



## Programare procedurală

### Afișare

- \* `print( )` → se parcurge primul argument, apoi la final
  - variabile, parametri → `sep = " "` ⇒ separator
  - (oriunde se parcurge primul singur)
  - `end = "\n"` ⇒ finalul afișării
  - caracter → poate fi videlicet
  - => mult mai puțin ambi
- \* `print(type(varialo))` ⇒ tipul variabilei
- \* `print(id(varialo))` ⇒ locul unde se află în memorie
- \* `print(" ")`
  - afisează fix cu entă impreună ghilimele
- \* `print(f" { } ")`
  - variabile, separator pe care îl împarteste, și calculează
  - nu modifică, doar afișează
- \* `print(" " + " ")` ⇒ concatenarea de stringuri

### Citire

- \* `input()`
  - primeste date de la tastatura, împărțește varii în funcție de caractere ⇒ oarecă memoria doar conține datele în tipul doar de cărți
  - în memoria, ex: `int()`, `float()`
- \* `input(" ")`
  - menajă pe ecran, de ex: introducând membru

### Atribuirile

`x = 5` → x este noul obiectul din memorie cu val 5  
 nu înlocuiește anteriorul

`x, y = 1, 2` ⇒ atribuire de tuplu  $\Leftrightarrow x = 1 \text{ și } y = 2$

## OPERATÙ ALGEBRICE

OBS! operatori cu float nu are precizia absolută

$$x = 0.5$$

print ( $x + x == 0.02$ )  $\Rightarrow$  FALSE

print ( $\text{abs}(x + x - 0.02) < \underbrace{1e-8}_{\text{modul}})$   $\Rightarrow$  True

modul

$\hookrightarrow 1 \cdot 10^{-8} \Rightarrow$  num pt. aproximare

$$\sqrt{a} \approx a^{0.5}$$

OBS!  $\sqrt{\phantom{x}}$  = math.sqrt(), din lib. imbrui no-i sau "import math"  
daca,  $\sqrt{a} \approx a$  ridicat la  $\frac{1}{2}$   $\Rightarrow \sqrt{a} = a \times 0.5$  mi pot evita import

a $^{0.5}$   $\Rightarrow$  rezultat float

a $^{0.5}$   $\Rightarrow$  parțea imbrui a imprestirii ; a $^{0.5}$   $\Rightarrow$  rezultat imp

## 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 oimbr.

este operatō,

dacă lampa este

nu imbrui po

termura ei.

while condiție:

instrucțiuni

for    im   :

↓      ↓  
variable    mir

im    medium pt.    variable

for i im range (m):  $\Leftrightarrow i \in \{0, 1, \dots, m-1\}$

range (a, b):  $\Leftrightarrow i \in \{a, \dots, b-1\}$

range (len(S)):  $\Leftrightarrow i \in \{0, 1, \dots, \text{lungimea lui } S-1\}$

range (a, b, pas):  $\Leftrightarrow i \in \{a, a+pas, a+2pas, \dots\}$

mu depende b-1

for i, c im enumerate (s):

print ("pozitia ", i, "elementul ", c)

oBS! În python, else este pt. folos în while nu în ca execută de către structura repetitivă și a termină cu break.

ex: for i im range (2, m):

: if mi.i == 0:

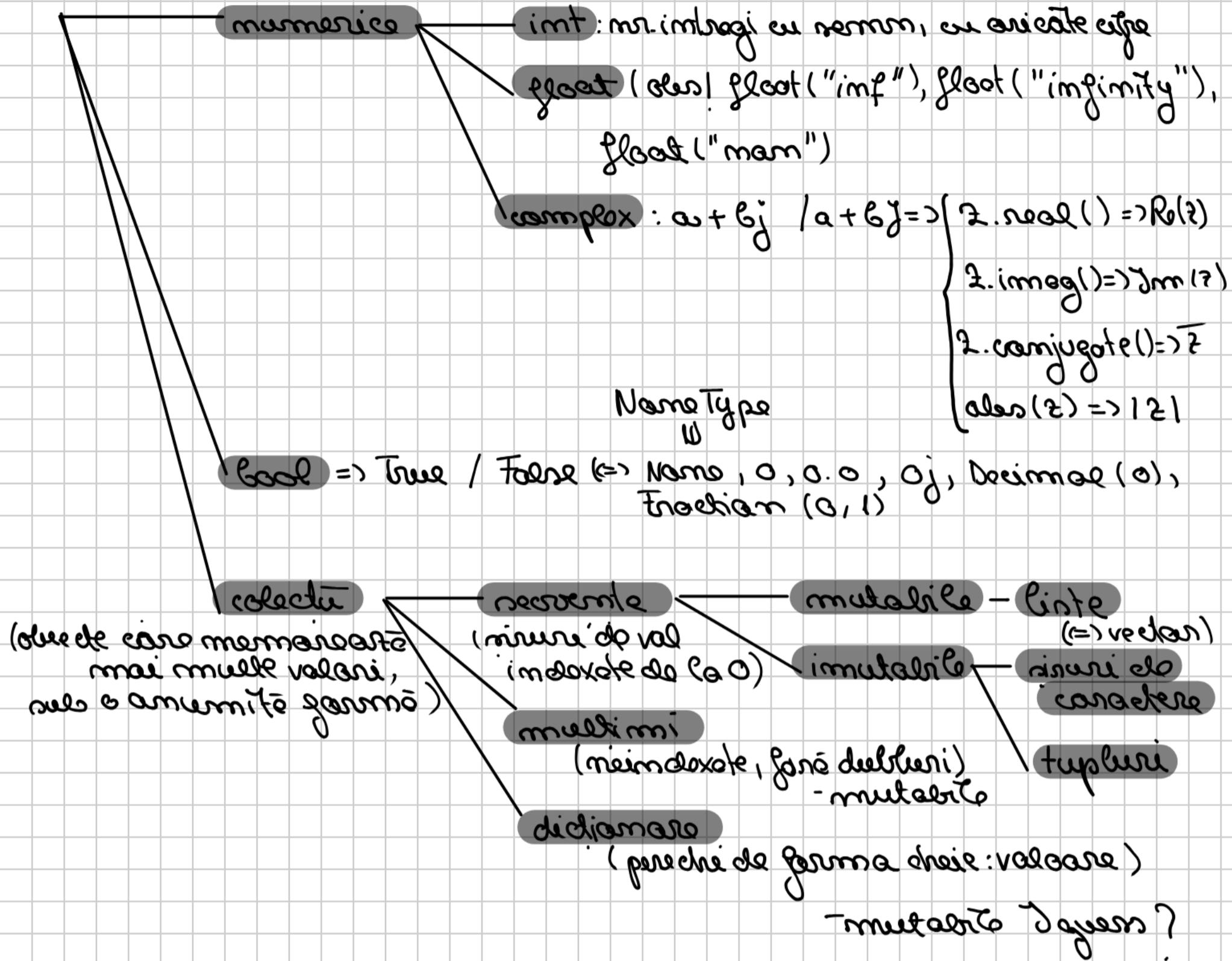
:     print ("mi este prim")

:     break; #iese din structura repetitivă

: else:

:     print ("nu este prim")

# TIPI DE DATE



## OPERATII CU SECVENTE

- \* fapt de apartenență  $\Rightarrow$  operanță: im, met im
- \* lungime:  $\text{len}()$   $\hookrightarrow$  număr elementă
- \*  $\min()$ ,  $\max()$   $\hookrightarrow$  cel mai mare/mic element din secvență
- \* număr element:  $\sum_{i=1}^n 1$

\* accesarea element:  $\text{a}[i]$   
 $\downarrow$  indice  $\rightarrow$  poate fi și negativ  $\Rightarrow$   
 se ia doar capăt  
 număr elementă

\* felicie:  $n[i:j] \Rightarrow$  subsecvență  $v[i], v[i+1], \dots, v[j-1]$   
 $\hookrightarrow$  indice pot fi negativi și lipsiți

i lipsente  $\Rightarrow$  implicit

! exemplu în powerpoint  
 curs 3-4

j lipsente  $\Rightarrow \text{len}(n)$  implicit

$n[:p]$   $\Rightarrow$  prefixul de lungime p al lui n

$n[p:]$   $\Rightarrow$  sufixul de lungime p al lui n

$n[2:3] \Rightarrow$  două elemente (la linie întoarsă și copie, deoarece)

$n[2:2] = n \Rightarrow$  True, dar  $n[2:2]$  în n  $\Rightarrow$  False

$n[1:-1]$   $\Rightarrow$  secvență n inversată

$n[i:j:k]$   $\Rightarrow$  secvență cu indice  $\{i, i+k, i+2k, \dots\}$

$\downarrow k$  poate fi și pozitiv și negativ

element  
 $\downarrow$  ultimul  
 $\uparrow$  primul indice

\* frecvență  $\Rightarrow$  metoda count  $\Rightarrow n.\text{count}(x, i, j)$  (exclusiv)

\* pozitie pe care apare o valoare  $\Rightarrow$  metoda index  $\Rightarrow$  ValueError

pozitie  $\rightarrow$  limită  $= -1$  cănd nu există

$n.\text{index}(x, i, j) \Rightarrow$  prima apariție a lui x în n începând

cu indicele i până la j-1

Concatenare : „+”, „\*”

„+”  $\Rightarrow$   $n = \text{"preprogramare"}$  |  $\Rightarrow \underbrace{n+t}_{\text{un obiect nou}} = \text{"preprogramareprogramarea"}$

$t = \text{"programarea"}$

“un obiect nou”

OBS! Pările obișnuite să simbolizeze linie este recomandată metoda extensă, fiind mai rapidă: (un singur element)

$ls.\text{extend}(lsL)$  sau  $ls.\text{append}(\text{element})$

↳ permite mai multe elemente (\*)

„\*”  $n = [0] + n \Rightarrow n = [0, 0 \dots 0]$

$n = "a" + h \Rightarrow n = "aaea"$

## Sortare

~) sortie:  $ls = ls.\text{sort}(\text{ls}, \text{key} = \text{None}, \text{reverse} = \text{False})$

↓

nu modifică pe ls

ls.sort( key = lambda x: (ceva care x, ca să e în ordine de egalitate))

modifică obiectul

Stergerea din listă:  $n = n[0:i] + n[i+1:n]$

del n[i:j]

n.remove(value)

n.pop(i) -> dacă nu pune i  $\Rightarrow$  implicit ultimul valoare

(\*) ls.extinct("all") ~) înni pune fiecare literă ca element independent

ls.extinct(["all"])  $\rightarrow$  un singur element

ls.opere("all")  $\rightarrow$  un singur element, preștează  
merig doar pt. un element

try:

0

x = n.index(element)

print("Prima oportura a lui", element, "este", x)

except ValueError:

# print("elementul", element, "nu apare im interval")

pass # instrucțiunea nu face nimic

! for i in range(0, len(lis)) → se poate ca să nu se modifică lungimea listei în for => eroare: lista nu poate fi modificată în timp ce este iterată

se poate să se modifice lungimea listei în for => eroare: lista nu poate fi modificată în timp ce este iterată

dim lungimea sa fie & multie

## Matrize

mat = [ [ 0 for i in range(m) ] for j in range(n) ]

↳ initializare cu 0

mat = [ [ 0 ] \* m for j in range(n) ]

! Acestea sunt de fapt ca obiecte

! Dacă vrem să modific elementele din vector sau  
vectorul său index pățește atât de multă să nu se suprascră  
în ceea ce privește, trebuie să adăm să fie mult  
modificarea să fie permanentă.

Cum creșt o matrice?

a = [] → șă fie o listă de liste

for i in range(m):

    limie = [ int(x) for x in input().split() ]

    a.append(limie)

Pentru:

că în C++: for i in range(m):

    for j in range(m):

că în Python: for limie in a:

    for x in limie:

~~~~~

## Copiere

### copiere superficială

- > creare un nou obiect
  - > obiectul nou este creat
- Iarne de memorie ca obiectul original => obiect modificare asupra uneia se reflectă în pe celălalt

`r = a.copy()`

### deep copy

- > creare un nou obiect
  - > copiază recursiv referințele obiectelor înainte în copie =>
  - => cele două devin independente
- import copy**
- `r = copy.deepcopy(a)`

## Siruri de caractere

~) imutabilitatea claselor / tipul de date str

~) cod ASCII → extensia lui UNICODE => 

|                |                               |
|----------------|-------------------------------|
| ord(character) | ↳ returnează valoarea corectă |
| chr(cod)       |                               |

$n = " \sim " \Leftrightarrow n = ' \sim '$

$n = " \sim "$

↳ nu poate înținde pe mai multe linii

$x = str(123) \Rightarrow x = "123"$

$str(1+2 == 3) \Rightarrow "True"$

$n[1] = "ava" \Rightarrow$  NU este  
imutabil => Type Error

### • acurenție ESCAPE

$n = " un \n nis" \rightsquigarrow un$

nis

$n = "un caracter UNICODE \u263A" \rightsquigarrow un caracter UNICODE ☺$

$n = "Titlul este "programare" " \Rightarrow$  EROARE (nu împerechează)

grafit) ↳ cel 1: folosește ' ' sau " " " " etc.

cel 2:  $n = "Titlul este \" programare \" "$

Metodele care au ca scop efectuarea unor modificări directe asupra obiectelor, nu schimbă noul obiect.

$n-mou = n.metoda()$  ↳  $n-mou$  poate fi chiar  $n$ , în sensul că n va rezulta alt obiect din memorie

$n = n.lower() \rightsquigarrow n.inlower()$

$n = n.upper() \rightsquigarrow n.inupper()$

$n.isdigit()$

$n.isnumber()$

## Split

$r = \text{"aaa1 aaa2 aaa3"}$

OBS! Pot avea un singur separator

$r = r.split(\text{sep}, maxsplit)$  → nr. max. de impărțiri

dacă tiparești  $\Rightarrow$  implică orice combinație de caractere albe. (ex: " ")

OBS!  $r = \text{"programare sau procedură"}$

$r = r.split() \Rightarrow r = [\text{"programare"}, \text{"procedură"}]$

$r = r.split(" ") \Rightarrow r = [\text{"programare"}, \text{""}, \text{""}, \text{""}, \text{""}, \text{""}, \text{"procedură"}]$

$r = r.split(maxsplit=1)$

↳ o împărțire inversă

## Join

$r = [\text{"aaa1"}, \text{"aaa2"}, \text{"aaa3"}]$

$r = \text{"} \underset{\|}{\text{|}} \text{". join(\text{ })}$

↳ rezultat pe care îl unim  
concedele pe care le punem între următoarele mișcări, poate fi și vid ("")

$r = \text{"aaa1, aaa2, aaa3"}$

## Replace

$r-new = r.replace(old, new, count) \rightarrow$  înlocuirea primelor

count aparitii ale mișcării old cu mișcarea new

## Înțelesuri

`n.count (subiect, start, end)` → de căte ori apare subiectul în subiect, începând cu poziția start până la end - 1.

✓ nu numără do mai multe ori dacă se suprapun

## Găsirea unui element în sir

`n.index (subiect, start, end)` → returnează poziția primei aparitii, dacă nu găsește => EROARE  
↳ în permisă lînce

`n.find (subiect, start, end)` → returnează poziția primei aparitii, dacă nu găsește, întoarcă -1

`n.findIndex` → poate că nu poate să copieze pt. că nu

intervalele și vectorii nu compunemini

$$V \text{-intervale} = \{x \in \mathbb{R} \times \text{intervali}\}$$

## Stergerea unui element:

dacă  $V[i]$  → sterge el. de pe lista.

dacă  $V[i:j]$  → sterge toate bucutele de la  $[i, j)$   $\Leftrightarrow$

$$\Leftrightarrow \sum [i:j] = \sum$$

`o.pop(i)` → sterge el. de pe lista.

`o.remove(x)` → sterge prima apariție a lui x

## Încuiere de elemente

$$[i:j] = [a, b, c, d, e] \rightarrow nu\ contează\ nici\ care\ este\$$

suntem, o se pună elementele în locul celor de pe pozitii i, i+1 ... j-1

$\text{lo}[1:2] = "abc"$  -> puno għiekkor l-eksejha es-ekġġġġi kien seppaq

$\text{lo}[0:1] = "abc"$  -> preċċie puno 3 elementi im-leaf id- $\text{lo}[0:1]$

$\text{lo}[1:2] = ["abc"]$  -> il-puno eww minn qiegħi element

### Imseċċarha u mui element

$\text{lo}[i:i] = [x]$  -> puno elementi kien tħalli fu position i-

$\Leftrightarrow \text{lo.insert}(i, x)$

## Tupleuri

$t = (2)$   $\Rightarrow$  nu este tuple

$t = (2,)$   $\Rightarrow$  este tuple

nu folosesc comprehensiune pentru tuple

$f = (i^* i \text{ for } i \text{ in range}(10)) \Rightarrow \text{GENERATOR}$

$\sim$  generatoarele elementelor altorui set nu sunt de tip

$\text{print}(t) \Rightarrow 0$

$\text{print}(\text{next}(t)) \Rightarrow 1$

$\text{print}(\text{next}(t)) \Rightarrow 4$

$\text{print}(\text{sum}(t)) \Rightarrow 280 \Rightarrow$  o expresie care elementele

$ls = [2, 4]$

$t = \text{tuple}(ls) \Rightarrow t = (2, 4)$

## Multimi

$\sim$  colectie de elemente distincte, nu sunt imbricate  $0 \dots len(m) - 1$

$\sim$  elementele multimii pot fi doar de tip immutabil (+ cu hsh code)

testul de operatatori la multimea nu intingeră unui element  $\Rightarrow O(1)$

$\boxed{\circ}$   $n = \{1, 2, 7\} \Rightarrow$  multimea DAR doar vir. multime vidé  $= 1$

$n = \text{set}() \quad (n = \{\}) \Rightarrow$  vide

$ls = [1, 1, 2, 5, 7, 1, 2, 8, 4]$

$n = \text{set}(ls) \Rightarrow$  multimea elementelor din  $ls$ , Paralelne cu este garantată.

$\text{print}(n_2 | n_1) \Rightarrow$  semimulțimea  $| \text{print}(n_2 \times n_1) \# n_2$  imbracat în

$\text{print}(n_2 \& n_1) \Rightarrow$  intersecția  $| \text{print}(n_2 \Delta n_1) \Rightarrow$  diferență

$\text{print}(n_2 - n_1) \Rightarrow$  diferență  $| \text{print}(A \oplus B) = (A \setminus B) \cup (B \setminus A)$

simetrică

Multimne do multimni  $\Rightarrow$  NU : multimne trie de contimă

elemente imutabile, dor multimnei in nme este mutabilă  
comprehensione  $\Rightarrow$  DA  $\rightarrow$  folosesc {} în loc de I

Metode care setările sunt rezultatul

n.union (casa, altceva), n.intersect(), n.difference()  $\rightarrow$  pot primi ca parametru orice, nu doar multimni

Metode care modifică multimne

n.intersection\_update(), n.difference\_update()

n.add (element)

## Fisiere text

### 1) Deschidere

`f = open("nume_fisier", mode)` de deschidere



asigură

deschiderea

primă

variantă

(implicit): "r" → citire



"w" → scriere



"a" → adaugare la final

### 2) Prelucrare (citire / scriere) → se metode ale obiectului

### 3) Încăidere → f. close()

Citire: (fisierul trebuie să existe)

⇒ mișcare

- `read()` ⇒ returnează ca str tot conținutul fisierului
- `readline()` ⇒ returnează ca str limia curentă
- `readlines()` ⇒ returnează o listă cu elemente de tip str. corespunzătoare limilor din fisier [limia 1, limia 2 ...]

Fisierul poate fi iterat prin limie:

for limie in f:

print(limie) ⇒ urmărește a enter

print(next(limie)) ⇒ urmărește a enter: în sfârșit do limie dar nu îl execută

Scriere: fisierul multib. să existe, dacă există este agit automat

f.write(str)

↳ și urmărește a enter

f.writelines(colectie de stringuri) ⇒ nu urmărește niciun separator.

IMPORT RE → limite 141 → 160

STRIP si RSTRIP → 162:00

import re

comunicação de memória → faz split duplo  
se.split("n,[\n]+",limite)

se.split("[\n]", limite)

faz split individual de cada string

strip() → remove caracteres sobre da string inputada no terminal  
sem nenhuma

re.rstrip() → remove os caracteres sobre da string inputada no terminal  
sem nenhuma

char p. d. imustat

## Dictionaries

↳ pt. eficiencia

Valores immutables  
=  $O(1)$

dictionary = {  
    "key1": "value1",  
    "key2": "value2",  
    3: "value3"}  
}

(4, 5, 6): [ "value4", "value5", "value6", {"key": "value"} ]

alt dictionary im  
3 value pointer (hier)

print(dictionary.keys())

dictionary.update({key: value})

print(dictionary.items())

print([(4, 5, 6)][3][1]) ~> value

print(dictionary["7"]) ~> Key Error

print(dictionary.get("7", "default value"))

dara mu gelingt check 7 =>  
=> interne default val.

dictionary[3] = "value3 modified"

dictionary[(4, 5, 6)][3] = "value5 modified"

print(dictionary)

dictionary.pop("key2")

← modificar links initial mu returniert mindestens

print(links.pop()) ~> None

print(links.pop(0))

print(links) → links[0] entfernt

intern ein Element entfernt, das mu  
modifizieren

links.pop(key = lambda x: x[1] == 2)

links.pop(key = lambda x: len(str(x)))

passt die Funktion auf alle  
werte im list an

links.pop(key = my\_funcie)  
[ funktie ]

`list. sort( key = lambda x: (len(str(x)), x) )` ( $\Rightarrow$

dintră ea  $\downarrow$  (aceeași ordine ca și după) (punem  
pe numere. păcălitățile nu sunt  
mai multe condiții)

$(1, 2) \sqsubset (2, 3) \rightarrow$  astfel sunt tupluri

`list. sort( key = lambda x: (len(str(x)), -x) )`

$\downarrow$  la conversie în tuple  $\rightarrow$  desenările  
după valoare

`list. sort( key = lambda x: (len(str(x)), x[-2], -x) )`

MFP:

`vector = list(map(int, vector))`  $\rightarrow$  transformam cîteva de

str. în liste de int.

`dictioanar. update({key: value})`  $\rightarrow$  adaugă o nouă cheie cu  
valoarea

$\downarrow$  imprimă valoarea do  
la cheie data cu  
noua valoare

$\Rightarrow$  `list = sorted(list, key=lambda x: (len(str(x)), x))`

`dictioanar. get(chie, valoare - implicit)`

$\rightarrow$  returnă valoarea corăbată la cheie dacă

aceasta există sau valoarea implicită dacă nu există  $\Rightarrow$

$\Rightarrow$  evită generația unei noi liste

Stergerea unei chei  $\Rightarrow$  del dict [cheie]  $\Rightarrow \Theta(1)$  (poate

deserabilitatea listă și ceea ce  $\Theta(m)$ )

percentage dict comprehension:

for x in dict:  $\Rightarrow$  percentage incomplete choice ( $\Leftarrow$ )

( $\Leftarrow$ ) for x in dict.keys():

for x in dict.values():  $\Leftarrow$  percentage redundant

ex: sample 434  $\rightarrow$  544

## Functie

definire (parametru formal):  $\rightarrow$  cantică funcției  
corespunzători de instrucțiuni

parametru formal  $\rightarrow$  dim întreb, analiză cu parametrii sau primii  
argument la apel

parametru actuali  $\rightarrow$  valori în cadrul unei apelările funcției

! Dacă funcție nu returnează explicit valori, returnarea None

obișnuit return  $x, y, z \Rightarrow$  tuplu  $= a, b, c = f(x, y, z) \Rightarrow$  returnarea  
de tupluri

## TIPOURI DE PARAMETRII

1) obligatorii  $\rightarrow$  trebuie să fie valoare la apel

• parametru actuali ne pot transmite printr-o parte

(adică în ordinea dim întreb), prim număr / odată cu un alt  
valori sau cu parametrii și ne ducă /, cum ar fi (dă prim  
număr obligatoriu la final)

2) ce valoare implicită: de ex:  $\text{def}(x, y, z=0) \rightarrow$  dacă la apel  
nu primește valoare pt z, atunci se va considera 0

Functie cu număr variabil de parametrii

$\rightarrow$  parametrii preiau și împerechesc imediat cu valori  
primești la apel sub formă de tuplu.

$\rightarrow$  poate fi urmat de alti parametrii, dar această funcție poate fi apelată  
prin număr

ex: def numar (\*numere, a=0, b=100): (implicit obligatoriu și trimite prim număr)  
 print (numere, type (numere))  
 return num (I  $x \in [a, b] \cap \text{numere} \wedge a \leq x \leq b$ )  
 numar (10, 20, 100, -5, 18)  $\rightsquigarrow$  0, 6 și valoare implicită  
 numar (10, 20, 100, -5, 18, a=10)  $\rightarrow$  obligatoriu prim număr

Functă ca parametru pentru alte funcții

ex: def numar (\*numere, f):

return num (I  $f(x) \in \text{numere}$ )

import math

a = numar (1, 2, 3, 4, 4, f = math.sqrt)

print (f" {a:.2f}"')

$\downarrow$  aproximare cu doară zecimale tip float.  
numărul pe care îl aproximăm

def invers (x):

return 1/x

numar (1, 4, 16, f = invers)

numar (1, 3, 4, f = lambda x: 1/x)

map (f, ls)  $\rightarrow$  aplică funcția f fiecărui element din ls

filter (f, ls)  $\rightarrow$  doar elementele din listă pentru care f

returnează True  $\rightarrow$  nu modifică obiectul

Pt. funcții recursive  $\rightarrow$  m. de apeluri recursive este limitat,

dacă nu poate modifica

import sys

sys.setrecursionlimit (2000)

## Fiziere

f = open ("finier.txt", "r")

↳ read

content = f.read() → citeste tot

print([content]) → se scrie un lm =>

split(\n) deoarece

content = f.readlines() → citeste din  
cate limbi sunt, sau  
ni lm la final

content = f.readlines() → face astfel  
split-ul, potrivit e f.readlines()

de cate limbi sunt

→ aruncați finierul și se va split  
după limbi

for i in range(len(content)):

content[i] = content[i].strip()

↳ eliminăm operația \n, fiecare

print(content) → ['avantură este o limbă', 'escante este alta']

print(" " lm operație " ".strip()) ~ operație

f.close() → obligatoriu

fout = open ("fizier.txt", "w")

fout.write("Hello, world!")

! nu punem lm asternut

fout.writelines(["H", "H", "\n"])

# Comprehension

`lintā = ["Ama", "ore", "mere", "m.", "pero"]`

Büte - versch - auswmt = [ für literē im auswmt

für uns im Eintöpf

[I literær genliterær i en væsentlig litterær i "AEIOUareiou" Ifølge væsentlig litterær]

al mei de ons far-〉 een dim dropten

[ x gan x im linken ig  $x/2 = 0$  ]

tip

(categorie) — numere — | pret.  
| comitata

## dichomas {

category: {mummet: (prob., comitote)} , mummia: - - ]

## Categorie 2:

## Kategorie 3:

numme }

7 daei veau no optimisee int auant le eau de la

figier tis no foleuse .read()), allig immi d's erage pte o deje  
linter.

list(mop(functioam, iterales))  
functie po  
cote veau  
no o opie  
o immi de mop-vel  
inti - o lento.

functie numar(x numerale, f = lambda x: 1/x)

$P = \begin{pmatrix} -1 & -10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix}$

- 013:7] ~ gram , oddica do Pa - 11 poms le - H EXCLUSIV  
 02:-4] ~ program T panâlau j EXCLUSIV.  
 02:3] ~ pris  
 025:3] ~ amarelo  
 028:3] ~ gramareo  
 025:2] ~ NiMiC  
 025:-2] ~ mar  
 0210:13] ~ NiMiC.

OBS! 021,100] → negramareo ↗

- 021:6:2] ~ nega ✓  
 026:1:-2] ~ mao ✓  
 021:2] ~ nagaage ✓  
 026:-2] ~ maoap ✓  
0212:7:-1] ~ also → pp. cae co aici  
inverso  
 02:9:-2] ~ aeramare → erro menor.  
 ↗ 02:-2:-2] ~ aeramareng o n p  
 02-2:-5:1] ~ NiMiC  
 02-2:-5:-1] ~ mao ✓

|         |    |    |    |    |    |    |    |
|---------|----|----|----|----|----|----|----|
|         | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
| $f_0 =$ | 3  | L  | 11 | 4  | 7  | 9  | 5  |
|         | 0  | 1  | 2  | 3  | 4  | 5  | 6  |

$\{1, 6 : 2\}$

$\{2, 6 : 1 : -2\}$

$\{1, L : : 2\}$

### Metoda greedy

#### Problema reprezentare

→ m reprezentează

→ o reprezentare

→ pt fiecare reprezentare există perioada de desfășurare  $(a, b)$

Să se determine o submulțime de condimente maximă de reprezentări care să pot fi realizate în același timp (sunt compatibile  $\Leftrightarrow$  sunt disjuncte)

$m = 5 \rightarrow [1, 5), [1, 3), [4, 8), [4, 7), [9, 11)$

$\Rightarrow$  soluție:  $\{1, 3), [4, 8), [9, 11)$

Soluție:

1) Care este primul reprezentat pe care să îl pun în rezolvare?

- cel mai scurt  $\Rightarrow$  nu este corect (ex:  $[4, 7)$  nu suprapune cu  $[1, 5)$  și  $[4, 8)$ )

- începe cu mai devreme  $\Rightarrow$  poate fi prelungit

$\Rightarrow$  într-o trebuire: INTEPE PERIODELE SE TERMINĂ CEL MAI REPEZUT

pentru a lăsa cele libere mai mult timp după ce se termină

OBS! Deoarece există puțină posibilitate de a intersecta între ele și că  
mai puține cele spectacole (pentru a săracire să fie mai multe  
spectacole compatibile cu acesta)  $\Rightarrow$  ERERAT pt că pot fi multe  
deasupra nenumărate intersecții între ele

## IMPLEMENTARE $\rightarrow$ pseudocod

- idee  $\rightarrow$  să găsim să se adauge spectacolul care să termine cu  
mai devreme și să nu intersecteze cu cele deja selectate  
 $\rightarrow$  să căutăm intervalele după timpul de terminare  
(ex:  $[1, 3], [1, 5], [5, 7], [4, 8], [9, 11]$ )  
 $\rightarrow$  să căutăm în ordine prima interval. Pentru fiecare interval  
să verificăm dacă se poate adăuga în soluție și dacă da, îl  
adăugăm

soluție = [primul interval]

pentru interval în lista\_endonote\_de\_intervalo:

înțelege că dacă intervalul nu se intersectează cu intervalele din soluție  
atunci este să căutăm în următoarea intervale

soluție:  $[1, 3], \sim [1, 3], [5, 7] \sim [1, 3], [5, 7], [9, 11]$

python:

for i in range(m):

n = input()

ls = n.split()

x = int(ls[0])

y = int(ls[1])

lista\_intervalo.append([x, y])

print( lista - intervale )

def cheie(x):

return x[2]

lista - intervale . sort( key = cheie )

solutie = [ lista - intervale [0] ]

gasu interval im lista - interv:

ult - interval = solutie [-1]

if interval [0] >= ult - interval [1]:

solutie.append( interval )

gasu interval im solutie:

print( interval )

## Probleme susacumă / culege

→ m intervale incluse (susacumă)

puncte im varo potem wie

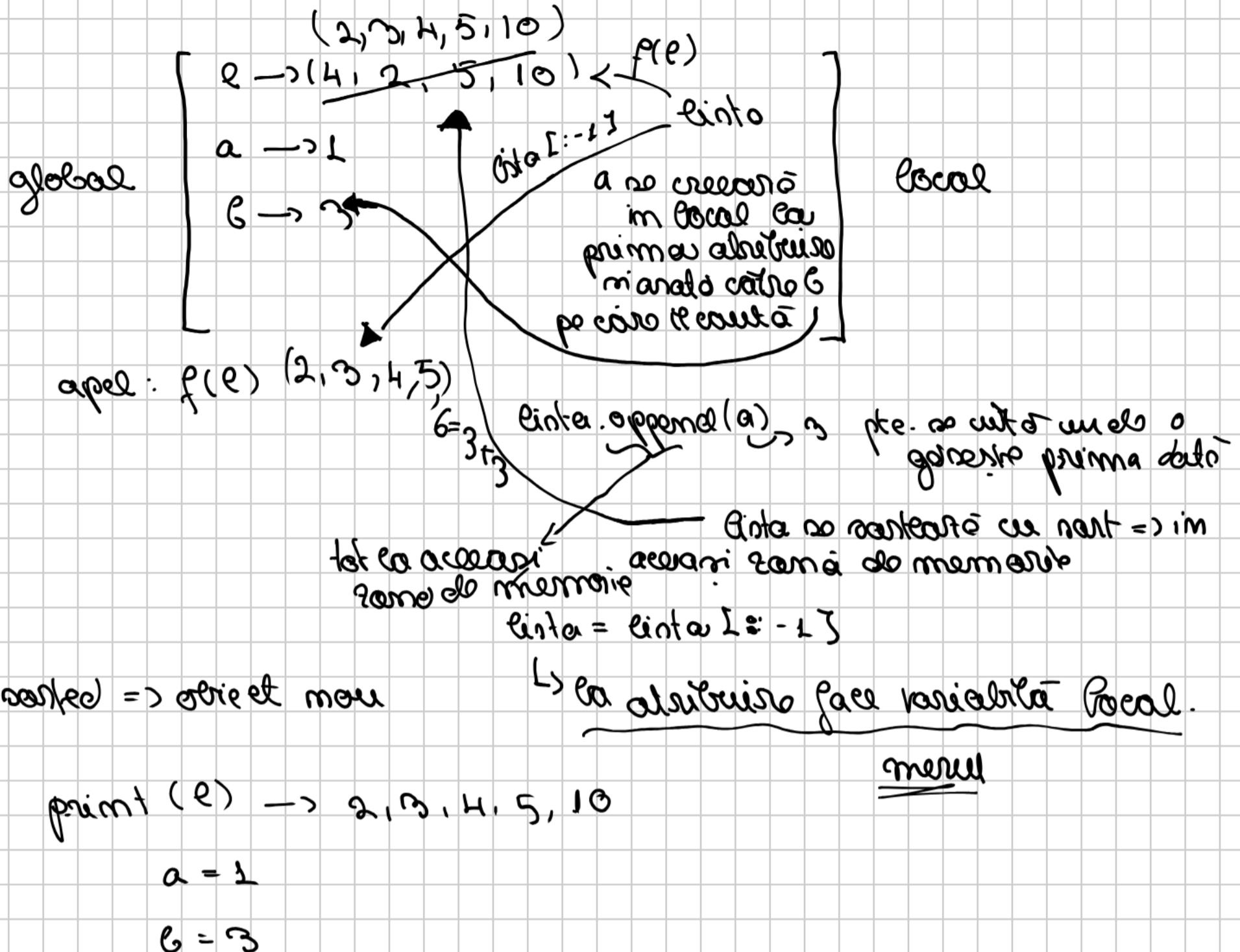
→ multime de cardinal minim cu proprietatea că orice

interval dintr-o colo date exceptia cel putin un punct din multime

(orică susacumă este fixată cu cel putin un cui)

OBS! Cardinalul solutiiei  $\geq$  numarul de intervale dintr-o dimensiune  
colo date (= cardinalul

## Model de subiect



marked  $\Rightarrow$  obiect nou

$\text{prim}(\ell) \rightarrow 2, 3, 4, 5, 10$

$a = 1$

$b = 3$

lîmînd 689 - 705.

lîmînd accessare:

valoarea unei jumătăți se cauță în local, apoi enclosing.  
apoi global apoi global.

(LEGIB)

lîmînd / modelare:  $\rightarrow$  lîmînd primă atrăuire și urmăre  
variații în spațiu sunt, dacă nu există în acel spațiu  
 $\rightarrow$  dacă nu se modifică o  
variație din global în jumătatea lui nu se propaga:

global 2

modificare asupra lui 2

## TRANSMITEREA PARAMETRILOR SI DOMENIUL DE VIZIBILITATE AL UNEI

VARIABILE

~> spatiu global

~> la fiecare apel al unei funcții => spațiu local al funcției

(pe cître programele nu cîntă contextul de apel: var. locale, numele  
funcției, adresa de rezervare)

- primul aburire, parametrul formal => parametru actual
- parametrului formal îi corespunde o variabilă locală în funcție

Stergerea elementului de la poziție k la dim niz.

$$n = n[1:k] + n[k+1:L]$$

Stergerea unei linii / rezervare  $\Rightarrow$  se stergă pe totă durată și e specifică  $\text{vector}$

$$n = n.\text{replace} ("rezervare", "")$$

! replace  $\rightarrow$  implementarea reprezentării cu repetiție nu se poate

folosi split îndeosebi

## Metoda Backtracking

$x = (x_1, x_2, \dots, x_m) \rightarrow$  rezultat este vector

$\rightarrow$  caută toate rezultatele cu o anumită proprietate

$\rightarrow$  Se completează vectorul  $x$  element cu element, se încearcă pe rând toate valoriile posibile

$\rightarrow$  Dacă valoarea verifică condiția de continuare trece la  $x_{k+1}$

$\rightarrow$  Dacă se termină valoarea permisă  $x_k$  nu poate fi parimpoziat  $x_{k+1}$

$\rightarrow$  Dacă  $x_1, \dots, x_k$  nu verifică condiția de continuare  $\Rightarrow$  nu există

$x_{k+1}, \dots, x_m$  astfel încât obținem o soluție

### Model

pozitie pe care o completează (w m. de lăs)

def back(k):

if  $k = m$ : # am completat  $x_0, \dots, x_{m-1}$  și ajungem la  $x_m$   
 $\Rightarrow$  vectorul are toate elementele complete

if soluție( $x$ ): # dacă cond. de eant nu e suficientă  
algoritm se termină și redăm ca nu suntem sol.  
algoritm se termină și redăm ca nu suntem sol.

else: # luăm po rândul valoarea pozibile pt  $x[k]$ .

for  $v$  in A $_m$ :  $\rightarrow$  depinde de problemă că

$$x[k] = v$$

if continuare( $k$ ):

back( $k+1$ )

Apel: back(0)  $\rightarrow$  prima pozitie

# Răzolvarea problemelor cu backtracking

Pasi:

- 1) Cum reprezentăm soluția?
- 2) Ce proprietăți trebuie să verifice soluția?
- 3) Când să se continuă?
- 4) Ce valori poate lua  $x_k$ ?

Ex: Problema aranjamentelor:  $A_m^m$

- 1)  $x = (x_1 \dots x_m)$
- 2)  $x_i \neq x_j, \forall i \neq j$
- 3)  $x_k \neq$  de ex. dejau complete
- 4)  $x_k \in \{1 \dots m\}$

def continuare( $x, k$ ):  $\rightarrow x$  este vectorul soluție,  $k$  este pasul la care am ajuns

foru  $i$  im range( $k$ ):

if  $x[i] = x[k]$ :

return False

return True

def Back( $k, x, m, m$ ):

if  $k = m$ :  $\rightarrow$  cumpărătire de cont. imi asigură că el are  
dintreto -> nu îl găsesc. să verifice alt eveneu  
point(\* $x$ )

else:

foru  $v$  im range( $m + 1$ ):

$x[k] = v$

if continuare( $x, k$ ):

Back( $k + 1, x, m, m$ )

def aranjamente (n, m)

$$x = [0]^+ m$$

return (0, x, n, m)

n = int(input())

m = int(input())

aranjamente (n, m)

=> mai mult ca ca nu fie  
nu se spune

OBS! Aceea va aranja pentru a multimea specială:

def alegere (x): #  $x[i] =$  indicele elementului din  
multimea A minus L.

for y in x:

print(A[v-L], end = " ")

print()

ex2: Problema combinatorilor  $C_m^n$

1)  $x = (x_1 \dots x_m)$

2)  $x_i \neq x_j \quad \forall i \neq j$

3)  $x_j$  nu poate fi de la mijlocul reșetuii

4)  $x_k \in \{1 \dots m\}$

Diferență față de aranjamente: datele obținute sunt egale doar  
cu acelasi elemente (indiferent de ordine)

def combinare (x, k):

if  $x[k] > x[k-1]$  sau  $k == 0$ :

return True

return False

→ se vede ca la aranjamente.

Ex 3: Probleme u. Rätsel mit Wörtern

v) mettia are lungime variabile

Ն Համբարձութեան Անդրանիկ Առաքելյան

sol 1: for  $m$  in range ( $l, m+l$ ):  
        combine ( $m, m$ )

Def:  $S = \text{submultim} \leq \Rightarrow$  reetor caracteristic (vector cu elemente din  $\mathbb{Z}$ , de lungime  $m$ )

$$S = \{2, 4, 5\} \rightarrow v = \left[ \begin{smallmatrix} 0 & 1 & 0 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 5 \end{smallmatrix} \right]$$

$$1) \quad x = [x_0 \dots x_{n-1}]$$

21 Jan è word. de eant

3)  $x_k \in \{0, 1\} \Rightarrow$  problema se reduce la generarea tuturor rectanglerilor binarii de lungime  $m$ .

def Bachu(k, x, m):

if  $K = m$ :

afisare ( $x, m$ )

*else*:

for us in name (2):

$$x[10] = 25$$

$\text{Gauss}(k+L, x, m)$

def adjoint ( $x, m$ ):

for : im namen (n) :

$$i \not\in x \sqcup i \not\in y = \perp$$

```
print(i+1, end=" ")
```

def nullmullime(m):

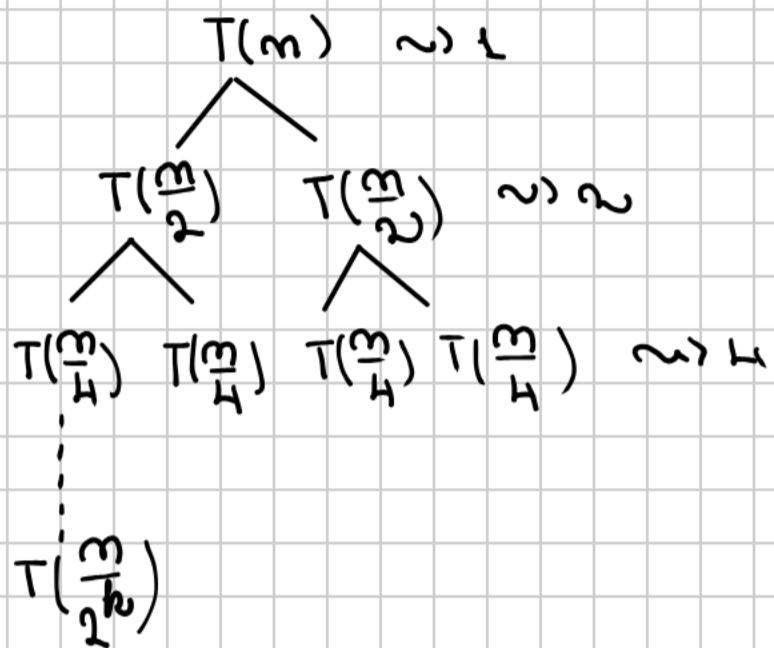
$$x = \{0\}^* m$$

backu(0, r, m)

nullmullime(r)

## Metoda Dividă și Împărță

-> se folosește pentru probleme care se pot împărti în probleme mai mici  
de același tip care nu se suprapun, păstrând ceea ce urmărește abordarea subproblemelor



$$\Rightarrow 2^0 + 2^1 + \dots + 2^{k-1} = 2^k - 1 = 2m - 1$$

$$\Rightarrow O(m)$$

## CĂUTARE BINARĂ

Se dă un vector ordonat (crescător) de numere. Se dă o valoare  $x$ .

Să se determine dacă  $x$  aparține sau nu în vectorul și dacă aparțin, pe ce poziție. Dacă vectorul nu este ordonat, folosind metoda imediat  $\rightarrow$  complexitate limitată, nu folosește ipoteza că vectorul este ordonat.

|        |   |   |         |    |    |
|--------|---|---|---------|----|----|
| 1      | 3 | ? | 10      | 11 | 12 |
| stanga |   |   | dreapta |    |    |

$\sim$  compară  $x$  cu mijlocul vectorului.

$x == mij$   $\Rightarrow$  găsit

$x < mij$   $\Rightarrow$  căut în stanga

$x > mij$   $\Rightarrow$  căut în dreapta

$\sim$  dacă nu reținem în casă căutăm din nou  $\Rightarrow x$  nu este în vector



$$\Rightarrow k = \log_2 n \Rightarrow O(\log n)$$

def cautare\_binara(v, p, u, x):

if  $p > u$ :

return -1

$$m = (p+u)/2$$

if  $v[m] == x$ :

return m

if  $x < v[m]$ :

return cautare\_binara(v, p, m-1, x)

if  $x > v[m]$ :

return cautare\_binara(v, m+1, u, x)

def cautare(v, x)

cautare\_binara(v, 0, len(v)-1, x)

## INTERCLASAREA

→ doi vectori mentindu-să se adună

$$\alpha = [4, 7, 8, 15]$$

$$\beta = [1, 2, 6, 7, 10, 18, 19]$$

⇒ să se formeze un vector

cine năște combinația elementelor din α și din β cind sunt corecte

$$\Rightarrow \mathcal{O}(m+n)$$

def interclasare(a, b):

i = 0 # indică pozitia din α

j = 0 # indică pozitia din β

$$m = \text{len}(a)$$

$$n = \text{len}(b)$$

$$c = \sum_{j=0}^k (m+jm)$$

$$h = 0$$

while  $i < m$  and  $j < m$ :

if  $a[i] < b[j]$ :

$$c[h] = a[i]$$

$$i += 1$$

$$h += 1$$

else:

$$c[h] = b[j]$$

$$j += 1$$

$$h += 1$$

while  $i < m$ :

$$c[h] = v[i]$$

$$i = i + 1$$

$$h = h + 1$$

while  $j < n$ :

$$c[h] = v[j]$$

$$j = j + 1$$

$$h = h + 1$$

## METODA GREEDY

→ se folosește în probleme de optimizare

→ rezultatul se construiește element cu element adăugând la fiecare pas

în ordine elementul care pare să mai fiu la acel moment

↳ problema care operează este corectădacă

ex : Se dă o mulțime  $A \subset \{1, 2, \dots, n\}$ . Se va determina o submulțime

a lui  $A$  de cardinal  $k$  care să număreze elementelor maximă

sol. greedy: pas 1 → adăugăm cel mai mare el. din  $A$

pas 2 → adăugăm elementul maxim dintr-o altă normală.

:

variantă ex: Se dă o mulțime  $A \subset \{1, 2, \dots, n\}$ . Se va determina o

submulțime a lui  $A$  cu număr maximă, dar  $\leq k$ .

$$A = \{1, 4, 5, 7\}$$

$$k = 9$$

$$\text{Greedy} \Rightarrow \{7, 1\}$$

$$\text{sol. optimă} \Rightarrow \{4, 5\}$$

## PGL: Problema spectacolelor

→ în spectacole , și nici. Pentru fiecare spectacol stim intervalul de desfășurare  $[r_i, t_i]$ . Se va determina o submulțime de spectacole maximă de care se pot programă în nici  
(se zice că sunt compatibile = au intervalul de desfășurare disjuncte)

## MERGE SORT

def interclasseare ( $v, p, m, u$ ):

$$i = p$$

$$j = m + 1$$

$$c = \sum_{i=p}^m (u - p + 1)$$

$$k_0 = 0$$

while  $i <= m$  and  $j <= u$ :

if  $v[i] < v[j]$ :

$$c[k_0] = v[i]$$

$$i = i + 1$$

$$k_0 = k_0 + 1$$

else:

$$c[k_0] = v[j]$$

$$j = j + 1$$

$$k_0 = k_0 + 1$$

while  $i < m$ :

$$c[k_0] = v[i]$$

$$i = i + 1$$

$$k_0 = k_0 + 1$$

while  $j <= u$ :

$$c[k_0] = v[j]$$

$$k_0 = k_0 + 1$$

$$j = j + 1$$

$$v[p : u+1] = c$$

def reverzna ( $v, p, u$ ):

if  $p >= u$ :

return

$$m = (p+u)/2$$

reverzna ( $v, p, m$ )

reverzna ( $v, m+1, u$ )

interclasseare ( $v, p, m, u$ )

def merge - sort ( $v$ ):

reverzna ( $v, 0, \text{corr}(v) - 1$ )