Unit 5. Intro to Python

Héctor D. Menéndez

Endless Science

endsci.net

Index

Index

Introduction to Python

Python is a high level programming language.

Python advantages:

- It is easy and stable.
- There are several programs in Python.
- It leverages libraries in several other languages such as C, C++, Java, etc.

Python disadvantages:

- It is not efficient per se.
- There are two versions of the language that are not 100% compatible.

Introduction to Python

Created by Guido van Rossum in CWI, Centrum Wiskunde & Informatica.

It was released at the end of the 80s.

Version 1.0 is released in 1994.

After, different communities created two different versions (Python 2 and Python 3) that diverge in their syntax.

First Program

Open by typing `python3` in your terminal.

Write the code: `print(“Hello World”)`.

```
$> python3
>>> print("Hello World")
>>> quit()
```

Listing 1: Opening the Interpreter

The program will simply print "Hello World". To close the interpreter use `quit()`.

First Script

Create a file called script.py (or choose any name you want).

Write the code: `print(“Hello World”)` inside of the file.

Run the file by typing `python3 script.py` in your terminal.

```
$> nano script.py
$> python3 script.py
```

Listing 2: Opening the Interpreter

The program will simply print "Hello World".

Basic Data types in Python

Programs need to use data which shall be stored. This information is saved into variables.

The type of data stored in the variable is called: data type or type.

Basic data types in Python are:

- Integer: `int` (5 19 50321)
- Floating point or real number: `float` (1.98 3.1415 1.6E19)
- Boolean: `bool` (True or False)
- String: `str` ("Hello")
- List: `list` ([1, 2, 3, 4])
- Dictionary: `dict` ("color":"red","age":18)

Python Variables

Variables are used to store data values.

Python variables use dynamic types.

The variable definition just needs the variable name and value, the type is internally assigned.

Python Variables: Important Facts

Once a variable is defined, its type can be changed. It is important to pay attention to the use of each variable.

When you need other type you can declared other variable.

Basic data types store one value. If you want to store several values, you should use lists or dictionaries.

Python Variables

The variable name is a reference to the stored value. It is important to have easy names.

Digits and letters can be used.

Upper and lower letters are different. Python is "case sensitive". "variable" is different to "Variable".

Definition examples are:

- `var1 = 5`
- `MiVariable = "Hello"`
- `myList = [1,2,3,4]`

Python Variables

Every variable in Python is initialized before being used, that means that a value is assigned to that variable.

The assignation is done using the '=' character, for example:

- The variable takes an integer value: `var1 = 5`
- The variable takes a string value: `var1 = "Hola"`

The value and type of a variable might change during the execution.

Python Variables

```
$> python3
>>> aNumber=5
>>> print(aNumber)
>>> aNumberDec=3.5
>>> print(aNumberDec)
>>> aList =[1,2,3,4]
>>> print(aList)
>>> aBool = False
>>> print(aBool)
>>> aString = "Hello"
>>> print(aString)
>>> aDictionary = {"color":"red","age":18}
>>> print(aDictionary)
>>> quit()
```

Listing 3: Declarations in the Interpreter

Python Variables

This is the a source of bugs, so be careful with type.

```
$> python3
>>> aNumber=5
>>> print(aNumber)
>>> type(aNumber)
>>> aNumber=3.5
>>> print(aNumber)
>>> type(aNumber)
>>> quit()
```

Listing 4: Changing Variables' types

User Inputs

When we are doing scripts or programs, we can ask the user to provide inputs.
For that, we normally print a message specifying the type of input that the program needs, we wait for the user to introduce the input, and we store it into a variable.

Create a file called greetings.py with the following code. After, run it with `python3 greetings.py`

```
1 name = input("Introduce your name:")
2 age = input("Introduce your age:")
3 print("Your name is: " + name)
4 print("Your age is: " + age)
```

Listing 5: greetings.py

User Inputs

When the user provides a number you might need to transform it, in order to use it for operations.
Modify the second part of the script as follows:

```
1 age = input("Introduce your age:")
2 print("Your age is: " + age)
3 print(type(age))
4 age = int(age)
5 print(type(age))
```

Listing 6: greetings.py

Python Operators: Arithmetic

Once the variables have been defined, the user can apply operations, such as:

Symbol	Operation
$+$	Sum
$-$	Subtraction
$*$	Multiplication
$/$	Division
%	Module

```
var1 = 5
var2 = var1 - 3
var3 = var1 / var2
var4 = 10
var5 = var1 * var4
var3 = var5 % var2
```

Python Arithmetic

```
$> python3
>>> aNumber=9
>>> otherNumber=2
>>> print(aNumber)
>>> print(otherNumber)
>>> print(aNumber+otherNumber)
>>> print(aNumber-otherNumber)
>>> print(aNumber*otherNumber)
>>> division = aNumber/otherNumber
>>> print(division)
>>> type(division)
>>> int_division = aNumber//otherNumber
>>> print(int_division)
>>> type(int_division)
>>> print(aNumber%otherNumber)
>>> quit()
```

Listing 7: Operations on Interger Number

Python Arithmetic

```
$> python3
>>> aNumber=9.5
>>> otherNumber=2.0
>>> print(aNumber)
>>> print(otherNumber)
>>> print(aNumber+otherNumber)
>>> print(aNumber-otherNumber)
>>> print(aNumber*otherNumber)
>>> print(aNumber/otherNumber)
>>> print(aNumber%otherNumber)
>>> quit()
```

Listing 8: Operations on Floating Point Number

Other Operators

Operator	Description	Example	Equivalent
+=	Sum and assign	`var1 += 3`	`var1 = var1 + 3;`
-=	Subtract and assign	`var1 -= 3`	`var1 = var1 - 3;`
*=	Multiply and assign	`var1 *= 3`	`var1 = var1 * 3;`
/=	Divide y assign	`var1 /= 3`	`var1 = var1 / 3;`

Other Operators

```
$> python3
>>> aNumber=2
>>> aNumber+=2
>>> print(aNumber)
>>> aNumber-=1
>>> print(aNumber)
>>> aNumber*=4
>>> print(aNumber)
>>> aNumber/=2
>>> print(aNumber)
>>> quit()
```

Listing 9: Other Operators

Python Operators: Concatenation

A string and list can be joined to another one with the '+' operator:

```
string1 = "Hello"
string2 = "World"
string3 = string1 + string2
list1 = [1,2,3]
list2 = [4,5,6]
list3 = list1 + list2
```

Python Operators: Simple Concatenation

```
$> python3
>>> string1 = "Hello"
>>> string2 = "World"
>>> print(string1 + " " + string2)
>>> list1 = [1,2,3]
>>> list2 = [4,5,6]
>>> print(list1+list2)
>>> quit()
```

Listing 10: Simple Concatenations

Python Operators: Complex Concatenation

```
$> python3
>>> aNumber=7
>>> list1=[1,2,3]
>>> print("A string and a number " + str(aNumber))
>>> print("A string and another number " + str(4.5))
>>> print("A string and an operation " + str(aNumber +
6))
>>> print("A string and two numbers " + str(aNumber)
+ " " +  str(6))
>>> print("A string and a list: " + str(list1))
>>> print(list1 + [aNumber] + list1)
>>> quit()
```

Listing 11: Complex Concatenations

Python Operators: Repetition

A string and list can be repeated multiple times by using the '*' operator with the number of times you want to repeat the string or list:

```
string1 = "Hello"
string2 = "World"
string3 = string1*2 + " " + string2*3
list1 = [1,2,3]
list2 = [4,5,6]
list3 = list1*2 + list2*3
```

Python Operators: Repetitions

```
$> python3
>>> string1 = "Hello"
>>> string2 = "World"
>>> print(string1*2 + " " + string2*3)
>>> list1 = [1,2,3]
>>> list2 = [4,5,6]
>>> print(list1*2 + list2*3)
>>> quit()
```

Listing 12: Repetitions

Python Operators: Logic

Logic values: `True` or `False`.

The logical variable type is `bool`.

The value of logic operators is defined by their 'truth table'.

Python Operators: Logic

A logic variable represents something that is either true or false, for example:

```
hasDrivingLicense= False
isOlderThan18= True
```

You can combine logic variables in logical operations. For example:

```
canDrive = hasDrivingLicense and isOlderThan18
```

Python Operators: Logic

There are three main logic operators:

- `condition1 and condition2`: It is true only when the two conditions are true.
- `condition1 or condition2`: It is true when one of the conditions is true.
- `not condition`: changes the logical value of a condition.

Python Operators: Logic

a	b	a and b	a or b	not a	not b
True	True	True	True	False	False
True	False	False	True	False	True
False	True	False	True	True	False
False	False	False	False	True	True

Python Operators: Logical Operations

```
$> python3
>>> MarioJumpsOnTime= True
>>> MarioRuns = False
>>> MarioEscapesMonsters = MarioRuns or
    MarioJumpsOnTime
>>> MarioWins = MarioRuns and MarioJumpsOnTime
>>> MarioDies = not MarioJumpsOnTime
>>> quit()
```

Listing 13: Logic Operations

Python Operators: Logic and Numbers

It is also possible to evaluate conditions of numbers and assign them to logical operations:

```
age=17
hasDrivingLicense= False
isOlderThan18= (age >= 18)
```

Python Operators: Logic and Numbers

Symbol	Operator
==	Equal
!=	Not equal
>	Greater than
>=	Greater or equal than
<	Less than
<=	Less or equal than

Python Operators: Logical Operations

```
$> python3
>>> age = 17
>>> brain = True
>>> canDrink = (age > 21)
>>> canVote = not (age < 18)
>>> willVoteTrump = canVote and (not brain)
>>> hasTotalFreedom = canVote and canDrink
>>> needAttendance = (age < 14) or (age >= 90)
>>> hasJesusAge = (age == 33)
>>> quit()
```

Listing 14: More Logic Operations

Dictionaries

Dictionaries are good to save some relevant information and are easy to manage. They follow the structure <Key,Value>. The key needs to be unique.

You can use them to store and print relevant information:

```
bestSong={"Madonna":"Like a Virgin",
"The Killers":  "Mr Brightside"}
print(bestSong["Madonna"])
```

You can also update the information easily:

```
bestSong["Madonna"]="Like a Prayer"
```

Or add new one:

```
bestSong["Justin Bieber"]="Beauty and a Beat"
```

Dictionaries

```
$> python3
>>> bestSong={"Madonna":"Like a Virgin",
"The Killers":"Mr Brightside"}
>>> print(bestSong)
>>> print(bestSong["Madonna"])
>>> bestSong["Madonna"]="Like a Prayer"
>>> print(bestSong["Madonna"])
>>> bestSong["Justin Bieber"]="Beauty and a Beat"
>>> print(bestSong)
>>> favouriteSong={"Madonna":["Like a Virgin",
"Like a Prayer"],
"The Killers":["Mr Brightside","Under the gun"],
"Justin Bieber":"Beauty and a Beat"}
>>> print(bestSong)
>>> quit()
```

Listing 15: Dictionaries

Index

Comments in Python

Sometimes you need to add notes to your programs to remind you what a specific line is doing.

At any time, you can add comment by using the # symbol before you write your comment. This is useful mainly for scripts.

```
1 #This program says hello to a person
2
3 name = input("Introduce your name: ") #Stores the name
4
5 #Now, the program prints the name
6 #for that, I use a print and string concatenation.
7 print("Hello " + name +", nice to meet you!")
```

Listing 16: greetingsComments.py

Index

Exercise 1: Calculator

Create a script that ask the user to introduce two numbers, and calculate their sum, difference, product and division. Assume that the numbers are floating points. Example:

```
Enter the first number:  4
Enter the second number:  2
The sum is:  6
The difference is:  2
The product is:  8
The division is:  2
```

Exercise 2: Driver

Create a script that ask the user to introduce the age of a person and whether she/he has a driving license (as True or False), print True if the person is able to drive (older than 18 and has a driving license) or False in the opposite case. Use `bool(variable)` to transform the user input into a logical variable. Example:

```
Enter your age:  4
Do you have a driving license?  (True or False):  False
Can drive?  False
```

```
Enter your age:  18
Do you have a driving license?  (True or False):  True
Can drive?  True
```