intro_python

October 13, 2018

1 INTRODUCTION TO PYTHON

1.1 Description

Python is a widely used high-level, general-purpose, interpreted, dynamic 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. (Source: https://anaconda.org/anaconda/python)

1.2 General advantages:

- Easy to use.
- High-level programming.
- Requires less lines of coding compared to other languages.
- Large online community for support.
- Capability of porting scripts to other platforms without Python interpreters.

1.3 Specific advantages:

- Preinstalled in UNIX systems.
- Capability to import more than 1000 pre-designed modules for penetration testing such as "hashlib".
- Capability to quickly build scripts to deliver exploits, manipulate well-known network protocols and create custom packets.
- Interactive shell lets programmer figure out how a specific function is working.
- Implementation of image recognition for biometrics.

1.4 Some statistics

- Python is the 2nd most demanded programming language in 2018 (https://www.codingdojo.com/blog/7-most-in-demand-programming-languages-of-2018/)
- It is the 2nd best paid programming skill in the market (https://medium.freecodecamp.org/best-programming-languages-to-learn-in-2018-ultimate-guide-bfc93e615b35)

- It is the best ranked programmin language according to IEEE (https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages)

2 How to install Python?

2.1 Option 1:

- Install Python from https://www.python.org
- Install a Python Integrated Development Environment (IDE) such as Pycharm (available at https://www.jetbrains.com/pycharm/)
- Install Jupytor Notebook, available at http://jupyter.org/

2.2 Option 2 (Better Solution):

All can be easily installed in a bundle called Anaconda https://www.anaconda.com/download/. Anaconda contains an IDE called Spyder and it even contains R!

2.3 Python in Linux

Also, keep in mind that Ubuntu already contains Python 2. To update to Python 3, use the command:

- sudo apt-get python3

3 Where to learn Python?

3.1 Books:

- Mark Lutz, David Ascher (1999). "Learning Python".

3.2 Forums:

- StackOverflow https://stackoverflow.com/questions/tagged/python The online community is always a great source of help.

3.3 Online

- https://docs.python.org/3/tutorial/
- Edx: They have a free course called "Python in Research" (https://www.edx.org/course/using-python-research-harvardx-ph526x-0) by Harvard University where you can get a certificate.
- DataCamp has developed the site learnpython.org (https://www.learnpython.org/en/). It contains an very complete online tutorial which even facilitates an online python IDE where you can run code and test it.
- Courses on Coursera and Datacamp.

3.4 Practice!

- DataCamp also has a mobile app with quick and interactive courses on Python.

4 Python Data Structures

4.1 Numbers

Python's most basic function is as a calculator. It handles multiple types of numbers such as integers, floating point or boolean operators.

```
In [1]: 8+9

Out[1]: 17

In [2]: 5-10.9/3

Out[2]: 1.3666666666666667

In [3]: True or False

Out[3]: True
```

Numbers can be assigned to variables using "=".

```
In [4]: X = 2
        Y = X**2 + 10
        print('Result is', Y)

Result is 14
```

By default, when defining a variable and storing a number, the variable's **type** will be integer (if the number has no decimals). Otherwise, the variable will contain a float.

```
In [5]: x = 1
        type(x)

Out[5]: int

In [6]: y = 2.1
        type(y)

Out[6]: float
```

A variable containing an integer can be converted into a float and viceversa.

```
In [7]: x=float(x)
        print(x, type(x))

1.0 <class 'float'>


In [8]: y=int(y)
        print(y, type(y))

2 <class 'int'>
```

4.2 Strings

Variables can also store strings (words). Strings are treated as a "list" of characters.

```
In [9]: Uni = "Robert Gordon University"
        print(Uni)

Robert Gordon University
```

Indexing: Python considers the first value to be in position **0** and the last one to be **-1**.

```
In [10]: len(Uni)
         print('Length of the string Uni is: ',len(Uni))
         print('First character of Uni is: ', Uni[0])
         print('Last character of Uni is: ',Uni[-1])

Length of the string Uni is:  24
First character of Uni is:  R
Last character of Uni is:  y
```

When requesting a range of characters, python "ignores" the last one.

```
In [11]: Uni[0:2] # characters from position 0 (included) to 2 (excluded)

Out[11]: 'Ro'

In [12]: Uni[2:5]  # characters from position 2 (included) to 5 (excluded)

Out[12]: 'ber'

In [13]: Uni[:5]

Out[13]: 'Rober'

In [14]: Uni[2:]

Out[14]: 'bert Gordon University'

In [15]: Uni[-2:]# last two characters

Out[15]: 'ty'
```

Just as with variables containing numbers, we can do operations with variables containing strings. For instance, we can add a string to another.

```
In [16]: Uni + ' Data Science'

Out[16]: 'Robert Gordon University Data Science'
```

4.3 Lists

Lists are **mutable** structures which contain items of equal or different types. Lists can even contain other lists within.

```
In [17]: squares = [1, 4, 9, 16, 25]
         print(squares, type(squares))

[1, 4, 9, 16, 25] <class 'list'>


In [18]: len(squares)

Out[18]: 5

In [19]: things = [1, 'John', 9.86, False, squares]
         things

Out[19]: [1, 'John', 9.86, False, [1, 4, 9, 16, 25]]
```

Similar to strings, we can request certain parts of the content of a list.

```
In [20]: squares[0] # returns first item

Out[20]: 1

In [21]: squares[:2] # returns items at position 0,1 (2 is exluded)

Out[21]: [1, 4]
```

Operations for lists.

```
In [22]: # add elements to the list
         squares = squares + [36, 49]
         squares

Out[22]: [1, 4, 9, 16, 25, 36, 49]

In [23]: # Appending data to a list
         things.append(['a','b'])
         things

Out[23]: [1, 'John', 9.86, False, [1, 4, 9, 16, 25], ['a', 'b']]

In [24]: # Compare two lists
         [1,2,3] == [1,3,3]

Out[24]: False

In [25]: # Check if an element is in the list
         4 in squares
```

```
Out[25]: True

In [26]: # replace elements from the list
         squares[2]=12
         print(squares)

[1, 4, 12, 16, 25, 36, 49]


In [27]: # delete elements from the list
         del squares[2]
         squares

Out[27]: [1, 4, 16, 25, 36, 49]
```

4.4 Tuples

Similar to a list, but **immutable**.

```
In [28]: tup = (1,2,3,'Data Science')
         tup

Out[28]: (1, 2, 3, 'Data Science')

In [29]: tup[1]

Out[29]: 2

In [30]: len(tup)

Out[30]: 4

In [31]: tup[2]=7


         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-31-1f03b11b5d1e> in <module>()
    ----> 1 tup[2]=7


         TypeError: 'tuple' object does not support item assignment
```

A tuple cannot be modified, but the values can be unpacked into variables.

```
In [ ]: num1,num2,num3,mystring = tup
        print(num1,num2,num3,mystring)
```

Example of using lists and tuples.

```
In [32]: list_physics = [("gravity_earth",9.81),("e",2.3),("pi",3.1416)]
         list_physics

Out[32]: [('gravity_earth', 9.81), ('e', 2.3), ('pi', 3.1416)]

In [33]: list_physics[0]=("gravity_moon",9.81/6)
         list_physics

Out[33]: [('gravity_moon', 1.635), ('e', 2.3), ('pi', 3.1416)]
```

4.5 Ranges

Ranges are "lists" of numbers which can be defined with one instruction.

```
In [34]: range(10)

Out[34]: range(0, 10)

In [35]: r = range(10)
         type(r)

Out[35]: range
```

Converting a range into a list.

```
In [36]: r = list(r)
         r

Out[36]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ranges between numbers.

```
In [37]: range(5,10)

Out[37]: range(5, 10)

In [38]: # Setting a range with a step
         list(range(5,10,2))

Out[38]: [5, 7, 9]

In [39]: list(range(9,1))

Out[39]: []

In [40]: list(range(9,1,-1))

Out[40]: [9, 8, 7, 6, 5, 4, 3, 2]
```

4.6 Sets

Unordered collection of elements.

```
In [41]: basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
         print(basket)
```

```
{'orange', 'apple', 'pear', 'banana'}
```

```
In [42]: # check if an element is in a set
         'grape' in basket
```

```
Out[42]: False
```

4.7 Dictionaries

"Lists" indexed with a key.

```
In [43]: dictStudent = {'Name': 'Alex', 'Age': 27, 'Grade': 'A'}
         dictStudent
```

```
Out[43]: {'Age': 27, 'Grade': 'A', 'Name': 'Alex'}
```

```
In [44]: len(dictStudent)
```

```
Out[44]: 3
```

```
In [45]: # Requesting a specific element from the dictionary by using the key
         print(dictStudent['Name'])
```

```
Alex
```

4.8 Numpy Arrays

Numpy is a python module which allows the use of "numpy arrays", which are easy to use vectors and matrices of numbers.

```
In [46]: !pip install numpy
```

```
Requirement already satisfied: numpy in d:\anaconda3\lib\site-packages


You are using pip version 9.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

```
In [47]: import numpy as np

         # Defining a vector
         N = np.array([1,3,5,7,9])
         print(N)
```

```
[1 3 5 7 9]
```

Operations for numpy arrays.

```
In [48]: N+1

Out[48]: array([ 2,  4,  6,  8, 10])

In [49]: sum(N)

Out[49]: 25

In [50]: N.shape

Out[50]: (5,)
```

Different to a list comparison, a comparison between numpy arrays delivers a one to one output.

```
In [51]: np.array([1,2,3])==np.array([1,3,3])

Out[51]: array([ True, False,  True])
```

Using comparisons in numpy arrays.

```
In [52]: N>6

Out[52]: array([False, False, False,  True,  True])

In [53]: N[N>6]

Out[53]: array([7, 9])
```

Defining "empty" vectors.

```
In [54]: zero_vector = np.zeros(5)
         zero_vector

Out[54]: array([0., 0., 0., 0., 0.])
```

Defining matrices.

```
In [55]: matrix1 = np.array([[1,2],[3,4]])
         matrix1

Out[55]: array([[1, 2],
                [3, 4]])
```

Defining "empty" matrices.

```
In [56]: zero_matrix = np.zeros((5,3))
         print(zero_matrix, zero_matrix.shape)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]] (5, 3)
```

Calculating the transpose matrix.

```
In [57]: zero_matrix_transpose = zero_matrix.transpose()
         print(zero_matrix_transpose, zero_matrix_transpose.shape)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]] (3, 5)
```

4.9 Loops & Conditions

This section contains a series of examples of how for and if statements can be used to generate code in Python.

```
In [58]: # print numbers in a list
         for i in range(10):
                 print('The number is: ',i)
```

```
The number is:  0
The number is:  1
The number is:  2
The number is:  3
The number is:  4
The number is:  5
The number is:  6
The number is:  7
The number is:  8
The number is:  9
```

```
In [59]: # print even numbers in a list
         for i in range(10):
             if i%2 ==0:
                 print('The even numer: ',i)
```

```
The even numer:  0
The even numer:  2
The even numer:  4
The even numer:  6
The even numer:  8
```

```
In [60]: # loop over a list of words and calculate their length
         words = ['Robert','Gordon','University','Data','Science']
         for w in words:
             print(w,len(w))
```

```
Robert 6
Gordon 6
University 10
Data 4
Science 7
```

```
In [61]: words = ['Robert','Gordon','University','Data','Science']
         for i in range(len(words)):
             print(i, words[i])
```

```
0 Robert
1 Gordon
2 University
3 Data
4 Science
```

```
In [62]: for i in range(5):
             print ('square of '+str(i)+' is',i**2)
```

```
square of 0 is 0
square of 1 is 1
square of 2 is 4
square of 3 is 9
square of 4 is 16
```

```
In [63]: # Loop over a dictionary
         for k, v in dictStudent.items():
              print(k, v)
```

```
Name Alex
Age 27
Grade A
```

```
In [64]: # Detect numbers
         x = int(input("Please enter number: "))
         if x < 0:
                 print('Negative')
         elif x == 0:
                 print('Zero')
         elif x == 1:
```

```
            print('Unit')
    else:
            print('Larger than 1')
```

```
Please enter number: 9
Larger than 1
```

4.10 Functions

To simplify code or when a series of instructions will be used several times, it is preferable to store them in a function.

```
In [65]: # Defining a function to calculate the distance between two points
         import numpy as np

         def distPoints(p1,p2):
             '''This function calculates the distance between two points p1 and p2 represented
             return np.sqrt(np.power(p1[0]-p2[0], 2)+np.power(p1[1]-p2[1], 2))

         # Use the function
         p1 = (3,4)
         p2 = (1,2)
         dist = distPoints(p1,p2)
         print(dist)
```

```
2.8284271247461903
```

```
In [66]: # Defining a function to calculate Fibonacci series numbers

         def fib(n):
              """Return a list containing the Fibonacci series up to n."""
              result = []
              a = 0
              b = 1
              while a < n:
                  result.append(a)
                  a, b = b, a+b
              return result

         # Call the function
         fib(10)
```

```
Out[66]: [0, 1, 1, 2, 3, 5, 8]
```

```
In [67]: # Why are these versions not working?
         def fib2(n):
              """Return a list containing the Fibonacci series up to n."""
```

```
        result = []
        a = 0
        b = 1
        while a < n:
            result.append(a)
            a = b
            b = a+b
        return result
    def fib3(n):
        """Return a list containing the Fibonacci series up to n."""
        result = []
        a = 0
        b = 1
        while a < n:
            result.append(a)
            b = a+b
            a = b
        return result

    print(fib2(10))
    print(fib3(10))
```

```
[0, 1, 2, 4, 8]
[0, 1, 2, 4, 8]
```

4.11 List Comprehension

Python allows to perform a loop and/or a conditional operation in one line.

```
In [68]: # Create a list containing the square of numbers from 0 to 9

         list(map(lambda x: x**2, range(10)))
```

```
Out[68]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
In [69]: # sum odd numbers from 0 to 9
         sum([i for i in range(10) if i%2==1])
```

```
Out[69]: 25
```

```
In [70]: # Create a list containing tuple pairs of numbers in the lists [1,2,3] and [3,1,4]
         ## excluding àirs of the same number.
         [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

```
Out[70]: [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

```
In [71]: # Transpose a matrix
         matrix = [
```

```
             [1, 2, 3, 4],
             [5, 6, 7, 8],
             [9, 10, 11, 12],]
         print(matrix)
         [[row[i] for row in matrix] for i in range(4)]
```

```
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

```
Out[71]: [[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

4.12 Typing

- Python is a dynamic language, which means that type check is performed at runtime.
- Command $x = 3$ first creates object 3, then creates variable x and then assigns a reference where variable x takes the value of object 3.
- References **ALWAYS** link to objects, never to other variables.

```
In [72]: # Example with integers
         x = 3
         y = x
         print('Value of x is', x)
         print('Value of y is', y)
```

```
Value of x is 3
Value of y is 3
```

```
In [73]: # Since integers are IMMUTABLE, a new object '2' is created. Python removes the refer
         y=y-1
         print('Value of x is', x)
         print('Value of y is', y)
```

```
Value of x is 3
Value of y is 2
```

```
In [74]: # Example with lists
         x = [1,2]
         y = x
         print('Value of x is', x)
         print('Value of y is', y)
```

```
Value of x is [1, 2]
Value of y is [1, 2]
```

```
In [75]: # Since lists are MUTABLE, the command y[0]=0 changed the list as an object, not x!
         y[0]=0
         print('Value of x is', x)
         print('Value of y is', y)
```

```
Value of x is [0, 2]
Value of y is [0, 2]
```

```
In [76]: # Solution: copy
         x = [1,2]
         y = x.copy()
         y[0]=0
         print('Value of x is', x)
         print('Value of y is', y)
```

```
Value of x is [1, 2]
Value of y is [0, 2]
```

5 How to convert the notebook into slides

5.1 Option 1

Install the "rise" plug-in in your Anaconda environment. This will enable a button in the top of jupyter notebook which will let you do a slideshow.

For more information about rise, check the following link:

- https://github.com/damianavila/RISE

5.2 Option 2

Open a terminal, go to the directory where the notebook is located and type:

- jupyter nbconvert **filename.ipynb** --to slides --post serve