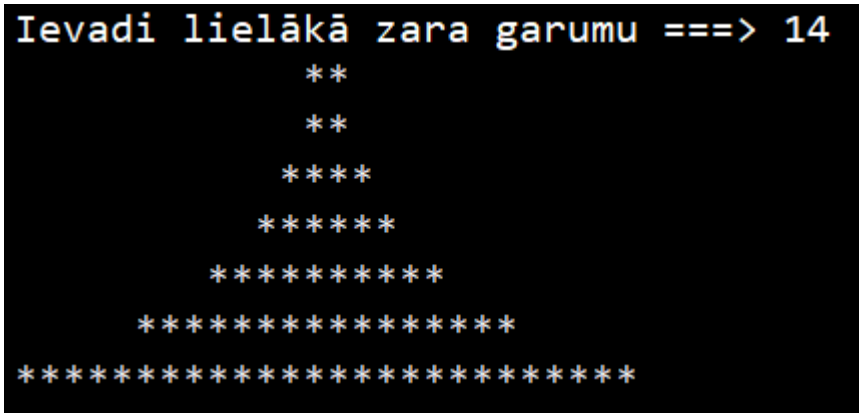
```
    zars = "**"*c                       # No kā ir veidots zars

    tuksumi = " " * int((garums-len(zars))//2)      # Tukšumu skaitīšana. Dalām ar 2 jo tukšumi ir no divām pusēm. Nakāmajā rindā tas ir parādīts

    egle = egle + tuksumi + zars + tuksumi + "\n" # _____****_____


    a = b

    b = c

    c = a + b


print(egle)
```

Testa piemēri:

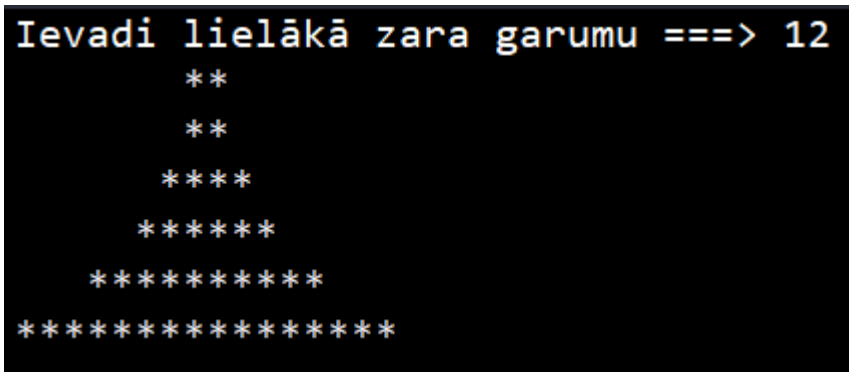
1)



2)



3)



4)

3. uzdevums

Sastādīt programmu, kura atrod skaitļa N pirmreizinātājus. Skaitli N ievada lietotājs + glīts noformējums.

Kods:

```
# Programmas nosaukums: 3. uzd MPR12

# 3. uzdevums MPR12

# Uzdevuma formulējums: Sastādīt programmu, kura atrod skaitļa N pirmreizinātājus. Skaitli N ievada lietotājs + glīts noformējums

# Versija 1.0


n = int(input("Ievadi naturālu skaitli ===> "))

a = n

j = 2


sv1 = str(n) + " | "


while a > 1 :

   k = 0
```

```
    while (a % j) == 0 :
        sv1 = str(a) + " | " + str(j)
        a = a // j
        k = k + 1
        print(sv1)

    j = j + 1

print("1 | 1")
```

Testa piemēri:

1)

```
Ievadi naturālu skaitli ===> 100
100 | 2
50 | 2
25 | 5
5 | 5
1 | 1
```

2)

```
Ievadi naturālu skaitli ===> 625
625 | 5
125 | 5
25 | 5
5 | 5
1 | 1
```

3)

```
Ievadi naturālu skaitli ===> 131
131 | 131
1 | 1
```

4. uzdevums

Sastādīt programmu, kura aprēķina laukumu zem funkcijas y=ax^3+bx^2+cx+d grafika intervālā [u,w] ar 1) taisnstūru metodi 2) trapeču metodi. a, b, c, d, u, w un precizitāti ievada lietotājs. Salīdzināt abas metodes pēc veikto iterāciju skaita pie vienas un tās pašas precizitātes.

Kods:

```
# Programmas nosaukums: 4. uzd MPR12

# 4. uzdevums MPR12

# Uzdevuma formulējums: Sastādīt programmu, kura aprēķina laukumu zem funkcijas y=ax^3+bx^2+cx+d grafija intervalā [u,w] ar 1) taisnstūru metodi 2) trapeču metodi. a, b, c, d, u, w un precizitāti ievada lietotājs. Salīdzināt abas metodes pēc veikto iterāciju skaita pie vienas un tās pašas precizitātes.

# Versija 1.0


print("Programma noteic laukumu zem funkcijas y = ax^3 + bx^2 + cx + d intervāla [u,w].\nJa laukums ir zem X ass, tad laukums ir negatīvs.\n")


q = float(input("Ievadi a ===> "))

w = float(input("Ievadi b ===> "))

e = float(input("Ievadi c ===> "))

r = float(input("Ievadi d ===> "))


a = float(input("Ievadi sākumpunktu u ===> "))

b = float(input("Ievadi galapunktu w ===> "))


pr = float(input("Ievadi precizitāti ===> "))


if a > b: # mainā vietām ja neprecīzi lietotājs ievadīja.
    k = a
    a = b
    b = k


#---------- ar taisnsturiem
```

```
p=0 # skaitītajs

s2 = 0
n=2
paz = True
while paz :
    p = p+1
    s1 = s2
    x = (b - a) / n
    s2 = 0
    for i in range (n) :

        s2 = s2 + (q*(a+i*x)*(a+i*x)*(a+i*x) + w*(a+i*x)*(a+i*x)+ e*(a+i*x) + r) * x

    n = n*2

    if abs(s1 - s2) < pr :
        paz = False

print("\nS ar taisnturiem = " + str(s2))
print("Iterāciju skaits: " + str(p) + "\n")



#------------------- ar trapecem

p=0 # skaitītajs

s2=0
n=2
paz = True
```

```
while paz :
    p = p+1
    s1 = s2
    x = (b - a) / n

    s2 = 0
    c = q*x*x*x + w*x*x+ e*x + r
    for i in range(n) :

        d = q*(a+i*x)*(a+i*x)*(a+i*x) + w*(a+i*x)*(a+i*x)+ e*(a+i*x) + r
        s2 = s2 + x * (c + d) / 2
        c = d

    n = n*2

    if abs(s1 - s2) < pr :
        paz = False

print("S ar trapecem = " + str(s2))
print("Iterāciju skaits: " + str(p)) # Iterāciju skaits
```

Testa piemēri:

1)

```
Programma noteic laukumu zem funkcijas y = ax^3 + bx^2 + cx + d intervāla [u,w].
Ja laukums ir zem X ass, tad laukums ir negatīvs.

Ievadi a ===> 1
Ievadi b ===> 0
Ievadi c ===> 0
Ievadi d ===> 0
Ievadi sākumpunktu u ===> 0
Ievadi galapunktu w ===> 2
Ievadi precizitāti ===> 0.001

S ar taisnturiem = 3.9990234971046448
Iterāciju skaits: 13

S ar trapecem = 3.999023541802672
Iterāciju skaits: 14
```

2)

```
Programma noteic laukumu zem funkcijas y = ax^3 + bx^2 + cx + d intervāla [u,w].
Ja laukums ir zem X ass, tad laukums ir negatīvs.

Ievadi a ===> 1
Ievadi b ===> 0
Ievadi c ===> -4
Ievadi d ===> 0
Ievadi sākumpunktu u ===> -2
Ievadi galapunktu w ===> 2
Ievadi precizitāti ===> 0.001

S ar taisnturiem = 0.0
Iterāciju skaits: 1

S ar trapecem = 0.0
Iterāciju skaits: 1
```

3)

```
Programma noteic laukumu zem funkcijas y = ax^3 + bx^2 + cx + d intervāla [u,w].
Ja laukums ir zem X ass, tad laukums ir negatīvs.

Ievadi a ===> 1
Ievadi b ===> 0
Ievadi c ===> -4
Ievadi d ===> 4
Ievadi sākumpunktu u ===> -2
Ievadi galapunktu w ===> 2
Ievadi precizitāti ===> 0.0001

S ar taisnturiem = 16.0
Iterāciju skaits: 2

S ar trapecem = 16.0
Iterāciju skaits: 2
```

Pie vienas un tās pašas precizitātes iterāciju skaits ir vai nu vienāds, vai nu praktiski vienāds, vai trapeču metodē iterāciju skaits ir mazāks nekā ar taisnstūru metodi.

5. uzdevums

Sastādīt programmu, kas aprēķina laukumu, kuru ierobežo divas parabolas. Parabolu koeficientus un precizitāti ievada lietotājs

Kods:

```
# Programmas nosaukums: 5. uzd MPR12

# 5. uzdevums MPR12

# Uzdevuma formulējums: Sastādīt programmu, kas aprēķina laukumu, kuru ierobežo divas parabolas. Parabolu koeficientus un precizitāti ievada lietotājs.

# Versija 1.0


import math


print("Programma noteic laukumu kuru ierobežo divas parabolas:\ny = ax^2 + bx + c \ny = dx^2 + ex + f\n")


a1 = float(input("Ievadi a ===> "))

b1 = float(input("Ievadi b ===> "))

c1 = float(input("Ievadi c ===> "))


a2 = float(input("Ievadi d ===> "))

b2 = float(input("Ievadi e ===> "))

c2 = float(input("Ievadi f ===> "))


pr =  float(input("Ievadi precizitāti ===> "))


a = a1 - a2

b = b1 - b2

c = c1 - c2


if a == 0 :
```

```
    print("Laukums ir 0")
    quit()
else :
    d = b * b - 4 * a * c # Diskriminants
    if d < 0:
        print("Laukums ir 0") # Kvadratvienādojumam realu saknu nav jo divam parabolam nav krustpunktu, tāpēc laukums neveidojas.
        quit()
    elif d == 0 :
        x12 = -b / 2 / a
        print("Laukums ir 0") # jo divas parabolas krustpunkts ir tikai viens, tāpēc laukums neveidojas.
        quit()
    else :
        x1 = (-b + math.sqrt(d)) / (2 * a) # Parabolu krustpunkti
        x2 = (-b - math.sqrt(d)) / 2 / a

# ---------  laukums zem pirmās funkcijas

a = x1
b = x2


s1 = 0
n = 2
x = (b - a) / n
s2 = 0

for i in range (n) :
    y = a+i*x
    s2 = s2 + (a1*y*y + b1*y + c1)*x
n = n * 2
```

```
while abs(s1 - s2) > pr :
    s1 = s2
    x = (b - a) / n
    s2 = 0
    for i in range (n) :
        y = a+i*x
        s2 = s2 + (a1*y*y + b1*y + c1)*x
    n = n * 2

S1 = s2

# --------- laukums zem otras funkcijas

a = x1
b = x2



s1 = 0
n = 2
x = (b - a) / n
s2 = 0

for i in range (n) :
    y = a+i*x
    s2 = s2 + (a2*y*y + b2*y + c2)*x
n = n * 2

while abs(s1 - s2) > pr :
    s1 = s2
    x = (b - a) / n
```

```
    s2 = 0
    for i in range (n) :
        y = a+i*x
        s2 = s2 + (a2*y*y + b2*y + c2)*x
    n = n * 2


S2 = s2


S_kop = abs(S1 - S2)


print("\nLaukums, kuru ierobežo divas parabolas, ir vienāds ar:\n" + str(S_kop))
print("Rezultāts ir iegūts ar precizitāti:\n" + str(pr))
```

Testa piemēri:

1)

```
Programma noteic laukumu kuru ierobežo divas parabolas:
y = ax^2 + bx + c
y = dx^2 + ex + f

Ievadi a ===> 1
Ievadi b ===> 2
Ievadi c ===> 0
Ievadi d ===> -1
Ievadi e ===> 0
Ievadi f ===> 0
Ievadi precizitāti ===> 0.001

Laukums, kuru ierobežo divas parabolas, ir vienāds ar:
0.3333320617675781
Rezultāts ir iegūts ar precizitāti:
0.001
```

2)

```
Programma noteic laukumu kuru ierobežo divas parabolas:
y = ax^2 + bx + c
y = dx^2 + ex + f

Ievadi a ===> 1
Ievadi b ===> 4
Ievadi c ===> 0
Ievadi d ===> 4
Ievadi e ===> 1
Ievadi f ===> 0
Ievadi precizitāti ===> 0.0001

Laukums, kuru ierobežo divas parabolas, ir vienāds ar:
0.4999999995343387
Rezultāts ir iegūts ar precizitāti:
0.0001
```

3)

```
Programma noteic laukumu kuru ierobežo divas parabolas:
y = ax^2 + bx + c
y = dx^2 + ex + f

Ievadi a ===> 4
Ievadi b ===> 4
Ievadi c ===> 0
Ievadi d ===> 4
Ievadi e ===> 4
Ievadi f ===> 0
Ievadi precizitāti ===> 0.1
Laukums ir 0
```