Fundamentals of Programming

Guiding teachers

Lect. PhD CZIBULA Istvan
Assist. PhD BOCICOR Maria-Iuliana
Assist. PhD MOLNAR Arthur
PhD student Sirbu Adela
PhD student Rimovecz Mihai
PhD student Coroiu Adriana

Schedule

Lectures: 2 hours/week
Seminars: 2 hours/week
Labs: 2 hours/week

Course URL

Course page: **http://cs.ubbcluj.ro/~istvanc/fp**

Email: **istvanc@cs.ubbcluj.ro**

Please use your ubbcluj.ro email.
Do not send emails from other accounts (yahoo,gmail, etc)

Aims of the activity

- To know the key concepts of programming
- To know the basic concepts of software engineering (design, implementation and maintenance of software systems)
- To understand the basic software tools
- To learn Python programming language, and to get used to Python programming, running, testing, and debugging programs.
- To acquire and improve programming style.

Course content

1. Introduction to software development processes
2. Procedural programming
3. Modular programming
4. User defined types - Object based programming
5. Development principles – Layered architecture
6. Development principles – UML Diagrams
7. Program testing and inspection
8. Recursion
9. Algorithms complexity
10. Search algorithms and their complexity
11 Sort algorithms and their complexity
12. Division method
13. Backtracking method
14 Overview

Activity and grading

Lab	(30%)	Lab documentations and programms
T	(30%)	Practical test (in the regular session)
E	(40%)	Written exam (in the regular session)
S	0 - 0.5	Seminar activity

Attendance is compulsory to all the activities

Minimum grade >= 5 at T,E and Lab

Lecture 1. Introduction to development processes

- What is programming
- Basic elements of a Python program
- A simple feature-driven development process

What is programming

Hardware and software

Hardware - *computers* (desktop, mobile, etc) and related *devices*
Software - *programs* or *systems* which run on hardware
Programming language - notation that defines syntax and semantics of programs
Python: a high level programming language [1]. It is a great language for beginner programmers [3].
Python interpreter: a program which allow us to run/interpret new programs.
Python standard library: built-in functions and types [2].

Program 1 - Hello world
print (‘Hello world’)

What computers do

- storage and retrieval
    - internal memory
    - hard disk, memory stick
- operations
    - processor
- communication
    - keyboard, mouse, display
    - network connector

Information and data

Data - collection of symbols stored in a computer (e.g. 123 decimal number or ‘abc’ string are stored using binary representations)
Information - interpretation of data for human purposes (e.g. 123, ‘abc’)
Information and data processing

- Input devices produce data from information (e.g. 123 taken from keyboard)
- Data is stored in memory (e.g. binary representation of 123)
- Output devices produce information from data (e.g. 123 displayed from a binary representation)

Basic operations of processors

- binary representation
- operations on binary representations (and, or, not; add, etc)

Basic elements of a Python program

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)
```

Lexical elements

A Python program is divided into a number of **lines**.
Comments start with a hash (#) character and ends at the end of the line.
Identifiers (or **names**) are sequences of characters which start with a letter (a..z, A..Z) or an underscore (_) followed by zero or more letters, underscores, and digits (0..9).
Literals are notations for constant values of some built-in types.

Data model

All data in Python programs is represented by objects, an **object** being “the Python’s abstraction for data”.
An **object** has:

- an *identity* - we may think of of it as the object’s address in memory;
- a *type* - which determines the operations that the object supports and also defines the possible values;
- a *value*.

Once an object has been created, its *identity and type cannot be changed*.
The value of some objects can change. Objects whose values can change - **mutable objects**.
Objects whose values cannot be changed - **immutable objects**.

Standard types

Types classifies values. A type denotes a **domain** (a set of values) and **operations** on those values.

Numbers - Numbers are immutable - once created, their values cannot be changed.

int (integers):

- Represents the mathematical set of integers (positive and negative, unlimited precision)
- Operations: +, -, *, /, see [2, section 4.4].
- Literals: 1

bool (booleans):

- Represents the the truth values True and False.
- Operations: [2, section 4.1].
- Literals: False, True; 0, 1

float (reals):

- Represents the mathematical set of double precision floating point numbers.
- Operations: +, -, *, /, see [2, section 4.4].
- Literals: 3.14

Sequences:

- Represents the finite ordered sets indexed by non-negative numbers.
- Let a be a sequence.
    - len(a) returns the number of items;
    - a[0], a[1], …, a[len(a)-1] represent the set of items.
- Examples: [1, ‘a’]

Strings:

- A string is an immutable sequence;
- The items of a string are Unicode characters.
- Literals: ‘abc’, “abc”

Variables

- Variables are reserved memory locations to store values.
- A **variable** has a name which refers to an object; so a variable has also a type, and a value.
- A variable is introduced in a program using a name binding operation – assignment.

Variable: • name • value • type ◦ domain ◦ operations • memory location	Variable in Python: • name • value ◦ type ▪ domain ▪ operations ◦ memory location

Expressions

An **expression** is a combination of explicit *values*, *constants*, variables, *operators*, and *functions* that are interpreted according to the particular *rules of precedence*, which computes and then *produces/returns* another value.
Examples:

- numeric expression: 1 + 2
- boolean expression 1 < 2
- string expression: ‘1’ + ‘2’

Statements

Statements are the basic operations of a program. A program is a sequence of statements.

- **Assignment**
    - Assignment is a statement.
    - Assignments are used to (re)bind names to values and to modify items of mutable objects.
    - Bind name:
        - x = 1 #x is a variable (of type int)
    - Rebind name:
        - x = x + 2 #a new value is assigned to x
    - Rebind name of mutable sequences:
        - y = [1, 2] #mutable sequence
        - y[0] = -1 #the first item is bound to -1

- **Block**
    - A *block* is a piece of Python program text that is executed as a unit.
    - A sequence of statements is a block.
    - Blocks of code are denoted by line indentation.

How to write programs

Basic roles in software engineering

Programmers/Developers

- Use computers to *write/develop* programs for users.

Clients/stakeholders:

- Who is affected by the outcome of a project.

Users

- *Run programs on their computers.*

A **software development process** is an approach to building, deploying, and possible maintaining software. It indicates:

- What tasks/steps can be taken during development.
- In which order?

We will use: a simple feature-driven development process

Problem statement

A **problem statement** is a short description of the problem being solved.

Calculator - Problem statement
A *teacher* (client) needs a program for *students* (users) who learn or use rational numbers. The program shall help students to make basic arithmetic operation

Requirements

Requirements define in details what is needed from the client perspective. Requirements define:

- What the clients need; and
- What the system must include to satisfy the client' needs.

Requirements guidelines

- **Good requirements ensure your system works like your customers expect. (Don't create problems to solve problems!)**
- Capture the **list of features** your software is supposed to do.
- The list of features must clarify the problem statement ambiguities.

Feature

Feature is a small, client-valued function:

- expressed in the form **<action>** <*result*> <object>,
    - action - a function that the application must provide
    - result - the result obtained after executing the function
    - object - an entity within the application that implements the function
- and typically **can be implemented within a few hours** (in order to be easy to make estimates).

Calculator - Feature list
F1. **Add** *a number* to calculator.
F2. **Clear** calculator.
F3. **Undo** *the last operation* (user may repeat this operation).

A simple feature-driven development process

- Build a feature list from the problem statement
- Plan iterations (at this stage, an iteration may include a single feature)
- For each iteration
    - Model planned features
    - Implement and test features

An **iteration** is a set period of time within a project in which you produce a stable, executable version of the product, together with any other supporting documentation.

Iteration will result in a working and usefoul program for the client (will interact with the user, perform some computation, show results)

Example: iteration plan

Iteration	**Planned features**
I1	F1. **Add** *a number* to calculator
I2	F2. **Clear** calculator
I3	F3. **Undo** *the last operation* (user may repeat this operation)

Iteration modeling

At the beginning of each iteration you must understand the work required to implement it. You must **investigate/analyze** each feature in order to determine work items/tasks. Then, work items are scheduled. Each work item will be independently implemented and tested.

Iteration 1 - Add a number to calculator

For simple programs (e.g. Calculator), running scenarios help developer to understand what must be implemented. A **running scenario** shows possible interactions between users and the program under development.

Running scenario for adding numbers

	User	Program	Description
a		0	Shows total
b	1/2		Add number to calculator
c		1/2	Shows total
d	2/3		Add number to calculator
e		5/6	Shows total
f	1/6		Add number to calculator
g		1	Shows total
h	-6/6		Add number to calculator
i		0	Shows total

Work items/tasks

Guidelines for tasks

- Define a task for each operation not already provided by the platform (Python), e.g. T1, T2.
- Define a task for implementing the interaction between User and Program (User Interface), e.g. T4.
- Define a task for implementing all operations required by UI, e.g. T3.
- Determine dependencies between tasks (e.g. T4 --> T3 --> T2 -->T1, where --> means depends on).
- Schedule items based on the dependencies between them.
    - bottom up: schedule first task without dependencies, or with dependencies scheduled before

T1	Compute the greatest common divisor of two integers (see g, i)
T2	Add two rational numbers (see c, e, g, i)
T3	Implement calculator: init, add, and total
T4	Implement user interface

Task 1. Compute gcd

Test case

A **test case** specifies a set of test inputs, execution conditions, and expected results that you identify to evaluate a particular part of a program.

Input params: a, b	gcd (a, b): c, where c is the greatest common divisor of a and b
2 3	1
2 4	2
6 4	2
0 2	2
2 0	2
24 9	3
-2 0	ValueError
0 -2	ValueError

If statement

The **if** statement is used for conditional execution.

gcd - if statement

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

print gcd(0,2)
print gcd(2,0)
```

While statement

The while statement is used for repeated execution as long as an expression is true.

gcd - while statement

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

```
print gcd(0,2)
print gcd(2,0)
print gcd(2,4)
print gcd(27,4)
```

References

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