Programming in Python

GAVRILUT DRAGOS

COURSE 1

Administrative

Final grade for the Python course is computed using Gauss over the total points accumulated.

One can accumulate a maximum of 100 of points:

- Maximum 70 points from the laboratory examination
- Maximum 30 points at the final examination (course)

The laboratory examination consists in 2 test:

- First test → 30 points
- Second test → 40 points

The minimum number of points that one needs to pass this exam:

- Minimum 25 points accumulated from the first and the seconds laboratory tests
- Minimum 10 points from the final examination (course)

Course page: https://sites.google.com/site/fiipythonprogramming/home

String Types

Strings also support different ways to access characters or substrings

Python 2.x / 3.x

```
s = "PythonExam"    #s is "PythonExam"

s[1]                #Result is "y" (second character, first index is 0)
s[-1]               #Result is "m" ➔ "PythonExam"(last character)
s[-2]               #Result is "a" ➔ "PythonExam"
s[:3]               #Result is "Pyt" ➔ "PythonExam"(first 3 characters)
s[4:]               #Result is "onExam" ➔ "PythonExam"
                    #(all the characters starting from the 4th character
                    #of the string until the end of the string)
s[3:5]              #Result is "ho" ➔ "PythonExam" (a substring that
                    #starts from the 3rd character until the 5th one)
s[2:-4]             #Result is "thon" ➔ "PythonExam"
```

String Types

Strings also support a variety of operators

Python 2.x / 3.x

```
s = "Python"+"Exam"#s is “PythonExam”
s = "A"+"12"*3     #s is “A121212” ➔ ”12” is multiplied 3 times
"A" in "Python"    #Result is False (”A” string does not exists in
                   #                 ”Python” string)
"A" not in "ABC"   #Result is False (”A” string exists in ”ABC”)
len (s)            #Result is 10 (10 characters in “PythonExam” string)
```

And slicing:

Python 2.x / 3.x

```
s = "PythonExam"   #s is “PythonExam”
s[1:7:2]           #Result is ”yhn” (Going from index 1, to index 7
                   #with step 2 (1,3,5) ➔ PythonExam
```

String Types

Every string is consider a class and has member functions associated with it. This functions are accessible through "." operator.

- Str.**startswith**("…") ➔ checks if a string starts with another one
- Str.**endswith**("…") ➔ checks if a string ends with another one
- Str.**replace**(toFind,replace,[count]) ➔ returns a string where the substring <*toFind*> is replaced by substring <*replace*>. Count is a optional parameter, if given only the firs <*count*> occurrences are replaced
- Str.**index**(toFind) ➔ returns the index of <*toFind*> in current string
- Str.**rindex**(toFind) ➔ returns the right most index of <*toFind*> in current string
- Other functions: **lower**(), **upper**(), **strip**(), **rstrip**(), **lstrip**(), **format**(), **isalpha**(), **isupper**(), **islower**(), **find**(…), **count**(…), etc

String Types

Strings splitting via .**split** function

Python 2.x / 3.x

```
s = "AB||CD||EF||GH"
s.split("||")[2]    #Result is "EF". Split produces an array of 4
                    #elements AB,CD,EF and GH. The second element is EF
s.split("||")[-1]   #Result is "GH".
s.split("||",1)[0]  #Result is "AB". In this case the second parameter
                    #tells the function to stop after <count> (in this
                    #case 1) splits. Split produces an array of 2
                    #elements AB and CD||EF||GH. The fist element is AB
s.split("||",2)[2]  #Result is "EF||GH". Split produces an array of 3
                    #elements AB, CD and EF||GH.
```

Strings also support another function .**rsplit** that is similar to .**split** function with the only difference that the splitting starts from the end and not from the beginning.

Built-in functions for strings

Python has several build-in functions design to work characters and strings:

- **chr** (*charCode*) ➔ returns the string formed from one character corresponding to the code *charCode. charCode* is an Unicode code value.
- **ord** (character) ➔ returns the Unicode code corresponding to that specific character
- **hex** (number) ➔ converts a number to a lower-case hex representation
- **oct** (number) ➔ converts a number to a base-8 representation
- **format** ➔ to format a string with different values

IF-Statement

Python 2.x/3.x

```
if expression:
        complex or simple statement
```

Python 2.x/3.x

```
if expression:
        complex or simple statement
else:
        complex or simple statement
```

Python 2.x/3.x

```
if expression:
        complex or simple statement
elif expression:
        complex or simple statement
```

Python 2.x/3.x

```
if expression:
        complex or simple statement
elif expression:
        complex or simple statement
elif expression:
        complex or simple statement
elif expression:
        complex or simple statement
…
else:
        complex or simple statement
```

WHILE - Statement

The **break** keyword can be used to exit the while loop. Using the **break** keyword will not move the execution to the **else** statement if present !

Python 2.x/3.x

```
a = 3
while a > 0:
        a = a - 1
        print (a)
        if a==2: break
else:
        print ("Done")
```

Output

```
2
```

WHILE - Statement

Similarly the **continue** keyword can be used to switch the execution from the while loop to the point where the while condition is tested.

Python 2.x/3.x

```
a = 10
while a > 0:
        a = a - 1
        if a % 2 == 0: continue
        print (a)
else:
        print ("Done")
```

Output

```
9
7
5
3
1
Done
```

FOR- Statement

A special keyword **range** that can be use to simulate a C/C++ like behavior.

Python 2.x/3.x

```
for index in range (0,3):
        print (index)
```

Output

```
0
1
2
```

Python 2.x/3.x

```
for index in range (0,3):
        print (index)
else:
        print (”Done”)
```

Output

```
0
1
2
Done
```

Functions

Function parameters can have default values. Once a parameter is defined using a default value, every parameter that is declared after it should have default values.

Python 2.x/3.x

```
def myFunc (x, y=6, z=7):
        return x * 100 + y * 10 + z
print (myFunc (1) )                                  #Output:167
print (myFunc (2,9) )                                #Output:297
print (myFunc (z=5,x=3) )                            #Output:365
print (myFunc (4,z=3) )                              #Output:463
print (myFunc (z=5) )                                #ERROR: missing x
```

Python 2.x/3.x

```
def myFunc (x=2, y, z=7):
        return x * 100 + y * 10 + z
```

Code will not compile as x has a default value, but Y does not !

Functions

A function can return multiple values at once. This will also be discuss in course no. 2 along with the concept of touple.

Python also uses **global** keyword to specify within a function that a specific variable is in fact a global variable.

Python 2.x/3.x

```
x = 10
def ModifyX ():
        x = 100
ModifyX ()
print ( x ) #Output:10
```

Python 2.x/3.x

```
x = 10
def ModifyX ():
        global x
        x = 100
ModifyX ()
print ( x ) #Output:100
```

Functions

Functions can have a variable – length parameter (similar to the … from C/C++).
It is precede by "*" operator.

Python 2.x/3.x

```
def multi_sum (*list_of_numbers):
        s = 0
        for number in list_of_numbers:
                s += number
        return s

print ( multi_sum (1,2,3) )                    #Output:6
print ( multi_sum (1,2) )                      #Output:3
print ( multi_sum (1) )                        #Output:1
print ( multi_sum () )                         #Output:0
```

Functions

It is recommended to add a short explanation for every defined function by adding a multi-line string immediately after the function definition
https://www.python.org/dev/peps/pep-0257/#id15

Python 2.x/3.x

```
def Fibonacci (n):
        """
        Computes the n-th Fibonaci number using recursive calls
        """
        if n == 1:
                return 1
        elif n == 2:
                return 1
        else:
                return Fibonacci (n-1) + Fibonacci (n-2)
```

How to create a python file

- Create a file with the extension .py
- If you run on a Linux/OSX operation system you can add the following line at the beginning of the file (the first line of the file):
  - `#!/usr/bin/python3` ➔ for python 3
  - `#!/usr/bin/python` ➔ for python (current version – usually 2)
- These lines can be added for windows as well ("#" character means comment in python so they don't affect the execution of the file too much
- Write the python code into the file
- Execute the file.
  - You can use the python interpreter directly (usually C:\Python27\python.exe or C:\Python36\python.exe for Windows) and pass the file as a parameter
  - Current distributions of python make some associations between .py files and their interpreter. In this cases you should be able to run the file directly without using the python executable.