

# **Fundamentele programării**

Prof. Dr. Czibula Istvan

Lect. Mircea Ioan Gabriel

Drd. Mihai Andrei

Drd. Briciu Anamaria

C. d. asociat Timiș Răzvan

C. d. asociat Mateș Ciprian

## **Orar**

**Curs:** 2 ore / săptămână

**Seminar:** 2 ore / săptămână

**Laborator:** 2 ore / săptămână

## **Pagina WEB**

<http://cs.ubbcluj.ro/~istvanc/fp>

Email: [istvanc@cs.ubbcluj.ro](mailto:istvanc@cs.ubbcluj.ro)

**Trimiteți emailuri doar de la adrese ubbcluj.ro.**

## Objective

- Cunoașterea conceptelor fundamentale programării
- Introducere concepte de bază legate de ingineria software (design, arhitectură, implementare și întreținere)
- Înțelegerea instrumentelor software folosite în dezvoltarea de aplicații
- Învățarea limbajului Python și utilizarea lui pentru implementarea, testarea, rularea, depanarea de programe.
- Însușirea/Îmbunătățirea stilului de programare.

Programming in the small	vs	Programming in the large
Algoritmica/Programare	vs	Inginerie Software

# Conținut

1. Introducere în procesul de dezvoltare software
2. Programare procedurală
3. Programare modulară
4. Tipuri definite de utilizator - Object based programming
5. Principii de dezvoltare – Arhitectură stratificată
6. Principii de dezvoltare – Șabloane GRASP, diagrame UML
7. Testarea și inspectarea programelor
8. Recursivitate
9. Complexitatea algoritmilor
10. Algoritmi de căutare
11. Algoritmi de sortare
12. Backtracking
13. Greedy, Programare dinamica
14. Recapitulare

## **Evaluare**

Lab (30%) - o notă pe activitatea de laborator din timpul semestrului.

T (30%) - examen practic (în sesiune)

E (40%) - examen scris (în sesiune)

### **Pentru a intra în examen:**

Minim 12 prezente la laborator

Minim 10 prezente la seminar

### **Pentru promovare trebuie să aveți cel puțin nota 5 la toate (Lab,T,E $\geq$ 5)**

Toate activitățile sunt obligatorii

Dacă nu obțineți nota 5 la laborator nu puteți intra în examen în sesiunea normală.

### **Restanțe**

În sesiunea de restanțe puteți preda laboratoare (maximă este 5).

Se poate re-susține examenul practic

Se poate re-susține examenul scris

# **Curs 1. Procesul de dezvoltare software**

- Ce este programarea
- Elemente de bază al limbajului Python
- Proces de dezvoltare bazat pe funcționalități

# Ce este programarea

## Hardware / software

**Hardware** - *computere*(desktop, laptop, etc) și alte dispozitive (mobile, atm, etc)

**Software** - *programe sau sisteme* ce rulează pe hardware

**Limbaj de programare** – Notății și reguli pentru scrierea de programe (sintaxă și semantică)

**Python:** Limbaj de programare de nivel înalt (high level programming language).

**Interpretor Python:** un program care permite rularea/interpretarea programelor scrise în limbajul Python.

**Biblioteci Python:** Funcții, module, tipuri de date disponibile în Python, scrise de alți programatori

Program 1 - Hello world
<pre>print ('Hello world')</pre>

## Python

Download: [python.org](https://python.org) - versiunea 3.x

După instalare:

- interpretor python – executa programe scrise in python
- IDLE Python – un mic mediu de dezvoltare pentru python

Autor principal limbaj (1989): **Guido van Rossum**

Rol curent: benevolent dictator for life (BDFL) – nu mai e cazul

**PEP** – Python Enhancement Proposal – design document providing information to the Python community, or describing a new feature for Python or its processes or environment.

PEP 20 The Zen of Python – câteva principii de baza

PEP 8 Style Guide for Python code

Documentație: [docs.python.org](https://docs.python.org)

<https://docs.python.org/3/reference/index.html> - core syntax&semantics

<https://docs.python.org/3/library/index.html> - standard library

## Ce fac computerele

- Stochează date
  - Memoria internă
  - Memoria externă (hard, stick, CD, etc)
- Operează
  - procesor
- Comunică
  - Prin tastatură, mouse, ecran
  - Conexiuni de tip rețea

## Informații și date

**Date** - o colecție de simboluri stocate într-un computer (Ex. 123 decimal sau șirul de caractere 'abc') sunt stocate folosind reprezentarea binară

**Informații** - interpretarea unor date (Ex. 123, 'abc')

### Procesarea datelor și informațiilor

- Dispozitivele de intrare transformă informațiile în date (ex. 123 citit de la tastatură)
- Datele sunt stocate în memorie (ex. 1111011 pentru numărul 123)
- Dispozitivele de ieșire produc informații din date

### Operații de bază ale procesoarelor

- În reprezentare binară
- Operații (and, or, not; add, etc)



## Elemente de bază ale unui program Python

### Program 2 - Adding two integers

```
# Reads two integers and prints the sum of them
a = input("Enter the first number: ")
b = input("Enter the second number: ")
c = int(a) + int(b)
print("The sum of ", a, " + ", b, " is ", c)
```

## Elemente lexicale

Un program Python este alcătuit din mai multe linii de cod

### Comentarii

- încep cu # și țin până la sfârșitul liniei
- încep cu ''' și țin mai multe rânduri, până la un nou '''

**Identificatori:** secvențe de caractere (litere, cifre, \_) care încep cu o literă sau cu \_

**Literali:** notații pentru valorile constante sau pentru tipuri definite de utilizator

# Modelul de date

Toate datele într-un program Python – **obiecte**

Un obiect are :

- **o identitate** – adresa lui în memorie
- **un tip** – care determină operațiile posibile precum și valorile pe care le poate lua obiectul
- **o valoare.**

Odată creat, **identitatea** și **tipul** obiectului nu mai pot fi modificate. Valoarea unor obiecte se poate modifica.

- Obiecte **mutable** - se poate modifica
- Obiecte **ne-mutable** – nu se poate modifica, orice operație efectuată creează un nou obiect

## Tipuri de date standard

**Tipul de date** definește **domeniul** de valori posibile și **operațiile** permise asupra valorilor din domeniu.

**Numerice** – Numerele sunt imutabile – odată create valoare nu se mai poate schimba (operațiile creează noi obiecte).

**int** (numere întregi):

- numerele întregi (pozitive și negative), dimensiune limitat doar de memoria disponibilă
- Operații: +, -, \*, /, //, \*\*, % comparare:==,!=,<, > operații pe biți: |, ^, &, <<, >>, ~
- Literal: 1, -3

**bool** (boolean):

- Valorile True și False.
- Operații: and, or, not
- Literal: False, True; 0, 1

**float** (numere reale):

- numerele reale (dublă precizie)
- Operații: +, -, \*, / comparare:==,!=,<, >
- Literal: 3.14

**NoneType**

- O singură valoare: None
- Operații:==, !=
- Literal: None

## Tipuri de date standard

### Secvențe:

- Mulțimi finite și ordonate, indexate prin numere ne-negative.
- Dacă `a` este o secvență atunci:
  - `len(a)` returnează numărul de elemente;
  - `a[0], a[1], ..., a[len(a)-1]` elementele lui `a`.
- Exemplu: `[1, 'a']`
- Exemplu de operații: accesare elemente, `+`, `*` cu un scalar, etc.

**String:** `'abc'`, `"abc"` - este o secvență imutabilă de caractere Unicode .

**List:** `[2,3]` , `[1, 'a', [1, 3]]` – secvență mutabilă

**Tuple:** `(2,3)`, `(1,'a',(1,3))` – secvență mutabilă

**Dicționar:** `{ 'num': 1, 'denom': 2 }`

- conține perechi (cheie – valoare)
- fiecare cheie apare o singură dată
- operația de găsim a unei valori după o cheie foarte eficient

### Funcție:

- funcțiile in Python pot fi tratate ca orice alt tip de valoare
- Operații: singura operație permisă este apelul funcției: `func(arg)`
- obiecte de tip funcție se creează prin definirea de funcții
- funcțiile in Python pot fi asignate la variabile, transmise ca parametri pentru o altă funcție, returnate dintr-o funcție.

# Liste

operații:

- creare [7, 9]
- accesare valori, lungime (**index**, **len**), modificare valori (**listele sunt mutabile**), verificare dacă un element este în lista (**2 in [1, 2, 'a']**)
- ștergere inserare valori (**append**, **insert**, **pop**) del a[3]
- slicing, liste eterogene
- listele se pot folosi în for
- lista ca stivă(**append**, **pop**)
- folosiți instrucțiunea **help(list)** pentru mai multe detalii despre operații posibile

<pre># create a = [1, 2, 'a'] print (a) x, y, z = a print(x, y, z)  # indices: 0, 1, ..., len(a) - 1 print (a[0]) print ('last element = ', a[len(a)-1]) # lists are mutable a[1] = 3 print (a)</pre>	<pre># slicing print (a[:2]) b = a[:] print (b) b[1] = 5 print (b) a[3:] = [7, 9] print(a) a[:0] = [-1] print(a) a[0:2] = [-10, 10] print(a)</pre>
<pre># lists as stacks stack = [1, 2, 3] stack.append(4) print (stack) print (stack.pop()) print (stack)</pre>	<pre># nesting a = [1, [1, 1, 9], 9] print (a) b = [1, 1, 9] c = [1, b, 9] print (c)</pre>
<pre>#generate lists using range l1 = range(10) print (list(l1)) l2 = range(0,10) print (list(l2)) l3 = range(0,10,2) print (list(l3)) l4 = list(range(9,0,-1)) print (l4)</pre>	<pre>#list in a for loop l = range(0,10) for i in l:     print (i)</pre>

# Tuplu

Sunt secvențe imutabile. Conține elemente, indexat de la 0

Operații:

- Crearea - packing (23, 32, 3)
- eterogen
- poate fi folosit in for
- unpacking

<pre># Tuples are immutable sequences # A tuple consists of a number # of values separated by commas  # tuple packing t = 12, 21, 'ab' print(t[0])  # empty tuple (0 items) empty = ()</pre>	<pre># tuple with one item singleton = (12,) print (singleton) print (len(singleton))  #tuple in a for t = 1,2,3 for el in t:     print (el)</pre>
<pre># sequence unpacking x, y, z = t print (x, y, z)</pre>	<pre># Tuples may be nested u = t, (23, 32) print(u)</pre>

# Dicționar

Un dicționar este o mulțime de perechi (cheie, valoare).

Cheile trebuie sa fie **imutable**.

Operații:

- creare {} sau { 'num': 1, 'denom': 2 }
- accesare valoare pe baza unei chei
- adăugare/modificare pereche (cheie, valoare)
- ștergere pereche (cheie, valoare)
- verificare dacă cheia există

<pre>#create a dictionary a = { 'num': 1, 'denom': 2 } print(a)  #get a value for a key print(a[ 'num' ])</pre>	<pre>#set a value for a key a[ 'num' ] = 3 print(a) print(a[ 'num' ])</pre>
<pre>#delete a key value pair del a[ 'num' ] print (a)</pre>	<pre>#check for a key if 'denom' in a:     print( 'denom = ', a[ 'denom' ]) if 'num' in a:     print( 'num = ', a[ 'num' ])</pre>

## Variabile

<b>Variabilă:</b> <ul style="list-style-type: none"><li>• nume</li><li>• valoare</li><li>• tip<ul style="list-style-type: none"><li>◦ domeniu</li><li>◦ operații</li></ul></li><li>• locație de memorie</li></ul>	<b>Variabilă in Python:</b> <ul style="list-style-type: none"><li>• nume</li><li>• valoare<ul style="list-style-type: none"><li>◦ tip</li><li>◦ domeniu</li><li>◦ operații</li><li>◦ locație de memorie</li></ul></li></ul>
---	---

Introducerea unei variabile într-un program – asignare

## Expresii

O combinație de valori, constante, variabile, operatori și funcții care sunt interpretate conform regulilor de precedență, calculate și care produc o altă valoare

Exemple:

- numeric :  $1 + 2$
- boolean:  $1 < 2$
- string : `'1' + '2'`

## Funcții utile:

**help**(instrucțiune) - ajutor

**id**(x) – identitatea obiectului

**dir**()

**locals**() / **globals**() - nume definite (variabile, funcții, module, etc)



# Instrucțiuni

Operațiile de bază ale unui program. Un program este o secvență de instrucțiuni

- **Atribuire/Legare**

- Instrucțiunea =.
- Atribuirea este folosit pentru a lega un nume de o variabilă
- Poate fi folosit și pentru a modifica un element dintr-o secvență mutabilă.
- Legare de nume:
  - `x = 1 #x is a variable (of type int)`
- Re-legare name:
  - `x = x + 2 #a new value is assigned to x`
- Modificare secvență:
  - `y = [1, 2] #mutable sequence`
  - `y[0] = -1#the first item is bound to-1`

- **Blocuri**

- Parte a unui program care este executată ca o unitate
- Secvență de instrucțiuni
- Se realizează prin indentarea liniilor (toate instrucțiunile indentate la același nivel aparțin aceluiași bloc)

# Instrucțiuni - If, While

```
if conditie:
    bloc de instructiuni
elif conditie:
    bloc de instructiuni
else:
    bloc de instructiuni
```

```
while conditie:
    bloc de instructiuni
    [break]
    [continue]
```

```
def gcd(a, b):
    """
    Return the greatest common divisor of two positive integers.
    """
    if a == 0: return b
    if b == 0: return a

    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a

print (gcd(7,15))
```

# Instrucțiuni – For

```
for el in secventa: #parcurgem element cu element
    bloc de instrucțiuni #el - element in secvența
    [break]
    [continue]
else:
    bloc de instrucțiuni #executat daca s-a dat break
```

```
#use a list literal
for i in [2,-6,"a",5]:
    print (i)

#using a variable
x = [1,2,4,5]
for i in x:
    print (i)

#using range
for i in range(10):
    print (i)

for i in range(2,100,7):
    print (i)

#using a string
s = "abcde"
for c in s:
    print (c)
```

# Parcuregere in Python

Pythonic	Programator C++/Java/C#/Pascal
<pre>for i in range(6):     print (i)</pre>	<pre>for i in [0,1,2,4,5]:     print (i)</pre>
<pre>x = [2,-6,"a",5] for el in x:     print (el)</pre>	<pre>x = [2,-6,"a",5] for i in range(len(x)):     print (x[i])</pre>
<pre>x = [2,-6,"a",5] for el in reversed(x):     print (el)</pre>	<pre>x = [2,-6,"a",5] for i in range(len(x)-1,-1,-1):     print (x[i])</pre>
<pre>x = [2,-6,"a",5] for i, el in enumerate (x):     print (i, "-&gt;", el)</pre>	<pre>x = [2,-6,"a",5] for i in range(len(x)):     print (i, "-&gt;", x[i])</pre>
<pre>#parcuregere 2 liste simultan x = [2,-6,"a",5] y = [2,-6,"a"] for elx, ely in zip (x,y):     print (elx, "&lt;-&gt;", ely)</pre>	<pre>#parcuregere 2 liste simultan x = [2,-6,"a",5] y = [2,-6,"a"] n = min(len(x),len(y)) for i in range(n):     print (x[i],y[i])</pre>

Transforming Code into Beautiful, Idiomatic Python <https://www.youtube.com/watch?v=OSGv2VnC0go>

**Dictionar:** dictionar = { 'num': 1, 'denom': 2 }

<pre>#parcuregere cheile din dictionar for cheie in dictionar:     print (cheie)</pre>
<pre>#parcuregere cheile din dictionar for cheie in dictionar.keys():     print (cheie)</pre>
<pre>#parcuregere valorile din dictionar for valoare in dictionar.values():     print (valoare)</pre>
<pre>#parcuregere perechile din dictionar for cheie,valoare in dictionar.items():     print (cheie,valoare)</pre>

# Cum se scriu programe

## Roluri în ingineria software

### Programator/Dezvoltator

- Folosește calculatorul pentru a scrie/dezvolta aplicații

### Client (stakeholders):

- Cel interesat/afectat de rezultatele unui proiect.

### Utilizatori

- Folosesc/rulează programul.

Un proces de dezvoltare software este o abordare sistematică pentru construirea, instalarea, întreținerea produselor software. Indică:

- Pașii care trebuie efectuați.
- Ordinea lor

Folosim la fundamentele programării: un proces de dezvoltare incrementală bazată pe funcționalități (simple feature-driven development process)

## Enunț (problem statement)

Enunțul este o descriere scurtă a problemei de rezolvat.

Calculator - Problem statement
Profesorul (client) are nevoie de un program care ajută <i>elevii</i> (users) sa invețe despre numere raționale. Programul ar trebui sa permita elevilor să efectueze operații aritmetice cu numere raționale

## Cerințe (requirements)

**Cerințele** definesc în detaliu de ce este nevoie în program din perspectiva clientului. Definește:

- Ce dorește clientul
- Ce trebuie inclus în sistemul informatic pentru a satisface nevoile clientului.

### Reguli de elaborare a cerințelor:

- **Cerințele exprimate corect asigură dezvoltarea sistemului conform așteptărilor clienților.** (Nu se rezolvă probleme ce nu s-au cerut)
- Descriu **lista de funcționalități** care trebuie oferite de sistem.
- Funcționalitățile trebuie să clarifice orice ambiguități din enunț.

## Funcționalitate

- O funcție a sistemului dorit de client
- descrie datele rezultatele și partea sistemul care este afectat
- este de dimensiuni mici, poate fi implementat într-un timp relativ scurt
- se poate estima
- exprimată în forma acțiune rezultat obiect
  - Acțiunea – o funcție pe care aplicația trebuie să o furnizeze
  - Rezultatul – este obținut în urma execuției funcției
  - Obiect – o entitate în care aplicația implementează funcția

Calculator – Listă de Funcționalități
F1. <b>Adună</b> un <i>număr rațional</i> în calculator.
F2. <b>Șterge</b> calculator.
F3. <b>Undo</b> – reface ultima operație (utilizatorul poate repeta această operație).

## Proces de dezvoltare incrementală bazată pe funcționalități

- Se creează lista de funcționalități pe baza enunțului
  - Se planifică iterațiile (o iterație conține una/mai multe funcționalități)
  - Pentru fiecare funcționalitate din iterație
    - Se face modelare – scenarii de rulare
    - Se creează o lista de taskuri (activități)
- Se implementează și testează fiecare activitate

**Iterație:** O perioadă de timp în cadrul căreia se realizează o versiune stabilă și executabilă a unui produs, împreună cu documentația suport

La terminarea iterației avem un program funcțional care face ceva util clientului

Exemplu: plan de iterații

Iteration	Planned features
I1	F1. <b>Adună</b> un <i>număr rațional</i> în calculator.
I2	F2. <b>Sterge</b> calculator.
I3	F3. <b>Undo</b> – reface ultima operație (utilizatorul poate repeta această operație).



## Modelare - Iteration modeling

La fiecare început de iterație trebuie analizat funcționalitatea care urmează a fi implementată.

Acest proces trebuie să asigure înțelegerea funcționalității și să rezulte un set de pași mai mici (work item/task), activități care conduc la realizarea funcționalității

Fiecare activitate se poate implementa/testa independent

### Iterația 1 - Adună un *număr rațional* în calculator.

Pentru programe mai simple putem folosi **scenarii de rulare** (tabelară) pentru a înțelege problema și modul în care funcționalitatea se manifestă în program. Un scenariu descrie interacțiunea între utilizator și aplicație.

**Scenariu pentru funcționalitatea de adăugare număr rațional**

	Utilizator	Program	Descriere
a		0	Tipărește totalul curent
b	1/2		Adună un număr rațional
c		1/2	Tipărește totalul curent
d	2/3		Adună un număr rațional
e		5/6	Tipărește totalul curent
f	1/6		Adună un număr rațional
g		1	Tipărește totalul curent
h	-6/6		Adună un număr rațional
i		0	Tipărește totalul curent

## Listă de activități

### Recomandări:

- Definiți o activitate pentru fiecare operație care nu este implementată deja (de aplicație sau de limbajul Python), ex. T1, T2.
- Definiți o activitate pentru implementarea interacțiunii program-utilizator (User Interface), ex. T4.
- Definiți o activitate pentru a implementa operațiile necesare pentru interacțiune utilizator cu UI, ex. T3.
- Determinați dependențele între activități (ex. T4 --> T3 --> T2 --> T1, unde --> semnifică faptul că o activitate depinde de o altă activitate).
- Faceți un mic plan de lucru (T1, T2, T3, T4)

T1	Determinare cel mai mare divizor comun (punctele g, l din scenariu)
T2	Sumă două numere raționale (c, e, g, i)
T3	Implementare calculator: init, add, and total
T4	Implementare interfață utilizator

## Activitate 1. Determinare cel mai mare divizor comun

### Cazuri de testare

Un **test case** conține un set de intrări și rezultatele așteptate pentru fiecare intrare.

Date: a, b	Rezultate: gcd (a, b): c, unde c este cel mai mare divizor comun
2 3	1
2 4	2
6 4	2
0 2	2
2 0	2
24 9	3
-2 0	ValueError
0 -2	ValueError

# Curs 1. Procesul de dezvoltare software

- Ce este programarea
- Elemente de bază al limbajului Python
- Proces de dezvoltare bazat pe funcționalități

## Curs 2. Programare procedurală

- Funcții în Python
- Cum se scriu funcții
- Dezvoltare dirijată de teste (Test Driven Development)

# Referințe

1. The Python language reference. <http://docs.python.org/py3k/reference/index.html>
2. The Python standard library. <http://docs.python.org/py3k/library/index.html>
3. The Python tutorial. <http://docs.python.org/tutorial/index.html>