P1_Starting_with_python_professor

September 19, 2019

1 A Brief Tutorial of Basic Python

```
Author: Jesús Fernández Bes
        Jerónimo Arenas García (jeronimo.arenas@uc3m.es)
        Jesús Cid Sueiro (jcid@tsc.uc3m.es)
        Vanessa Gómez Verdejo (vanessa@tsc.uc3m.es)
        Óscar García Hinde (oghinnde@tsc.uc3m.es)
        Simón Roca Sotelo (sroca@tsc.uc3m.es)
```

```
Notebook version: 1.1 (Sep 20, 2017)

Changes: v.1.0 - First version.
         v.1.1 - Compatibility with python 2 and python 3
         v.2.0 - (Sep 3, 2018) - Adapted to TMDE
         v.2.1 - (Sep 2, 2019) - Dictionaries added
         v.2.2 - (Sep 6, 2019) - Exercises
```

From Wikipedia: "Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale."

To easily work with Python from any computer you can use Google Colaboratory tool, which provides a free envoriment to run Python Jupyter notebooks.

Throughout this tutorial, students will learn some basic characteristics of the Python programming language, as well as the main characteristics of the Python notebooks.

2 A Python Jupyter Notebook

A python noteboook is an interactive web page made up of cells where each cell can contain text (like this one) or executable code. For instance, the next cell contains excutable code:

[0]:
```
a = 'house'
print(a)
```

To run this code and check its output, you firstly have to be conected to a Python runtime server (check the top right pannel) and then you can: * Select the cell to be run and click the Play button (on the left) * Select the cell to be run and type Cmd/Ctrl+Enter * Select the cell to be run and Type Shift+Enter to run the cell. In this case, you additionaly will be moved to the next cell.

Besides, you can use the options of the Runtime (on the top menu) to run several cells at the same time or stop a execution.

3 Introduction to Python

3.1 1. Introduction to Strings

Among the different native python types, we will focus on strings, since they will be the core type that we will recur to represent text. Essentially, a string is just a concatenation of characters.

[0]:
```
str1 = '"Hola" is how we say "hello" in Spanish.'
str2 = "Strings can also be defined with quotes; try to be sistematic and␣
↪consistent."
```

It is easy to check the type of a variable with the type() command:

[0]:
```
print(str1)
print(type(str1))
print(type(3))
print(type(3.))
```

The following commands implement some common operations with strings in Python. Take a look at them, and try to deduce what the result of each operation will be. Then, execute the commands and check what the actual results are.

[0]:
```
print(str1[0:5])
```

[0]:
```
print(str1 + str2)
```

[0]:
```
print(str1.lower())
```

[0]:
```
print(str1.upper())
```

[0]:
```
print(len(str1))
```

[0]:
```
print(str1.replace('h','H'))
```

[0]:
```
str3 = 'This is a question'
str3 = str3.replace('i', 'o')
str3 = str3.lower()
print(str3[0:3])
```

It is interesting to notice the difference in the use of commands 'lower' and 'len'. Python is an object-oriented language, and str1 is an instance of the Python class 'string'. Then, str1.lower() invokes the method lower() of the class string to which object str1 belongs, while len(str1) or type(str1) imply the use of external methods, not belonging to the class string. In any case, we will not pay (much) attention to these issues during the session.

Finally, we remark that there exist special characters that require special consideration. Apart from language-oriented characters or special symbols (e.g., €), the following characters are commonly used to denote carriage return and the start of new lines

[0]:
```
print('This is just a carriage return symbol.\r Second line will start on top of␣
  ↪the first line.')
```

[0]:
```
print('If you wish to start a new line,\r\nthe line feed character should also␣
  ↪be used.')
```

[0]:
```
print('But note that most applications are tolerant\nto the use of \'line feed\'␣
  ↪only.')
```

Exercise 1: *Define variables* `str1` *and* `str2` *containing strings "The greatest glory in living lies not in never falling, " and "but in rising every time we fall", respectively. Concatenate both strings to compose a sentence, transform it to capital letters and print the result.*

[0]:
```
# Write your solution here
# <SOL>
str1 = "The greatest glory in living lies not in never falling, "
str2 = "but in rising every time we fall"
str3 = (str1 + str2).upper()
print(str3)
# </SOL>
```

3.2 2. Working with Python lists and dictionaries

3.2.1 2.1. Lists

Python lists are iterable containers that hold a number of other objects, in a given order. To create a list, just put different comma-separated values between square brackets:

[0]:
```
# A list of integers
mylist0 = [1, 2, 3, 4, 5]
print(mylist0)

# A list of trings
mylist1 = ['a', 'b', 'c', 'd']
print(mylist1)

# A list of strings, integers and float
mylist2 = ['student', 'teacher', 1997, 2000, 8.9]
print(mylist2)
```

2.1.1. Indexing lists To check the value of a list element, indicate between brackets the index (or indices) to obtain the value (or values) at that position (positions).

Run the code fragment below, and try to guess what the output of each command will be.

Note: Python indexing starts from 0!!!!

[0]:
```
print(mylist0[0])
print(mylist1[2:4])
print(mylist2[-1])
```

2.1.2. Modifying lists To add elements in a list you can use the method append() and to remove them the method remove()

[0]:
```
mylist = ['student', 'teacher', 1997, 2000]
mylist.append(3)
print(mylist)
mylist.remove('teacher')
print(mylist)
```

You can also concatenate lists by simple "addition"

[0]:
```
fruits = ['apple', 'orange']
drinks = ['water',  'juice']
fruits_and_drinks = fruits + drinks
print(fruits_and_drinks)
```

Other useful functions are:

```
len(list):    Gives the number of elements in a list.
max(list):    Returns item from the list with max value.
min(list):    Returns item from the list with min value.
```

[0]:
```
mylist = [1, 2, 3, 4, 5]
print(len(mylist))
print(max(mylist))
print(min(mylist))
```

Exercise 2: *Define variable* `odds` *containing all odd numbers betewen 1 and 9. Also, define variable* `prime` *containing all prime numbers between 1 and 9. Now, define variable* `odd_or_prime` *as the concatenation of* `odd` *and* `prime`*. Some numbers are repeated. User method* `remove` *to supress elements appearing twice. Print the result.*

[0]:
```
# Write your solution here
# <SOL>
odd = [1, 3, 5, 7, 9]
prime = [2, 3, 5, 7]
odd_or_prime = odd + prime
odd_or_prime.remove(3)
odd_or_prime.remove(5)
odd_or_prime.remove(7)
print(odd_or_prime)
# </SOL>
```

3.3 2.2. Dictionaries

Dictionaries are a different type of iterable in which data is stored with a `key:value` format. Creating dictionaries is similar to creating lists except that we use curly brackets: {}.

[0]:
```
my_marks = {"calculus": 9.5, "programming": 7.5, "statistics": 4.5}
print(my_marks)
```

In order to access a particular value in the dictionary we need to use the corresponding key:

[0]:
```
print(my_marks["programming"])
```

If we want to display all the keys in a dictionary we can use the `.keys()` method. Likewise we can view all the stored values in the dictionary by using the `.values()` method:

[0]:
```
print(my_marks.keys())
print(my_marks.values())
```

Exercise 3: *After the exam review, you teacher has updated your mark in Statistics to 7.0. Also, the scores of "communication theory" have been just published, and you've got 10.0. How would you modify your dictionary with the new marks?*

[0]:
```
# Write your solution here
# <SOL>
my_marks['statistics'] = 7.0
my_marks['communication theory'] = 10.0
# </SOL>
print(my_marks)
```

3.4 3. Flow control (with 'for' and 'if')

As in other programming languages, python offers mechanisms to loop through a piece of code several times, or for conditionally executing a code fragment when certain conditions are satisfied.

3.4.1 3.1. Conditional execution

For conditional execution, you can we use the 'if', 'elif' and 'else' statements.

Try to play with the following example:

[0]:
```
x = int(input('Please enter an integer: '))
if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

The above fragment, allows us also to discuss some important characteristics of the Python language syntaxis:

- Unlike other languages, Python does not require to use the 'end' keyword to indicate that a given code fragment finishes. Instead, Python recurs to indentation
- **Indentation in Python is mandatory**, and consists of 4 spaces (for first level indentation)
- The condition lines conclude with ':', which are then followed by the indented blocks that will be executed only when the indicated conditions are satisfied.

3.4.2 3.2. Loops

The statement 'for' lets you iterate over the items of any iterable (a list, a string, a dictionary...), in the order that they appear.

[0]:
```
words = ['cat', 'window', 'open-course']
for w in words:
     print (w, len(w))
```

[0]:
```
my_dict = {"key1": 10, "key2": 25, "key3": 30}
for key in my_dict.keys():
     print (key, my_dict[key])
```

In combination with enumerate(), you can iterate over the elements of the sequence and have a counter over them

[0]:
```
words = ['cat', 'window', 'open-course']
for (i, w) in enumerate(words):
     print ('element ' + str(i) + ' is ' + w)
```

Python is very expressive and flexible when it comes to list creation and manipulation. Check out this interesting block of code and try to figure out what it's doing:

[0]:
```
# Python one-liners are great:
x = [1, 2, 3, 4, 5]
y = [i**2 for i in x]
print('x = ', x, '\ny = ', y)
```

Exercise 4: *Create a list* `powers2` *of all powers of 2 between* 2^0 *(*`2**0`*) and* 2^{16} *(*`2**16`*). Loop over the list to create a new list* `y` *with all powers of 2 that are greater than 100 and lower than 1000*

[0]:
```
# Write your solution here
# <SOL>
powers2 = [2**x for x in range(16)]
y = []
for p in powers2:
    if p > 100 and p < 10000:
        y.append(p)
# </SOL>
print(y)
```

3.5 4. File input and output operations

First of all, you need to open a file with the open() function (if the file does not exist, the open() function will create it).

[0]:
```
f = open('workfile', 'w')
```

The first argument is a string containing the filename. The second argument defines the mode in which the file will be used:

```
'r' : only to be read,
'w' : for only writing (an existing file with the same name would be erased),
'a' : the file is opened for appending; any data written to the file is automatically appended t
'r+': opens the file for both reading and writing.
```

If the mode argument is not included, 'r' will be assumed.

Use f.write(string) to write the contents of a string to the file. When you are done, do not forget to close the file:

[0]:
```
f.write('This is a test\n with 2 lines')
f.close()
```

To read the content of a file, use the function f.read():

[0]:
```
f2 = open('workfile', 'r')
text = f2.read()
f2.close()
print(text)
```

You can also read line by line from the file identifier

[0]:
```
f2 = open('workfile', 'r')
for line in f2:
    print(line)

f2.close()
```

3.6 5. Importing modules

Python lets you define modules which are files consisting of Python code. A module can define functions, classes and variables.

Most Python distributions already include the most popular modules with predefined libraries which make our programmer lives easier. Some well-known libraries are: time, sys, os, numpy, etc.

There are several ways to import a library:

1) Import all the contents of the library: import lib_name

Note: You have to call these methods as part of the library

[0]:
```
import time
print(time.time())  # returns the current processor time in seconds
time.sleep(2) # suspends execution for the given number of seconds
print(time.time()) # returns the current processor time in seconds again!!!
```

2) Define a short name to use the library: import lib_name as lib

[0]:
```
import time as t
print(t.time())
```

3) Import only some elements of the library

Note: now you have to use the methods directly

[0]:
```
from time import time, sleep
print(time())
sleep(2)
```

```
print(time())
```

3.7 6. User defined functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity and a high degree of code reusing.

As we have seen, Python gives us many built-in functions like print(), etc. but you can also create your own functions.

3.7.1 Defining a function:

- Function blocks begin with the keyword `def` followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring. Docstring blocks start and ed with triple quotes (`"""`).
- The code block within every function starts with a colon (`:`) and is indented.
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

[0]:
```
# Example of a function:

def add_five(a):
  """This function adds the number five to its input argument."""
  return a + 5

# Let's try it out!
x = int(input('Please enter an integer: '))
print('Our add_five function is super effective at adding fives:')
print(x, '+ 5 =', add_five(x))
```

Exercise 5: *Now try it yourself! Write a function called my_factorial that computes the fatorial of a given number. If you're feeling brave today, you can try writing it recursively since Python obviously supports recursion. Remember that the factorial of 0 is 1 and the factorial of a negative number is undefined.*

[0]:
```
# Write your solution here
# <SOL>
def my_factorial(n):
    if n > 1:
        return n * my_factorial(n - 1)
    else:
        return 1
# </SOL>
```

[0]:
```
# Let's try it out!
from math import factorial
x = int(input('Please enter an integer: '))
if my_factorial(x) == factorial(x):
  print(x, '! = ', my_factorial(x))
  print('It works!')
else:
  print("Something's not working..." )
```

[0]:
```

```