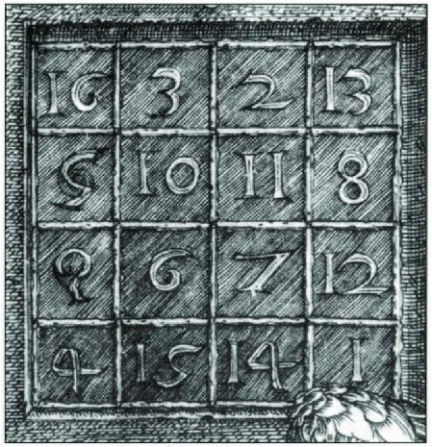
Logică și Structuri Discrete -LSD



Cursul 2

dr. ing. Cătălin Iapă

e-mail: catalin.iapa@cs.upt.ro

facebook: Catalin Iapa

cv: Catalin Iapa

Ce am făcut data trecută?

Ne-am cunoscut
Ce facem la LSD?
Demonstrații
Mulțimi
Funcții
Proprietăți ale funcțiilor
Funcțiile în programare

Ce am făcut data trecută?

Mulțimi

- Elementul unei mulțimi - Submulțimi

Funcții

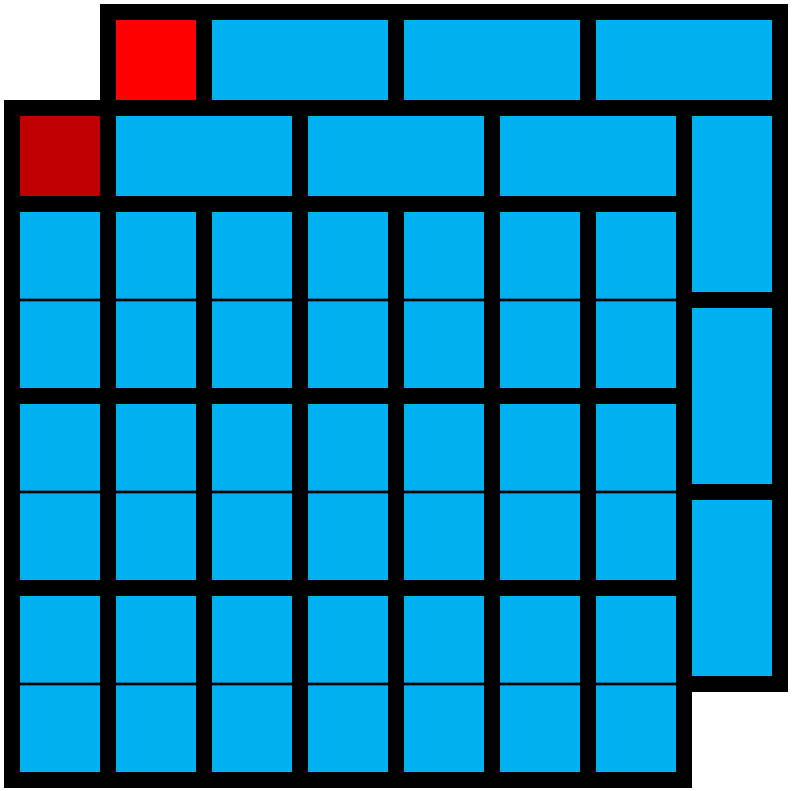
- Definiții, Domeniul, Codomeniul, Asocierea,
- Exemple care nu sunt funcții,
- injective, surjective, bijective

Funcții în PYTHON

```
def f(x):
    return x + 3
```

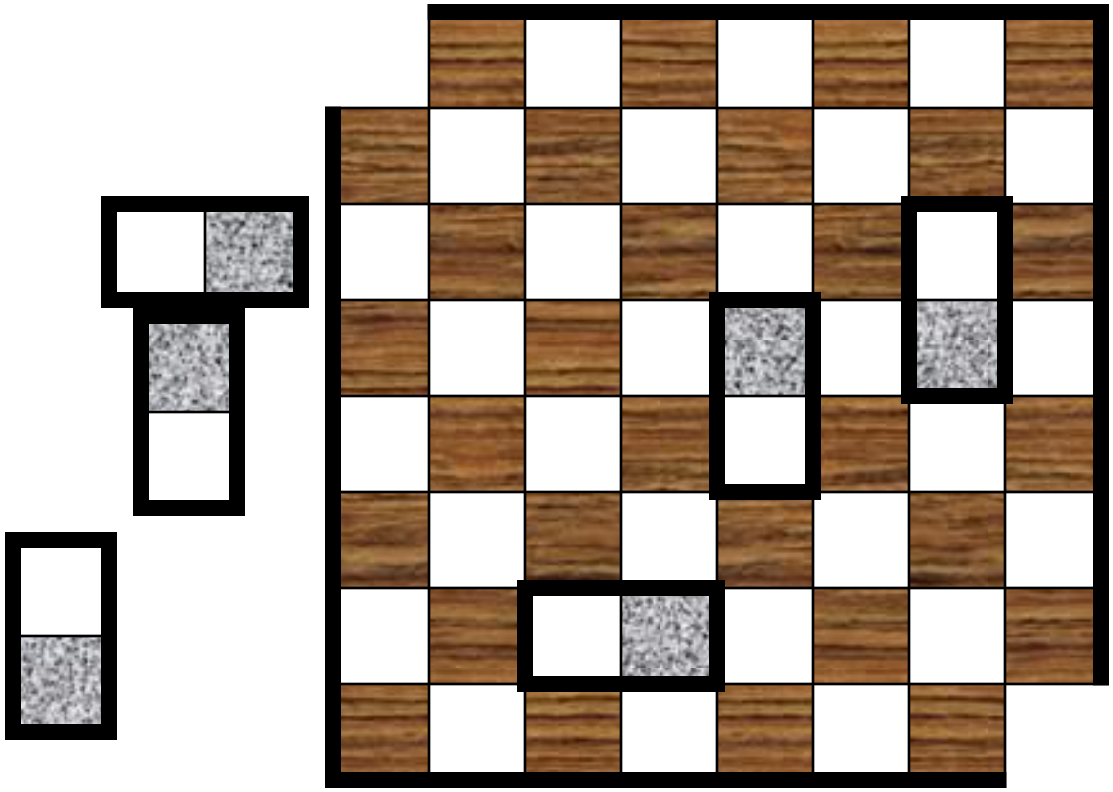
Ce am făcut data trecută?

Demonstrații



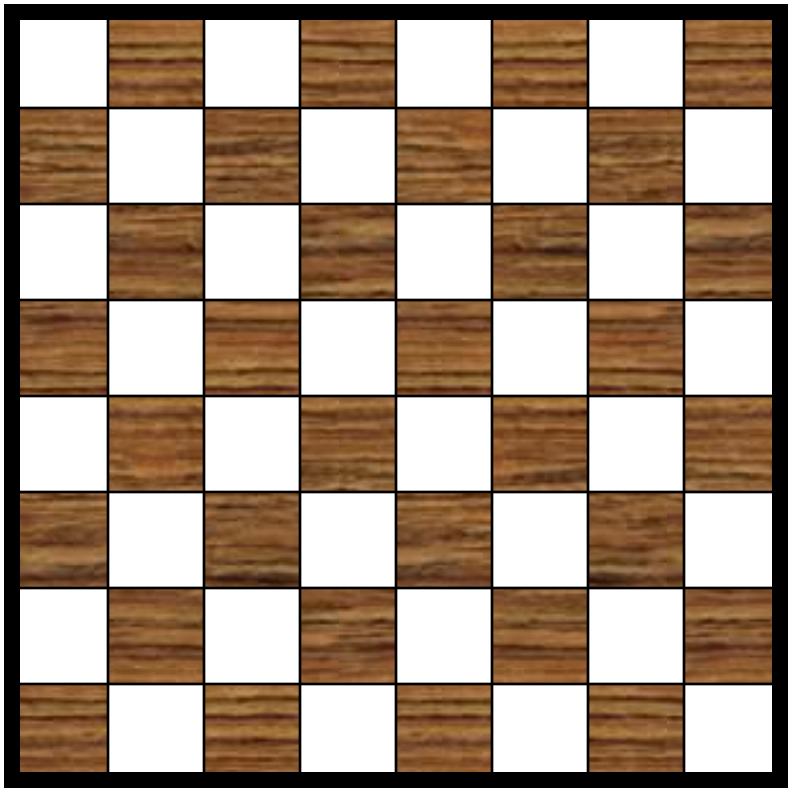
Ce am făcut data trecută?

Putem demonstra că orice pereche de pătrate, unul alb și unul negru, am șterge, tot putem acoperii tabla?



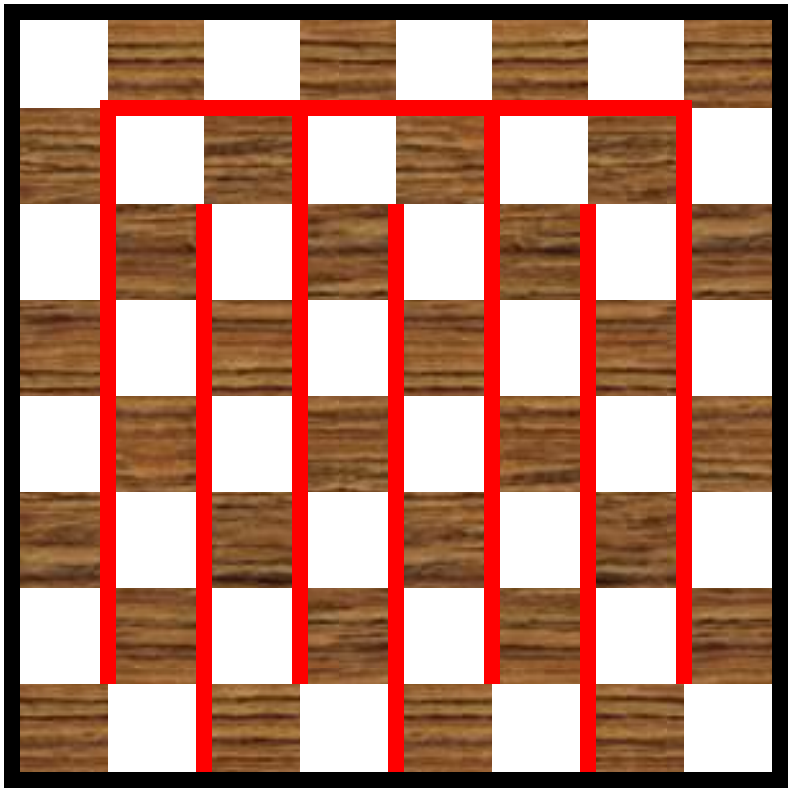
Ce am făcut data trecută?

Putem demonstra că orice pereche de pătrate, unul alb și unul negru, am șterge, tot putem acoperii tabla?



Ce am făcut data trecută?

Putem demonstra că orice pereche de pătrate, unul alb și unul negru, am șterge, tot putem acoperii tabla?



Ce am făcut data trecută?

Demonstrația prin exemplu

Demonstrația prin reducere la absurd

o afirmație e echivalentă cu contrapozitiva ei:

$$P \Rightarrow Q \quad \Leftrightarrow \quad \neg Q \Rightarrow \neg \mathrm{P}$$

afirmația — contrapozitiva

Demonstrația prin inducție matematică

Daca o propoziție P(n) depinde de un număr natural n și:

1) cazul de baza : P(1) e adevarată

2) pasul inductiv : pentru orice n≥1

$$P(n) \Rightarrow P(n+1)$$

atunci P(n) e adevarată pentru orice n.

Ce am făcut data trecută?

Demonstrația prin inducție matematică

Daca o propoziție P(n) depinde de un număr natural n și:

1) cazul de baza : P(1) e adevarată

2) pasul inductiv : pentru orice n≥1

$$P(n) \Rightarrow P(n+1)$$

atunci P(n) e adevarată pentru orice n.

Pe ce se bazază raționamentul inducției matematice?

- P(1) am demonstrat că e adevărată

- $P(1) \Rightarrow P(2)$, $P(2) \Rightarrow P(3)$, ... , $P(n-1) \Rightarrow P(n)$, ...

Inducția matematică completă

Daca o propoziție P(n) depinde de un număr natural n și:

1) cazul de baza : P(1) e adevarată

2) pasul inductiv : pentru orice n≥1

P(1) și P(2) și P(3) și ... și P(n) ⇒ P(n + 1) e adevărată

atunci P(n) e adevarată pentru orice n.

Inducția matematică completă

Să demonstrăm: Orice strategie am alege să jucăm, vom obține același punctaj pentru același număr de piese, S(n)

Ex: S(8)=28 S(8)=56/2 S(8)=(7*8)/2

S(n)=((n-1)*n)/2

P(n)

cazul de bază P(1)=0

Pasul inductiv: P(1) și P(2) și P(3) și ... și P(n) ⇒ P(n + 1)

P(n+1)=P(k)+P(n+1-k), k<n

Scorul S(n+1)= k*(n+1-k) + P(k) + P(n+1-k)

Ar trebui să demonstrăm că acest scor depinde doar de n, nu și de k

Inducția matematică completă

$$S(n+1)= k*(n+1-k) + P(k) + P(n+1-k)$$

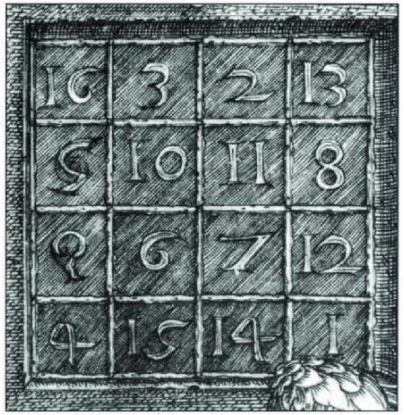
$$S(n+1)= k*(n+1-k) + \frac{k(k-1)}{2} + \frac{(n+1-k)(n-k)}{2}$$

$$S(n+1)= \frac{2kn+2k-2k^2+k^2-k+n^2-nk+n-k-nk+k^2}{2}$$

$$S(n+1)= \frac{(2kn-nk-nk)+(2k-k-k)-(2k^2+k^2+k^2)+n^2+n}{2}$$

$$S(n+1)= \frac{n^2+n}{2}$$

$$S(n+1)= \frac{n(n+1)}{2}$$



Ce am făcut săptămâna trecută?

Inducția matematică completă

Mulțimi, Tupluri, Produs cartezian

Funcții – compunere, inversabile

Probleme de numărare

Compunerea funcțiilor în PYTHON

Mulțimi definite inductiv

Cardinalul unei mulțimi

Cardinalul (cardinalitatea) unei mulțimi A e numărul de elemente al mulțimii.

Cardinalul unei mulțimi A se notează $|A|$.

Putem avea mulțimi finite: $|\{1, 2, 3, 4, 5\}| = 5$ sau infinite: N, R, etc.

Care e cardinalul unei mulțimi infinite? $|N| = |R| = \infty$? Nu:

$|N| = \aleph_0$ $\aleph_0$ – cel mai mic cardinal infinit

$|R| = 2^{\aleph_0}$

TUPLURI

Un *n-tuplu* e un șir de *n* elemente $(x_1, x_2, \ldots, x_n)$

Proprietăți:

- elementele nu sunt neapărat distincte
- ordinea elementelor în tuplu contează

Cazuri particulare: *pereche* (*a, b*), *triplet* (*x, y, z*), etc.

Produs cartezian

Produsul cartezian a două mulțimi e mulțimea perechilor

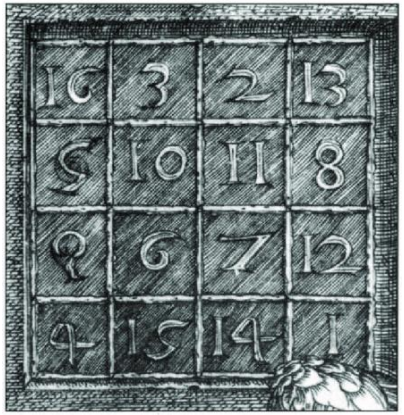
$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

Produsul cartezian a n mulțimi e mulțimea n – *tuplelor*

$$A_1 \times A_2 \times \ldots \times A_n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in A_i, 1 \leq i \leq n\}$$

Dacă mulțimile sunt finite, atunci

$$|A_1 \times A_2 \times \ldots \times A_n| = |A_1| \cdot |A_2| \cdot \ldots \cdot |A_n|$$



Ce am făcut săptămâna trecută?

Inducția matematică completă

Mulțimi, Tupluri, Produs cartezian

Funcții – compunere, inversabile

Probleme de numărare

Compunerea funcțiilor în PYTHON

Mulțimi definite inductiv

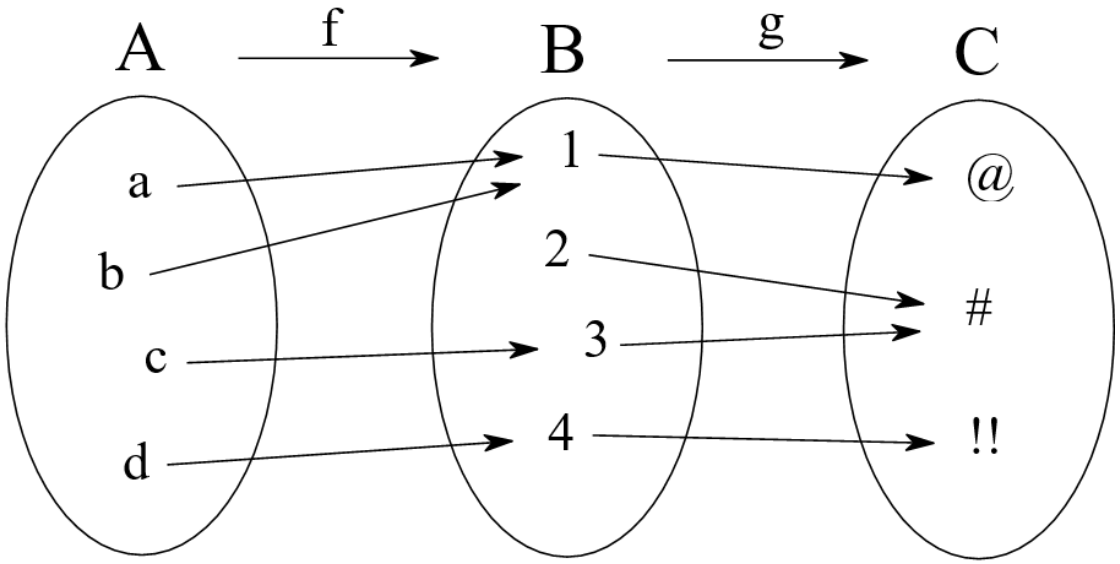
Compunerea funcțiilor

Fie functiile $f : A \to B$ și $g : B \to C$.

Compunerea lor este funcția

$$g \circ f : A \to C \; (g \circ f)(x) = g(f(x))$$

Putem compune $g \circ f$ doar când *codomeniul* lui f = *domeniul* lui g !



Imagine: http://en.wikipedia.org/wiki/File:Compufun.svg

Proprietăți ale compunerii funcțiilor

- Compunerea a două funcții e *asociativă*:

$$(f \circ g) \circ h = f \circ (g \circ h)$$

- *Demonstrație*: fie x oarecare din domeniul lui h. Atunci:

$((f \circ g) \circ h)(x) =$
rescriem $\circ$ $= (f \circ g)(h(x))$
rescriem $\circ$ $= f(g(h(x)))$

$(f \circ (g \circ h))(x) =$
rescriem $\circ$ $= f((g \circ h)(x))$
rescriem $\circ$ $= f(g(h(x)))$

- Compunerea a două funcții *nu* e neaparat *comutativă*
- Puteți da un exemplu pentru care $f \circ g \neq g \circ f$?

Funcții inversabile

- Pe orice mulțime A putem defini *funcția identitate* $id_A : A \to A$
- $id_A(x) = x$ (notată adeseori și $\mathbf{1}_A$)
-
- O funcție $f : A \to B$ e *inversabilă* dacă există o funcție
- $f^{-1} : B \to A$ astfel încât
- $f^{-1} \circ f = id_A$ și
- $f \circ f^{-1} = id_B$.

Funcții inversabile

O funcție e inversabilă dacă și numai dacă e *bijectivă*. Demonstrăm:

Dacă f e inversabilă:

pentru $y \in B$ oarecare, fie $x = f^{-1}(y)$.

Atunci $f(x) = f(f^{-1}(y)) = y$, deci f e surjectivă

dacă $f(x_1) = f(x_2)$, atunci $f^{-1}(f(x_1)) = f^{-1}(f(x_2))$,

deci $x_1 = x_2$ (injectivă)

Reciproc, dacă f e bijectivă:

- f e surjectivă $\Rightarrow$ pentru orice $y \in B$ *există* $x \in A$ cu $f(x) = y$
- f fiind injectivă, dacă $f(x_1) = y = f(x_2)$, atunci $x_1 = x_2$.

Deci $f^{-1} : B \to A$, $f^{-1}(y)$ = acel x a. î. $f(x) = y$

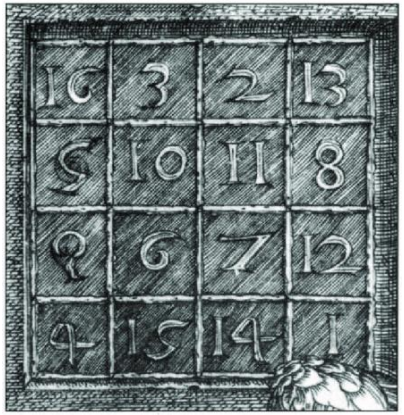
e o funcție bine definită, $f^{-1}(f(x)) = x$, și $f(f^{-1}(y)) = y$.

Imagine și preimagine

- Fie $f : A \to B$.
- Dacă $S \subseteq A$, mulțimea elementelor $f(x)$ cu $x \in S$ se numește *imaginea* lui S prin f, notată $f(S)$.
- Dacă $T \subseteq B$, mulțimea elementelor x cu $f(x) \in T$ se numește *preimaginea* lui T prin f, notată $f^{-1}(T)$.

$$f^{-1}(f(S)) \supseteq S$$

Aplicând întâi funcția și apoi inversa ei se pierde precizie. (*nu orice calcul e reversibil*).



Ce am făcut săptămâna trecută?

Inducția matematică completă

Mulțimi, Tupluri, Produs cartezian

Funcții – compunere, inversabile

Probleme de numărare

Compunerea funcțiilor în PYTHON

Mulțimi definite inductiv

Câte funcții există de la A la B?

Dacă *A* și *B* sunt mulțimi finite există $|B|^{|A|}$ funcții de la *A* la *B*. (în fiecare element din *B* se poate mapa orice element din *A*)

Demonstrație: prin *inducție matematică* după $|A|$

Mulțimea funcțiilor $f : A \rightarrow B$ se notează uneori B^A

Notația ne amintește că numărul acestor funcții e $|B|^{|A|}$

Câte funcții injective există de la A la B?

Dacă A și B sunt mulțimi finite și $f : A \to B$ injectivă

$\Rightarrow |f(A)| = |A|$ (imaginea lui f va avea $|A|$ elemente).

Ordinea în care alegem elementele *contează* !
(ordini diferite $\Rightarrow$ funcții diferite)

... deci avem aranjamente de $|B|$ luate câte $|A|$

$\Rightarrow$ există $A_{|B|}^{|A|} = \frac{|B|!}{(|B|-|A|)!}$ funcții injective

Câte funcții bijective există de la A la B?

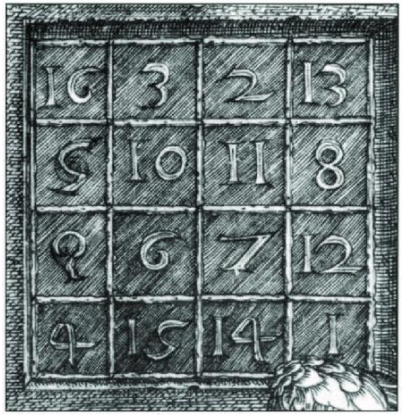
DacăA și B sunt mulțimi finite și $f : A \to B$ bijectivă

$\Rightarrow |f(A)| = |A| = |B|$ (imaginea lui f va avea $|A|$ elemente).

Ordinea în care alegem elementele *contează* !

... deci avem permutări de $|A|$ elemente

$\Rightarrow$ există $P|A| = |A|$! Funcții bijective



Ce am făcut săptămâna trecută?

Inducția matematică completă

Mulțimi, Tupluri, Produs cartezian

Funcții – compunere, inversabile

Probleme de numărare

Compunerea funcțiilor în PYTHON

Mulțimi definite inductiv

Timpuri de date

Python pune la dispozitie 4 tipuri de date primitive:

- Integer
- Float
- String
- Boolean

Tipurile de date primitive sunt imutabile, adică, odată ce acestea au fost create ele nu se mai pot modifica. Dacă o variabilă x = 3 își modifică valoarea în 4, de fapt un nou întreg este creat și atribuit variabilei x.

Integer

Tipul de date integer (int) reprezintă datele numerice cu sau fără semn, fără zecimale și de lungime nelimitată (numere întregi cu valori cuprinse între -∞ și ∞).

Ex: 3, 6, -234.

Float

Tipul de date float este folosit pentru reprezentarea numerelor în virgulă flotantă, cu sau fără semn.

Ex: 3.34, -0.123456.

Float

2 e o valoare întreagă. Pentru o valoare reală (float) trebuie să scriem 2.0 (sau prescurtat 2.).

În Python conversia de tip de la int la float se face automat. Astfel, rezultatul operațiilor ce conțin atât numere întregi cât și numere reale va fi un număr real (de exemplu 5 + 2.0 va da 7.0).

Putem folosi și funcția float() dacă dorim să facem o conversie în mod explicit:

```
>>> float (3 * 2)
    6.0
```

String

Un string (șir de caractere) reprezintă o colecție de caractere, cuvinte sau fraze. Pentru a crea un string în Python, folosim semnele '' (apostrof) sau "" (ghilimele). Exemplu: 'carte' , "23abc".

Una din cele mai comune operații este concaterarea șirurilor de caractere. Aceasta se face prin operatorul + :

```
>>> 'abc' + 'def'
    'abcdef'
```

String

- Caracterele dintr-un string se pot accesa direct prin "string"[index]:

```
>>> 'A string'[2]
's‘
```

```
>>> 'A string'[-8]
'A'
```

- Rezultatul este al treilea caracter (numerotarea caracterelor începe de la 0). Python permite și accesarea de caractere folosind valori negative pentru index. Pentru exemplul de mai jos, selectarea oricărui alt întreg care se află în afara intervalului [-8; 7] va genera o excepție:

'A '	' '	's '	't '	'r'	'i'	'n'	'g'
0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

Boolean

- Tipul de date **boolean** reprezintă valoarea de adevăr a unei expresii. Acesta poate avea valorile *True* sau *False*. De obicei, un boolean se folosește în scrierea condițiilor sau în compararea unor expresii.

```
>>> 3 == 4
    False
```

- Folosind operatorii de comparație == (egal), != (diferit), > (mai mare), < (mai mic), >= (mai mare sau egal), <= (mai mic sau egal), putem face comparații între diferite expresii.
- Pentru a scrie condiții mai complicate puteți folosi cuvintele cheie **and**, **or** și **not**.

Tipuri predefinite pentru colecții de date

Python mai oferă 4 tipuri predefinite pentru a putea reține colecțiile de date. Astfel, în Python putem lucra cu următoarele:

- Listă
- Tuplu
- Mulțime (Set)
- Dicționar

Listă

Dacă dorim să definim o listă vom enumera între paranteze pătrate elementele acesteia, elementele fiind despărțite între ele prin virgulă:

pare = [0, 2, 4, 6, 8]

Dacă dorim să accesăm un element al listei procedăm la fel ca și în cazul șirurilor de caractere (Exemplu: dacă dorim să accesăm elementul 2 din lista *pare* vom scrie *pare[1]*, numerotarea făcându-se și în acest caz de la 0).

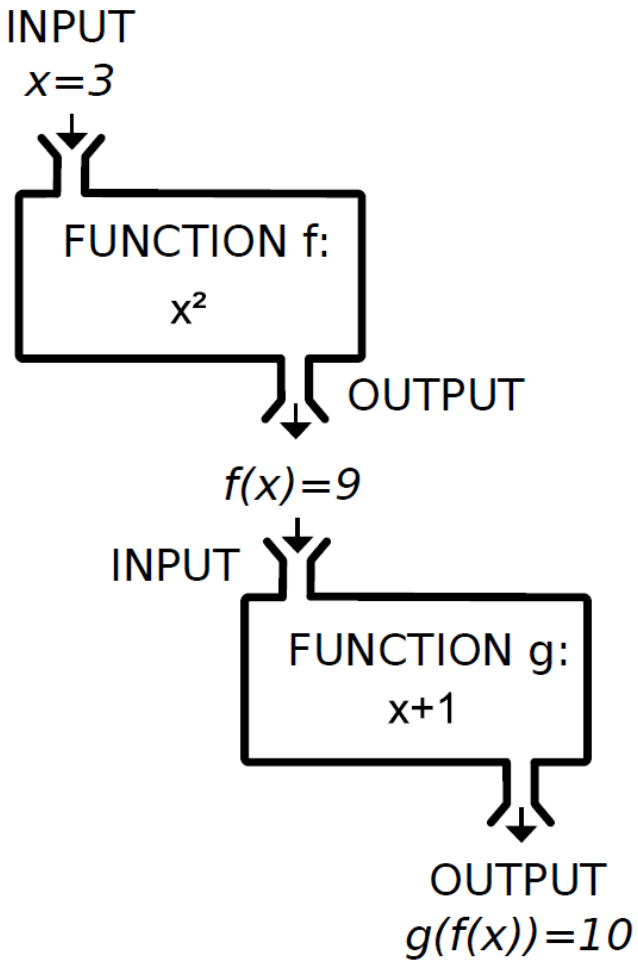
Tuplu

Un tuplu se definește folosind paranteze rotunde:

tuplu_pare = (0, 2, 4, 6, 8)

Compunerea funcțiilor

- Rezultatul funției f devine argument pentru funcția g
- Prin compunere, construim funcții complexe din funcții mai simple.



Imagine: http://en.wikipedia.org/wiki/File:Function_machine5.svg

Compunerea funcțiilor în PYTHON

Compunerea funcțiilor este modul prin care rezultatulprodus de o funcție săfie folosit ca parametru într-o altă funcție

Dacă avem 2 funcții, f și g, compunerea lor e reprezentată astfel: f(g(x)), unde x e argument pentru funcția g, iar rezultatul funcției g(x) devine argument pentru funcția f

Compunerea funcțiilor în PYTHON

```
def suma(x):
    return x + 10

def produs(x):
    return x * 10

print(produs(suma(5)))   # (5+10)*10
150
```

Compunerea funcțiilor în PYTHON

Putem, de asemenea, să creem o funcție nouă care să combine 2 funcții existente:

```
def suma(x):
    return x + 10
def produs(x):
    return x * 10

def functie_compusa(f, g):
    return lambda x : f(g(x))

suma_produs= functie_compusa(produs, suma)
print(suma_produs(2))
```

Compunerea a 3 funcții în PYTHON

```
def suma(x):
    return x + 10
def produs(x):
    return x * 10
def difetenta(x):
    return x - 2

def functie_compusa(f, g):
    return lambda x : f(g(x))
suma_diferenta_produs= functie_compusa(produs, functie_compusa(diferenta,suma))
print(suma_diferenta_produs(2))
```

Funcția poate fi returnată de altă funcție în PYTHON

```
# Python program to illustrate functions
# Functions can return another function

def create_adder(x):
        def adder(y):
                return x+y
        return adder

add_15 = create_adder(15)
print (add_15(10))
```

OUTPT:
25

Exemplu preluat: https://www.geeksforgeeks.org/first-class-functions-python/?ref=lbp

Funcții- informații extra

Putem scrie în cod explicit valoarea argumentelor și a rezultatului:

```
def suma(a: int, b: int) -> int:
    c = a + b
    return c
```

Funcții- informații extra

Putem scrie funcții cu argumente presetate:

```
def suma(x, y=10):
    return x+y
```

Putem apela funcția doar cu un parametru:

```
suma(8)
```

OUTPUT:

18

Funcții- informații extra

La apelare, putem scrie parametrii în altă ordine decât în definiția funcției:

```
def diferenta(n1, n2):
    return n1-n2
```

Putem apela funcția:

```
diferenta(n1=10, n2=3)
diferenta(n2=3, n1=10)
```

OUTPUT:

7

7

Rezumat funcții

Prin *funcții* exprimăm calcule în programare.

Domeniile de definiție și valori corespund *tipurilor* din programare.

În limbajele funcționale, funcțiile pot fi manipulate ca orice *valori*. Funcțiile pot fi *argumente* și *rezultate* de funcții.

Funcțiile de mai multe argumente (sau de tuple) pot fi rescrise ca funcții de un singur argument care returnează funcții.

Ce știm până acum?/ Ce ar trebui să știm?

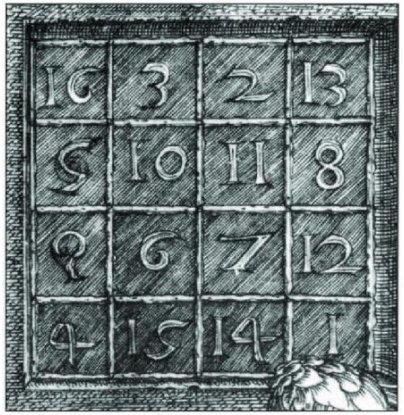
Știm *proprietățile funcțiilor* și cum să *ne folosim de ele:* funcții injective, surjective, bijective, inversabile;

Să *construim* funcții cu anumite proprietăți;

Să *numărăm* funcțiile definite pe mulțimi finite (cu proprietăți date);

Să *compunem* funcții simple pentru a rezolva probleme;

Să identificăm *tipul* unei funcții.



Ce am făcut săptămâna trecută?
Inducția matematică completă
Mulțimi, Tupluri, Produs cartezian
Funcții – compunere, inversabile
Probleme de numărare
Compunerea funcțiilor în PYTHON

Mulțimi definite inductiv

Mulțimi definite inductiv

Fie mulțimea $A = \{3, 5, 7, 9, \ldots\}$

O putem defini $A = \{x \mid x = 2k + 3, k \in \mathbb{N}\}$

Alternativ, observăm că:

- $3 \in A$
- $x \in A \Rightarrow x + 2 \in A$
- un element ajunge în A doar printr-unul din pașii de mai sus

$\Rightarrow$ putem defini *inductiv* mulțimea A

Mulțimi definite inductiv

$A = \{3, 5, 7, 9, 11, 13, \ldots\}$

- $3 \in A$ – **elementul de bază**: $P(0) : a_0 \in A$
- $x \in A \Rightarrow x + 2 \in A$ – **construcția de noi elemente**:

$P(k) \Rightarrow P(k+1) : a_k \in A \Rightarrow a_{k+1} \in A$

- un element ajunge în A doar printr-unul din pașii de mai sus – **închiderea** (niciun alt element nu e în mulțime)

$\Rightarrow$ definiția *inductivă* a lui A

$\Rightarrow$ spunem că A e o *mulțime inductivă*

Mulțimi definite inductiv

O definiție inductivă a unei mulțimi *S* constă din:

- *bază*: Enumerăm **elementele de bază** din *S* (minim unul).
- *inducția*: Dăm cel puțin o **regulă de construcție** de noi elemente din *S*, pornind de la elemente deja existente in *S*.
- *închiderea*: *S* conține **doar** elementele obținute prin pașii de bază și inducție (de obicei implicită).

Elementele de bază și regulile de construcție de noi elemente constituie **constructorii** mulțimii *S*.

Mulțimi definite inductiv - exemplu

Mulțimea numerelor naturale N e o mulțime inductivă:

- *bază*: $0 \in \mathbb{N}$
- *inducția*: $n \in \mathbb{N} \Rightarrow n + 1 \in \mathbb{N}$

Constructorii lui N:

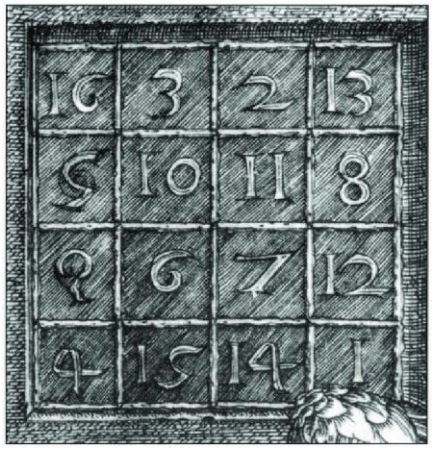
- baza 0
- operația de adunare cu 1

Mulțimi definite inductiv - exemplu

$A = \{1, 3, 7, 15, 31, \ldots\}$ e o mulțime inductivă:

- *bază*: $1 \in A$
- *inducția*: $x \in A \Rightarrow 2x + 1 \in A$

- Constructorii lui A:
  - baza 1
  - operat, ia de înmulțire cu 2 și adunare cu 1



O seară bună în continuare!

Bibliografie

- Jocul de la inducție matematică completă a fost inspirate din cursul ***Mathematics for Computer Science*** de la Massachusetts Institute of Technology (de pe https://ocw.mit.edu/)
- Conținutul cursului se bazează preponderent pe materialele de anii trecuți de la cursul de LSD, predat de conf. dr. ing.Marius Minea și ș.l. dr. ing. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)