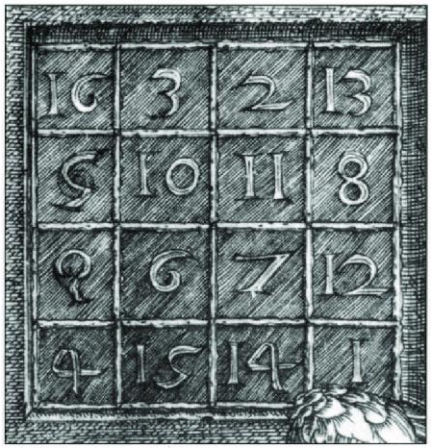
Logică și Structuri Discrete -LSD

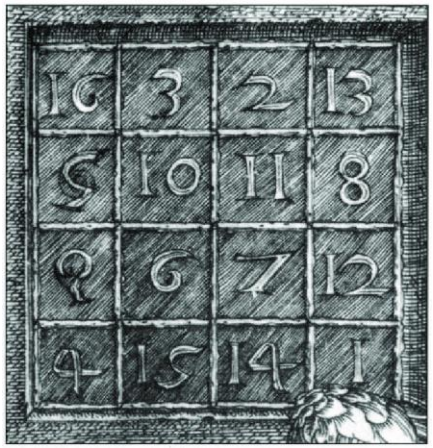


Cursul 5 – Mulțimi

dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro

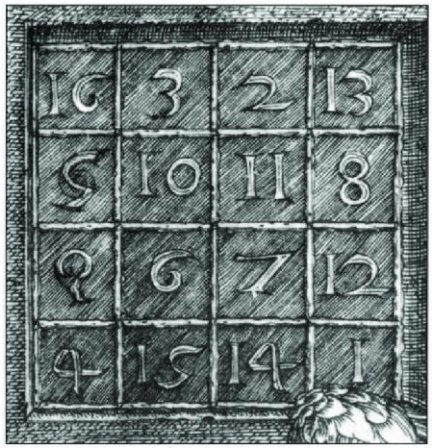
Ce am parcurs până acum?



Funcții

Funcții recursive

Liste



Ce sunt mulțimile?

Operații de bază cu mulțimi

Mulțimile, fundament al matematicii

Mulțimile în PYTHON

Operații cu mulțimi în PYTHON

Exerciții cu mulțimi în PYTHON

Ce sunt mulțimile?

Mulțimea e un concept matematic *fundamental*.

Am putea spune:

Definiție: O mulțime este o colecție de obiecte numite elementele mulțimii.

Dar ca să fie riguroasă, ar trebui să definim precis ce e o *colecție*.

Definiția e *informală*. Vom vedea că e important s-o formalizăm.

Ce sunt mulțimile?

Avem două noțiuni distincte: *elemente și mulțime*

- $x \in S$: elementul x aparține mulțimii S
- $y \notin S$: elementul y nu aparține mulțimii S

Spre deosebire de liste:

Ordinea elementelor nu conteaza {1, 2, 3} = {2, 1, 3}

Un element nu apare de mai multe ori {1, 2, 3, 2}

Cum definim mulțimile?

Prin *descriere*: { mulțimea divizorilor lui 6 }

Prin *enumerarea* elementelor:

$A = \{a, b, c\}$, $D = \{1, 2, 3, 6\}$ = mulțimea divizorilor lui 6

Elementele mulțimii se scriu între acolade, separate prin virgulă.

Printr-o *proprietate* caracteristică:

$S = \{x \mid x \text{ are proprietatea } P(x)\}$

$D(n) = \{d \in \mathbb{N} \mid n \bmod d = 0\}$ (mulțimea divizorilor lui n)

Știm: mulțimea numerelor naturale N, întregi Z, raționale Q, reale R, ...

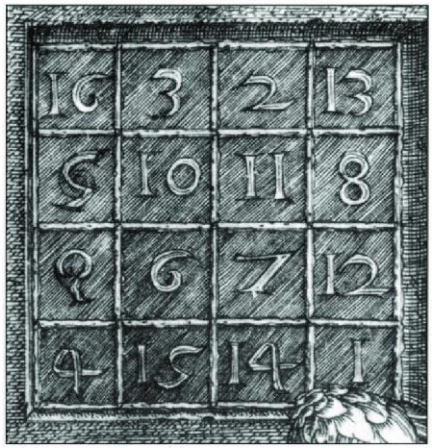
Submulțimi

A e o *submulțime* a lui B: $A \subseteq B$
dacă fiecare element al lui A e și un element al lui B.
A e o *submulțime proprie* a lui B: $A \subset B$
dacă $A \subseteq B$ și există (măcar) un element $x \in B$ astfel ca $x \notin A$.

$\in$ e o relație între un *element* și o mulțime.
$\subseteq$ s, i $\subset$ sunt relat, ii între *două mulțimi*.

Ca să demonstrăm $A \not\subseteq B$ e suficient să găsim un element $x \in A$ pentru care $x \notin B$.

Dacă $A \subseteq B$ și $B \subseteq A$, atunci $A = B$ (mulțimile sunt egale)



Ce sunt mulțimile?

Operații de bază cu mulțimi

Mulțimile, fundament al matematicii

Mulțimile în PYTHON

Operații cu mulțimi în PYTHON

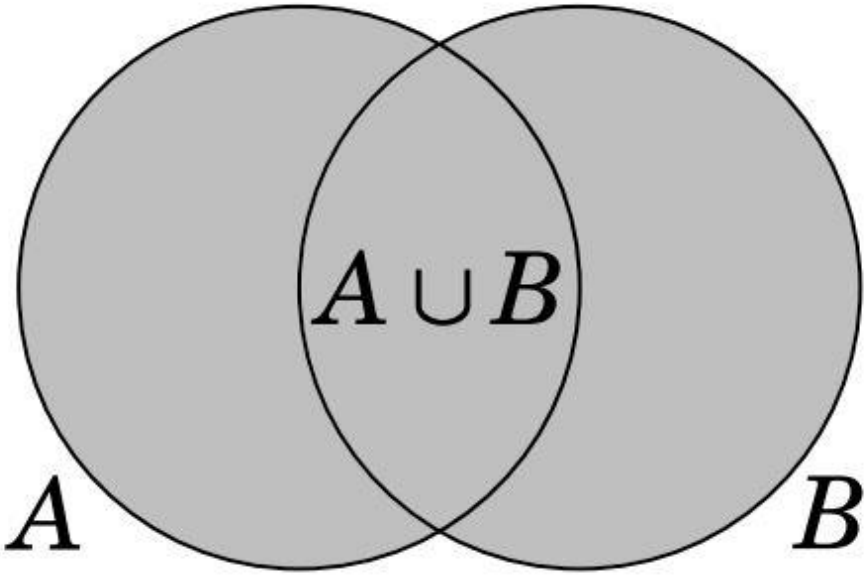
Exerciții cu mulțimi în PYTHON

Operații de bază cu mulțimi

Reuniunea a două mulțimi:

$$A \cup B = \{x \mid x \in A \text{ sau } x \in B\}$$

Reprezentare cu diagrame Venn:

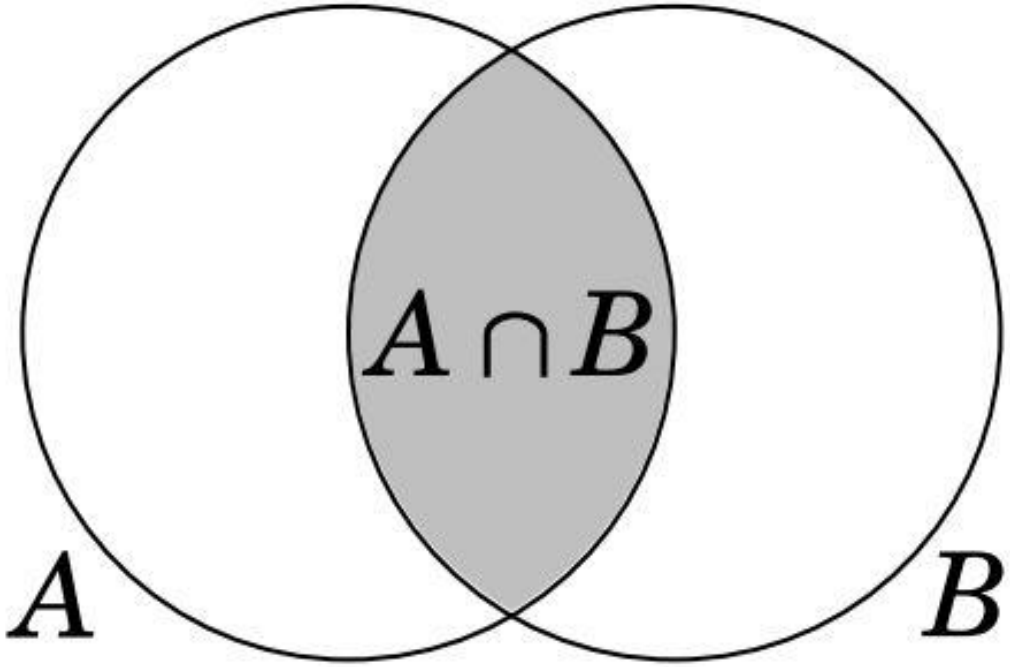


https://en.wikipedia.org/wiki/Venn_diagram

Operații de bază cu mulțimi

Intersecția a două mulțimi:

$$A \cap B = \{x \mid x \in A \text{ și } x \in B\}$$

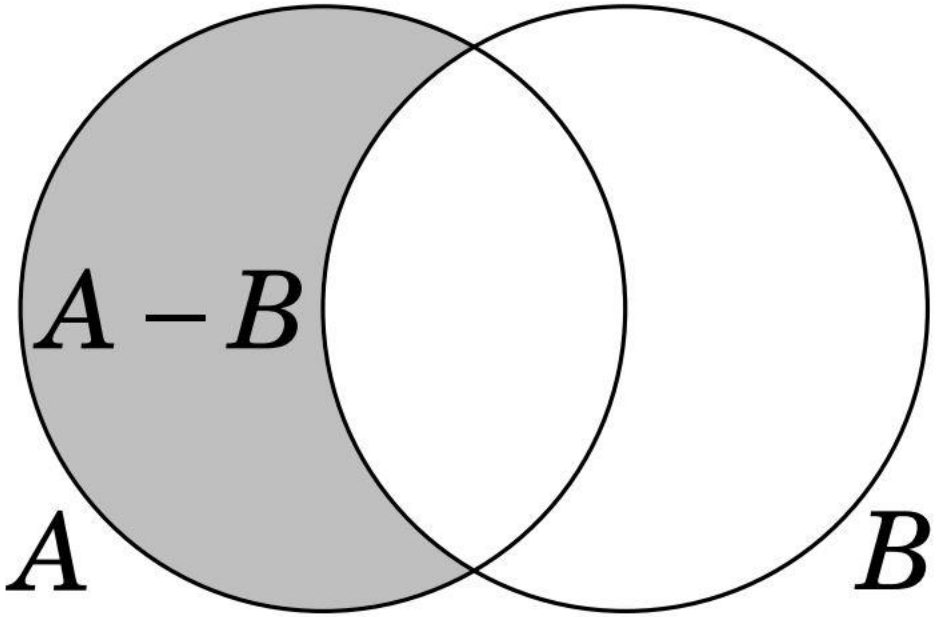


https://en.wikipedia.org/wiki/Venn_diagram

Operații de bază cu mulțimi

Diferența a două mulțimi:

$$A \setminus B = \{x \mid x \in A \text{ și } x \notin B\}$$



https://en.wikipedia.org/wiki/Venn_diagram

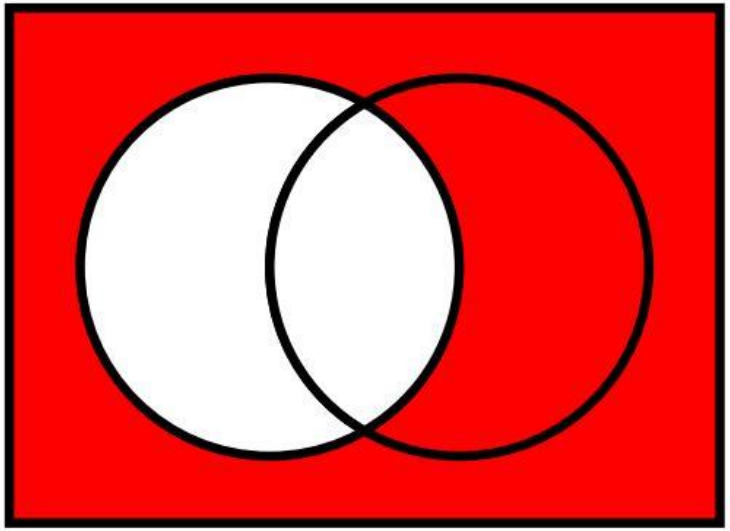
Operații de bază cu mulțimi

Uzual, discutăm într-un context: avem un univers U al tuturor elementelor la care ne-am putea referi.

Complementul unei mulțimi (față de universul U):

$A^C = \{x \in U \mid x \notin A\} = U \setminus A$ *(notat și* $\bar{A}$*)*

.



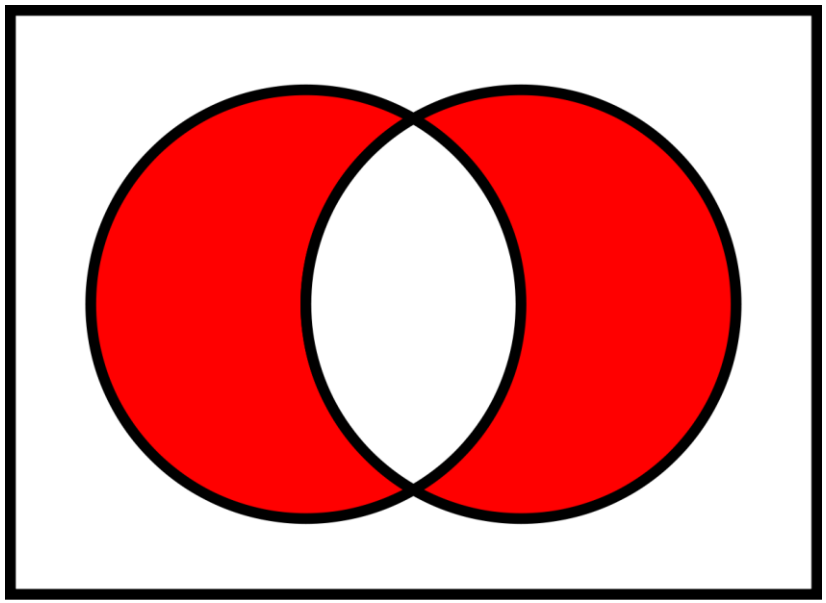
https://en.wikipedia.org/wiki/Venn_diagram

Operații de bază cu mulțimi

Diferența simetrică a două mulțimi:

$$A \Delta B = (A \setminus B) \cup (B \setminus A)$$

.



https://en.wikipedia.org/wiki/Venn_diagram

Operații de bază cu mulțimi

Dacă fixăm universul U al elementelor, putem reprezenta orice mulțime $S \subseteq U$ prin *funcția caracteristică*

$$f_S : U \to \mathrm{B}: f(x) = \begin{cases} True \text{ dacă } x \in S \\ False \text{ altfel (dacă } x \notin S) \end{cases}$$

.

Algebra Booleană a mulțimilor

Noțiune datorată matematicianului George Boole (sec. 19)

Operațiile unei algebre Boolene (aici ∪ și ∩) satisfac proprietățiile:

Comutativitate: $A \cup B = B \cup A$ $\quad\quad$ $A \cap B = B \cap A$

Asociativitate:

$(A \cup B) \cup C = A \cup (B \cup C)$ și $(A \cap B) \cap C = A \cap (B \cap C)$

Distributivitate: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ și

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Algebra Booleană a mulțimilor

Operațiile unei algebre Boolene (aici ∪ și ∩) satisfac proprietățiile:

Identitate: există două valori (aici ∅ și universul *U*) astfel ca:

$$A \cup \emptyset = A \qquad A \cap U = A$$

Complement: orice *A* are un complement A^c (sau $\bar{A}$) astfel ca:

$$A \cup A^c = U \qquad A \cap A^c = \emptyset$$

Algebra Booleană a mulțimilor

Alte proprietăți (pot fi deduse din cele de mai sus):

Idempotență: $A \cup A = A$ $\quad\quad$ $A \cap A = A$

Absorbție: $A \cup (A \cap B) = A$ $\quad\quad$ $A \cap (A \cup B) = A$

Dublu complement: $(A^c)^c = A$

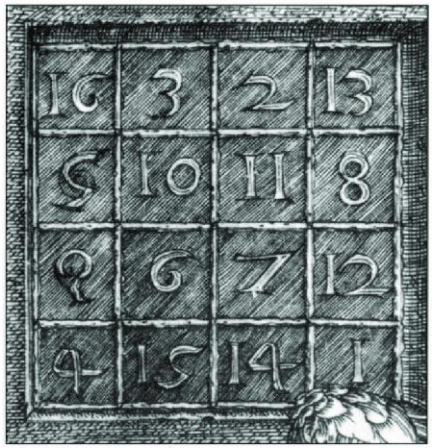
Algebra Booleană a mulțimilor

Alte proprietăți (pot fi deduse din cele de mai sus):

Complementele elementelor identitate: $\emptyset^c = U$
$U^c = \emptyset$

Limită universală: $A \cup U = U \quad A \cap \emptyset = \emptyset$

Legile lui de Morgan:
$(A \cup B)^c = A^c \cap B^c$ $\qquad$ $(A \cap B)^c = A^c \cup B^c$
Vom revedea aceste legi la *logica propozițională*.



Ce sunt mulțimile?

Operații de bază cu mulțimi

Mulțimile, fundament al matematicii

Mulțimile în PYTHON

Operații cu mulțimi în PYTHON

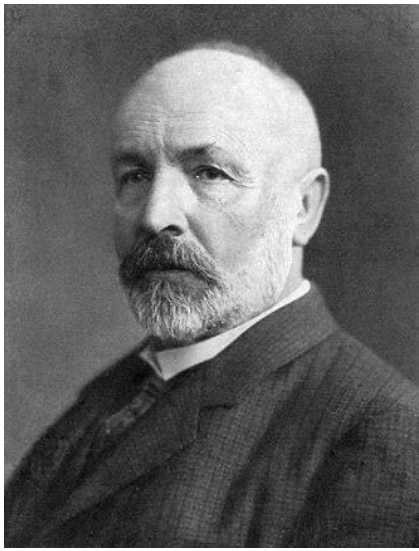
Exerciții cu mulțimi în PYTHON

Mulțimile, fundament al matematicii

Mulțimea este unul dintre cele mai *importante concepte* ale matematicii moderne.

Georg Cantor (1874) - a creat teoria mulțimilor, care a devenit *o teorie fundamentală a matematicii* în lucrarea „Uber eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen" („*Despre o proprietate a colecției tuturor numerelor reale algebrice*").

Cunoscută ca *teoria naivă a mulțimilor*.



https://ro.wikipedia.org/wiki/Georg_Cantor

Mulțimile, fundament al matematicii

Cantor a stabilit importanța *corespondențelor unu-la-unu* între membrii a două mulțimi, a definit *mulțimile infinite* și pe cele *bine ordonate*, și a demonstrat că numerele reale sunt mult *mai numeroase* decât numere naturale.

Metoda de demonstrație a teoremelor elaborate de Cantor implică existența unui „*infinit de infinituri*”.

El a definit numerele cardinale și ordinale și aritmetica lor.

Opera lui Cantor este de mare interes filosofic.

https://ro.wikipedia.org/wiki/Georg_Cantor

Mulțimile, fundament al matematicii

Practic toată matematica poate fi formalizată în *teoria mulțimilor*.

(sau în *logică*, de care e strâns legată, după cum vom vedea)

Exemplu: *o pereche* (ordonată) poate fi definită ca:

(a, b) ={{a} , {a, b}} - cum putem extrage pe *a* și *b* din (*a, b*)?

Intersecție, diferență

(1921, Kazimierz Kuratowski - matematician polonez ce a adus contribuții importante în topologie, teoria mulțimilor și teoria grafurilor.)

Însă, pornind de la definiții *imprecise*, în limbaj natural, în teoria naivă a mulțimilor apar *paradoxuri*.

https://ro.wikipedia.org/wiki/Georg_Cantor

Paradoxul lui Russell

Fie *R* mulțimea tuturor mulțimilor care nu se conțin pe ele însele:

$$R = \{X \mid X \notin X\}$$

Mulțimea *R* se conține pe ea însăși?

- dacă $R \in R$, pentru a satisface condiția de definiție, avem $R \notin R$.
- dacă $R \notin R$, atunci *R* satisface condiția, deci $R \in R$: *paradox*!

O formulare intuitivă (paradoxul bărbierului):

- Bărbierul bărbierește exact oamenii care nu se bărbieresc singuri. Bărbierul se bărbierește pe el însuși sau nu?

Paradoxul lui Russell

Paradoxul a pus probleme serioase formalizării logicii matematice.

Poate fi *evitat* în mai multe feluri, impunând *restricții* asupra modului în care se poate defini o mulțime.

de ex.: Nu putem defini o mulțime doar printr-o proprietate $P(x)$, trebuie să *specificăm universul* din care își poate lua elementele:

$$R = \{X \mid X \subseteq U \text{ și } X \notin X\}$$

Teoria axiomică a mulțimilor

O *axiomă* e o propoziție presupusă adevărată. E un punct de plecare pentru un raționament.

Sistemele axiomatice au fost dezvoltate pentru a evita paradoxurile din *teoria naivă* a mulțimilor (cu noțiuni definite în limbaj natural)

Cel mai răspândit: *sistemul Zermelo-Fraenkel* (1907-1930).

Teoria axiomică a mulțimilor

Câteva axiome:

Axioma extensionalității:

Două mulțimi sunt egale dacă și numai dacă au aceleași elemente

(dacă fiecare element al lui *A* e și un element al lui *B*, și reciproc)

$$\forall A, \forall B\ (A = B \Leftrightarrow \forall c\ (c \in A \Leftrightarrow c \in B))$$

Axioma mulțimii vide (existență):

Există o mulțime care nu are niciun element

$$\exists E \forall X\ \neg(X \in E\ )$$

Teoria axiomică a mulțimilor

Axioma regularității (a fundației)

Orice mulțime nevidă are un element $x \in A$ disjunct de ea:

$x \cap A = \emptyset$

$$\forall X\,(X \neq \emptyset) \Rightarrow \exists Y\,(Y \in X \ \wedge \ \neg\exists Z\,(Z \in X \ \wedge \ Z \in Y\,))$$

Rezultă că nu există un șir infinit $A_0, A_1, \ldots A_n \ldots$ astfel încât

$$A_0 \ni A_1 \ \ni \ \ldots \ni \ A_n \ \ni \ \ldots$$

($\{A_0, A_1, \ldots\}$ ar fi o astfel de mulțime)

Rezultă că *nicio mulțime nu se poate avea ca element*, $X \notin X$,

altfel $X \ni X \ni X \ldots$ ar fi un astfel de șir

Intuitiv: orice mulțime e formată din elemente (posibil mulțimi) mai simple, care la rândul lor conțin elemente mai simple, până ajungem la *elemente fundamentale*

⇒ *elimină* paradoxul lui Russell

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.

Cardinalul reuniunii, intersecției, diferenței

Pentru mulțimi *finite*:

Legea reuniunii:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Legea diferenței:

$$|A \setminus B| = |A| - |A \cap B|$$

Mulțimea submulțimilor

Mulțimea submulțimilor (engl. *power set*) unei mulțimi S, notată $\mathrm{P}(S)$ (uneori 2^S):

$$\mathrm{P}(S) = \{X \mid X \subseteq S\}$$

Exemplu, pentru $S = \{a, b, c\}$, avem:

$\mathrm{P}(S) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

Dacă S e finită, atunci $|\mathrm{P}(S)| = 2^{|S|}$

Tupluri și produs cartezian

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)

Tupluri și produs cartezian

Produsul cartezian a două mulțimi e mulțimea perechilor

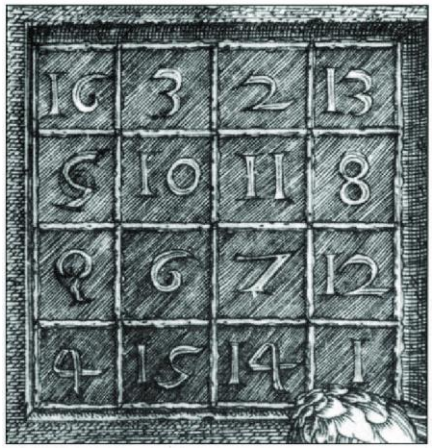
$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

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 sunt mulțimile?

Operații de bază cu mulțimi

Mulțimile, fundament al matematicii

Mulțimile în PYTHON

Operații cu mulțimi în PYTHON

Exerciții cu mulțimi în PYTHON

Mulțimile în PYTHON

Mulțimile sunt colecții care rețin mai multe elemente într-o singură variabilă.

Sunt una dintre *colecțiile de bază* în PYTHON (pe lângă Liste, Tupluri și Dicționare)

Mulțimile sunt colecții:

- *neordonate*
- *neindexate*
- *nu permit duplicate* printre elemente
- pot conține doar elemente *nemodificabile* (eng. *immutable*)

Mulțimile în PYTHON

Pentru *a crea* o mulțime vom enumera elementele mulțimii între *acolade* { } sau vom folosi constructorul *set()*

multime = {1, 2, 3, 4, 5}

multime2 = set ((1, 2, 3, 4, 5))

multime3 = set ([1, 2, 3, 4, 5])

Mulțimile în PYTHON

Pentru *a crea o mulțime vidă* vom putea folosi doar funcția *set()*, dacă definim o mulțime vidă cu două acolade, PYTHON o va interpreta ca fiind dicționar.

```
X = set()
print(type(X))
# <class 'set'>

Y = {}
print(type(Y))
#<class 'dict'>
```

Mulțimile în PYTHON

Ordinea elementelor nu contează. La afișare elementele pot apărea în *ordine diferită*.

```
multime = {1, 11, 4, 5, 3, 2}
print(multime)

# {1, 2, 3, 4, 5, 11}
# {1, 2, 3, 4, 11, 5}
# {1, 3, 4, 2, 11, 5}
```

Mulțimile în PYTHON

Elementele duplicate vor fi reținute *o singură dată*

```
multime = {1, 11, 4, 5, 3, 2, 1, 1, 2}
print(multime)

# {1, 2, 3, 4, 5, 11}
```

Mulțimile în PYTHON

Pentru a afla numărul de elemente dintr-o mulțime putem folosi funcția *len()*

```
multime = {1, 11, 4, 5, 3, 2}
print(len(multime))

# 6
```

Mulțimile în PYTHON

Elementele unei mutimi pot avea *diferite tipuri de date*:

zile = {“luni”, “marti”, “miercuri”}
numere = {1, 2, 3}
valori = {True, False}

O mulțime poate conține diferite tipuri de date *în același timp*:

multime = {“luni”, 1, True}

Mulțimile în PYTHON

Elementele unei mușimi sunt nemodificabile (eng. immutable)

Un tuplu poate fi element al unei mulțimi:

x = {3, 4, (1, 2, 3)}

O listă (sau un dicționar) nu poate fi element:

A={3, [4, 5,6]}

Va genera eroare

TypeError: unhashable type: 'list'

Accesul la elementele mulțimii

Accesul la elementele mulțimii *nu se poate face prin index*.

Putem verifica dacă un element este într-o mulțime cu *in*:

```
if (x in {1, 2, 3})
        print(”elementul face parte din multime”)
else
        print(”elementul nu face parte din multime”)
```

Accesul la elementele mulțimii

Putem parcurge element cu element cu *for in*:

```
multime = {1, 11, 4, 5, 3, 2}
for x in multime:
        print(x)
```

Afișează:

1
2
3
4
5
11

Adăugarea elementelor noi

Odată ce o mulțime este creată, elementele sale nu se pot modifica, în schimb *se pot adăuga* sau *șterge* elemente din mulțime.

Putem adăuga elemente noi în mulțime cu metoda *add()*

```
multime = {1, 11, 4, 5, 3, 2}
multime.add(29)
print(multime)
# {1, 2, 3, 4, 5, 11, 29}
```

Adăugarea elementelor noi

Putem adăuga *toate elementele unei alte mulțimi* în mulțimea curentă cu metoda *update()*

```
zile = {"luni","marti","miercuri"}
zile_weekend = {"sambata","duminica"}
zile.update(zile_weekend)
print(zile)
```

```
#{'duminica', 'marti', 'miercuri', 'luni', 'sambata'}
```

Ștergerea elementelor

Putem șterge elementedintr-o mulțime utilizând metodele remove() sau discard()

```
zile = {"luni","marti","miercuri"}
zile.remove("marti")
zile.discard("miercuri")
print(zile)

# {'luni'}
```

Ștergerea elementelor

Metoda *remove()* va genera eroare dacă se apelează cu un element inexistent, în timp ce metoda *discard()* nu va genera eroare.

```
zile = {"luni","marti","miercuri"}
zile.remove("joi")          # va genera eroare
zile.discard("joi")         #nu va genera eroare
print(zile)
```

Ștergerea elementelor

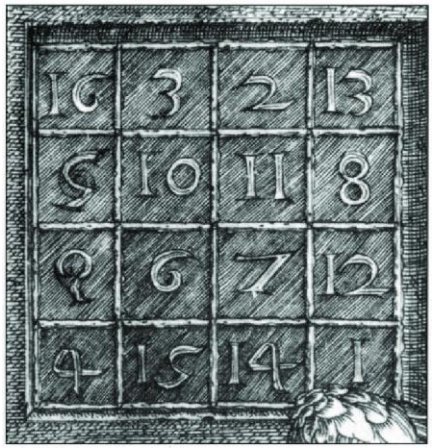
Pentru a șterge toate elementele unei mulțimi folosim metoda *clear()*

```
zile = {"luni","marti","miercuri"}
zile.clear()
print(zile)

# set()
```

Ștergerea elementelor

Pentru a șterge complet mulțimea putem folosi del

```
zile = {"luni","marti","miercuri"}
del zile
print(zile)

# print(zile)
#NameError: name 'zile' is not defined
```



Ce sunt mulțimile?

Operații de bază cu mulțimi

Mulțimile, fundament al matematicii

Mulțimile în PYTHON

Operații cu mulțimi în PYTHON

Exerciții cu mulțimi în PYTHON

Operații cu mulțimi

Putem calcula reuniunea a două mulțimi cu metoda *union()*. Metoda union() va crea *o nouă mulțime* care va conține elementele ambelor mulțimi.

```
multime1 = {"a", "b", "c"}
multime2 = {1, 2, 3}
multime3 = multime1.union(multime2)
print(multime3)

# {'a', 1, 'b', 2, 3, 'c'}
```

Operații cu mulțimi

Putem calcula intersecția a două mulțimi cu metoda *intersection()*.

```
multime1 = {2, 3, 4, 5}
multime2 = {1, 2, 3}
multime3 = multime1.intersection(multime2)
print(multime3)

# {2, 3}
```

Operații cu mulțimi

Putem calcula diferența a două mulțimi cu metoda *difference()*.

```
multime1 = {2, 3, 4, 5}
multime2 = {1, 2, 3}
multime3 = multime1.difference(multime2)
print(multime3)

# {4, 5}
```

Operații cu mulțimi

Putem calcula diferența simetrica a două mulțimi cu metoda *symmetric_difference()*.

```
multime1 = {2, 3, 4, 5}
multime2 = {1, 2, 3}
multime3 = multime1.symmetric_difference(multime2)
print(multime3)

# {1, 4, 5}
```

Operații cu mulțimi

Metodele *union(), intersection(), difference()* pot fi folosite și cu mai multe argumente:

```
A = {1, 2, 3, 4}
B = {2, 3, 4, 5}
C = {3, 4, 5, 6}
D = {4, 5, 6, 7}
E = A.union(B, C, D)          # E = {1, 2, 3, 4, 5, 6, 7}
F = A.intersection(B, C)      # F = {3, 4}
G = A.difference(C, D)        # G = {1, 2}
```

Metoda *symmetric_difference()* are doar un singur argument.

Operații cu mulțimi

Putem verifica dacă o mulțime este submulțime sau supramulțime pentru o altă mulțime cu metodele issubset() și issuperset()

A = {1, 2, 3, 4}

B = {2, 3}

```
print(B.issubset(A))   # True
print(A.issubset(B))   # False

print(B.issuperset({1, 2, 3, 4}))   # False
print(A.issuperset({1, 2, 3, 4}))   # True

print(A.issubset(A))    # True
```

Operații cu mulțimi

În PYTHON putem folosi și *operatori pentru lucrul cu mulțimi*:

Reuniune	*operatorul |*	*C = A | B*
Intersecție	*operatorul &*	*D = A & B*
Diferența	*operatorul –*	*E = A - B*
Diferența simetrică	*operatorul ^*	*F = A ^ B*
Submulțime	*operatorii < și <=*	*A < B*
Supramulțme	*operatorii > și >=*	*B >= A*

Operații cu mulțimi

Exemplu:

A = {1, 2, 3, 4}

B = {3, 4, "a", "b", "c"}

C = A | B *#C = {1, 2, 3, 4, 'a', 'b', 'c'}*

D = A & B *#D = {3, 4}*

E = A – B *#E = {1, 2}*

F = A ^ B *#F = {1, 2, 'a', 'b', 'c'}*

Operații cu mulțimi

Dacă vrem ca rezultatul unei operații să se regăsească în *mulțimea curentă* și să nu genereze *o nouă mulțime* cu rezultatul operației vom folosi metodele: *update()*, *intersection_update()*, *difference_update()*, *symmetric_difference_update()* astfel:

```
A = {1, 2, 3, 4}
B = {2, 3, 4, 5}
A.update(B)                    # A = {1, 2, 3, 4, 5}
A.intersection_update(B)       # A = {2, 3, 4, 5}
A.difference_update(C)         # A = {2}
```

Atenție, fiecare operație de mai sus va modifica mulțimea A, iar următoarea metodă va folosi *noua componență* a lui A

Operații cu mulțimi

Dacă vrem ca o mulțime să aibă ca *element o altă mulțime* și scriem:

$$A = \{\{1,2,3\}, \{7\}\}$$

Vom primi eroarea *TypeError: unhashable type: 'set‘*

Nu ne lasă să scriem această sintaxă pentru că elementele mulțimii A trebuie să fie obiecte *neschimbabile* (eng. *immutable*).

Pentru a putea face astfel de operații, PYTHON ne pune la dispoziție o colecție mulțime care nu permite modificări odată ce a fost creată: *frozenset*

Operații cu mulțimi

Mulțimile de tip *frozenset* pot folosi toate metodele și operatorii pentru tipul de date *set cu excepția* celor care modifică structura mulțimii.

Exemplu:

```
A = frozenset({'a', 'b', 'c'})
print(A)                                  # frozenset({'a', 'b', 'c'})
print(len(A))                             # 3
print(A & {'a', 'b','z'})                 # frozenset({'a', 'b'})
A = {frozenset({1,2,3}), frozenset({7})}

print(A.add('d'))        # acest apel va genera eroarea:
AttributeError: 'frozenset' object has no attribute 'add'
```

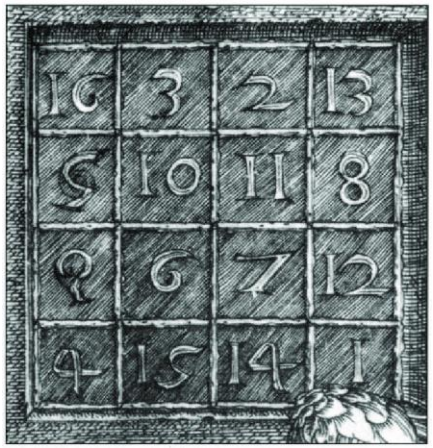
Operații cu mulțimi

Funcțiile *map(), filter() și reduce()* discutate pentru lucrul cu liste se pot folosi similar și pentru mulțimi. Cele 3 funcții au ca parametru un *iterabil*.

Elementele unei mulțimi le putem parcurge (în stilul programării funcționale) cu ajutorul funcției *reduce():*

```
import functools
M = {1, 2, 3, 4}
functools.reduce(lambda acc, elem: print(elem), M, 0)
```

Ultimul argument al funției reduce (0, în exemplul de mai sus) indică de la ce *valoare inițială* să se parcurgă funcția.



Ce sunt mulțimile?

Operații de bază cu mulțimi

Mulțimile, fundament al matematicii

Mulțimile în PYTHON

Operații cu mulțimi în PYTHON

Exerciții cu mulțimi în PYTHON

Exerciții cu mulțimi

1. Scrieți o funcție care returnează mulțimea tuturor divizorilor unui număr pozitiv *n* dat ca argument. Parcurgerile se vor realiza cu ajutorul funcților recursive.

Exerciții cu mulțimi

1. Scrieți o funcție care returnează mulțimea tuturor divizorilor unui număr pozitiv *n* dat ca argument. Parcurgerile se vor realiza cu ajutorul funcților recursive.

```
def multimea_divizorilor(n, A=set(), i=2):
    if(i>n/2):
        return A
    else:
        if(n % i == 0):
            A.add(i)
        return multimea_divizorilor(n, A, i+1)

print(multimea_divizorilor(20))
```

Exerciții cu mulțimi

O rezolvare mai eficientă a problemei precedente:

```
import math as m
def multimea_divizorilor(n, A=set(), i=2):
    if(i>m.sqrt(n)):
        return A
    else:
        if(n % i == 0):
            A.update({i, int(n/i)})
        return multimea_divizorilor(n, A, i+1)

print(multimea_divizorilor(20))
```

Exerciții cu mulțimi

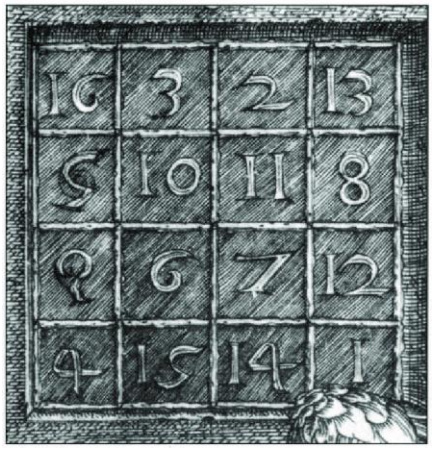
2. Implementați funcția standard filter care ia ca parametri o funcție booleană (condiție, predicat) f și o mulțime s și returnează mulțimea elementelor din s care satisfac funcția f. Parcurgeți mulțimea inițială cu funcția reduce().

Mai jos e un exemplu de folosire a funcției *filter()*, pe care trebuie să o implementăm noi:

```
def impar(x):
    return x % 2

B=set(filter(impar, {1, 2, 3, 4, 5, 6}))
print(B)
# {1, 3, 5}
```

Exerciții cu mulțimi

```
import functools
def my_filter(f, A, B=set()):
    def functie(acc, elem):
        if(f(elem)):
            B.add(elem)
        return f(elem)
    functools.reduce(functie, A, 0)
    return B

def impar(x):
    return x % 2
B=set(my_filter(impar, {1, 2, 3, 4, 5, 6}))
print(B)
# {1, 3, 5}
```



Vă mulțumesc!

Bibliografie

- 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)