



Rešene naloge

Python urejevalnik se nahaja na strani:

<https://trinket.io/python3>

Kazalo

1	Uvodne naloge	5
1.1	Standardni vhod in izhod	5
1.2	Tipi spremenljivk	5
1.3	Niz iz standardnega vhoda	5
1.4	Pretvorba v celo število	5
1.5	Dolžina niza	5
1.6	Veriženje nizov	6
1.7	Obratni vrstni red	6
1.8	Vsota	6
1.9	Največje število	6
1.10	Stranice trikotnika	6
1.11	Tretja stranica trikotnika	7
1.12	Pitagorov izrek	7
1.13	Ura	7
1.14	Collatzevo zaporedje	8
1.15	Členi Collatzevega zaporedja	8
1.16	Faktoriel	8
2	Seznami, množice in slovarji	9
2.1	Seznam celih števil	9
2.2	Vsota števil v seznamu	9
2.3	Produkt števil v seznamu	9
2.4	Vsota kvadratov sodih števil	9
2.5	Filtriranje seznama	10
2.6	Izpeljani sezname s pogojem	10
2.7	Izpeljani sezname s pogojem	10
2.8	Indeks velikih tiskanih črk v nizu	10
2.9	Pari prvič	10
2.10	Pari drugič	11
2.11	Več nivojska iteracija	11
2.12	Množica	11

2.13	Unija in presek	12
2.14	Kumulativna vsota	12
2.15	Operator <i>mroža</i> :=	12
2.16	Cikel v seznamu	13
2.17	Pogostost črk v besedilu	14
2.18	Pari sosednjih črk v besedilu	14
3	Šifriranje besedil	15
3.1	Cesarjeva šifra	15
3.2	Koda TAP	16
3.3	Koda DIF	17
4	Naravna števila	18
4.1	Prafaktorji	18
4.2	Eratostenovo rešeto	19
4.3	Evklidov algoritem	19
5	Rekurzija	21
5.1	Gnezden seznam	21
5.2	Sploščitev gnezdenega seznama	21
5.3	Hanojski stolpi	21
5.4	Le tri in pet	23
5.5	Fibonaccijevo zaporedje	23
6	Zaporedja	24
6.1	Collatzevo zaporedje	24
6.2	Benfordov zakon	24
6.3	Fareyevo zaporedje	25
6.4	Fordove krožnice	26
6.5	Paskalov trikotnik	27
6.6	Zaporedje EKG	28
6.7	Kaprekarjevo zaporedje	29
7	Urejanja	30
7.1	Urejanje	30
7.2	Mehurčkasto urejanje	30
7.3	Urejanje z zlivanjem	31
8	Kombinatorika	33
8.1	Razporedbe ali permutacije	33
8.2	Cikli v razporedbi	33
8.3	100 zapornikov	34
8.4	Kovanci	35
9	Številski sestavi	36
9.1	Drugi številski sestavi	36
9.2	Številski sistem Majev	37
9.3	Uravnotežen mestni trojiški sestav	38

9.4	Rimski številski sestav	40
9.5	Fibonaccijev številski sestav	41
10	Verižni ulomki in zlati rez	43
10.1	Verižni ulomki	43
10.2	Razmerje zlatega reza	44
10.3	Sončnica	45
10.4	Virahanka	46
11	Fraktali	48
11.1	Fibonaccijeve generacije	48
11.2	Fibonaccijev fraktal	48
11.3	Kochova krivulja	49
12	Objekti	50
12.1	Peanovi aksiomi	50
12.2	Rešitev enačbe zlatega reza	52
12.3	Permutacijska grupa	53
13	Okraski ali dekoratorji	56
13.1	Funkcija deljenja	56
13.2	Timer	57
14	Generatorji in iteratorji	59
14.1	Funkcija	59
14.2	Sploščeni seznam	59
15	Igre	60
15.1	Monty Hall	60
15.2	An ban pet podgan	61
15.3	Umri pokončno	62
15.4	Rojstni dnevi učencev	64
15.5	Skrito število	65
16	Numerične metode	67
16.1	Bisekcija	67
16.2	Bisekcija v zlatem rezu	69
16.3	Trisekcija	70
16.4	Sekantna metoda	71
17	Razne naloge	72
17.1	Sredinska črka	72
17.2	Naredimo palindrom	72
17.3	Učenci	72
17.4	Slovar ocen učencev	73
17.5	Samoglasniki	74
17.6	Dešifriranje besedila	74
17.7	Najdi kroglico	75

17.8	Ena črka več	76
17.9	Bankovci	76
17.10	Sedem segmentni prikazovalnik	77
18	Naloge na vajah	78
18.1	Le tri in pet	78
18.2	Hanojski stolpi	78
18.3	Maya	79
18.4	Rimske številke	79
18.5	Koda TAP	80
18.6	Koda DIF	80
18.7	Dekodiranje besedila	80
18.8	Naravna števila	80
18.9	Bisekcija	81
18.10	Najdi kroglico	81
18.11	Ena črka več	82

1 Uvodne naloge

Vhodne podatke preberete iz standardnega vhoda z ukazom `input`. Rezultate izpišete na standardni izhod z ukazom `print`.

1.1 Standardni vhod in izhod

Preberite niz iz standardnega vhoda in ga izpišite na standardni izhod.

```
a = input("Vpiši besedilo: ")
print(a)
```

1.2 Tipi spremenljivk

Spremenljivke so različnih tipov. Celó število `int`, "decimalno število" `float`, niz `str`,...

```
a, b, c, d = 12, 3/4, 3.14, 'število 1'
print(type(a), type(b), type(c), type(d))
```

1.3 Niz iz standardnega vhoda

Kakšnega tipa je spremenljivka v katero preberemo vrstico iz standardnega vhoda?

```
a = input("Vpisi besedilo: ")
print(type(a))
```

1.4 Pretvorba v celo število

Celemu številu, ki ga preberete iz standardnega vhoda prištejte 1 in rezultat izpišite na standardni izhod.

```
a = int(input("Vpiši število: "))
a += 1      # a = a + 1
print(a)
```

1.5 Dolžina niza

Preberite niz iz standardnega vhoda in izpišite njegovo dolžino, število znakov na standardni izhod.

```
s = input("Vpiši besedo: ")
print(len(s))
```

1.6 Veriženje nizov

Dva niza se združita nov niz.

```
a, b = 'abc', 'cdef'
c = a + b
print(c)
```

1.7 Obratni vrstni red

Iz standardnega vhoda preberite niz in izpišite na standardni izhod niz z znaki v obratnem vrstnem redu.

```
s = input("Vpiši besedo: ")
ns = '' # prazen niz
for ch in s:
    ns = ch + ns
print(ns)
```

1.8 Vsota

Iz standardnega vhoda preberite dve števili ločeni s presledkom in na standardni izhod izpišite njuno vsoto.

```
a, b = input("Števili a in b ločeni s presledkom -> ").split()
print(int(a) + int(b)) # split() razcepi niz po presledku
```

1.9 Največje število

Iz standardnega vhoda preberite tri števila ločena s presledkom. Na standardni izhod izpišite največje med njimi.

```
a, b, c = input("Števila a, b, c ločena s presledkom -> ").split()
d, b, c = int(a), int(b), int(c)
if b > d:
    d = b
if c > d:
    d = c
print(d)
```

1.10 Stranice trikotnika

Iz standardnega vhoda preberite tri naravna števila a , b in c , ki predstavljajo dolžine daljic. Na standardni izhod izpišite **True**, če daljice lahko tvorijo trikotnik in **False**, če to ni mogoče.

```
a, b, c = input('Vpiši a, b in c').split() # split() razcepi niz po presledkih
a, b, c = int(a), int(b), int(c)
print(a + b > c & b + c > a & c + a > b)
```

1.11 Tretja stranica trikotnika

Iz standardnega vhoda preberite dve naravni števili a in b , ki predstavljata dolžini dveh stranic trikotnika in na standardni izhod izpišite največjo možno celoštevilčno dolžino tretje stranice trikotnika.

```
a, b = input('Vpiši dolžini a in b ').split()
c = int(a) + int(b) - 1
print(c)
```

1.12 Pitagorov izrek

Iz standardnega vhoda preberite tri naravna števila a, b in c , v naraščajočem vrstnem redu, ki predstavljajo dolžine daljic trikotnika. Na standardni izhod izpišite **True**, če je trikotnik pravokotni in **False** v nasprotnem primeru.

```
a, b, c = input('Vpiši a, b in c --> ').split()
print(int(a)**2 + int(b)**2 == int(c)**2)
```

```
Vpiši a, b in c --> 3 4 5
True
```

1.13 Ura

Mali kazalec na uri kaže točno na številko $1 \leq n \leq 12$. Na katero številko bo kazal kazalec čez $m > 0$ ur, oziroma na katero številko je kazal kazalec pred $|m|$ urami, če $m < 0$. Vrednost $|m|$ je absolutna vrednost celega števila m .

```
n, m = input('m -> ').split()
nn = (int(n) + int(m)) % 12
if nn == 0:
    nn == 12
print(nn)
```

1.14 Collatzevo zaporedje

Iz standardnega vhoda preberite naravno število n in na standardni izpišite vrednost:

1. $n/2$, če je število n sodo in
2. $3n + 1$, če je število liho.

collatz.py

```
def collatz(x):
    if x % 2 == 0:    # ali je število sodo?
        return x // 2
    else:            # sicer je število liho
        return 3 * x + 1
```

1.15 Členi Collatzevega zaporedja

$$x_{i+1} = \text{collatz}(x_i), \quad i \geq 0$$

Iz standardnega vhoda preberite število x_0 , na standardni izhod izpišite zaporedno število člena $x_n = 1$.

```
from collatz import collatz
x = int(input('Število x -> '))
i = 0
while x != 1:    # dokler x ne postane 1
    i += 1      # števec členov i = i + 1
    x = collatz(x)
print(i)
```

```
Število x -> 27
111
```

1.16 Faktoriel

Iz standardnega vhoda preberite naravno število in $n > 0$ izpišite na standardni izhod produkt

$$n! = 1 \times 2 \times 3 \times \dots \times n$$

```
n = int(input('Vpiši n -> '))
res = 1
for i in range(1, n + 1)
    res *= i    # res = res * i
print(res)
```


2 Seznami, množice in slovarji

2.1 Seznam celih števil

Preberite vrstico z zaporedjem celih števil, ločenih s presledki in jo spremenite v seznam celih števil. Nato izpišite seznam na standardni izhod.

```
vrstica = input('Vpisi števila: ')
stevila = vrstica.split()           # naredi seznam
stevila = [int(x) for x in stevila] # elemente pretvori v cela števila
print(stevila)
```

2.2 Vsota števil v seznamu

Iz standardnega vhoda preberi vrstico s celimi števili ločenimi z vejico in na standardni izhod izpiši njihovo vsoto.

```
#!/usr/bin/env python3
stevila = input("Vpisi števila ločena z vejico -> ")
stevila = [int(x) for x in stevila.split(",")]
print(sum(stevila))
```

2.3 Produkt števil v seznamu

Iz standardnega vhoda preberi vrstico s celimi števili ločenimi presledkom in na standardni izhod izpiši njihov produkt.

```
#!/usr/bin/env python3
stevila = input("Vpisi števila ločena s presledkom -> ")
stevila = [int(x) for x in stevila.split()]
prod = 1
for x in stevila:
    prod *= x
print(prod)
```

2.4 Vsota kvadratov sodih števil

```
#!/usr/bin/env python3
n = int(input("Vpisi število -> "))
kvadrati = range(2, n, 2)
suma = sum([x**2 for x in kvadrati])
print(suma)
```

2.5 Filtriranje seznama

Dan je seznam elementov, ki so različnih tipov. Izberi iz seznama cela števila.

```
L = [1, 2, 4, 3/2, "python", 7]
print([x for x in L if type(x) is int])
```

2.6 Izpeljani seznama s pogojem

Ustvari seznam števil med $1 \leq n < 20$, ki niso deljiva s 3.

```
#!/usr/bin/env python3
L = [i for i in range(1,20) if i % 3 > 0]
print(L)
```

2.7 Izpeljani seznama s pogojem

Preberite vrstico z zaporedjem celih števil, ločenih s presledki in jo spremenite v seznam celih števil. Nato izpišite seznam števil, ki so večja od zadnjega števila v seznamu.

```
vrstica = input("Vpisi števila: ") # števila so ločena s presledkom
stevila = vrstica.split() # naredi seznam in elemente
stevila = [int(x) for x in stevila] # spremeni v cela števila
stevila = [x for x in stevila if x > stevila[-1]]
print(stevila)
```

2.8 Indeks velikih tiskanih črk v nizu

Napišite funkcijo z imenom `capital_indexes`, ki sprejema niz črk in vrne seznam indeksov, kjer se v nizu nahajajo velike tiskane črke.

```
def capital_indexes(s: str) -> list:
    ss = dict(enumerate(s)) # slovar indeksov in črk
    return [x for x in ss if ss[x].isupper()]
print(cpital_indexes('PericaRežeRaciRep'))
```

```
[0, 6, 10, 14]
```

2.9 Pari prvič

Iz standardnega vhoda preberi seznam števil in na standardni izhod izpiši seznam parov sosednjih elementov.

```

stevila = eval(input("Seznam --> ")) # niz "[a1,a2,...,an]"
# funkcija eval evalira vhodni niz
dvojice = [stevila[i:i+2] for i in range(len(stevila)-1)]
print(dvojice)

```

```

Seznam --> [1, 2, 3, 4, 5, 6, 7]
[[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7]]

```

2.10 Pari drugič

Iz standardnega vhoda preberi seznam števil in na standardni izhod izpiši seznam parov sosednjih elementov. Elementi se v parih ne smejo ponavljati.

```

stevila = eval(input("Vpiši seznam -> "))
dvojice = [stevila[i:i+2] for i in range(0,len(stevila),2)]
print(dvojice)

```

```

Seznam --> [1, 2, 3, 4, 5, 6, 7]
[[1, 2], [3, 4], [5, 6], [7]]

```

2.11 Več nivojska iteracija

```

L = [(i, j) for i in range(2) for j in range(3)]
print(L)

```

```

[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]

```

2.12 Množica

Preberite seznam in izločite vse ponavljajoče elemente.

```

seznam = [2, 3, 5, 2, 1, 3, 4, 8, 7, 9, 2, 4]
mnozica = set(seznam) # v množici se elementi ne ponavljajo
print(mnozica)

```

```

{1, 2, 3, 4, 5, 7, 8, 9}

```

2.13 Unija in presek

Za dani dve množici določite unijo, presek in njuno razliko.

```
m1 = {2, 5, 7, 9, 12, 6, 1}
m2 = {3, 6, 8, 1, 10, 11, 5}
print(m1 | m2) # unija, združeni množici
print(m1 & m2) # presek, elementi skupni obema množicama
print(m1 - m2) # razlika množic
```

```
{1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12}
{1, 5, 6}
{9, 2, 12, 7}
```

2.14 Kumulativna vsota

Napišite funkcijo z imenom `cumsum`, ki sprejema seznam celih števil in vrne seznam kumulativnih vsot elementov.

```
l = [3, 4, 3, 4, 2]
cl = cumsum(l) -->
cl = [3, 3 + 4, 3 + 4 + 3, 3 + 4 + 3 + 4, 3 + 4 + 3 + 4 + 2]
cl = [3, 7, 10, 14, 16]
```

```
def cumsum(l):
    res = [l[0]]
    for i in range(1, len(l)):
        res.append(res[-1] + l[i])
    return res
print(cumsum([1, 3, 2, 7, 4, 5, 6, 3]))
```

```
[1, 4, 6, 13, 17, 22, 28, 31]
```

2.15 Operator *mroža* :=

Kumulativna vsota

```
nums = [1, 3, 2, 7, 4, 5, 6, 3] # začetni seznam nums
v = 0 # uvede spremenljivo v in jo postavi na 0
cumsum = [v := v + n for n in nums] # sproti prireja spremenljivki v -> v + n
print(f'cumsum: {cumsum}') # in tvori seznam kumulativnih vsot iz nums
```

```
cumsum: [1, 4, 6, 13, 17, 22, 28, 31]
```

Kumulativni produkt

```
nums = [1, 3, 2, 7, 4, 5, 6, 3]    # začetni seznam nums
v = 1
cumprod = [v := v * n for n in nums]
print(f'cumprod: {cumprod}')
```

```
cumprod: [1, 3, 6, 42, 168, 840, 5040, 15120]
```

2.16 Cikel v seznamu

Podan je neurejen seznam l n števil k , $0 \leq k \leq n$. Izberite prvi element $l[0]$ in gredite na mesto v seznamu na katerega kaže vrednost. Nadaljujete postopek, dokler se ne vrnete na začetek. Izpišite seznam vrednosti, ki ste jih obiskali na poti.

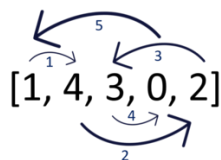
```
l = [3, 2, 0, 1, 5, 4]
0 -> 3 -> 1 -> 2 -> 0 cikel se je zaključil
rezultat je [3, 1, 2, 0]
```

Cikel

```
def one_cycle(l):
    dic = dict(enumerate(l))
    print(dic)
    a = dic[0]
    s = []
    while a not in s:
        s.append(a)
        a = dic[a]
    return s

print(one_cycle([1, 3, 2, 7, 4, 5, 6, 0]))
```

```
{0: 1, 1: 3, 2: 2, 3: 7, 4: 4, 5: 5, 6: 6, 7: 0}
[1, 3, 7, 0]
```



Slika 1: Cikel v seznamu $[1, 4, 3, 0, 2]$, $(0 \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 0)$

2.17 Pogostost črk v besedilu

Za dano besedilo v slovenskem jeziku, določi za vsako črko, kolikokrat se pojavi v besedilu. Predpostavimo, da je besedilo zapisano z malimi tiskanimi črkami brez ločil, le s presledki.

Črke

```
abeceda = 'abcčdefghijklmnoprsštuvzž '
besedilo = '''edinost sreča sprava k nam naj nazaj
se vrnejo otrok kar ima slava vsi naj si v roke sežejo
de oblast in z njo čast ko pred spet naša boste last'''
besedilo = ' '.join(besedilo.split('\n'))
freq = dict(zip(abeceda, [0] * len(abeceda)))
for ch in besedilo:
    freq[ch] += 1
print(freq)
```

```
{'a': 17, 'b': 2, 'c': 0, 'č': 2, 'd': 3, 'e': 11, 'f': 0,
'g': 0, 'h': 0, 'i': 5, 'j': 6, 'k': 5, 'l': 3, 'm': 2,
'n': 9, 'o': 10, 'p': 3, 'r': 7, 's': 13, 'š': 1, 't': 7,
'u': 0, 'v': 5, 'z': 2, 'ž': 1, ' ': 30}
```

2.18 Pari sosednjih črk v besedilu

Koliko je različnih parov sosednjih črk v zgornjem besedilu, če ne štejemo pare s presledkom.

Pari

```
besedilo = '''edinost sreča sprava k nam naj nazaj
se vrnejo otrok kar ima slava vsi naj si v roke sežejo
de oblast in z njo čast ko pred spet naša boste last'''
besedilo = ' '.join(besedilo.split('\n'))
pairs = [besedilo[i:i+2] for i in range(0, len(besedilo)-1)\
         if ' ' not in besedilo[i:i+2]]
print(len(set(pairs)))
```

53

3 Šifriranje besedil

3.1 Cezarjeva šifra

Julij Cezar (100–44 pr. n. št.) je kodiral sporočila, ki so si jih izmenjavale vojaške enote, za primer, če bi se sporočilo znašlo v nasprotnikovih rokah. Kodiranje si je domislil Marcus Tullius Tiro. Kodiranje je temeljilo na zamenjavi črk abecede. Zamenjava je bila kar se da preprosta. Uporabil je ciklično zamenjavo črk abecede za dano število mest. Število mest zamika je ključ za kodiranje in dekodiranje sporočil. Vzemimo za primer male črke slovenske abecede:

a,b,c,č,d,e,f,g,h,i,j,k,l,m,n,o,p,r,s,š,t,u,v,z,ž, .

Zadnji znak je znak za presledek. Vseh znakov je 26. Če izberemo ključ $k = 7$, potem v besedilu zamenjamo črke iz zgornje vrstice z istoležnimi črkami v spodnji vrstici.

š t u v z ž _ a b c č d e f g h i j k l m n o p r s
a b c č d e f g h i j k l m n o p r s š t u v z ž _

```
abeceda = 'abcčdefghijklmnoprsštuvzž '  
abeceda_idx = list(enumerate(abeceda))  
idx_abc = dict(abeceda_idx)  
abc_idx = dict([x[::-1] for x in abeceda_idx])  
len_abc = len(abeceda)  
  
def cesar_code(abc: str, key: int) -> str:  
    res = ''  
    for ch in abc:  
        i = abc_idx[ch]  
        i = (i + key) % len_abc  
        res += idx_abc[i]  
    return res
```

```
besedilo = '''narveč sveta otrokam sliši slave  
tje bomo najdli pot kjer nje sinovi  
si prosti voljo vero in postave'''  
  
if __name__ == '__main__':  
    besedilo = ' '.join(besedilo.split('\n'))  
    code = cesar_code(besedilo, 7)  
    print(code)  
    print(cesar_code(code, -7))
```

ugžčljf člbgfvbžvsgtf špapf šgčlfbrlfhvtv
fugrkšpfzvbfsrlžfurlf puvčpf pfzžv bpfčvšrvfčlžvfpufzv bgčl

narveč sveta otrokam sliši slave tje bomo
najdli pot kjer nje sinovi si prosti voljo vero in postave

3.3 Koda DIF

Koda **DIF** kodira besedilo tako, da dano črko nadomesti s črko, katere mesto v abecedi je enako razliki med mesti v abecedi dane črke in njej predhodne črke v besedilu. Razliko računamo po modulu dolžine abecede.

Koda DIF

```
abeceda = 'ABCČDEFGHIJKLMNOPRSŠTUVZŽ '
def diff_code(text: str) -> str:
    res = text[0]
    for i in range(1, len(text)):
        res += abeceda[(abeceda.index(text[i]) - abeceda.index(text[i-1])) %
            ↪ len(abeceda)]
    return res

def diff_decode(text: str) -> str:
    res = text[0]
    for ch in text[1:]:
        res += abeceda[(abeceda.index(res[-1]) + abeceda.index(ch)) % len(abeceda)]
    return res

if __name__ == '__main__':
    code = diff_code('AEIOU MAME NI DOMA OČE JE DOMA PA KUHATI NE ZNA')
    print(code)
    decode = diff_decode(code)
    print(decode)
```

```
AEDFFDNMMSTOUPEKŽM PNCTKUTEKŽM RJ LJMSTOPORTŽRL
AEIOU MAME NI DOMA OČE JE DOMA PA KUHATI NE ZNA
```

4 Naravna števila

4.1 Prafaktorji

Praštevilo je naravno število, ki ga ne moremo zapisati v obliki produkta dveh števil, ki sta različni od 1. Program bo naredil seznam prafaktorjev, katerih produkt je dano število:

$$n = p_0^{e_0} \cdot p_1^{e_1} \cdot \dots \cdot p_k^{e_k}.$$

Prafaktorji prvič

```
def prime_factors_(n):
    res = [] # pripravi seznam prafaktorjev
    while n > 1:
        for i in range(2, n + 1): # če število <= n
            while n % i == 0: # deli dano število brez ostanka
                n //= i # potem n -> n / i in
                res.append(i) # število i se shrani kot praštevilo
    return res
```

Prafaktorji drugič

```
from math import sqrt

def factors(n):
    res = []
    mult = 0
    # obdela prafaktorje 2
    while not n & 1: # ali je n deljivo z 2
        mult += 1 # če je poveča kratnost
        n >>= 1 # deli n / 2
    if mult != 0: # če je kratnost > 0
        res.append((2, mult)) # vpiše 2 in kratnost v seznam
    limit = round(sqrt(n)) # zgornja meja faktorji so <= sqrt(n)
    test_prime = 3 # začne s 3
    while test_prime <= limit: # dokler ne doseže mejo
        mult = 0
        while n % test_prime == 0:
            mult += 1 # če se deljenje izide
            n //= test_prime # poveča kratnost
        if mult != 0: # če je na koncu kratnost > 0
            res.append((test_prime, mult)) # vpiše v seznam
            if n == 1: # ko postane n == 1
                break # se konča izvajanje zanke
            limit = round(sqrt(n)) # se popravi meja za nov n
            test_prime += 2 # in poveča testno praštevilo
    if n != 1: # če n != 1
        res.append((n, 1)) # ga doda kot zadnjega v seznam
    return res # na koncu vrne seznam (p, k)
```

Slednji primer se najprej znebi vseh potenc števila 2 in upošteva zgornjo mejo prafaktorjev p števila n ($p \leq \sqrt{n}$).

4.2 Eratostenovo rešeto

Iz standardnega vhoda se prebere število n in na standardni izhod se izpiše *praštevila*, ki so manjša ali enaka n .

- Najprej se ustvari seznam celih števil od 2 do n
- in prazen seznam kamor se bodo shranjevala praštevila.
- V zanki se vnese v seznam vsakokrat prvo število iz prvotnega seznama števil,
- nato se v seznamu obdrži le tista števila, pri katerih je ostanek pri deljenju (%) s prvim številom v seznamu različen od nič.
- Ko se izprazni prvotni seznam števil, ostanejo v drugem seznamu praštevila.

Eratostenovo rešeto

```
n = int(input('Vpiši število n -> '))
prastevila = []
stevila = list(range(2, n+1))
while stevila:
    prastevila.append(stevila[0])
    stevila = [x for x in stevila if x % stevila[0] > 0]
print(prastevila)
```

4.3 Evklidov algoritem

Evklidov algoritem poišče največji skupni delitelj dveh števil.

- Označimo števili z a in b . Kot samo ime pove je to največje število, ki ju deli brez ostanka.
- Preverimo, če sta $a > b > 0$. Če ne, ju zamenjamo.
- Vstopimo v zanko, dokler ni $b = 0$.
- Izračunamo ostanek pri deljenju a/b .
- Shranimo b v spremenljivko a in
- ostanek pri deljenju v spremenljivko b
- Nadaljujemo v zanki.

Evklidov algoritem

```
def evklid(a, b):
    a, b = int(a), int(b)
    assert a > 0 and b > 0, print('pozitivni celi stevili')
    if a < b: a, b = b, a
    while b > 0: a, b = b, a % b
    return a
```

```
if __name__ == "__main__":  
    a, b = [int(x) for x in input('Vpisi stevili a in b --> ').split()]  
    print(evklid(a, b))
```

5 Rekurzija

5.1 Gnezden seznam

Izračunaj vsoto števil v gnezdenem seznamu.

`l = [2, 3, [4, 5,[1,2,3]], [3, 5], 1] -> ssum(l) = 29`

```
#!/usr/bin/env python3
def ssum(l):
    res = 0
    for s in l:
        if type(s) == list:
            res += ssum(s)
        else:
            res += s
    return res

if __name__ == '__main__':
    l = eval(input('l = '))
    print(ssum(l))
```

5.2 Sploščitev gnezdenega seznama

Sploščili bomo gnezden seznam, tako da bomo ustvarili seznam elementov gnezdenega seznama in obdržali vrstni red.

```
#!/usr/bin/env python3
def flatten(l):
    res = []
    for s in l:
        if type(s) == list:
            res += flatten(s)
        else:
            res += [s]
    return res

if __name__ == '__main__':
    l = eval(input('l = '))
    print(flatten(l))
```

5.3 Hanojski stolpi

Na tri palice, imenujmo jih *A*, *B* in *C*, se natikajo okrogle ploščice različnih premerov, ki imajo na sredini luknjo.

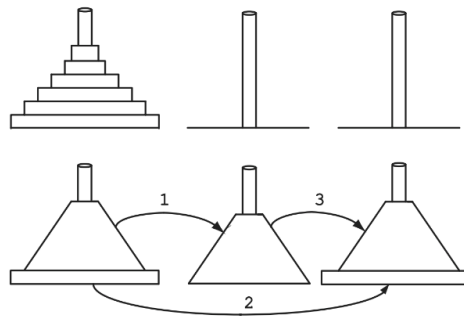
Na začetku so ploščice nataknjene na palici *A*. Urejene od tal do vrha v vrstnem redu od največje do najmanjše, tako ima kup obliko stožca. Cilj igre je prestaviti ploščice na palico *C*, pri tem si pomagamo z

začasnim odlaganjem ploščic na palico *B*. Uporabiti moramo najmanjše možno število potez, z upoštevanjem naslednjih pravil:

1. Naenkrat lahko premaknemo le eno ploščico.
2. Na dano ploščico lahko prestavimo le ploščico, ki ima manjši premer.

Strategija:

Iz palice *A* prestavimo, s pomočjo rekurzivnega klica procedure, vse ploščice razen spodnje na palico *B*, pri tem pa začasno odlagamo ploščice na palico *C*. Spodnjo ploščico prestavimo iz palice *A* na palico *C*, nato pa z rekurzivnim klicem prestavimo ploščice iz palice *B* na palico *C*, pri tem uporabimo palico *A* za začasno odlaganje.



```
#!/usr/bin/env python3
def TowerOfHanoi(n, start, cilj, umik):
    if n == 1:
        print("Prestavi disk 1 iz", start, "na", cilj)
        return
    TowerOfHanoi(n - 1, start, umik, cilj)
    print("Prestavi disk", n, "iz", start, "na", cilj)
    TowerOfHanoi(n - 1, umik, cilj, start)

if __name__ == '__main__':
    n = int(input('Število diskov: --> '))
    TowerOfHanoi(n, 'A', 'C', 'B')
```

<https://www.youtube.com/watch?v=BalWjeY209g>

```
Stevilo diskov: --> 3
Prestavi disk 1 iz A na C
Prestavi disk 2 iz A na B
Prestavi disk 1 iz C na B
Prestavi disk 3 iz A na C
Prestavi disk 1 iz B na A
Prestavi disk 2 iz B na C
Prestavi disk 1 iz A na C
```

5.4 Le tri in pet

Začeni s 3 ali 5 in z uporabo operacij:

1. prištejemo 5 oziroma
2. množimo s 3,

poskušamo doseči ciljno vrednost n .

Primeri:

```
only_5_and_3(14) -> True
# 14 = 3*3 + 5
only_5_and_3(25) -> True
# 25 = 5+5+5+5+5
only_5_and_3(7) -> False
# Števila 7 ni mogoče doseči
```

```
def can_reach_target(start, target): # Je mogoče doseči?
    if start == target:             # Če je cilj enak startu ,
        return True                 # je problem rešen.
    elif start > target:             # Če start presega cilj,
        return False                # ni mogoče doseči cilja.
    else:                             # Sicer pa rekurzivni klic
        return can_reach_target(start + 5, target) or
               can_reach_target(start * 3, target)

def only_5_and_3(target):            # Poskuša z 3 ali s 5
    return can_reach_target(3, target) or
           can_reach_target(5, target)
```

5.5 Fibonaccijevo zaporedje

Definicija: $F_1 = F_2 = 1$, $F_n = F_{n-1} + F_{n-2}$, $n \geq 3$.

Funkcija `fibonacci` se ima dva rekurzivna klica, zato se izračunane vrednosti ponavljajo. Ponavljanju se izognemo, če se izračunane vrednosti hranimo v spremenljivki `cache`.

Fibonacci

```
cache = {1: 1, 2: 1}

def fibonacci(n):
    if n in cache:                   # Pogleda, če je vrednost že bila izračunana
        return cache[n]             # Če je, jo prebere iz slovarja cache
    else:
        f = fibonacci(n - 1) + fibonacci(n - 2)
        cache[n] = f                 # Če ni jo izračuna z rekurzivnim klicem funkcije
        return f                     # in vrne izračunano vrednost,
                                     # hkrati pa jo shrani v cache.
```

6 Zaporedja

6.1 Collatzevo zaporedje

je zaporedje naravnih števil, ki ga tvorimo na naslednji način:

- Izberimo si začetno naravno število $k_0 > 0$.
- Naslednika člena k_n izračunamo takole: $k_{n+1} = \begin{cases} 3k_n + 1, & \text{če je } k_n \text{ liho število,} \\ k_n/2, & \text{če je } k_n \text{ sodo število.} \end{cases}$

Kolikor vemo, se je do sedaj v vseh primerih zaporedje končalo s ciklom (4, 2, 1). Collatzeva domneva je, da se to zgodi vedno, ne glede na začetno število. V matematiki je to nerešena domneva.

Glej <https://www.youtube.com/watch?v=094y1Z2wpJg>.

Prvi jo je leta 1937 postavil Lothar Collatz.

Iz standardnega vhoda preberite naravno število $n > 0$, tvorite Collatzevo zaporedje in ga končajte, ko postane izračunani člen prvič enak 1. Na standardni izhod izpišite število izračunanih členov in največjo vrednost člena v zaporedju.

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt

def collatz(n):
    m = n % 2
    if m: return 3 * n + 1
    else: return n // 2

def list_collatz(n):
    res = [n]
    while True:
        n = collatz(n)
        res.append(n)
        if n == 1: break
    return res

if __name__ == '__main__':
    n = int(input('n --> '))
    x = list_collatz(n)
    print(len(x), max(x))
    plt.plot(x)
    plt.show()
```

6.2 Benfordov zakon

Zakon je leta 1881 prvič omenil ameriški astronom Simon Newcomb, ko je ugotovil, da frekvence vodilnih števk, pri naključnem izboru vrednosti iz tabele s fizikalnimi konstantami, niso porazdeljene enakomerno. Največkrat, to je približno v 1/3 primerih, nastopi številka 1, na drugem mestu je številka 2 in tako naprej do številke 9, ki se pojavlja v manj kot 1/20 primerov.

Določil je teoretično porazdelitev

$$f(d) = \frac{1}{\log 10} \log \left(1 + \frac{1}{d} \right).$$

Glej <https://www.youtube.com/watch?v=XXjLR20K1kM>

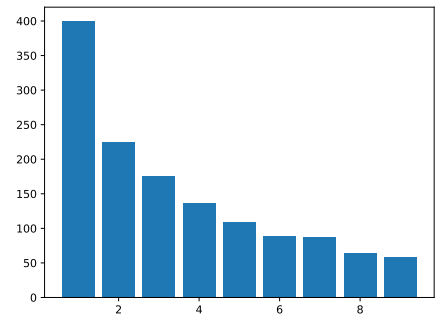
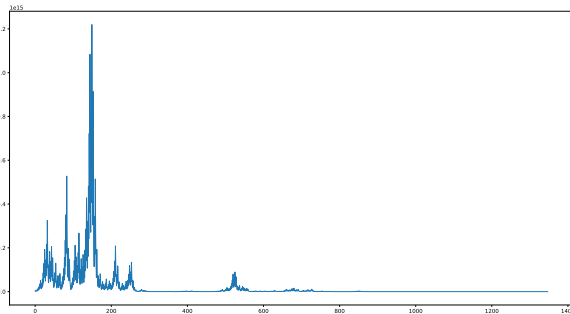
```
#!/usr/bin/env python3
from collatz import *

def first_digit(n):
    assert type(n) == int, print('no go')
    return int(str(n)[0])

def benford_collatz(n):
    benfords = 10 * [0]
    for m in collatz_list(n):
        benfords[first_digit(m)] += 1
    return benfords

if __name__ == '__main__':
    a = benford_collatz(989345275647)
    print(a)
```

```
{'1':400, '2':225, '3':176, '4':137, '5':109, '6':89, '7':88, '8':65, '9':59}
```



6.3 Fareyvo zaporedje

je imenovano po britanskem geologu Johnu Fareyju. Fareyjevo zaporedje reda n , označeno z $F(n)$, je sestavljeno iz vseh okrajšanih (števec in imenoalec sta tuji števili) ulomkov med 0 in 1 z imenovalci, manjšimi ali enakimi n .

Mediant dveh ulomkov a/b in c/d je ulomek $(a+c)/(c+d)$. Fareyjevo zaporedje reda n dobimo z naslednjimi koraki:

- Začnemo z ulomkoma $0/1$ in $1/1$, ta sta vedno vključena v Fareyjevo zaporedje.

- Iz $F(n - 1)$ dobimo $F(n)$, če med dva člena v zaporedju $F(n - 1)$ vrinemo njun *mediant* v primeru, če je imenovalc manjši ali enak n

Koda za izračun Fareyevih zaporedij.

```
def mediant(a, b):
    return (a[0] + b[0], a[1] + b[1])

def quotient(a):
    return a[0] / a[1]

def farey_step(q):
    q0 = q[:-1]
    q1 = q[1:]
    return sorted(q + [mediant(a, b) for a, b in zip(q0, q1)], key=quotient)

def farey_n(n):
    q = [(0, 1), (1, 1)]
    for i in range(n):
        q = [x for x in farey_step(q) if x[1] <= n]
    return q
```

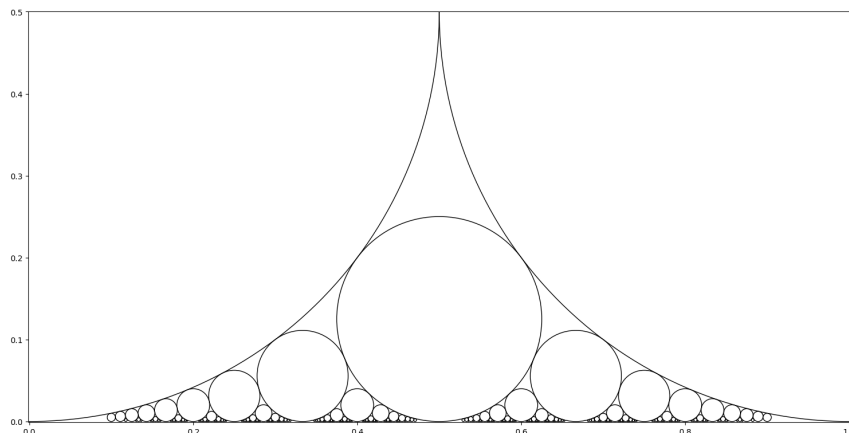
```
if __name__ == '__main__':
    print(farey_n(5))
```

Program izračuna primer Fareyvega zaporedja reda 5 ($F(5)$):

0/1, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 1/1

Zaporedje vsebuje vse okrajšane ulomke med 0 in 1 z imenovalci, manjšimi ali enakimi 5.

6.4 Fordove krožnice



```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
from farey import farey_n

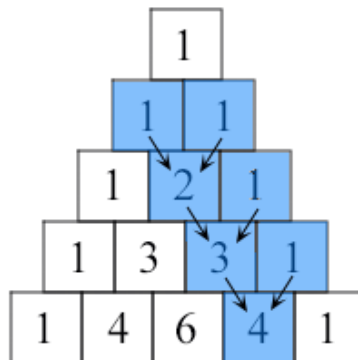
def plot_circles(n):
    fig, ax = plt.subplots()
    ax.set(xlim=(-0.001, 1.001), ylim=(-0.001, 1 / 2),
           autoscale_on=False, clip_on=True, aspect='equal')

    def circle_(x, y, r, fill, color):
        return plt.Circle((x, y), r, color=color, clip_on=True, fill=fill)
    seq = farey_n(n)
    for ss in seq:
        cc = circle_(ss[0] / ss[1], 1 / 2 / ss[1] ** 2, 1 / 2 / ss[1] ** 2,
                    False, 'black')
        ax.add_patch(cc)
    plt.show()
```

```
if __name__ == '__main__':
    print(plot_circles(9))
```

6.5 Paskalov trikotnik

Program naj izpiše prvih n vrstic Paskalovega trikotnika.



```
#!/usr/bin/env python3

def pascal(n):
    l=[1]
    print(l)
    for _ in range(n):
        l = [i + j for i, j in zip([0] + l, l + [0])]
        print(l)
```

6.6 Zaporedje EKG

Zaporedje se začne z [1, 2]. Število, ki sledi je najmanjše število s sklada, ki ima skupni faktor različen od 1 s predhodnim členom. Če takega števila na skladu ni, nadaljujemo zaporedjem naravnih števil, če naslednje število nima skupnega faktorja s predhodnim ga dodamo na sklad in nadaljujemo dokler ne naletimo na pravo število, ki ga dodamo zaporedju.

```
zaporedje = [1, 2]
sklad = []
3 = nima skupnega faktoja z 2
4 = ima skupni faktor 2 s številom 2
zaporedje = [1, 2, 4]
sklad = [3]
3 = nima skupnega faktorja s 4
5 = nima skupnega faktorja s 4
6 = ima skupni faktor 2 s 4
zaporedje = [1, 2, 4, 6]
sklad [3, 5]
3 = ima skupni faktor 3 s 6
zaporedje = [1, 2, 4, 6, 3]
sklad [5]
```

Število 3 je na 4. mestu v EKG zaporedju. Napišite funkcijo `ekg_indeks(n)`, ki za dan indeks n poišče vrednost `zaporedje(n)`.

Primeri:

```
ekg_indeks(4) --> 3
ekg_indeks(9) --> 5
ekg_indeks(13) --> 7
```

```
from math import gcd

def ekg_index(n):
    i, ekg, stack = 3, [1, 2], []
    while len(ekg) < n + 1:
        for x in stack:
            if gcd(ekg[-1], x) > 1:
                ekg.append(x)
                stack.pop(stack.index(x))
                break
        else:
            if gcd(ekg[-1], i) > 1:
                ekg.append(i)
            else:
                stack.append(i)
            i += 1
    return ekg
```

```
if __name__ == '__main__':
    print(ekg_index(30))
```

```
[ 1, 2, 4, 6, 3, 9, 12, 8, 10, 5, 15, 18,
 14, 7, 21, 24, 16, 20, 22, 11, 33, 27, 30,
 25, 35, 28, 26, 13, 39, 36, 32]
```

6.7 Kaprekarjevo zaporedje

Členi Kaprekarjevega zaporedja.

1. Izberemo poljubno število. To število je prvi člen zaporedja.
2. Njegove številke v desetiškem zapisu uredimo po velikosti od največje do najmanjše.
3. Temu številu odštejemo število, ki se zapiše z istimi števkami v obratnem vrstnem redu.
4. Razlika je naslednji člen zaporedja.
5. Ponavljamo, dokler se ne pojavi število, ki se še nahaja v zaporedju.

```
#!/usr/bin/env python3

def kaprekar(x):
    n, mi, ma = len(x), ''.join(sorted(x)), ''.join(sorted(x, reverse=True))
    x = str(int(ma) - int(mi)) # x je razlika
    x = (n - len(x)) * '0' + x # doda vodilne ničle, da dolžina ostane n
    return x # vrne naslednji člen zaporedja

def fix_pt(x): # zaporedmo računa člene zaporedja
    res = [] # hrani člene v seznamu res
    while x not in res: # računa nove člene, dokler ne neleti
        res.append(x) # na člen, ki se že nahaja v seznamu res
        x = kaprekar(x) # spravi novi izračunani člen v seznam
    return res # vrne seznam
```

```
if __name__ == '__main__':
    x = input('nn -> ')
    print(fix_pt(x))
```

```
nn -> 3452
['3452', '3087', '8352', '6174']
nn -> 4128
['4128', '7173', '6354', '3087', '8352', '6174']
nn -> 23432
['23432', '20988', '95931', '85932', '74943', '62964', '71973', '83952']
```

7 Urejanja

7.1 Urejanje

Kaj je metoda in kaj je funkcija.
Funkcija `sorted` in metoda `sort`.

```
#!/usr/bin/env python3
a = evaluate(input('a --> '))
print(sorted(a))
print(a)
print(a.sort())
print(a)
a.append('end')
print(a)
```

7.2 Mehurčkasto urejanje

Eden od najpreprostejših algoritmov za urejanje seznamov je naslednji:

- Naredimo prehod skozi seznam, pri tem
- primerjamo dva sosednja elementa seznama,
- če nista urejena v predpisanem vrstnem redu, ju zamenjamo,
- če sta enaka, enega izločimo.
- To ponavljamo toliko časa, dokler v prehodu skozi seznam ni bila narejena nobena sprememba.

```
import random

def bubble_sort(lst: list) -> list:
    change = True
    while change:
        change = False
        i = 0
        while i < len(lst) - 1:
            if lst[i] > lst[i + 1]:
                lst[i:i+2] = reversed(lst[i:i+2])
                change = True
                i += 1
            elif lst[i] == lst[i + 1]:
                del lst[i]
                change = True
            else:
                i += 1
    return(lst)
```

```

if __name__ == '__main__':
    lst = list(range(10))
    random.shuffle(lst)
    print(bubble_sort(lst))

```

7.3 Urejanje z zlivanjem

Algoritem urejanje z zlivanjem štejemo med algoritme tipa deli in vladaj. Seznam, ki ga ureja razdeli na manjše podseznama in jih uredi na koncu pa jih ponovno združi v celoten urejen seznam. Glavni funkciji v kodi sta `merge`, (zlivanje), ki združi dva podseznama, in `mergeSort`, ki rekurzivno razdeli seznam na pol dokler ne dobimo podseznama z enim samim elementom, ki jih nato združi v urejen seznam.

```

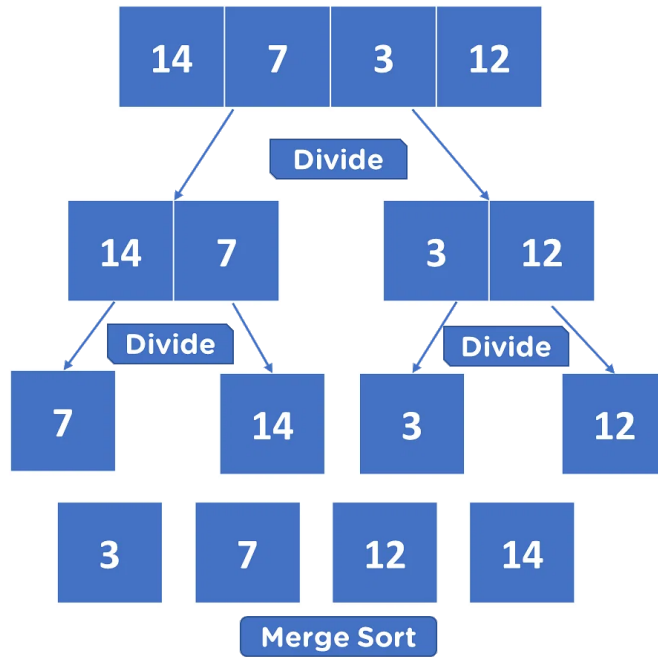
def merge_sort(unsorted_list):
    # Razdeljevanje
    if len(unsorted_list) <= 1:
        return unsorted_list
    middle = len(unsorted_list) // 2
    # Sredinski element
    # Dva seznama
    left_list = unsorted_list[:middle]
    right_list = unsorted_list[middle:]

    left_list = merge_sort(left_list)
    # Rekurzivni klic
    right_list = merge_sort(right_list)
    return list(merge(left_list, right_list)) # Klic zlivanja

def merge(left_half, right_half):
    # Zlivanje urejenih seznamov
    res = []
    while len(left_half) != 0 and len(right_half) != 0:
        if left_half[0] < right_half[0]:
            # Primerjanje prvih elementov
            res.append(left_half[0])
            # Vpis v končni seznam
            left_half.remove(left_half[0])
            # po vrstnem redu
        else:
            res.append(right_half[0])
            right_half.remove(right_half[0])
    if len(left_half) == 0:
        # Primer, če je eden od seznamov
        # že izčrpan, drugega dodamo
        res = res + right_half
    else:
        # h končnemu urejenemu seznamu.
        res = res + left_half
    return res

if __name__ == '__main__':
    unsorted_list = [64, 34, 25, 12, 22, 11, 90]
    print(merge_sort(unsorted_list))

```



8 Kombinatorika

8.1 Razporedbe ali permutacije

Vzemimo za primer niz 'abcd'. Elementi niza so različni. Vprašamo se na koliko načinov lahko razporedimo elemente tega niza. Vse razporedbe niza so:

```
'abcd', 'abdc', 'acbd', 'acdb', 'adbc', 'adcb',  
'bacd', 'badc', 'bcad', 'bcda', 'bdac', 'bdca',  
'cabd', 'cadb', 'cbad', 'cbda', 'cdab', 'cdba',  
'dabc', 'dacb', 'dbac', 'dbca', 'dcab', 'dcba'
```

Naša naloga bo napisat program, ki sprejme niz in izpiše vse razporedbe znakov v nizu. Če pogledamo natančneje je prvi element v vsaki vrstici po vrsti 'a', 'b', 'c', 'd', sledijo ostali elementi urejeni v abecednim vrstnem redu. Če odstranimo prvi element prvem stolpcu dobimo spet urejene permutacije ostalih treh elementov. Slutimo, da bomo problem lahko rešili rekurzivno.

Izločimo po vrsti elemente iz začetnega niza, ostanek uredimo po vrstnem redu rekurzivno izračunamo permutacije preostanka. Na koncu dodamo izločeni element na prvo mesto permutacij preostanka.

```
def permute(word):  
    res = []  
    if len(word) == 1:  
        return [word]  
    else:  
        for i in range(len(word)):  
            sub_permute = permute(word[:i]+word[i+1:])  
            for item in sub_permute:  
                res.append(word[i]+item)  
    return res
```

```
if __name__ == '__main__':  
    print(permute('abcd'))
```

8.2 Cikli v razporedbi

Iskanje ciklov v razporedbi je poseben način kako razbijemo razporedbo na manjše dele, ki se imenujejo cikli. Cikel je skupina elementov v razporedbi, kjer se elementi ciklično zamenjujejo. Cikel sestavlja vsaj en element, to je element, ki ostane na svojem mestu.

Razporedba $\pi = \{1:3, 2:1, 3:2\}$, ima en sam cikel $\pi = (1, 3, 2)$, kjer se 1 zamenja s 3 in 2 zamenja z 1 in 3 se zamenja z 2.

Razporedba ima lahko tudi več ciklov na primer $\pi = \{1:3, 2:1, 3:2, 4:5, 5:4\}$ se zapiše s cikli takole $\pi = (1, 3, 2)(4, 5)$.

find_cycles.py

```
#!/usr/bin/env python3
import random
def perm_cycles(perm: list):
    a: Dict[int, Any] = dict(enumerate(perm, start=1))
    cycles = []
    while a:
        cycle = []
        temp = next(iter(a))
        while temp not in cycle:
            cycle.append(temp)
            temp_ = a[temp]
            del a[temp]
            temp = temp_
        cycles.append(cycle)
    return cycles
```

```
if __name__ == '__main__':
    x: list[int] = list(range(1, 31))
    random.shuffle(x)
    print(perm_cycles(x))
```

8.3 100 zapornikov

<https://scanalyst.fourmilab.ch/t/the-100-prisoners-puzzle-a-combinatorial-conundrum/1625>

Ustvarili bomo seznam ciklov razporedb 100 elementov statistično določili relativno frekvenco razporedb, katerih dolžina ciklov je manj ali enako 50.

```
import find_cycles
if __name__ == '__main__':
    j = 0
    m = 1000
    for _ in range(m):
        x: list[int] = list(range(1, 101))
        random.shuffle(x)
        c: list[list[int]] = perm_cycles(x)
        n = max([len(x) for x in c])
        if n <= 50:
            j += 1
    print(f'{j/m:.2f}')
```

0.32

8.4 Kovanci

Na razpolago imamo naslednje denarne enote v obliki kovancev. Vrednosti kovancev izrazimo v evrskih centih.

1, 2, 5, 10, 20, 50, 100, 200

Na koliko načinov lahko s temi kovanci plačamo dan račun. Na primer ceno 2 evra lahko plačano na naslednji način:

$$200 = 100 + 50 + 2 \cdot 20 + 5p + 2 + 3$$

Seveda pa vrstni red kovancev ni pomemben.

```
coins = [1, 2, 5, 10, 20, 50, 100, 200]
import functools

def nways(n):
    ways = [1] + [0 for i in range(n + 1)]
    for coin in coins:
        for i in range(coin, n + 1):
            ways[i] += ways[i - coin]
    return ways[n]
```

```
if __name__ == '__main__':
    n = int(input('n --> '))
    x = nways(n)
    print("Načini plačila = ", n)
```

9 Številski sestavi

Številski sestavi določajo, kako se zapisujejo števila. Mi uporabljamo desetiški številski sestav. Osnova tega sestava je $b=10$. Števk, znakov s katerimi zapisujemo števila, je tudi 10, 0,1,2,3,4,5,6,7,8,9. Poglejmo kako zapišemo število v našem desetiškem sestavu. Vzemimo število 53107:

$$53107 = 5 \cdot 10^5 + 3 \cdot 10^4 + 1 \cdot 10^3 + 0 \cdot 10^2 + 7 \cdot 10^0.$$

V tem zapisu je zelo pomembno, na katerem mestu stoji posamezna številka. Manjkajoče potence števila 10 v razvoju napovemo s številko 0.

9.1 Drugi številski sestavi

V uporabi so tudi drugi številski sestavi, na primer šestnajstiški z osnovo $b = 16$ in dvojiški z osnovo $b = 2$. Prvi potrebuje za zapis šestnajst števk 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F, medtem ko drugi le dve 0,1.

```
cifers = '0123456789ABCDEF'
d_c = dict(enumerate(cifers))
c_d = dict(zip(d_c.values(),d_c.keys()))

def dec2base(n, b):
    if b > 16:
        return None
    if n < b: return d_c[n]
    else:
        return dec2base(n // b, b) + d_c[n % b]
def base2dec(s, b):
    if b > 16:
        return None
    n = 0
    for ch in s:
        n = n * b + c_d[ch]
    return n
```

```
if __name__ == '__main__':
    n, b, bd = input('convert n, b, bd -> ').split()
    if bd == '1':
        m = base2dec(n, int(b))
    else:
        m = dec2base(int(n), int(b))
    print(m)
```

```
convert n, b, bd -> 123 12 1
171
convert n, b, bd -> 123 12 0
A3
```

Program sprejema tri argumente: število, bazo in smer pretvorbe. Ko preberemo argumente iz koman- dne vrstice, dobimo vse zapise v obliki niza. Program v primeru, ko je `db = '1'` predpostavlja, da je število `n` zapisano v bazi `b`, zato pretvori zapis z bazo `10`. V drugem primeru pa predpostavlja, da je število zapisano v desetiškem sistemu in ga pretvori v zapis v bazi `b`.

9.2 Številski sistem Majev

Kultura Majev je uporabljala številski sistem z osnovo 20. To je bil aditivni pozicijski sistem, ki je upora- bljal tri simbole, piko za ena, vodoravno črto za pet in školjko za ničlo.

Napiši funkciji za pretvorbo iz majevskega sistema v naš desetiški sistem `maya2dec()` in obratno `dec2maya()`.

Maya





















```
#!/usr/bin/env python3

def maya2dec(s):
    base = 20
    def maya_number(s):
        return x.count('o') + 5 * x.count('-')
    l = s.split(' ')
    res = 0
    for x in l:
        res = base * res + maya_number(x)
    return res

def dec2maya(d):
    base = 20
    def number_maya(c):
        if c == 0:
            return '@'
        else:
            return (c % 5) * 'o' + (c // 5) * '-'
    res = ''
    if d == 0:
        return('@')
    while d:
        res = number_maya(d % base) + ' ' + res
        d //= base
    return res[:-1]

if __name__ == '__main__':
    print(maya2dec('o oooo---'), maya2dec('oo @ ---'), maya2dec('oo @ o- -'))
    print(dec2maya(16125))
```

```
39 815 16125
oo @ o- -
```

				
0	1	2	3	4
				
5	6	7	8	9
				
10	11	12	13	14
				
15	16	17	18	19

Slika 2: Številski sistem Mayev

https://www.youtube.com/watch?v=Ybvb7oy_WV0

<https://www.youtube.com/watch?v=VmZjPVT2M20>

Po navodilih v filmu sestavi funkcijo seštevanja.

Maya add

```
def simplify(z):
    i = len(z) - 1
    while i > 0:
        one_y = z[i].count('o')
        five_y = z[i].count('-')
        if one_y > 4:
            one_y -= 5
            five_y = five_y + 1
        if five_y > 3:
            five_y -= 4
            z[i-1] = 'o' + z[i-1]
        z[i] = one_y * 'o' + five_y * '-'
        i -= 1
    return ' '.join([t if len(t) > 0 else '@' for t in z])

def maya_add(x: str, y: str) -> str:
    xl = [t.replace('@', '') for t in x.split(' ')]
    yl = [t.replace('@', '') for t in y.split(' ')]
    if len(yl) < len(xl):
        xl, yl = yl, xl
    xl = (len(yl) - len(xl)) * [''] + xl
    sl = [''] + [''.join(sorted(a[0] + a[1],\
        reverse=True)) for a in zip(xl, yl)]
    return simplify(sl)
```

9.3 Uravnovežen mestni trojiški sestav

Mestni zapis v trojiškem sistemu potrebuje tri številke. Običajno jih zapišemo z 0, 1 in 2. Poseben primer trojiškega sistema pa je uravnovežen trojiški sistem. Tudi ta ima tri številke, vendar pa so njihove številke vrednosti drugačne. Prvo je številka 0 njena številka vrednost je pričakovana, je enaka 0. Potem

pa sta še dve števki, ena ima številsko vrednost 1, medtem ko ima druga številsko vrednost -1 . Zapišemo ju z znakoma $+$ in $-$. V uravnoveženem številskem sestavu lahko predstavimo vsa cela števila in pri tem ne potrebujemo posebnega znaka za negativna števila. Uravnovežen trojiški sistem je primer nestandardnega mestnega zapisa števil. Uporabljen je bil v zgodnjem obdobju razvoja računalništva.

Ta številski sestav je bil prvič omenjen v Stifelovi knjigi *Aritmetica Integra* (1544). Najdemo ga tudi v delih Johanna Keplera.

Kot smo že omenili v običajnem trojiškem številskem sestavu potrebujemo tri števke $0, 1, 2$, katerih številsko vrednosti so enake $0, 1, 2$. Zapis $'2101'$ v tem sestavu predstavlja število

$$2101_3 = 2 \cdot 3^3 + 1 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 = 2 \cdot 27 + 9 + 1 = 54 + 9 + 1 = 64_{10}.$$

V uravnoveženem trojiškem številskem sestavu koeficienti pred potencami števila 3 niso $0, 1$ in 2 , ampak $-1, 0$ in 1 .

Na primer zapis $+00-00$ predstavlja število

$$(+00-00) \rightarrow 3^5 - 3^2 = 234_{10}.$$

Če v zapisu zamenjamo med seboj znaka $-$ in $+$, dobimo ustrezen zapis števila, pomnoženega z -1 .

Uravnovežen trojiški zapis števila

```
#!/usr/bin/env python3
tri_digit = {'+': 1, 'o': 0, '-': -1}
digit_tri = {1: '+', 0: 'o', -1: '-'}

def to_tri(n):
    res = ''
    while True:
        a = n // 3
        b = n % 3
        if b == 2:
            res += digit_tri[-1]
            a += 1
        else:
            res += digit_tri[b]
        if a == 0:
            return res[::-1]
        n = a

def to_dec(s):
    print(s)
    res = 0
    res = tri_digit[s[0]]
    for x in s[1:]:
        res = res * 3 + tri_digit[x]
    return res
```

9.4 Rimski številski sestav

Pred letom 1202, ko je Fibonacci izdal knjigo Liber Abacci, kjer je uvedel indijsko-arabski desetiški mestni številski sistem, ki ga uporabljamo še danes, so števila zapisovali z rimskimi številkami.

Rimski sistem je bil desetiški. Ni bil mestni zapis, ni poznal znaka za 0, zato se v tem sistemu ni dalo zapisati poljubno velikega števila z omejenim naborom števk.

V rimskem zapisu števil so uporabljali znake (števke) z naslednjimi številskimi vrednostmi:

I, V, X, L, C, D, M
1, 5, 10, 50, 100, 500, 1000.

Pravila za zapis rimskih števil so naslednja:

- Znaki se združijo v pare, znak za enoto in znak za 5 enot, (I, V), (X, L), (C, D), (M, [V]).
- Pri zapisu števil se znak za enoto ponavlja od 1 do 3 krat.
- Štiri enote se zapiše tako, da se znak za enoto postavi pred znak za 5 enot.
- Pet se zapiše z znakom za 5 enot.
- Od 6 do 8 se zapiše z znakom za 5 in ponavljanjem znaka za enoto od 1 do 3.
- Devet se zapiše z znakom za enoto, ki stoji pred znakom za višjo enoto.

Števila od 1 do 9: I, II, III, IV, V, VI, VII, VIII, IX.

Desetice od 10 do 90: X, XX, XXX, XL, L, LX, LXX, LXXX, XC.

Stotice 100 do 900: C, CC, CCC, CD, D, DC, DCC, DCCC, CM.

Tisočice od 1000 do 5000: M, MM, MMM.

Številki 1634 ustreza rimska številka, ki je sestavljena takole:

4 -> IV, 30 -> XXX, 600 -> DC, 1000 -> M => 1634 -> MDCXXXIV

Rimski številski sestav

```
#!/usr/bin/env python3
def roman_to_int(roman):
    roman_dict = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
    result, prev_value = 0, 0

    for numeral in roman[::-1]:
        value = roman_dict[numeral]
        if value < prev_value:
            result -= value
        else:
            result += value
            prev_value = value

    return result
```



```

def int_to_roman(num):
    roman_numerals = {
        1: 'I', 4: 'IV', 5: 'V', 9: 'IX', 10: 'X', 40: 'XL',
        50: 'L', 90: 'XC', 100: 'C', 400: 'CD', 500: 'D',
        900: 'CM', 1000: 'M'
    }

    result = ''
    for value, numeral in sorted(roman_numerals.items(), reverse=True):
        while num >= value:
            result += numeral
            num -= value

    return result

```

```

if __name__ == '__main__':
    # Test the function
    integer_value = int(input("Vpiši naravno število: "))
    roman_numeral = int_to_roman(integer_value)
    print(f"Zapis z rimsko številko {integer_value} je {roman_numeral}")

    # Test the function
    roman_numeral = input("Vpiši rimsko številko: ").upper()
    integer_value = roman_to_int(roman_numeral)
    print(f"Rimska številka {roman_numeral} predstavlja število {integer_value}")

```

```

Vpiši naravno število: 2024
Zapis z rimsko številko 2024 je MMXXIV
Vpiši rimsko številko: MMXXIV
Rimska številka MMXXIV predstavlja število 2024

```

<https://trinket.io/python3/8cd3a2ae81>

9.5 Fibonaccijev številski sestav

```

#!/usr/bin/env python3

cache = {1: 1, 2: 1}
def fibonacci_sequence(n):
    if n in cache:
        return cache[n]
    else:
        f = fibonacci_sequence(n - 1) + fibonacci_sequence(n - 2)
        cache[n] = f
        return f

```

```

def fibonacci_to_decimal(fib_representation):
    decimal, i = 0, 1
    for bit in reversed(fib_representation):
        decimal += bit * fibonacci_sequence(i)
        i += 1
    return decimal

def decimal_to_fibonacci(decimal):
    i = 1
    while fibonacci_sequence(i) < decimal:
        i += 1
    fib_representation = []
    for fib_number in reversed(sorted(cache.values())):
        if fib_number <= decimal:
            fib_representation.append(1)
            decimal -= fib_number
        else:
            fib_representation.append(0)
    return fib_representation

decimal_number = int(input('-> n '))
fib_representation = decimal_to_fibonacci(decimal_number)
decimal_number_again = fibonacci_to_decimal(fib_representation)
print(decimal_number_again)

```

10 Verižni ulomki in zlati rez

10.1 Verižni ulomki

ulomek $r = p/q$ zapišemo v obliki končnega verižnega ulomka.

$$r = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}} = [a_0; a_1, a_2, a_3, \dots],$$

kjer so a_i koeficienti verižnega ulomka. Koeficient a_0 je celo število, medtem so ko ostali koeficienti a_i , $i > 0$, cela pozitivna števila.

Primer

Izrazimo racionalno število r v obliki verižnega ulomka. $r = \frac{415}{93}$:

- $\frac{415}{93} = 4 + \frac{43}{93} = 4 + \frac{1}{\frac{93}{43}}$,
- $\frac{93}{43} = 2 + \frac{7}{43}, \rightarrow \frac{415}{93} = 4 + \frac{1}{2 + \frac{1}{\frac{43}{7}}}$,
- $\frac{415}{93} = 4 + \frac{1}{2 + \frac{1}{6 + \frac{1}{7}}}$.

Evklid (300 p. n. š.) je v knjigi Elementi zapisal algoritem za iskanje največjega skupnega delitelja dveh števil. Koeficienti verižnega ulomka so zaporedje celih delov pri deljenju med dvema zaporednima ostankoma v Evklidovem algoritmu. Verižni ulomek iracionalnega števila in neskončen. Rešitve kvadratne enačbe se izražajo z periodičnem verižnem ulomkom.

Definirajmo funkcijo, ki sprejema dva argumenta, to sta števec in imenovalec ulomka, in vrne seznam koeficientov verižnega ulomka.

Dodali bomo še funkcijo, ki sprejema seznam koeficientov verižnega ulomka in vrne števec in imenovalec ulomka, ki ga predstavlja seznam.

Funkcija `fc` pretvori verižni ulomek v navadnega. Začnemo računat od iz zadnjega konca. Zato vrstni red v seznamu `l` obrnemo. Na začetku je m zadnji koeficient verižnega ulomka in $n = 1$. En korak pri računanju:

$$a_k + \frac{1}{\frac{m_k}{n_k}} = a_k + \frac{n_k}{m_k} = \frac{a_k m_k + n_k}{m_k}$$

Na to vrnemo recipročno vrednost rezultata, (zamenjamo števec in imenovalec v ulomku). Na koncu dobimo rezultat v okrajšano obliki, števec in imenovalec sta tuji števili.

```

def cf(m, n):          # pretvorba ulomka v verižni ulomek
    assert m > 0 and n > 0, "m in n naj bosta pozitivna"
    res = []          # koeficienti verižnega ulomka
    if m < n:         # večji od njiju postane m
        res.append(0) # prvi koeficient v. ulomka je 0
        m, n = n, m
    while n > 0:      # doklar je n različen od 0
        res.append(m // n) # celi del je koeficient v. ulomka
        m, n = n, m % n  # m postane n in n postane ostanek
    #print(m)         # pri deljenju
    return res        # koeficienti verižnega ulomka

def fc(l):            # pretvorba verižnega ulomka v ulomek
    l = l[::-1]       # koeficienti verižnega ulomka
    m = l[0]          # prvi približek končnega ulomka
    n = 1
    for a in l[1::]:  # zaporedni približki končnega ulomka
        n, m = m, a * m + n
    return m, n

if __name__ == '__main__':
    m, n = [int(x) for x in input('m n --> ').split()]
    l = cf(m, n)
    print(l)
    print(fc(l))

```

10.2 Razmerje zlatega reza

Dve pozitivni števili $a > b$ sta v razmerju zlatega reza, če velja:

$$\frac{a}{b} = \frac{b}{a-b} \quad \text{vzemimo, da je } a = x, \quad b = 1,$$

$$x = \frac{1}{x-1}, \quad \frac{1}{x} = x-1, \quad x = 1 + \frac{1}{x}$$

Po zaporednem vstavljanju $x \rightarrow 1/x$ dobimo neskončni verižni ulomek:

$$r = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}} = [1; 1, 1, 1, \dots],$$

Ki reši enačbo $x = 1 + 1/x$. Rešitev enačbe poiščemo iterativno.

Vzamemo začetni približek $x_0 = 1$ in sprožimo iteracijo

$$x_{n+1} = 1 + 1/x_n.$$

Nekaj približkov zapisanih v obliki dvojic števca in imenovalca ulomka:

```
#!/usr/bin/env python3

def add_q(x, y):      # seštevanje ulomkov
    return((x[0]*y[1] + y[0]*x[1], x[1]*y[1]))

def rec_q(x):        # recipročna vrednost
    return x[::-1]

def gold_q(n):
    res = []
    x = (1, 1)      # začetni približek
    res.append(x)
    for _ in range(n): # iteracija
        x = add_q((1, 1), rec_q(x))
        res.append(x)
    return res

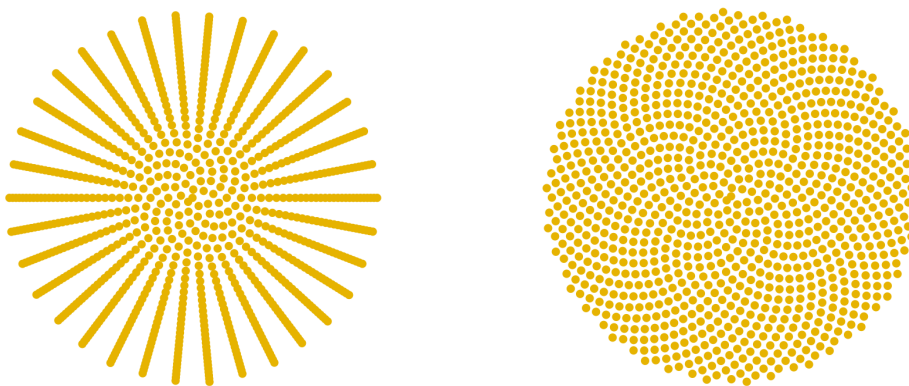
print(gold_q(10))
```

```
[(1,1),(2,1),(3,2),(5,3),(8,5),(13,8),(21,13),(34,21),(55,34),(89,55)]
```

V dvojicah prepoznamo, dva sosedna člena Fibonaccijevega zaporedja. Zaporedje ulomkov dveh sosednjih členov Fibonaccijevega zaporedja se približuje razmerju zlatega reza.

10.3 Sončnica

Sončnično socvetje je vizualni prikaz dejstva, kaj pomeni to, da je razmerje zlatega reza najbolj iracionalno število v smislu, da se najboljši možni zaporedni približki, ki jih dobimo iz verižnega ulomka, kot je le mogoče, izogibajo končni vrednosti, ker so vsi koeficienti najmanjši možni, torej enaki 1.



Slika 3: Razporeditev v odvisnosti od kota

<https://scratch.mit.edu/projects/344169437/editor>

Novi brstiči na sončničnem socvetju nastajajo v bližini središča socvetja. Kot ϕ med dvema zaporednima brstičema odloča o enakomernosti porazdelitve brstičev oziroma semen. Če je razmerje $2\pi/\phi$ blizu zlategu rezu je porazdelitev bolj enakomerna. Vzeli smo kvocient dveh sosednjih Fibonaccijevih števil F_{10}/F_9 levo in F_{15}/F_{14} desno.

Sončnica

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt, numpy as np

def fibonacci(n):
    a, b = 0, 1;
    for i in range(n - 1):
        a, b = b, a + b
    return (a, b)

def sunflower(n, m=987):
    r, alpha, x, y = 5, 0, [], []
    a, b = fibonacci(n)
    phi = 2*np.pi * a/b

    for n in range(m):
        R = np.sqrt(n) * r
        x.append(R * np.cos(alpha))
        y.append(R * np.sin(alpha))
        alpha += phi

    plt.scatter(x, y, s=20, c=[[0.9, 0.7, 0]])
    plt.xticks([]); plt.yticks([]); plt.axis('equal')
    plt.show()

if __name__ == '__main__':
    n = int(input('n -> '))
    sunflower(n)
```

<https://www.youtube.com/watch?v=CaasbfdJdJg>

10.4 Virahanka

Virahanka je bil indijski matematik prozodist. Živel je v 6. stoletju našega štetja.

Kaj je prozodija? V Wikipediji preberemo:

Verzologija (stihoslovje, metrika, prozodija) je teorija verza, del literarne teorije.

Virahanka je 500 let pred Fibonaccijem omenil zaporedje, ki danes nosi njegovo ime. Nanj je naletel pri stihoslovju. Zanimali so ga različni vzorci, sestavljeni iz kratkih in dolgih zlogov. Dolgi zlogi *guru* so bili dvakrat daljši od kratkih *lahu*. Zanimalo ga je, koliko je različnih vzorcev dane dolžine, sestavljenih iz dolgih in kratkih zlogov. Odgovor na to vprašanje ga je pripeljal do Fibonaccijevega zaporedja. Označimo

kratki zlog z L in dolgi z G. Obstaja en sam vzorec dolžine kratkega zloga, dva dolžine dolgega zloga itd. Poglejmo:

```
virahanka.py
```

```
#!/usr/bin/env python3

def virahanka(n):
    res = ["l", "ll", "g"]
    if n < 3:
        return res
    for _ in range(n - 2):
        res.append([x + "l" for x in res[-1]] + [x + 'g' for x in res[-2]])
    return res

print(virahanka(5))
```

```
[['l'], ['ll', 'g'], ['lll', 'gl', 'lg'], ['llll', 'gll', 'lgl', 'llg', 'gg'],
['lllll', 'glll', 'lgl', 'llgl', 'ggl', 'lllg', 'glg', 'lgg']]
```

Začnemo z zaporedjem `['l'], ['ll', 'g']`. Nato pa na koraku dobimo nov člen zaporedja tako, da dodamo nizom v zadnjem členu 'l' in nizom v predzadnjem členu dodamo 'g'. Število nizov v posameznem členu je enako členom Fibonaccijevega raporedja.

11 Fraktali

11.1 Fibonaccijeve generacije

Zgodba, ki se skriva za Fibonaccijevem zaporedjem govori o naraščanju zajčje populacije. Na začetku imamo mlad zajčji par, ki ga označimo z '0'. Po preteku prve časovne enote zajčji par odraste, označimo ga z '1' in družina se poveča za mlad zajčji par '0'. Družino sedaj opišemo z '10'. V vsaki naslednji časovni enoti vsak par 1 se poveča v '10' in par '0' dozori v par '1'.

Spodaj je prikazano nastajanje Fibonaccijevih generacij.

```
'0', '1', '10', '101', '10110', '10110101',  
'1011010110110', '101101011011010110101',
```

Napiši program, ki se začne z nizom '0' in se konča z n -to generacijo.

```
fibonacci_generations.py  
  
rules = {'0': '1', '1': '10'}  
  
def fibonacci_generation(gen):  
    new_gen = ''  
    for ch in gen:  
        new_gen += reules[ch]  
    return new_gen  
  
def fibonacci_generations(n):  
    gen = '0'  
    for _ in range(n):  
        gen = fibonacci_generation(gen)  
    return gen
```

11.2 Fibonaccijev fraktal

Niz z zapisom n -te Fibonaccijeve generacije interpretiramo takole:

1. Niz z Fibonaccijevo generacijo prebiramo znak za znakom.
2. Postavimo se v točko $(0, 0)$, $x = [0]$, $y=[0]$.
3. Pomiki so v smereh $s = [[1, 0], [0, 1], [-1, 0], [0, -1]]$.
4. Kazalec na pomike je na začetku enak $i = 0$.
5. Vsakokrat se pomaknemo v smeri $s[i]$.
 $x.append(x[-1] + s[i][0])$ in $y.append(y[-1] + s[i][1])$
6. Če je znak enak '1' na lihem mestu, postane $i = (i + 1) \% 4$ in če se znak '1' nahaja na sodem mestu, postane $i = (i - 1) \% 4$.

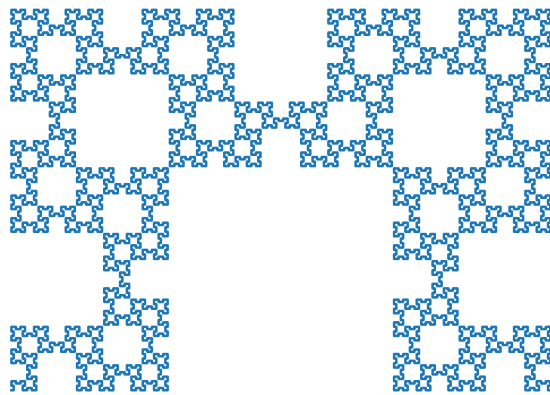
Fibonaccijev fraktal

```
import matplotlib.pyplot as plt
from fibonacci_generations import *

def fibonacci_fractal(gens):
    i, x, y, s = 0, [0], [0], [[1, 0], [0, 1], [-1, 0], [0, -1]]

    for j in range(len(gens)):
        x.append(x[-1] + s[i][0])
        y.append(y[-1] + s[i][1])      # pomik (x, y) v dani smeri
        if gens[j] == '1':            # spremeni smer i
            if j % 2: i = (i - 1) % 4 # znak 1 na lihem mestu
            else: i = (i + 1) % 4     # znak 1 na sodem mestu
    return x, y
```

```
gens = fibonacci_generations(20)
x, y = fibonacci_fractal(gens)
plt.axes().set_aspect('equal')
plt.plot(x, y)
plt.show()
```



11.3 Kochova krivulja

12 Objekti

12.1 Peanovi aksiomi

Peanovi aksiomi za množico naravnih števil.

Giuseppe Peano (1858–1932) italijanski matematik. Je zapisal aksiome, ki določajo množico naravnih števil:

1. Število 0 je naravno število.
2. Vsako naravno število n ima naslednika n' .
3. Če za dvoje naravnih števil m in n velja, da je $m' = n'$, potem ste števili enaki.
4. Vsako naravno število razen števila 0 je naslednik nekega naravnega števila.
5. Če podmnožica naravnih števil vsebuje število 0 in za vsako število n v tej množici velja, da je n' tudi element te množice, potem množica vsebuje vsa naravna števila.

Nemški matematik Leopold Kronecker (1823–1891) je nekoč izjavil, da je Bog ustvaril naravna števila $\{1, 2, 3, 4, 5, \dots\}$ vse ostalo v matematiki je delo človeka. Ameriški matematik madžarskega rodu Von Neumann (1903–1957) je uporabil teorijo množic Georgea Cantorja (1845–1920) in naredil konstrukcijo množice naravnih števil s pomočjo prazne množice. Za Cantorja je torej dovolj, da bi Bog ustvaril le prazno množico.

Von Neumannova konstrukcija temelji na predpostavki, da je množica množice različna od same množice $\{A\} \neq A$.

Prazna množica $\phi = \{\}$ nima elementov, vendar pa ima množica $\{\phi\}$ natanko en element.

Konstrukcija množice naravnih števil:

1. $\phi = \{\}$ je prazna množica
2. $1 = \phi \cup \{\phi\} = \{\phi\} = \{\{\}\}$
3. $2 = 1 \cup \{1\} = \{\phi, \{\phi\}\} = \{\{\}, \{\{\}\}\}$
4. $n' = n \cup \{n\}$, kjer je število n' naslednik števila n , $(n + 1)$.

Zapiši funkcijo, ki sprejme končno množico, in vrne Von Neumannovo konstrukcijo naravnega števila, ki predstavlja število elementov te množice.

Von Neumannova naravna števila

```
import copy
class natural:
    def __init__(self, num=None):
        if isinstance(num, list):
            self.num = ()
            for _ in num:
                self.prim()
        else:
            self.num = num

    def prim(self):
        self.num = self.num + (self.num,)
    def iprim(self):
        if self.num:
            self.num = self.num[:-1]
        else: self.num = None

    def __add__(self, other):
        res = copy.deepcopy(self)
        oth = copy.deepcopy(other)
        while oth.num:
            res.prim()
            oth.iprim()
        return res
    def __sub__(self, other):
        res = copy.deepcopy(self)
        oth = copy.deepcopy(other)
        if res >= oth:
            while oth.num:
                res.iprim()
                oth.iprim()
        else: res = natural()
        return res

    def __str__(self):
        if self.num: return f'{len(self.num)}'
        else: return 'None'

    def __gt__(self, other):
        return (self >= other) and (self != other)
    def __ge__(self, other):
        return set(other.num).issubset(set(self.num))
    def __lt__(self, other):
        return (self <= other) and (self != other)
    def __le__(self, other):
        return set(self.num).issubset(set(other.num))
    def __eq__(self, other):
        return set(self.num) == set(other.num)
    def __ne__(self, other):
        return set(self.num) != set(other.num)
```

```

if __name__ == '__main__':
    a = ['a', 'b', 'c']
    b = ['d', 'e']
    na = natural(a)
    nb = natural(b)
    print((na < nb))

```

12.2 Rešitev enačbe zlatega reza

Pogledali si bomo, kako definiramo objekte in kako prenalozimo operatorje. Vzemimo primer, ki smo ga že srečali. Enačbo

$$x = 1 + \frac{1}{x}$$

rešujemo iterativno. Vendar pa bi v tem primeru želeli, da se je spremenljivka x vedno zapisana v obliki ulomka, dvojice $x = (\text{števec}, \text{imenovallec})$. Zato bomo definirali nov objekt, objekt racionalni števil prenalozili operatorje, ki jih bomo potrebovali.

```

class rat:
    def __init__(self, a, b=1):
        self.value = (a, b)

    def __add__(self, other):
        return rat(self.value[0] * other.value[1] +
                  self.value[1] * other.value[0], self.value[1] * other.value[1])

    def __truediv__(self, other):
        return rat(self.value[0] * other.value[1],
                  self.value[1] * other.value[0])

    def __str__(self):
        return "{0:d}, {1:d}".format(*self.value)

def convergent(x: rat) -> rat:
    return one + (one / x)

one = rat(1)

for _ in range(13):
    x = convergent(x)
    print(x, end=' ')

```

```

(2, 1) (3, 2) (5, 3) (8, 5) (13, 8) (21, 13) (34, 21)
(55, 34) (89, 55) (144, 89) (233, 144) (377, 233) (610, 377)

```

Kot smo že omenili je rešitev enačbe $x = 1 + 1/x$ razmerje zlatega reza. Konvergenti so zaporedje ulomkov, ki jih dobimo v iteraciji in so kvocienti dveh sosednjih členov v Fibonaccijevem zaporedju.

12.3 Permutacijska grupa

Grupa je množica elementov med katerimi je definirana operacija, ki dvema elementoma grupe predpiše spet element grupe. Če grupo označimo z G in operacijo z \circ zapišemo:

$$\circ : G \times G \rightarrow G$$

Dvočlena operacija \circ v grupi mora ustrezat naslednjim pravilom (aksiomom).

1. **Asociativnost** $a \circ (b \circ c) = (a \circ b) \circ c, \quad a, b, c \in G$
2. **Nevtralni element** $\exists! e \in G, \forall a \in G \Rightarrow a \circ e = e \circ a = a$
3. **Inverzni element** $\forall a \in G, \exists a^{-1} \in G \Rightarrow a \circ a^{-1} = a^{-1} \circ a = e$

Vzemimo množico razporedb n elementov. Običajno so razporedbe predstavljene s seznamom števil $1 \leq i \leq n$. Nad to množico definiramo transformacije oziroma permutacije, ki premetavajo elemente razporedb in tako preslikajo eno razporedbo v drugo. Vsako tako permutacijo (transformacijo) lahko enolično predstavimo z razporedbo, v katero le-ta prevede razporedbo, ki ima elemente urejene v naravnem vrstnem redu. V množici permutacij je definiran produkt to je zaporedno učinkovanje dveh permutacij. Produkt permutacij je spet permutacija, ki jo lahko predstavimo z razporedbo v katero le-ta prevede razporedbo, ki ima elemente v naravnem vrstnem redu kot smo že omenili zgoraj. Ker lahko razporedbe in permutacije opišemo z enako podatkovno strukturo se lahko na koncu posvetimo enemu samemu pojmu permutacijami ali razporedbami. Množica permutacij ima grupno strukturo. Grupna operacija je produkt permutacij, nevtralni element je permutacija, ki je podana z razporedbo elementov v naravnem vrstne redu. Inverzni element je permutacija, ki prevede razporedbo, s katero je predstavljena dana permutacija v razporedbo, ki ima elemente urejene v naravnem vrstnem redu. Produkt transformacij je vedno asociativen, ni pa komutativen.

Pokažimo na primeru:

$$a = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 2 & 3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix}, \quad a \circ b = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$$
$$b \circ a = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 3 \end{pmatrix}, \quad a^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 2 \end{pmatrix}, \quad (a \circ b)^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

Pri zapisu permutacij običajno izpuščamo zgornjo vrstico, ker ta je vedno enaka.

Permutacijska grupa

```
class permutation:
    def __init__(self, x):
        if isinstance(x, list):
            if isinstance(x[0], int):
                self.value = dict(enumerate(x, start=1))
            else:
                self.value = {}
                for c in x:
                    res, n = {}, len(c)
                    for i in range(n):
                        res[c[i]] = c[(i + 1) % n]
                    self.value.update(res)
        elif isinstance(x, dict):
            self.value = x
        else:
            self.value = None

    def __str__(self):
        res = []
        for i in sorted(self.value.keys()):
            res += [str(self.value[i])]
        res = "[" + ", ".join(res)
        return res + "]"

    def __mul__(self, other):
        res = {}
        x = self.value.copy()
        while x:
            item = x.popitem()
            res[item[0]] = other.value[item[1]]
        return permutation(res)

    def __truediv__(self, other):
        return (self * ~other)

    def __invert__(self):
        return permutation(dict(zip(self.value.values(), self.value.keys())))

    def __xor__(self, other):          # potenciranje ^
        assert isinstance(other, int)
        a = self
        if other == 0: return a.unit() # ^0 je identiteta
        if other == 1: return a
        if other < 0: a = ~a          # negativna potenca,
        n, b = abs(other), a
        for _ in range(n - 1): a *= b
        return a
```

```

def unit(self):
    return permutation(list(range(1, len(self.value) + 1)))

def to_cycles(self):
    cycles = []
    a = self.value.copy()
    while a:
        cycle, temp = [], next(iter(a))
        while temp not in cycle:
            cycle.append(temp)
            temp_ = a[temp]
            del a[temp]
            temp = temp_
        cycles.append(cycle)
    return cycles

```

```

if __name__ == '__main__':
    import random
    x = list(range(1, 11))
    random.shuffle(x)
    p = permutation(x)
    c = p.to_cycles()
    q = permutation(c)
    print(p, c, q, q.value)

```

```

[1, 3, 2, 6, 7, 10, 9, 8, 5, 4]
[[1], [2, 3], [4, 6, 10], [5, 7, 9], [8]]
[1, 3, 2, 6, 7, 10, 9, 8, 5, 4]
{1: 1, 2: 3, 3: 2, 4: 6, 6: 10, 10: 4, 5: 7, 7: 9, 9: 5, 8: 8}

```

13 Okraski ali dekoratorji

Dekoratorji dodajo funkciji nove lastnosti, ne da bi spreminjali samo kodo v definiciji funkcije.

13.1 Funkcija deljenja

Definiramo funkcijo, ki deli dve pozitivni števili. Naknadno bomo funkciji dodali lastnost, da je rezultat deljenja vedno enak deljenju večjega od števil z manjšim.

Dekorator sprejema kot argument funkcijo ji doda nove lastnosti.

```
def divide_function(x, y):
    assert x > 0 and y > 0, 'x in y morata biti pozitivna'
    return x/y
def decorator_function(original_function):
    def wrapper_function(x, y):
        if x < y:
            x, y = y, x
        return original_function(x, y)
    return wrapper_function

w = decorator_function(divide_function)(3, 4)
print(w)
```

Vendar bi želeli, da to novo dekorirano funkcijo kličemo z istim imenom kot prvotno.

Python nam omogoča tudi to. Pred definicijo funkcije dodamo ime dekoratorja.

```
@decorator_function
def divide_function(x, y):
    assert x > 0 and y > 0, 'x in y morata biti pozitivna'
    return x/y

w = divide_function(3, 4)
print(w)
```

Rezultat je enak v obeh primerih.

13.2 Timer

Dani funkciji dodamo dekorator `timer`, ki bo meril čas izvajanja funkcije.

```
def timer(f):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = f(*args, **kwargs)
        stop_time = time.time()
        dt = stop_time - start_time
        print(f"delta t = {dt}")
        return result
    return wrapper
```

Funkcija `wrapper` je definirana tako, da lahko prenese poljubno število argumentov `*args` in argumentov podanih v obliki slovarja (ključ, vrednost). Poglejmo

```
def alpha(*args, **kwargs):
    print(f'args = {args}')
    print(f'kwargs = {kwargs}')

alpha('a', 2, None, x=7, y=11, z=28)
```

```
args = ('a', 2, None)
kwargs = {'x': 7, 'y': 11, 'z': 28}
```

Meritev:

```
@timer
def prime_factorisation(n):
    factors = []
    divisor = 2
    while n > 1:
        while n % divisor == 0:
            factors.append(divisor)
            n //= divisor
        divisor += 1
    return factors

print(prime_factorisation(2**29+1))
```

```
delta t = 0.9186558723449707
[3, 59, 3033169]
```

```

from functools import cache
import time

def timer(f):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = f(*args, **kwargs)
        stop_time = time.time()
        dt = stop_time - start_time
        print(f"delta t = {dt}")
        return result
    return wrapper

#@cache
@timer
def fibonacci(n):
    if not isinstance(n, int) or n < 1:
        raise ValueError(f'{n} is not positive integer')

    if n == 1 or n == 2:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

for i in range(1, 10):
    print(fibonacci(i))

```

Dekorator `cache` poskrbi za pomnjenje vmesnih rezultatov. Poženite program in primerjajte izvajanje z izvajanjem v primeru, ko odstranite znak za komentar `#` pred dekratorjem `cache`.

14 Generatorji in iteratorji

14.1 Funkcija

```
def f():  
    return 1  
    return 2  
print(f())
```

Pri vsakem klicu dobimo le prvo vrednost. Generatorska funkcija namesto stavka `return` uporablja stavke `yield`.

```
def g():  
    yield 1  
    yield 2  
print(g())
```

```
<generator object g at 0x7f2b6c54c200>
```

Dobilo le kazalec na funkcijo. Če pogledamo v slovar in se prepričamo, kaj *yield* pomeni v angleščini dobimo odgovor *pridelek, izkoristek, donos*, v tem primeru ustreza bolj glagol *izvleči*. Poglejmo

```
for x in g():  
    print(x, end=' ')
```

```
1 2
```

14.2 Sploščeni seznam

Sploščitev gnezdenega seznama lahko realiziramo s generatorsko funkcijo, ki se kliče rekurzivno.

```
def flatten(l):  
    for x in l:  
        if isinstance(x, list): yield from flatten(x)  
        else: yield x  
l = [1, 2, 3, [4, 5, [6, 7], 8], 9]  
print(list(flatten(l)))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

15 Igre

15.1 Monty Hall

Vse se je začelo z ameriškim televizijskim programom v 70-tih letih prejšnjega stoletja z naslovom *Let's Make a Deal*, ki ga je na začetku vodil Monty Hall. V studiju je bilo troje vrat. Za enimi od treh vrat se je nahajala nagrada. Gost je izbral vrata in prejel nagrado v primeru, ko je pokazal na vrata za katerimi je bila nagrada.

Da bi bila igra bolj zanimiva so nekoliko spremenili pravila. Igra je potekala v dveh fazah:

1. Najprej je gost naključno izbral vrata. Vendar teh vrat niso odprli, ampak so odprli druga vrata za katerimi ni bilo nagrade.
2. V drugi fazi se je igralec odločil ali vztraja pri prvotni odločitvi ali pa izbere druga zaprta vrata.

V drugi fazi ima gost gost ima tri možnosti,

1. da vztraja pri prvotni izbiri ali,
2. da se premisli in pokaže na druga zaprta vrata,
3. oziroma naključno izbere ena od teh dveh vrat.

<https://scratch.mit.edu/projects/226103761/editor>

Ugotovi optimalno strategijo, da bo imel gost največjo možno verjetnost, da dobi nagrado.

Napiši program, ki bo statistično pregledal katera od teh možnosti ima največ zadetkov v 100 poskusih.

Monty Hall

```
from random import randint

def set_car():
    doors = [0 for i in range(3)] # Postavijo se troje vrat
    car = randint(0, 2)          # Izbira vrat
    doors[car] = 1               # Za temi vrati je avto
    return doors

def choose_door():
    doors = set_car()           # Začetna konfiguracija
    door = randint(0, 2)        # Naključna izbira vrat
    open_door = (door + 1) % 3  # Odprejo se druga vrata
    if doors[open_door] == 1:   # za katerimi ni avta
        open_door = (open_door + 1) % 3
    door = pop({0, 1, 2} - {door, open_door})
    if doors[door] == 1:       # Kaj je za izbranimi vrati
        return True
    else:
        return False
```

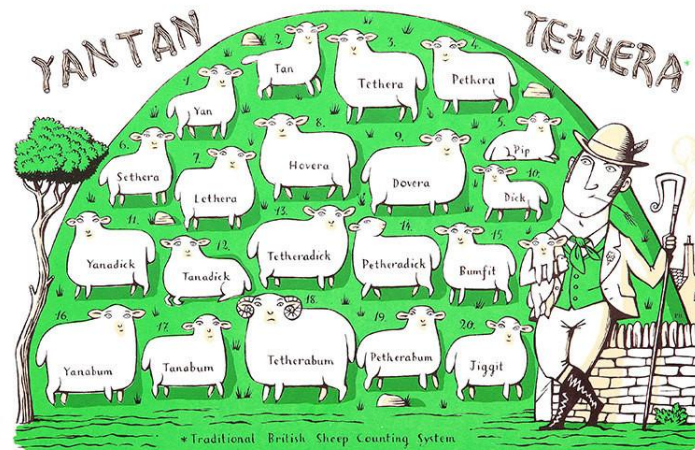
Statistika

```
statistics = 0
for _ in range(1000):
    statistics += choose_door()
print(statistics/10)
```

65.2 %

<https://trinket.io/python/c1a6b1f0d6>

15.2 An ban pet podgan



Ste kdaj pomislili, od kod čudne besede pri otroških izštevankah, kot je na primer tale:

*En ten tenera
en ten tenera, bum čin kalvara,
kalvarina bundočina, bundočina kalvara,
ekate pekate cukate me, fibe fabe domine,
ektum pektum kufer štuc, kvinte kvante fingo puc.*

Nekatere od teh besed spominjajo na besede, ki jih najdemo v starem dvajsetiškem keltskem štetju ovac na britanskem otočju (glej sliko). Ta je bil v uporabi do začetka dvajsetega stoletja. Sledove keltskega dvajsetiškega sistema najdemo v irskem in francoskem jeziku.

V francoščini štejemo:

$\text{soixante-dix} = 60 + 10 = 70$, $\text{quatre-vingts} = 4 \times 20 = 80$ in $\text{quatre-vingt-dix} = 4 \times 20 + 10 = 90$.

Programsko bomo simulirali izštevanko.

Razporedimo n otrok v krog. Zaradi enostavnosti označimo otroke s številkami od 0 do $n - 1$. Vzemimo, da ima izštevanka k besed. Začnemo s štetjem pri otroku, ki smo ga označili s številko 0. Tega ne izločimo

v prvem koraku, ampak šele otroka, ki je na k tem mestu. Med obema je $k - 2$ mest. Če je $k < n$, je ta označen s številko $k - 1$. In tako naprej. Program naj izračuna, s katero številko je označen otrok, ki ostane zadnji.

Primer, če je $n = 41$ in ima izštevanka $k = 3$ besed, bo na zadnje ostal otrok označen s številko 30.

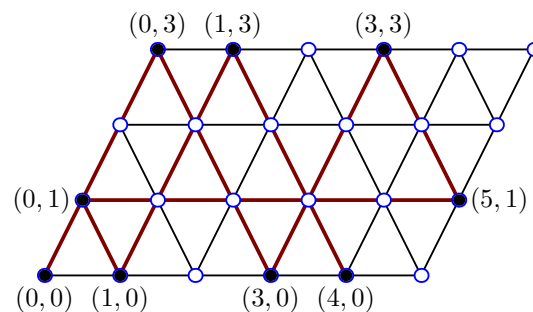
Odštevanka

```
#!/usr/bin/env python3
def odstevanka(n, k):
    nn, k, r = list(range(n)), 0, k - 1
    while len(nn) > 1:
        r = (r + k) % len(nn)
        del nn[r]
    return "lovi: {}".format(nn[0])
```

```
if __name__ == "__main__":
    line = input('otroci besede --> ')
    n, k = [int(p) for p in line.split()]
    print(odstevanka(n,k))
```

15.3 Umri pokončno

V filmu Umri pokončno 3, morata Bruce Willis in Samuel Jackson, prvi v vlogi new yorškega policaja, drugi v vlogi črnega aktivista, razreševati uganke, ki jima jih nastavlja Jeremy Irons, ki igra vlogo negativca Petra. Če ne bi razrešita uganke pravočasno, bi povzročila eksplozijo bombe. Prizorišče, kjer morata razrešiti eno od uganek, je vodnjak ob katerem se nahaja kovček in dve posodi, ena drži 5 galon, druga pa 3 galone. Ko odpreta kovček se zavrti posnetek s Petrovim glasom, ki pravi, da je v kovčku bomba, in da imata 30 sekund časa, da jo deaktivirata, tako da postavita posodo s štirimi galonami vode na tehtnico, ki se nahaja v kovčku. Ker nimata dovolj časa, si pomagata le s posodama, ki sta na razpolago. Naredi program ki simulira kako odmerita 4 galone vode. Koraki za doseg cilja so: ali napolnita eno posodo do vrha, prelijeta vodo iz ene posode v drugo in, da spraznita posodo.



Program naj bo splošen. Kdaj je problem rešljiv? Imamo dve posodi a in b in odmerek k , ki se mora znajti na koncu v večji posodi.

Primer za dve posodi 7 in 5 enot narejen s pomočjo programa [scratch](https://scratch.mit.edu/projects/144375715/editor/).
<https://scratch.mit.edu/projects/144375715/editor/>

```

#!/usr/bin/env python3

def gcd(m, n):
    if m < n: m, n = n, m
    while n:
        m, n = n, m % n
    return m

def extract(x, y, k):
    if x < y: x, y = y, x
    if x < k: return [(0, 0)]
    if y == k: return [(0, k)]
    elif x == k: return [(k, 0)]

    d = gcd(x, y)
    if k % d: return [(0, 0)]
    else: x //= d; y //= d; k //= d

    res, xx, yy = [], 0, 0
    res.append((xx,yy))
    while True:
        yy = y
        res.append((xx, yy))
        while xx < x:
            xx += y
            yy = 0
            if xx <= x:
                res.append((xx, yy))
                if xx == k: return res
                res.append((xx, y))
            else:
                yy, xx = xx - x, x
                res.append((xx, yy))
                xx = 0
                res.append((xx, yy))
                xx, yy = yy, 0
                res.append((xx, yy))
                if xx == k: return res
                res.append((xx, y))
        return res

if __name__ == '__main__':
    a, b, c = [int(x) for x in input('a, b, c -> ').split()]
    print(extract(a, b, c))

```

```

a, b, c -> 5 3 4
[(0, 0), (0, 3), (3, 0), (3, 3), (5, 1), (0, 1), (1, 0), (1, 3), (4, 0)]

```

15.4 Rojstni dnevi učencev

Vprašali smo ChatGPT, da napiše program simulacije rojstnodnevnega protislovja.

Če se vprašamo kolika je verjetnost da imata dva učenca v razredu z 23 učencev rojstni dan na isti dan, dobimo nepričakovano veliko verjetnost. Ta je za 23 učencev že več kot polovica. To pomeni, da v povprečju v naključno izbranem razredu s 23 učenci je verjetnost, da imata vsaj dva učenca na isti dan rojstni dan večja od 0.5. Odgovor ChatGPT je naslednji.

Rojstni dan

```
import random

def generate_birthdays(num_people):
    """Generate random birthdays for a given number of people."""
    birthdays = [random.randint(1, 365) for _ in range(num_people)]
    return birthdays

def has_duplicate(birthdays):
    """Check if there are any duplicate birthdays in the list."""
    unique_birthdays = set(birthdays)
    return len(birthdays) != len(unique_birthdays)

def birthday_paradox_simulation(num_simulations, num_people):
    """Simulate the Birthday Paradox."""
    count_duplicates = 0
    for _ in range(num_simulations):
        birthdays = generate_birthdays(num_people)
        if has_duplicate(birthdays):
            count_duplicates += 1
    probability = count_duplicates / num_simulations
    return probability

if __name__ == "__main__":
    num_simulations = 10000
    num_people = 23
    probability = birthday_paradox_simulation(num_simulations, num_people)
    print(f"Probability of a shared birthday among {num_people} people after
    ↪ {num_simulations} simulations: {probability:.2f}")
```

Probability of a shared birthday among 23 people after 10000 simulations: 0.51

1. Leto ima 365 dni. Ustvarimo seznam z naključno izbranimi 23 števili iz intervala (1, 365).
2. Preverimo, ali se števila ponavljajo. To storimo, tako da seznam spremenimo v množico. V množici se elementi ne ponavljajo.
3. Na koncu primerjamo število elementov v prvotnem intervalu in v množici. Če se število elementov ne ujema, pomeni, da se elementi v seznamu ponavljajo, kar pomeni, da imata vsaj dva učenca rojstni dan na isti dan v letu.

https://www.youtube.com/watch?v=Y_shcEgdhI8

15.5 Skrito število

2 3 6 7 10 11 14 15 16 17 18 19 20 21 22 23 32 33 34 35 36 37 38 39
18 19 22 23 26 27 30 31 24 25 26 27 28 29 30 31 40 41 42 43 44 45 46 47
34 35 38 39 42 43 46 47 48 49 50 51 52 53 54 55 48 49 50 51 52 53 54 55
50 51 54 55 58 59 62 63 56 57 58 59 60 61 62 63 56 57 58 59 60 61 62 63

1 3 5 7 9 11 13 15 4 5 6 7 12 13 14 15 8 9 10 11 12 13 14 15
17 19 21 23 25 27 29 31 20 21 22 23 28 29 30 31 24 25 26 27 28 29 30 31
33 35 37 39 41 43 45 47 36 37 38 39 44 45 46 47 40 41 42 43 44 45 46 47
49 51 53 55 57 59 61 63 52 53 54 55 60 61 62 63 56 57 58 59 60 61 62 63

Zamislite si eno število med $0 < n < 64$,

Program naključno prikaže eno za drugo tabele, ki so zapisane zgoraj in vas za vsako vpraša ali se vaše število nahaja na tabeli ali ne.

Na začetku postavi končno vrednost na nič, potem pa za vsak odgovor **da**, prišteje končni vrednosti prvo število v tabeli.

Na koncu je vsota enaka številu, ki ste se ga zamislili. Zakaj to deluje?

Število med $0 < n < 64$ zapišemo v binarnem številskem sistemu z kot vsoto potenc števila 2, 2^m , kjer je $0 \leq m \leq 5$:

$$n = x_5 * 2^5 + x_4 * 2^4 + x_3 * 2^3 + x_2 * 2^2 + x_1 * 2^1 + x_0 * 2^0$$
$$n = x_5 * 32 + x_4 * 16 + x_3 * 8 + x_2 * 4 + x_1 * 2 + x_0 * 1 \quad x_i \in \{0, 1\}.$$

Na primer število 42 se zapiše kot vsota potenc števila 2 kot:

$$42 = 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = 32 + 8 + 2 \quad (1)$$

V dvojiškem številskem sistemu ga zapišemo s šestimi števki **0** in **1**, $42 = 101010$. V tabelah je število na prvem mestu enako potenci števila 2. Dano število se nahaja na tabelah, kjer v njegovem zapisu v binarnem številskem sistemu, stoji številka **1** pred dano potenco števila 2. Število 42 se nahaja na tabelah, kjer je prvo število v tabeli 2, 8 in 32.

Pretvorba iz desetiškega sistema v binarnega in obratno [dec2bin.py](#).

```
dec2bin.py
```

```
def dec2bin(n):  
    res = ''  
    if n < 2: return str(n)  
    while n:  
        res += str(n % 2)  
        n //= 2  
    return res[::-1]  
  
def bin2dec(s):  
    res = 0  
    for ch in s: res = 2 * res + int(ch)  
    return res
```

stevilo.py

```
from dec2bin import dec2bin, bin2dec
from random import shuffle

table = 4*(8*"{:4d} "+"\\n")
bin_nums = [(6 - len(y)) * '0' + y for y in [dec2bin(x) for x in range(64)]]
tablex = \
    [[bin2dec(x) for x in bin_nums if x[i] == '1'] for i in range(6)][::-1]

number = 0
print("Zamislite si številko med 1 in 63")
lst = list(range(6))
shuffle(lst)
for i in lst:
    print(table.format(*tablex[i]))
    answer = input('Ali je vaša številka v tabeli? ')
    if answer == 'da':
        number += tablex[i][0]
print('Vaša številka je {}'.format(number))
```

<https://trinket.io/python3/69955a0098>

Kako delujeta programa `dec2bin` in `bin2dec`?

Funkcija `dec2bin`:

1. Znak `%` deli celi števili in vrne ostanek pri deljenju.
2. Znak `//` vrne celi del pri deljenju dveh celih števil.
3. Ostanek pri deljenju z 2 sta 0 ali 1.
4. V zanki `while` poiščemo ostanek pri deljenju z 2, ga spremenimo v znak `char` in ga dodamo k končnemu nizu `res`.
5. Vsakokrat število `n` zamenjamo z celim delom pri deljenju z 2
6. Zanka se zaključi, ko je število `n` enako 0.
7. Na koncu program vrne niz enic in ničel v obratnem vrstnem redu.

Funkcija `bin2dec`:

1. Rezultat `res` na začetku postavimo na 0.
2. Prebiramo vsebino niza znak za znakom pomnožimo `res` z 2 in prištejemo prebrani znak spremenjen v `int`.
3. Če je niz $s = [x_n, x_{n-1}, \dots, x_1, x_0]$, kjer je $x_i \in \{0, 1\}$, je rezultat enak:

$$n = 2(2(\dots(2(2x_n + x_{n-1}) + x_{n-2}) \dots) + x_1) + x_0 = x_n 2^n + x_{n-1} 2^{n-1} \dots + x_1 2^1 + x_0 2^0$$

16 Numerične metode

16.1 Bisekcija

Reševanje nelinearnih enačb oblike:

$$f(x) = 0$$

Najpreprostejša metoda je metoda bisekcije (razpolavljanja). Iščemo točko, v kateri je vrednost funkcije enaka nič. Graf funkcije mora v tej točki sekati os x . To pomeni, da funkcija spremeni znak, ko neodvisna spremenljivka x prečka to točko. Poleg funkcije moramo pri tej metodi podati interval, po možnosti naj bo točka, ki jo iščemo edina v kateri je vrednost funkcije na podanem intervalu enaka 0. Ker govorimo o zveznih funkcijah, to so funkcije katerih graf je nepretrgana krivulja, potem vrednosti funkcije v krajiščih intervala morata biti različnega znaka. Algoritem:

Bisekcija

```
def bisect(f, a, b, eps=1e-5):
    if f(a) * f(b) < 0:           # funkcijski vrednosti
                                # v krajiščih različnega znaka
        c = (a + b) / 2          # točka na sredini intervala
        while abs(f(c)) > eps:   # je vrednost f(c) dovolj majhna
            if f(a) * f(c) < 0:  # izberemo podinterval,
                b = c            # kjer sta funkcijski vrednosti
            elif f(b) * f(c) < 0: # različnega znaka
                a = c
            else:
                return c
        c = (a + b) / 2          # središče novega intervala
    return c                     # vrne c, f(c) approx 0
return False                    # če ni izpolnjen pogoj
```

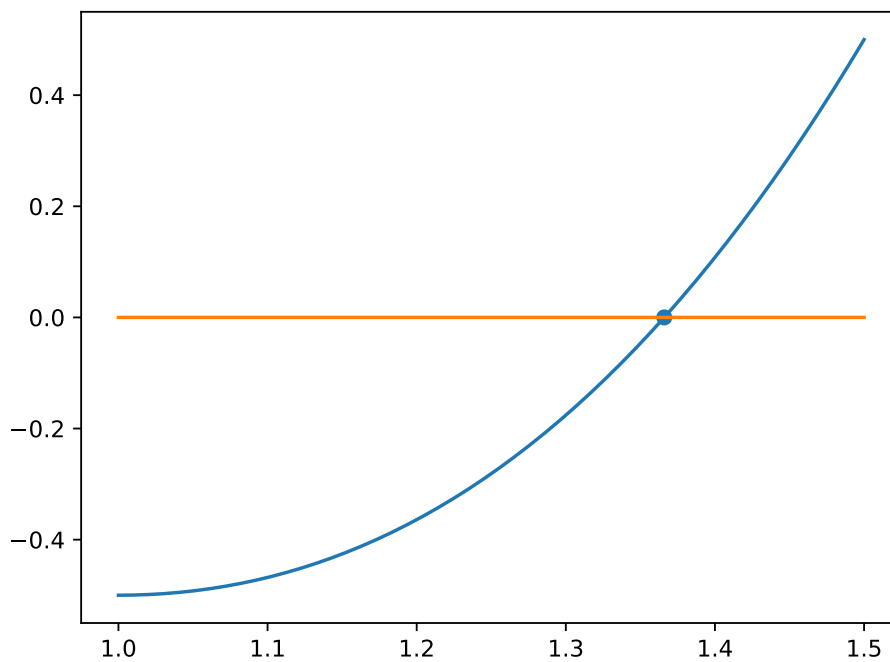
- Podana je funkcija $f(x)$ in interval $[a, b]$.
- Vrednosti funkcije v krajiščih intervala sta različno predznačeni. $f(a)f(b) < 0$
- Ponavljamo v zanki:
 - Razpolovimo interval. Razpolovišče je točka $c = (a + b)/2$
 - Če je $f(a)f(c) < 0$ potem postane $b = c$,
 - če je $f(c)f(b) < 0$ potem postane $a = c$,
 - sicer pa je $f(c) = 0$ in vrnemo točno rešitev $x = c$
- Ponavljamo dokler ni vrednost $|f(c)|$ dovolj majhna.
- Na koncu vrnemo približno rešitev $x = c$.

```
if __name__ == '__main__':
    import numpy as np
    import matplotlib.pyplot as plt

    f, a, b = lambda x: 2 * x ** 3 - 3 * x ** 2 + 0.5, 1, 1.5
    x = np.linspace(a, b)
    y = f(x)
    z = bisect(f, 1, 1.5)

    plt.plot(x, y)
    plt.scatter([z], [0])
    plt.plot([a, b], [0, 0])
    plt.show()
    print(z)
```

1.3660202026367188



Slika 4: $f(x) = 2x^3 - 3x^2 + 1/2$

16.2 Bisekcija v zlatem rezu

Interval $[a, b]$ ne razdelimo na polovico ampak v zlatem rezu. Krajši del ob krajišču intervala, kjer je funkcijska vrednost po absolutni vrednosti manjša.

```
#!/usr/bin/env python3
import numpy as np

fi = (1 + np.sqrt(5)) / 2 # zlati rez
f1 = 1 / fi              # daljši interval
f2 = 1 / fi ** 2        # krajši interval   f1/f2 = fi

def bisect_gold(f, a, b, eps=1e-5):
    ya = f(a)
    yb = f(b)
    if ya * yb < 0:
        i = 0
        while abs(b - a) > eps:
            if abs(ya) < abs(yb): # se postavi bližje k a
                c = a + (b - a) * f2
            else: # sicer bližje k b
                c = a + (b - a) * f1
            yc = f(c)
            if ya * yc < 0:
                b = c
                yb = yc
            elif yb * yc < 0:
                a = c
                ya = yc
            else:
                return c
            i += 1
        print(i)
        print(abs(b - a))
        return c
    else:
        return None
```

16.3 Trisekcija

Poiščimo minimalno vrednost funkcije $f(x)$ na intervalu $[a, b]$. Na tem intervalu ima funkcija natanko en relativni minimum. Rešujemo z metodo trisekcije.

- Interval $[a, b]$ razdelimo na tri dele. Notranja krajišča intervalov po delitvi, določimo v razmerju zlatega reza $x_1 = a + d/\varphi$ in $x_2 = a + d/\varphi^2$, kjer je d dolžina prvotnega intervala $d = b - a$ in $\varphi = (1 + \sqrt{5})/2$
- Izračunamo funkcijski vrednosti $y_1 = f(x_1)$ in $y_2 = f(x_2)$.
- Primerjamo vrednosti y_1 in y_2 . Če je $y_1 < y_2$, leži minimalna vrednost levo od x_1 , zato popravimo vrednosti $b = x_2$ in premaknemo $x_2 = x_1$ in $y_2 = y_1$.
- Izračunamo dolžino novega intervala $d = b - a$ in gremo v zanko.
- Če je $y_1 > y_2$, potem postopamo simetrično.
- Na vsakem koraku izračunamo le eno novo funkcijsko vrednost.

Trisekcija

```
def trisect_gold(f, a, b, eps=1e-5):
    phi1 = (1 + 5 ** 0.5) / 2 # Zlati rez
    phi2 = phi1**2
    d = b - a
    x1, x2 = a + d / phi2, a + d / phi1
    y1, y2 = f(x1), f(x2)

    while abs(d) > eps:
        if y1 < y2:
            b, x2, y2 = x2, x1, y1
            d = b - a
            x1 = a + d / phi2
            y1 = f(x1)
        else:
            a, x1, y1 = x1, x2, y2
            d = b - a
            x2 = a + d / phi1
            y2 = f(x2)

    return (a + b) / 2

if __name__ == '__main__':
    import numpy as np
    import matplotlib.pyplot as plt
    f, a, b = lambda x: 2 * x ** 3 - 3 * x ** 2 + 0.5, -0.5, 1.5
    x, y = np.linspace(a, b), f(x)
    z = trisect_gold(f, 0.5, 1.4)
    plt.plot(x, y); plt.scatter([z], [f(z)]); plt.plot([a, b], [0, 0])
    plt.show()
```

16.4 Sekantna metoda

- Izberemo dva začetna približka x_0 in x_1 . Po možnosti taka, da sta funkcijski vrednosti $f(x_0)$ in $f(x_1)$ različna predznaka. Z drugimi besedami, $f(x_0)f(x_1) < 0$. Rešitev $f(x) = 0$ se potemtakem nahaja na intervalu $x \in [x_0, x_1]$.

- Računamo zaporedne približke:

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

- Ponavljamo, dokler ni približek x_n dovolj blizu dejanski rešitvi, z vnaprej določeno toleranco ali številom ponovitev.

Sekantna metoda

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import numpy as np

def secant_method(f, x0, x1, tol, max_iter):
    n = 0
    while n < max_iter:
        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        if abs(x2 - x1) < tol:
            return x2 # Metoda konvergira
        x0, x1 = x1, x2
        n += 1
    return None # Po max_iter iteracijah, vrednost
                # abs(x1 - x2) ni dovolj blizu 0.
```

```
if __name__ == '__main__':
    # Example usage:
    def f(x):
        return x**2 - 4 # Enačba x^2 - 4 = 0

    root = secant_method(f, 1.0, 3.0, 1e-6, 100)
    if root is not None:
        print("Približna rešitev:", root)
    else:
        print("Metoda ne konvergira.")

    if root:
        a, b = 1, 3
        x = np.linspace(a, b)
        y = f(x)

        plt.plot(x, y); plt.plot([a, b], [0, 0]); plt.scatter([root], [0])
        plt.show()
```

17 Razne naloge

17.1 Sredinska črka

Napišite funkcijo z imenom `mid`, ki sprejema niz in vrne črko na sredini niza. Srenja črka je črka v nizu, kjer je število črk levo od nje enako številu črk desno od nje. Če take črke v nizu ni, potem vrne niz `'-?-'`.

```
def mid(s: str) -> str:
    n = len(s)           # dolžina niza
    if n % 2:           # ali je dolžina liho število?
        return s[n // 2] # vrne sredinsko črko
    else:               # če je dolžina niza sodo število,
        return '-?-'    # sredinske črke ni.
print(mid(input('Beseda -> ')))
```

17.2 Naredimo palindrom

Ali je mogoče v dani besedi premetati črke tako, da postane palindrom. Palindrom je beseda, ki se bere enako od leve proti desni, kot od desne proti levi.

Vsaka črka, razen kvečjemu ene same, mora nastopati v besedi sodo število krat. V tem primeru lahko sestavimo palindrom.

Iz standardnega vhoda preberite besedo. Napišite funkcijo, ki bo vrnila `True`, če je mogoče sestaviti palindrom in `False`, če to ni mogoče.

```
def is_palindrome_possible(s):
    s = s.lower()      # pretvori v male tiskene črke
    m = [s.count(x) % 2 for x in set(s)] # set(s) množica različnih črk v s
    return m.count(1) < 2 # % ostanek pri deljenju testira sodo/liho
```

```
if __name__ == '__main__':
    print(is_palindrome_possible("rearcac"))
    print(is_palindrome_possible("suhbeusheff"))
    print(is_palindrome_possible("palindrome"))
```

```
True
True
False
```

17.3 Učenci

Imamo spisek učencev s podatki o spolu in predmetu, ki ga imajo najraje. Drugi seznam je seznam predmetov s podatkom `J` za jezikoslovni predmet in `N` za naravoslovnega.

Poišči med učenci tiste, ki se zanimajo za jezikoslovne predmete (**jezikoslovci**) in tiste, ki so jim bolj všeč naravoslovni predmeti (**naravoslovci**).

```
#!/usr/bin/env python3

ucenci = {('Maja','Ženski','Angleščina'),
          ('Igor','Moški','Fizika'),
          ('Olga','Ženski','Matematika'),
          ('Marija','Ženski','Kemija'),
          ('Sonja','Ženski','Slovenščina'),
          ('Mojca','Ženski','Fizika'),
          ('Ivan','Moški','Ruščina'),
          ('Marjan','Moški','Kemija'),
          ('Simon','Moški','Kemija')}

predmeti = {('Slovenščina','J'), ('Ruščina','J'), ('Angleščina','J'),
            ('Matematika','N'), ('Fizika','N'), ('Kemija','N')}
```

```
jezikoslovje = {x for x in predmeti if x[1] == 'J'}
naravoslovje = {x for x in predmeti if x[1] == 'N'}
jezikoslovci = {x for x in ucenci if x[-1] in [x[0] for x in jezikoslovje]}
naravoslovci = ucenci - jezikoslovci
print(jezikoslovci, naravoslovci)
```

```
{('Sonja', 'Ženski', 'Slovenščina'),
 ('Ivan', 'Moški', 'Ruščina'),
 ('Maja', 'Ženski', 'Angleščina')}
{('Marjan', 'Moški', 'Kemija'),
 ('Marija', 'Ženski', 'Kemija'),
 ('Igor', 'Moški', 'Fizika'),
 ('Olga', 'Ženski', 'Matematika'),
 ('Simon', 'Moški', 'Kemija'),
 ('Mojca', 'Ženski', 'Fizika')}
```

17.4 Slovar ocen učencev

Izločite tiste, ki so dosegli oceno več kot 3.

```
#!/usr/bin/env python3
ocene = {'Marjan': 4, 'Teja': 5, 'Oskar': 5, 'Miha': 3}
pridni = [x for x in ocene if ocene[x] > 3]
```

17.5 Samoglasniki

Simon se zanima za jezike. Veliko je prebral o kulturi v starem Egiptu, o njihovi pisavi in jeziku. Celo egipčanskih hieroglifov se je lotil. Presenečen je bil, ko je prvič slišal, da samoglasnikov običajno niso zapisovali. Enako velja za sorodna semitska jezika arabščino in hebrejščino. Sprva si ni mogel predstavljati, da soglasniki hranijo dovolj informacije, da se besedilo lahko razume, če pri pisanju izpuščamo samoglasnike. Zelo ga zanima, kako je s slovenščino. Pomagajte mu.

```
besedilo = '''
V jesenski tihi čas prileti brinovka na Kras.
Na polju že nikogar več ni, le ona preko gmajne leti.
In samo lovec ji sledi ...
Strel v tišino; droben curek krvi;
brinovka obleži, obleži.'''

aeiou=['a','e','i','o','u', 'A','E','I','O','U']
def primer1(besedilo):
    besedilo = list(besedilo)
    i=0;
    while i < len(besedilo):
        if besedilo[i] in aeiou:
            del besedilo[i]
        else:
            i+=1
    return ''.join(besedilo)
```

```
def primer2(besedilo):
    for x in aeiou:
        besedilo = besedilo.replace(x, '')
    return besedilo

print(primer1(besedilo))
```

```
V jsnsk th čs prlt brnkv n Krs.
N plj ž nkgr vč n, l n prk gmjn lt.
n sm lvc j sld ...
Strl v tšn; drbn crk krv;
brnkv blž, blž.
```

17.6 Dešifriranje besedila

Dobili ste besedilo je šifrirano s Cezarjevo šifro. Pričakujete, da je besedilo šifrirano skupaj s presledki. Ker so presledki v besedilu najpogostejši, lahko ugotovite, pod katero črko se skriva presledek. S primernim zamikom črk v abecedi dobite nazaj prvotno besedilo.

Besedilo

```
besedilo = '''gšjožvskožmšshmjhdsfskeclcske  
šožsmčžsthfhsgščzecsihmsdčžjsgčžskcghocskcs  
ijhkmcsöhečhsožjhscgsihkmšož'''  
besedilo = ' '.join(besedilo.split('\n'))  
  
f = freq(besedilo)  
ch = max(list(f.items()), key = lambda x: x[-1])  
n = abeceda.index(ch[0])  
  
print(ch)  
print(n)  
print(cesar_code(code, -n - 1))
```

```
('s', 17)  
18  
narveč sveta otrokam sliši slave tje bomo najdli pot kjer  
nje sinovi si prosti voljo vero in postave
```

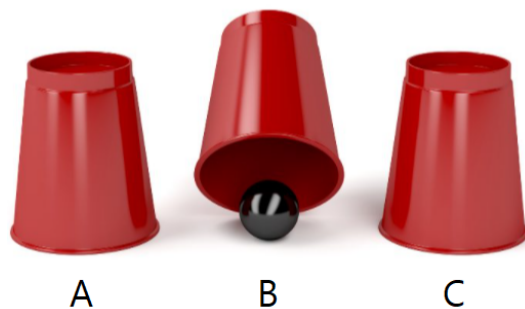
Najpogostejša črka je bila 's'. Črka se je pojavila v besedilu 17. krat. Njeno mesto v abecedi je 18. Zamik je negativen -19 , absolutno za eno večji kot je mesto v abecedi.

Funkciji `cesar_kode` in `freq` se nahajajo v nalogah Cezarjeva šifra in pogostost črk v besedilu.

17.7 Najdi kroglico

Na mizi se nahajajo tri čaše.

Položaje čaš označimo z A, B in C. Kroglica se na začetku nahaja pod čašo B.



Čaše menjajo položaje. Menjavo položaja označimo s parom črk, na primer AB pomeni, da čaši na položajih A in B zamenjata svoji mesti. Zamenjava AB pomeni isto kot BA.

Napišite funkcijo, ki sprejema seznam parov črk A, B in C in vrne položaj kroglice na koncu.

Primer:

```
cup_swapping(["AB", "CA", "AB"]) --> "C"
```

```

def cup_swapping(lst):
    l = {'A': 0, 'B': 1, 'C': 0}
    for st in lst:
        l[st[0]], l[st[1]] = l[st[1]], l[st[0]]
    return {i for i in l if l[i] == 1}.pop()

print(cup_swapping(["BA", "AC", "CA", "BC"]))

```

A

17.8 Ena črka več

Podana sta dva niza, ker je drugi dobljen, tako da prvemu dodamo eno naključno črko. Poiščite katera črka je ta. Vrstni red črk v nizih je naključni.

Napišite funkcijo z imenom `find_the_difference(p, q)`, ki sprejeme dva niza malih črk angleške abecede, ker ima drugi niz dodano eno črko. Funkcija na vrne to odvečno črko.

```

find_the_difference('abcd', 'acbed') --> 'e'
find_the_difference('abcd', 'adcbd') --> 'd'
find_the_difference('', 'y') --> 'y'

```

```

def find_the_difference(p, q):
    ql = list(q)
    for ch in p:
        ql.remove(ch)
    return ql[0]

```

17.9 Bankovci

Račun $R \in$ je bil plačan z N bankovci po 20 € in 50 €. Zanima nas, ali je mogoče ugotoviti, koliko je bilo bankovcev za 20 € in koliko je bilo bankovcev za 50 €.

Funkcija `bankovci` sprejema dva argumenta n število bankovcev in r račun, ter vrne dve celi števili `euro20` in `euro50`, ki povesta koliko je bilo bankovcev za 20 in 50 evrov. Če problem nima rešitve, vrnite obe števili (0, 0).

```

def solve(n, r):
    k = [2, 5]
    if r % 10 == 0: r //= 10
    for i in range(n + 1):
        j = n - i
        if k[0] * i + k[1] * j == r: return (i, j)
    return (0, 0)

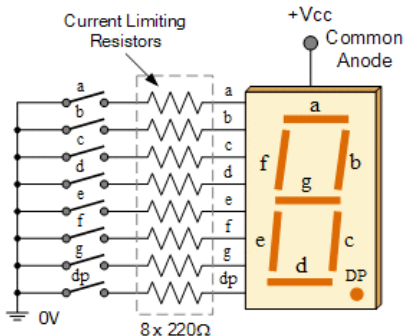
```

```

if __name__ == '__main__':
    s = input('število N račun R --> ')
    s = [int(x) for x in s.split()]
    euro20, euro50 = solve(*s)
    print( 'euro 20 {0:d}, euro 50 {1:d}'.format(euro20, euro50))

```

17.10 Sedem segmentni prikazovalnik



Segmenti so označeni od **a** do **g**. Na prikazovalniku se izpišejo številke od **0** do **9**.

Številko se izpiše tako, da se na prikazovalniku zaporedno izpisujejo številke ena za drugo. Na prehodu se ugasnejo segmenti, ki ne pripadajo dani številki in se prižgejo manjkajoči segmenti, tako da je prehod izpisa ene številke do izpisa naslednje številke čimbolj gladek.

Funkcija sprejema zaporedje števk in za vsak prehod med eno in drugo številko izpiše niz s spremembami stanja segmentov. Če se segment prižge, v nizu nastopa velika tiskana črka ustreznega segmenta. Če pa se segment ugasne, potem se to označi z ustrezno malo tiskano črko.

```

def seven_segment(s):
    digits = {'0':'abcdef', '1':'bc', '2':'abdeg', '3':'abcdg', '4':'bcfg',
             '5':'acdfg', '6':'acdefg', '7':'abc', '8':'abcdefg', '9':'abcfg', ' ':''}
    def transition(s):
        return ''.join(sorted(list(set(digits[s[0]]) - set(digits[s[1]]))) +
                        '\n'.join(sorted(list(set(digits[s[1]]) - set(digits[s[0]])))).upper()
    l2 = [s[i:i+2] for i in range(len(s)-1)]
    res = [digits[s[0]].upper()]
    for s2 in l2:
        res.append(transition(s2))
    return res

print(seven_segment(' 8055 '))

```

```
[' ', 'ABCDEFG', 'g', 'beG', '', 'acdfg']
```

18 Naloga na vajah

18.1 Le tri in pet

Ali je mogoče, če začnemo zaporedje s tri ali pet, z dvema operacijama

- prištejemo pet,
- množimo s tri,

doseči dano ciljno vrednost n

Primeri

14 = 3*3 + 5 -> True

25 = 5+5+5+5+5 -> True

7 -> False

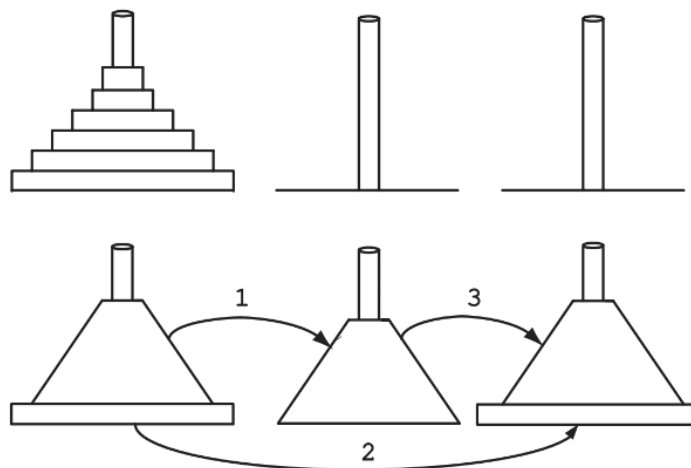
```
only_3_and_5
```

```
def can_reach_target(start, target):  
    pass  
  
if __name__ == '__main__':  
    print([x for x in range(1,100) if not only_5_and_3(x)])
```

```
[1, 2, 4, 6, 7, 11, 12, 16, 17, 21, 22, 26,  
31, 36, 41, 46, 51, 56, 61, 66, 71, 76]
```

<https://trinket.io/python3>

18.2 Hanojski stolpi



<https://trinket.io/python/c11837e8c3>

Na tri palice, imenujmo jih **A**, **B** in **C**, se natikajo okrogle ploščice različnih premerov, ki imajo na sredini luknjo.

Na začetku so ploščice natakne na palico **A**. Urejene od tal do vrha v vrstnem redu od največje do najmanjše, tako ima kup obliko stožca. Cilj igre je prestaviti ploščice na palico **C**, pri tem si pomagamo z začasnim odlaganjem ploščic na palico **B**.

18.3 Maya

Dopolni s funkcijo, `maya2dec`.

<https://trinket.io/python3/8ed0467637>

<https://www.youtube.com/watch?v=VmZjPVT2M20>

Po navodilih v filmu sestavi funkcijo seštevanja.

Maya add

```
def simplify(z):
    i = len(z) - 1
    while i > 0:
        one_y = z[i].count('o')
        five_y = z[i].count('-')
        if one_y > 4:
            one_y -= 5
            five_y = five_y + 1
        if five_y > 3:
            five_y -= 4
            z[i-1] = 'o' + z[i-1]
        z[i] = one_y * 'o' + five_y * '-'
        i -= 1
    return ' '.join([t if len(t) > 0 else '@' for t in z])

def maya_add(x: str, y: str) -> str:
    xl = [t.replace('@', '') for t in x.split(' ')]
    yl = [t.replace('@', '') for t in y.split(' ')]
    if len(yl) < len(xl):
        xl, yl = yl, xl
    xl = (len(yl) - len(xl)) * [''] + xl
    sl = [''] + [''.join(sorted(a[0] + a[1],\
        reverse=True)) for a in zip(xl, yl)]
    return simplify(sl)
```

18.4 Rimske številke

Števila od 1 do 9:	I, II, III, IV, V, VI, VII, VIII, IX.
Desetice od 10 do 90:	X, XX, XXX, XL, L, LX, LXX, LXXX, XC.
Stotice 100 do 900:	C, CC, CCC, CD, D, DC, DCC, DCCC, CM.
Tisočice od 1000 do 5000:	M, MM, MMM.

<https://trinket.io/python3/716ccaf797>

Objekt rimske številke. Vsota rimskih številke:
<https://trinket.io/python3/1c6153a86d>

18.5 Koda TAP

Koda TAP ali koda s trkanjem je način sporočanja s trkanjem črk sporočila. Ta način sporočanja so uporabljali zaporniki, ki so si izmenjevali sporočila tako, da so trkali po steni celice. Črke so razporejene v kvadrat 5×5 . Vsaka črka je podana z dvema zaporedjema trkov, ki sta ločeni z dvojnimi premorom. V našem primeru trke označimo s piko in dvojni premor s presledkom. Prvo zaporedje vsebuje toliko pik, kot je številka vrstice v tabeli, kjer se nahaja črka. Drugo zaporedje pik pa pove, v katerem stolpcu je črka. Kodiramo posamezne besede, ki so izpisane z velikimi tiskanimi črkami. Male tiskane črke pretvorimo v velike tiskane. Napišite funkcijo `tap(s)`, ki vrne zakodirano besedo, če je ta izpisana s črkami. Besede, izpisane s pikami, izpiše v dekodirani obliki z velikimi tiskanimi črkami.

```
  1 2 3 4 5
1 A B C Č D      . .      . .      . . . .      . . . . .      . . . . .
2 E F G H I      . . .      . . .      . . . .      . . . . .      . . . . .
3 J K L M N      . . . .      . . . .      . . . .      . . . . .      . . . . .
4 O P R S Š      . . . . .      . . . . .      . . . . .      . . . . .      . . . . .
5 T U V Z Ž      . . . . .      . . . . .      . . . . .      . . . . .      . . . . .
```

<https://trinket.io/python3/5afe272ac4>

18.6 Koda DIF

Koda **DIF** kodira besedilo tako, da dano črko nadomesti s črko, katere mesto v abecedi je enako razliki med mesti v abecedi dane črke in njej predhodne črke v besedilu. Razliko računamo po modulu dolžine abecede.

Nastavek naloge je na naslovu:

<https://trinket.io/python3/0b3d607384>

https://sl.wikipedia.org/wiki/Frekvence_črk

18.7 Dekodiranje besedila

Dobili ste besedilo je šifrirano s Cezarjevo šifro. Pričakujete, da je besedilo šifrirano skupaj s presledki. Ker so presledki v besedilu najpogostejši, lahko ugotovite, pod katero črko se skriva presledek. S primernim zamikom črk v abecedi dobite nazaj prvotno besedilo.

<https://trinket.io/python3/76a4e7a237>

18.8 Naravna števila

Prazna množica $\phi = \{\}$ nima elementov, vendar pa ima množica $\{\phi\}$ natanko en element.

Konstrukcija množice naravnih števil:

1. $\phi = \{\}$ je prazna množica
2. $1 = \phi \cup \{\phi\} = \{\phi\} = \{\{\}\}$
3. $2 = 1 \cup \{1\} = \{\phi, \{\phi\}\} = \{\{\}, \{\{\}\}\}$

4. $n' = n \cup \{n\}$, kjer je število n' je naslednik števila n , $(n + 1)$.

<https://trinket.io/python3/02dc7bab2d>

18.9 Bisekcija

Algoritem:

- Podana je funkcija $f(x)$ in interval $[a, b]$.
- Vrednosti funkcije v krajiščih intervala sta različno predznačeni. $f(a)f(b) < 0$
- Ponavljamo v zanki:
 - Razpolovimo interval. Razpolovišče je točka $c = (a + b)/2$
 - Če je $f(a)f(c) < 0$ potem postane $b = c$,
 - če je $f(c)f(b) < 0$ potem postane $a = c$,
 - sicer pa je $f(c) = 0$ in vrnemo točno rešitev $x = c$
- Ponavljamo dokler ni vrednost $|f(c)|$ dovolj majhna.
- Na koncu vrnemo približno rešitev $x = c$.

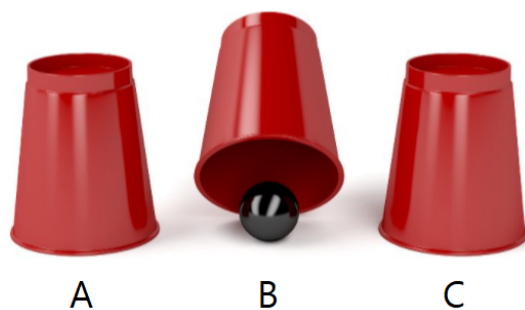
<https://trinket.io/python3/43008b405b>

<https://trinket.io/python3/b617e70377>

18.10 Najdi kroglico

Na mizi se nahajajo tri čaše.

Položaje čaš označimo z **A**, **B** in **C**. Kroglica se na začetku nahaja pod čašo **B**.



Čaše menjajo položaje. Menjavo položaja označimo s parom črk, na primer **AB** pomeni, da čaši na položajih **A** in **B** zamenjata svoji mesti. Zamenjava **AB** pomeni isto kot **BA**.

Napišite funkcijo, ki sprejema seznam parov črk **A**, **B** in **C** in vrne položaj kroglice na koncu.

Primer:

```
cup_swapping(["AB", "CA", "AB"]) --> "C"
```

<https://trinket.io/python3/99fbb02e5f>

<https://www.youtube.com/watch?v=5aAa6Icy8lw>

18.11 Ena črka več

Podana sta dva niza, ker je drugi dobljen, tako da prvemu dodamo eno naključno črko. Poiščite katera črka je ta. Vrstni red črk v nizih je naključni.

Napišite funkcijo z imenom `find_the_difference(p, q)`, ki sprejeme dva niza malih črk angleške abecede, ker ima drugi niz dodano eno črko. Funkcija na vrne to odvečno črko.

```
find_the_difference('abcd', 'acbed') --> 'e'  
find_the_difference('abcd', 'adcbd') --> 'd'  
find_the_difference('', 'y') --> 'y'
```

<https://trinket.io/python3/471ec128b9>