Programare procedurală

AFIȘARE

* print( ) → separate prin spațiu, enter la final
  - variabile, parametrii (scrise separate prin virgulă)
  - sep = " " → separator
    - caracter → poate fi vid
  - end = " " → finalul afișării
    - caracter → poate fi vid ⇒ nu mai pune enter

* print(type(variabilă)) ⇒ tipul variabilei
* print(id(variabilă)) ⇒ locul unde se află în memorie
* print(" ")
  - ↳ afișează fix ce este între ghilimele
* print(f" { } ")
  - ↳ variabile, operații pe care le implementezi, le calculează
  - ↳ nu modifică, doar afișează
* print(" " + " ") ⇒ concatenare de șiruri

CITIRE

* input( )
  - ↳ primește date de la tastatură, implicit ele vor fi șiruri de caractere ⇒ am nevoie de funcții care să transforme datele în tipul de dată de care am nevoie, ex: int( ), float( )
* input(" ")
  - ↳ mesaj pe ecran, de ex: introduceți un număr

ATRIBUIRI

x = 5 ↳ x arată spre obiectul din memorie cu val 5
nu trb. declarată anterior

x, y = 1, 2 ⇒ atribuire de tupluri ⇔ x = 1 și y = 2

OPERAȚII ALGEBRICE

OBS! operațiile cu float nu au precizie absolută

```
x = 0.1
print (x*x == 0.01)  => FALSE
print( abs(x*x - 0.01) < 1e-8)  => True
```

abs = modul

1e-8 → $1 \cdot 10^{-8}$ => bun pt. aproximări

* a ** b <=> a^b

OBS! $\sqrt{\ }$ = math.sqrt( ), dar tb. întâi să-i dau "import math"

dar, $\sqrt{\ }$ <=> a ridica la $\frac{1}{2}$ => $\sqrt{a}$ = a ** 0.5 și pot evita import

a/b => rezultat float

a//b => partea întreagă a împărțirii, a%b => restul împ

INSTRUCȚIUNI

```
if cond:
    instrucțiuni
else:
    alte-instrucțiuni
```

```
if cond:
    instrucțiuni
elif cond2:
    instrucțiuni 2
elif cond3:
    instrucțiuni 3
else:
    instrucțiune 4
```

OBS! nu va da eroare dacă o instr. este greșită, atâta timp cât nu intră pe ramura ei.

```
while condiție:
    instrucțiuni
```

for ______ in ______ :
↓ variabilă ↓ șir

instrucțiuni pt. variabilă

for i in range(n): <=> i ∈ {0, 1 ..., n-1}
range(a,b): <=> i ∈ {a, ... b-1}
range(len(S)) <=> i ∈ {0, 1 ..., lungimea lui S-1}
range(a, b, pas) <=> i ∈ {a, a+pas, a+2pas ... }
↓ nu depășește b-1

```
for i,c in enumerate(s):
    print("poziția",i, "elementul",c)
```

OBS! În python, else este și pt. for și while și se execută dacă structura repetitivă nu s-a terminat cu break.

ex:
```
for i in range(2, n):
    if n%i == 0:
        print("nu este prim")
        break; #iese din structura repetitivă
else:
    print("este prim")
```

TIPURI DE DATE

- **numerice**
  - **int**: nr. întregi cu semn, cu precizie infinită
  - **float** (also: float("inf"), float("infinity"), float("nan")
  - **complex**: $a + bj$ / $a + bj \Rightarrow$
    - z.real() => Re(z)
    - z.imag() => Im(z)
    - z.conjugate() => $\bar{z}$
    - abs(z) => |z|
  - NoneType
  - **bool** => True / False (<=> None, 0, 0.0, 0j, Decimal(0), Fraction(0,1))
- **colecții** (obiecte care memorează mai multe valori, sub o anumită formă)
  - **secvențiale** (șiruri de val indexate de la 0)
    - **mutabile** – liste (<=> vectori)
    - **imutabile**
      - șiruri de caractere
      - tupluri
  - **multimi** (neindexate, fără duplicări)
    - mutabile
  - **dicționare** (perechi de forma cheie:valoare)
    - mutabile începând?

OPERAȚII CU SECVENȚE

* test de apartenență => operatorii: in, not in
* lungime: len(_)   OBS! len => O(1) complexitate
  ↳ nume secvență
* min(_), max(_) — nume secvență | => cel mai mare/mic element din secvență
* accesare element: s[ ] — s = nume secvență; [ ] ↳ indice -> poate fi și negativ => se ia de la capăt
* feliere: s[i:j] => subsecvența v[i], v[i+1] ... v[j-1]
  ↳ indicii pot fi negativi + pot lipsi

i lipsește => 0 implicit   ! exercițiu în powerpoint curs 3_4

j lipsește => len(s) implicit

s[:p] => prefixul de lungime p al lui s

s[-p:] => sufixul de lungime p al lui s

s[:] => toată secvența (la liste întoarce o copie, deci s[:] == s => True, dar s[:] is s => False)

s[::-1] => secvența s inversată

s[i:j:k] => secvența cu indicii {i, i+k, i+2k ... }

! k poate fi și pozitiv și negativ

* frecvența => metoda count => s.count(x, i, j) — x = element; i ↳ primul indice; j = ultimul (exclusiv)
* poziția pe care apare o valoare => metoda index => ValueError
  la șiruri -> find => -1 când nu găsește

s.index(x, i, j) => prima apariție a lui x în s începând cu indicele i până la j-1

Concatenare: „+", „*"

„+" => s = "programare" | => s+t = "programareprocedurală"
t = "procedurală" | → un obiect nou

OBS! Pentru adăugarea la finalul unei liste este recomandată metoda extend, fiind mai rapidă:

`ls.extend(ls2)` sau `ls.append(element)` ← un singur element

↳ pentru mai multe elemente (*)

„*" s = [0] * n => s = [0, 0 ... 0]

s = "a" * 4 => s = "aaaa"

Sortarea

→ sorted: `ls = sorted(ls, key=None, reverse=False)` → nu modifică pe ls

ls.sort(key = lambda x: (ce fac ex, ce fac în caz de egalitate))
↳ sort modifică obiectul

Ștergere din șir:

`s = s[:i] + s[:i+1]`

`del v[i:j]`

`s.remove(valoare)`

`s.pop(i)` → dacă nu pun i => implicit ultima valoare

(*) ls.extend("alt") → îmi pune fiecare literă ca element independent

ls.extend(["alt"]) → un singur element

ls.append("alt") → un singur element, că append merge doar pt. un element

```
try:
    x=n.index( element )
    print("Prima apariție a lui", element, "este", x)
except ValueError:
    # print("elementul", element, "nu apare in sirul n")
    pass # instructiunea nulă
```

! for i in range(len(sir)) -> reține len => dacă modific lungimea șirului în for => eroare: la să iasă din lungimea reală a șirului