Laboratory 1: Introduction. Functions in Python

1. About Python

Overview

Pythonit is a programming language interpreted,dynamic and high level. It supports multiple programming paradigms such as:

- procedural programming;
- object oriented programming;
- functional programming.

a interpreter can run (evaluate) program fragments (expressions or statements) as they are entered.

Work environment

Once we write sample programs it is useful to save them so that we can reuse them. Python programs are written in text files with the extension .py. They can be edited with anything text editor (for example Notepad++,not Word that edits files.dock with specific formatting). These programs saved in files.pyit is run from the command line (cmd/terminal) with command python 3 file.py

To simplify the process we can use an IDE (Integrated Development Environment). An IDE is a program that usually contains at least 3 things: code editor, compiler and debugger. One such IDE is IDLE. It can be installed very easily

IDLE

When opening the IDLE program, a window called IDLE Shell. The interpreter runs in this window. Here we can run code snippets. If we want to write programs in a file that we can save and later run, we can press File->New File to open another window. Here we can write code that can be saved to a file. To compile and run a program press*Run->Run Modules*or the F5 key. Recently used files can be easily accessed from*Files->Recent Files*. To navigate the command history in the interpreter we can use:

- Alt-p : previous command
- Alt-n : next command

2. Python - Fundamentals Evaluation of some expressions

At the prompter>>>of the interpreter we can in general evaluate expressions. The simplest expressions are calculations with numbers, written in the usual mathematical notation. For example, we can enter:

```
> > > 2+3
```

and the interpreter answers:

```
5
```

In the first frame >>> represents the interpreter's prompter (so it should not be entered), and in the second frame, without the prompter, is the answer given by the interpreter, namely 5.

Spaces within the expression do not matter, we can also write:

```
> > > 2 + 3
```

In addition to the addition operation, there are also subtraction (-), multiplication (*), division (/), modulo (%) and exponentiation(**).

Data types

Python provides 4 types of primitive data:

- Integer
- Float
- String
- Boolean

Primitive data types are immutable, that is, once they have been created they cannot be changed. If a variable x = 3 changes its value to 4, a new integer is actually created and assigned to the variable x. We cannot do:

```
> > > 2 = 5
      Syntax Error: cannot assign to literal here. Maybe you meant '==' instead of '='?
```

1. Integer

The data type integer (int)represents unsigned or unsigned numeric data, without decimals, and of unlimited length (integers with values between -∞ and ∞).

```
Ex: 3, 6, -234.
```

2. Float

The data type float is used to represent signed or unsigned floating point numbers.

```
Ex: 3.34, -0.123456.
```

! Careful2 is an integer value. For a real value (float) we must write 2.0 (or abbreviated 2.). In Python the type conversion from int to float is done automatically. Thus, the result of operations containing both integers and real numbers will be a real number (eg 5 + 2.0 will give 7.0).
We can also use the float() function if we want to do a conversion explicitly:

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

3. String

String(string) represents a collection of characters, words, or phrases. To create a string in Python, we use the signs '' (apostrophe) or "" (quotes).
Example: 'book' , "23abc".
We can perform various operations on a character string. One of the most common operations is string concatenation. This is done through the + operator:

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

The characters in a string can be accessed directly by "string"[index]:

```
> > > 'A string'[2] 's'
```

The result is the third character (character numbering starts at 0). Python also allows accessing characters using negative index values. For the example below, selecting any other integer that is outside the range [-8; 7] will throw an exception:

```
'A' ' ' 's' 't' 'r' 'i' 'n' 'g'
  0 1 2 3 4 5 6 7
- 8 -7 -6 -5 -4 -3 -2 -1
> > > 'A string'[-8]
```

```
'A'
```

Careful Because Python is a programming language strongly typed, if you want to perform mathematical operations with character strings containing numeric values, the conversion will not be done automatically, but you will have to do it using one of the int() or float() functions.

```
> > > 5 + int('2') 7
```

For more operations that can be applied to character strings (such as converting a string to lowercase/uppercase, splitting a string, replacing a certain character in a string, etc.) you can consult page.

4. Boolean

The data type boolean represents the truth value of an expression. It can have the values false or False. A boolean is usually used in writing conditions or comparing expressions.

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

Using comparison operators ==(equal), !=(different), >(bigger), <(smaller), >=(greater than or equal to), <=(less than or equal to), we can make comparisons between different expressions. To write more complicated conditions you can use keywords and, hours and note.

When using and, if both operands have the logical value True, then the final result will be True. If at least one operand is False, the entire expression will evaluate to False.

```
> > > 3 == (2 + 1) and (3 + 1) != (2 ** 2) False
```

In the case of the logical operator or, it is sufficient for at least one of the two operands to be true for the resulting expression to be true. If both operands are False, then the result of the expression will also be False.

```
> > > 3 == (2 + 1) or (3 + 1) != (2 ** 2) True
```

The logical operator not applies to a single operand. If it has the value True, then the final result will be false. If the operand is False, then the final result will be True.

```
> > > not (3 + 1) != (2 ** 2)
```

```
false
```

In addition to these fundamental data types, Python provides 4 other predefined types to hold collections of data. Thus, in Python we can work with the following:

- List
- Tuple
- Set
- Dictionary

In the following we only present how to define the data types list and tuple in Python (more details about these types, as well as the other two, will be presented in future labs).
Thus, if we want to define a list, we will enumerate its elements between square brackets, the elements being separated from each other by a comma:

```
Example: even_digits_list = [0, 2, 4, 6, 8]
```

If we want to access an element of the list, we proceed in the same way as in the case of character strings (Example: if we want to access element 2 of the list*evenDigits*we will write *evenDigits[1]*, the numbering also in this case starting from 0).

Similar to lists, we can define a tuple using round brackets:

```
Example: even_digits_tuple = (0, 2, 4, 6, 8)
```

print

So far we have used the fact that the interpreter automatically displays the result of an evaluation. Writing and running standalone programs, however, requires functions that print values.
Python has the print() function, which can be used to print on the screen various messages or data types that will be converted to string. The newline character is represented as in C: '\n'.
We can switch to a new line by calling`print('\n')`.
The print() function accepts a variable number of parameters. Thus, we can print different strings as follows:

```
> > > print("Hello.", "How are you?") Hello. How
      are you?
```

We can also specify a separator between printed strings by assigning a string to the`September`of the print function:

```
> > > print("Hello.", "How are you?", sep="\n-----\n") Hello.

        - - - - -
        How are you?
```

We can also use the print() function in a way similar to the printf function in C, as follows:

```
> > > print("Result: %d + %.2f = %.2f" % (2, 3.25, 2 + 3.25)) Result: 2 + 3.25 = 5.25
```

Conditional structure

By default, Python code executes sequentially, line by line, but there may be situations where a statement is executed only if a certain condition is met. To achieve this we can use the following structure:

```
if condition_1:
    instruction_block_1
elif condition_2:
    instruction_block_2
otherwise:
    instruction_block_3
```

Remarks

- there can be any number of elif branches;
- the elif branch may be completely absent from the structure;
- the else branch is optional (may occur once or not at all).

Example:

```
if (x < 0):
    print("x < 0")
elif (x <= 1):
    print("0 <= x <= 1") else:

    print("x > 1")
```

The above example checks if the variable x is less than 0, in which case "x<0" is printed. If x is greater than or equal to 0 (that is, the first condition is not checked), the condition on the elif branch will be checked, namely if x is less than or equal to 1. If this condition is also not checked (i.e. x is greater than 1) the statement corresponding to the else branch will be executed.

Peculiarities of syntax

1. Indentation

In Python, indentation is very important in writing code. Unlike other programming **languages that use braces { }, in Python indentation is used for blocks of code.**
Indentation is always preceded by the sign:(two points).

2. Comments

Comments are used to explain portions of code. In Python comments start with the # symbol:

```
# This is a single line comment.
```

Comments can also be placed at the end of a line:

```
print("Python") #This is another example.
```

If the comment spans multiple lines, each line can start with
or we can opt to use """ (3 quotes at the beginning of the comment and 3 quotes at the end of it):

```
"""This is
  A
  multiline comment"""
  print(5 + 2)
```

Inserting external code (Module)

As in other programming languages, functions and constants from other files can be entered in Python. A useful example of this is given by the inclusion of the math module.

```
> > > import math
```

To now use features like floor or ceil:

```
> > > math.floor(3.7) 3

> > > math.ceil(3.7) 4
```

If we want to rename the imported module we can use:

```
> > > import math as m
```

Now we can use the functions by calling:

```
> > > m.floor(3.7) 3
```

Observation!functionint()can be used to truncate the fractional part.

3. Functional programming

In functional programming, programs are built by applying and composing functions. Unlike procedural programming, where sequences of instructions are used that change the state of the program, functional programming is based on the evaluation of mathematical functions, thus avoiding mutable state and data.
Although Python allows us to work directly with data and modify it, in order to program functionally it is very important not to write functions that change the global state of the program when they are called.

4.Functions in Python

Defining

In Python functions are defined using the key word def. After the keyword, the name of the function is passed, followed by round brackets where its parameters are placed separated by a comma, and at the end the symbol :(two dots) is added. If the function has no parameters, you will not put anything between the brackets, but their presence is mandatory.

Observation!Don't forget to indent the entire function body!

```
def function_name(param_1, param_2, param_3, ..., param_n):
    # some_instructions
```

Thus, the function$f:\mathbb{Z}\to\mathbb{Z}, f(x) = x + 3$it is written in Python:

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

We can also define the function with the help of the interpreter:

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

We press enter twice to signal to the interpreter that we have finished writing the code. The function will remain defined in memory until the interpreter is closed. The interpreter's ... prompter indicates that the code continues on the next line.

Calling functions

Once the function is defined, it is called as follows:

```
> > > f(1)
        4
```

When we call a function we can also specify the name of the parameter at the call:

```
> > > f(x=5)
        8
```

We can also give the function a complex expression as a parameter:

```
> > > f(2*3)
        9
```

Functions with multiple parameters

Let the function in mathematics: $integer\_add: \mathbb{Z}x\mathbb{Z} \to \mathbb{Z}, integer\_add(x, y) = x + y$ The most common way to write this function in a Python program is:

```
def integer_add(x, y):
          return x + y
```

Functions that return multiple values

In Python we can implement functions that return multiple values using lists. For example, if we want to implement the function $f: \mathbb{Z}x\mathbb{Z} \to \mathbb{Z}x\mathbb{Z}, f(x, y) = [x + y, x - y]$ we can write like this:

```
def f(x, y):
        return [x + y, x - y]
```

Another way we can write such functions is by using the Tuple data type:

```
def f(x, y):
    return x + y, x - y
```

Calling example:

```
result_1, result_2 = f(2, 1)
```

Example

In functional programming we want to evaluate expressions, not execute a series of statements. Thus, the value returned by one function will be used as a parameter by another function.

However, in the example below, although print() is executed inside a function, this call does not affect system state.

```
def print_abs(x):
    if (x > 0):
        print("Positive: " + str(x)) return x

    else:
        print("Negative: " + str(x)) return -x
```