Laboratorul 2 - probleme rezolvate

Problema #1 l laboratorul #2

Se citesc de la tastatura, de pe cate o linie, trei numere naturale z, l, a, reprezentand ziua, luna si anul unei date calendaristice. Sa se afiseze data zilei urmatoare, in formatul zi.luna.an.

Reamintim ca un an este bisect daca:

- este divizibil cu 4 si nu este divizibil cu 100 sau
- este divizibil cu 400

Rezolvarea problemei #1 l laboratorul #2

```
# Solutia problemei #1 I laboratorul #2

z = int(input("z: "))
l = int(input("l: "))
a = int(input("a: "))

# Cream o lista care sa functioneze astfel:
# zile_in_luna[n] = cate zile are luna n
# Intervine problema: listele sunt indexate (incep de la) pozitia 0
# Pentru a rezolva asta, punem o valoare aleatoare la inceput, pe pozitia 0
zile_in_luna = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

# Verificam daca anul este bisect.
if (a % 4 == 0 and a % 100 != 0) or (a % 400 == 0):
    zile_in_luna[2] = 29

nr_zile_luna_curenta = zile_in_luna[l]

if z < nr_zile_luna_curenta:            # verificam daca suntem in ultima zi din luna s
au nu
    z_urm = z + 1                       # nu suntem, deci nu va trebui sa trecem in lun
a urmatoare
    l_urm = l
    a_urm = a                           # cum nu trecem in luna urmatoare, nu avem cum
sa trecem nici in anul urmator
else:                                   # aici vom fi in ultima zi din luna, deci ziua
urmatoare va fi 1 din luna urmatoare
    z_urm = 1
    if l == 12:                         # daca luna actuala e ultima din an, trecem ina
poi la prima luna
        l_urm = 1
        a_urm = a + 1                   # crestem si anul
    else:                               # daca luna actuala nu e ultima din an doar adu
nam 1 la luna si lasam anul la fel
```

```
        l_urm = l + 1
        a_urm = a

print(f"Data zilei urmatoare este: {z_urm}.{l_urm}.{a_urm}")
```

Problema #1 II laboratorul #2

Fie x un numar natural, citit. Sa se verifice daca x este:

- numar perfect (egal cu suma divizorilor lui, mai putin numarul insusi, ex: 6, 28, 496, 8128);
- palindrom (este egal cu oglinditul sau)

Rezolvarea problemei #1 II laboratorul #2

Pentru primul subpunct, vom folosi un `for` care va determina divizorii lui `x` si ii va adauga la suma finala.

Pentru rezolvare, am folosit rezultatul matematic urmator:

$$\text{Fie } n \in \mathbb{N},\ n \geq 2. \text{ Atunci:}$$
$$n \text{ este compus } \iff \exists\, d \in \mathbb{N},\ 2 \leq d \leq [\sqrt{n}] \text{ cu } d \mid n.$$

Corolar: e suficient sa mergem pana la $[\sqrt{n}]$ pentru a cauta divizori.

Pentru al doilea subpunct, folosim operatiile urmatoare pe cifrele unui numar natural:

- `x % 10` : restul impartirii lui x la 10; rezultatul este **ultima cifra** a lui x
- `x // 10` : partea intreaga a impartirii lui x la 10, care ne da acelasi numar x, dar **fara ultima cifra** ("taie" ultima cifra)
- asemanator, `x % 100` ne va da ultimele doua cifre si `x // 1000` va "taia" ultimele 3 cifre

```
# Solutia problemei #2 laboratorul #2
import math

x = int(input("x: "))
if x <= 1:
  print("{x} nu este numar perfect")
else:
  suma_div = 1
  r = math.isqrt(x)

  for i in range(2, r + 1):       # i va lua valori de la 2 la [radical din x] inclusiv
. [] este partea intreaga
      if x % i == 0:              # daca x se imparte la i => i e divizor => si x/i e d
ivizor
          j = x // i              # j este acel al doilea divizor adica x/i
          if i == j:              # daca sunt egale: i == j => i^2 = x => x e patrat pe
rfect
              suma_div += i       # x e patrat perfect, deci adaug doar i, o data, si n
u i + j pentru ca as adauga i de doua ori la suma (cum i=j)
          else:
              suma_div += i + j   # adaug ambii divizori la suma

  print(f"{x} este numar perfect (True/False): {suma_div == x}")
```

```python
# Este x palindrom?

oglindit = 0
copie_x = x     # pastrez valoarea lui x aici deoarece voi schimba mai jos variabila x

while x != 0:                       # echivalent cu "while x"
  uc = x % 10                       # uc = ultima cifra (x % 10 este restul impartirii lu
i x la 10, care imi da ultima cifra)
  oglindit = oglindit * 10 + uc   # "lipesc" ultima cifra a lui x la finalul lui oglin-
dit
  x //= 10                          # "tai" ultima cifra din x (ex: 1245 // 10 = 124)

print(f"{copie_x} este palindrom (True/False): {copie_x == oglindit}")
```

Problema #2 II laboratorul #2

Fie n numar natural. Sa se afiseze:

- $n!$
- toate numerele prime mai mici decat n
- primele n numere naturale prime
- primele n patrate perfecte
- toate patratele perfecte mai mici decat n

```python
n = int(input("n: "))

# Factorial "de mana"
factorial = 1
for i in range(1, n + 1):
  factorial *= i

print(f"factorial({n}) = {factorial}")

# Factorial cu biblioteca math
import math
print(f"factorial({n}) = {math.factorial(n)}")

# Toate numerele prime mai mici decat n
# Vom folosi o metoda ineficienta, dar limpede de inteles.
# Pentru eficienta, folositi ciurul lui Eratostene.
prime_mai_mici_decat_n = []
for i in range(2, n):
  for j in range(2, math.isqrt(i) + 1):
    if i % j == 0:
      break
  else:
    prime_mai_mici_decat_n.append(i)

print(f"Numerele prime mai mici decat {n}: ", *prime_mai_mici_decat_n) # * face operati
a de unpacking
print(f"Numerele prime mai mici decat {n}: {prime_mai_mici_decat_n}")  # fara unpacking
 ca sa vedeti diferenta
```

```python
# Primele n numere naturale prime
primele_n_prime = []
nr_de_verificat = 2
if n != 0 and n != 1:
  while len(primele_n_prime) < n:
    # Verificam daca nr_de_verificat e prim
    for i in range(2, math.isqrt(nr_de_verificat) + 1):
      if nr_de_verificat % i == 0:
        break
    else:
      primele_n_prime.append(nr_de_verificat)
    nr_de_verificat += 1

print("Primele %d numere naturale prime: %s" % (n, primele_n_prime))


# Primele n patrate perfecte
primele_n_patrate = []
for i in range(n):
  primele_n_patrate.append(i ** 2)

print(f"Primele {n} patrate perfecte:", *primele_n_patrate)


# Toate patratele perfecte mai mici decat n
i = 0
patrate_mai_mici_decat_n = []
while i * i < n:
  patrate_mai_mici_decat_n.append(i * i)
  i += 1
print(f"Patratele perfecte mai mici decat {n}:", *patrate_mai_mici_decat_n)
```

Problema #3 II laboratorul #2

Se citeste un sir de cel putin 3 valori intregi pana la intalnirea lui 0. Sa se afiseze:

a., b. nu le voi rezolva (daca aveti nevoie de ajutor la ele ne puteti intreba)

c. numarul valorilor de tip deal, vale (citeste pdf cu cerinte pt. explicatie)

Rezolvarea problemei #3 II laboratorul #2

Exemplu pt. c:

INPUT: 3 9 10 -1 3 4 5 4 2 0

OUTPUT: 2, 3 (10, 5 sunt dealuri; 3, -1, 2 sunt vai)

```python
# c.

nr_dealuri = 0
nr_vai = 0
```

```python
a = int(input())
b = int(input())
c = int(input())

# Verificam daca prima valoare citita (a) este deal sau vale (sau niciuna)
if a > b:
  nr_dealuri += 1
elif a < b:
  nr_vai += 1

# Citim noi numere cat timp c nu e 0
while c != 0:
  if b > a and b > c:       # Verificam daca b este deal
    nr_dealuri += 1
  elif b < a and b < c:     # Verificam daca b este vale
    nr_vai += 1
  a = b
  b = c
  c = int(input())

# La finalizarea loop-ului while, c va fi 0
# Verificam ultima valoare nenula citita (b-ul)
if b > a:
  nr_dealuri += 1
elif b < a:
  nr_vai += 1

print(f"Dealuri: {nr_dealuri}; Vai: {nr_vai}")
```

Problema #4 II laboratorul #2

Sa se genereze toate submultimile multimii $A = \{1, 2, \ldots, n\}$ unde numarul natural nenul n se citeste de la tastatura (fara backtracking - folosind scrierea binara a unui numar si operatii pe biti).

Exemplu:

INPUT: n = 3

OUTPUT: {}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3} afisate in orice ordine.

Rezolvarea problemei #4 II laboratorul #2

Stim ca trebuie sa folosim reprezentarea in baza doi.

Q: Cum Dumnezeu ajungem de la biti la submultimi?

Simplu, deoarece noua ne trebuie un set foarte particular de submultimi: cele cu numere naturale pana la n.

Hai sa zicem ca ne luam 4 biti. Putem forma numere precum:

$$1111, 1010, 0011, 0110 \text{ etc.}$$

A: Codificam o submultime dupa pozitiile***** bitilor egali cu 1 din numarul nostru pe 4 biti. Iata:

$$\text{daca avem } \underline{1}0\underline{11} \rightarrow \{1,2,4\}$$

*****: De fapt, pozitiile acelor biti sunt 0, 1, 3 (incepe numerotarea de la 0), dar nu ne injura nimeni daca ne imaginam ca se numara incepand cu 1.

Explicatie: bitii de pe pozitiile 1, 2 si 4 sunt egali cu 1 deci multimea pe care o codifica este {1,2,4}. Alte exemple:

$$\begin{gathered} 000\underline{1} \rightarrow \{1\} \\ \vdots \\ \underline{1}0\underline{1}0 \rightarrow \{2,4\} \\ \vdots \\ \underline{1111} \rightarrow \{1,2,3,4\} \end{gathered}$$

```python
n = int(input("n: "))

nr_submultimi = 1 << n      # echivalent cu 2 ** n

# Iteram prin toate numerele de la 0 la 2^n - 1
# Fiecare i reprezinta o submultime
for i in range(nr_submultimi):
  submultime = []
  # Verificam fiecare bit al numarului i de la 0 la n-1
  for j in range(n):
    # Daca bitul de pe pozitia j din numarul i este 1
    if (i >> j) & 1:
      submultime.append(j + 1)
  print(f"{set(submultime)}")
```