Lecture-1

Introduction to Python Programming

Table of Contents

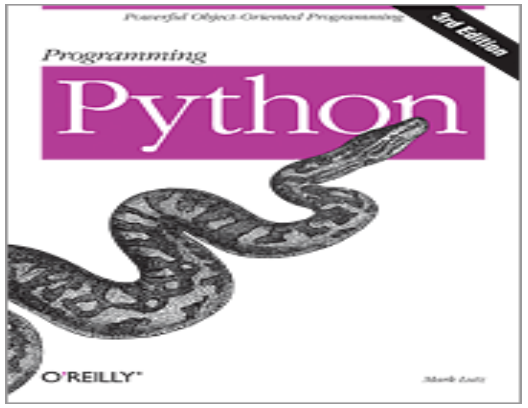
(Ref. Text, 50-70)

Introduction to Python

- **Python** is an interpreted, interactive, object-oriented, and high-level programming language developed in early 90's.
- Python is a general purpose programming language that is often applied in scripting roles.
- Python is designed to be highly readable. It uses English keywords frequently whereas other languages use punctuation, and it has fewer syntactical constructions than other languages.

History

- Invented in the Netherlands, early 90s by Guido van Rossum. Python was conceptualized in the late 1980s and its implementation was started in December 1989.
- Guido Van Rossum is fan of 'Monty Python's Flying Circus', a famous TV show in Netherlands, So he named Python after Monty Python.
- Nearly all known Python software initially developed by Guido van Rossum and released in 1991.



Difference of Programing & Scripting Language

Programming

- a **program is executed** *(i.e. the source is first compiled, and the result of that compilation is expected)*
- A "program" in general, is **a sequence of instructions written so that a computer can perform certain task**.

Scripting

- a **script is interpreted** A "script" is code written in a scripting language.
- A scripting language is nothing but **a type of programming language in which we can write code to control another** software application.

Scope of Python

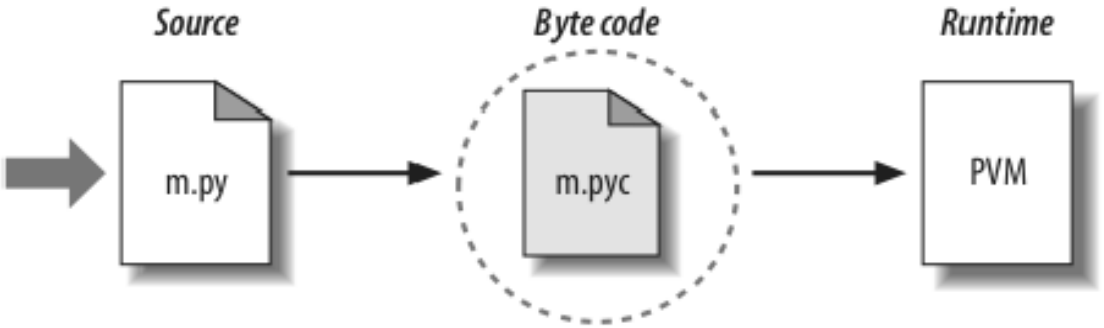
- Science
  - -Bioinformatic
  - -Artificial Intelligence
- System Administration
- Web Application Development
  - -CGI
- Software Development
- System Scripting.
- Gaming, Robotics and more

Why Python ?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, (etc.)
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Python Code Execution

- Python's traditional runtime execution model: source code you type is translated to byte code, which is then run by the Python Virtual Machine. Your code is automatically compiled, but then it is interpreted.



Source code extension is .py
Byte code extension is .pyc (compiled python code)

Syntax of Python

```
>>> print("Hello World!")
Hello World!

>>> 2+3
5
```

Relevant output is displayed on subsequent lines without the >>> symbol

Once you install any of the Python IDE, you can type code easily then before.

Example:

```
print (“Hello world!”)
Output: Hello world!
```

Variables in Python

Variables are containers for storing data values. Python has no command for declaring a variable. A variable is defined the moment you assign a value in it. Depending on the value the variable types is define.

Example:

```
x = 40                    # x is of type int
x =  "Bangladesh"         # x is now of type str
print(x)
Print(type(x))
```

Built-in Data Types

Formation	Data Types
Text Types:	str
Numeric Types:	Int, float, complex
Sequence Types:	List, tuple, range
Mapping Types:	dict
Set Types:	Set, frozenset
Boolean Types:	bool
Binary Types:	bytes, bytearray, memoryview

Arithmetic operators

OPERATOR	DESCRIPTION	SYNTAX
+	Addition: adds two operands	x + y
-	Subtraction: subtracts two operands	x - y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when first operand is divided by the second	x % y
**	Power : Returns first raised to power second	x ** y

Relational Operators

OPERATOR	DESCRIPTION	SYNTAX
>	Greater than: True if left operand is greater than the right	x > y
<	Less than: True if left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to: True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to: True if left operand is less than or equal to the right	x <= y

Assignment operators

OPERATOR	DESCRIPTION	SYNTAX
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b

Print Function

The built-in **print function** displays its argument(s) as a line of text:

```
In [1]: print('Welcome to Python!’)
        Welcome to Python!
```

Printing a Comma-Separated List of Items

```
In [2]: print('Welcome', 'to', 'Python!’)
        Welcome to Python!
```

Printing Many Lines of Text with One Statement, When a backslash (\) appears in a string, it’s known as the **escape character**.

```
In [3]: print('Welcome\nto\n\nPython!’)
        Welcome
        To

        Python!
```

Other Escape Sequences

Escape sequence	Description
\n	Insert a newline character in a string. When the string is displayed , for each newline, move the screen cursor to the beginning of the next line.
\t	Insert a horizontal tab. When the string is displayed, for each tab, move the screen cursor to the next tab stop.
\\	Insert a backslash character in a string.
\”	Insert a double quote character in a string.
\’	Insert a single quote character in a string.

Printing Expression with Escape Sequences

```
In [1]: print('Sum is', 7 + 3)
        Sum is 10

In [2]: print('Display \'hi\' in quotes’)
        Display 'hi' in quotes

Ln [3]: print("Display \"hi\" in quotes")
        Display "hi" in quotes

In [4]: print("""Display "hi" and 'bye' in quotes""")
        Display "hi" and 'bye' in quotes
```

Getting Input from the User

```
In [1]: name = input("What's your name? ")
        What's your name? Paul
In [2]: print(name)
        Paul

In [3]: value1 = input('Enter first number: ‘)
        Enter first number: 7
In [4]: value2 = input('Enter second number: ‘)
        Enter second number: 3
In [5]: add = value1 + value2
Out[6]: print(add)
        73
```

Multiple Input from the User

```
In [1]: a , b = input().split()
        5 10
In [2]: Print(a , b)
        5 10
In [3]: Print(type(a))
        <class ‘str’>
```

Objects and Dynamic Typing

- `In [1]: type(7)`
- `Out[1]: int`
- `In [2]: type(4.1)`
- `Out[2]: float`
- `In [3]: type('dog')`
- `Out[3]: str`

Values such as 7 (an integer), 4.1 (a floating-point number) and 'dog' are all objects. Every object has a type and a value. An object's value is the data stored in the object. The snippets above show objects of Python built-in types **int** (for integers), **float** (for floating-point numbers) and **str** (for strings).

Variables Refer to Objects

Assigning an object to a variable **binds** (associates) that variable's name to the object. As you've seen, you can then use the variable in your code to access the object's value:

- `In [4]: x = 7`
- `In [5]: x + 10`
- `Out[5]: 17`
- `In [6]: x`
- `Out[6]: 7`

After snippet [4]'s assignment, the variable x **refers** to the integer object containing 7. As shown in snippet [6], snippet [5] does not change x's value. You can change x as follows:

- `In [7]: x = x + 10`
- `In [8]: x`
- `Out[8]: 17`

Dynamic Typing

Python uses **dynamic typing**—it determines the type of the object a variable refers to while executing your code. We can show this by rebinding the variable x to different objects and checking their types:

```
• In [9]: type(x)
• Out[9]: int
• In [10]: x = 4.1
• In [11]: type(x)
• Out[11]: float
• In [12]: x = 'dog'
• In [13]: type(x)
• Out[13]: str
```

Getting Your Questions Answered

Online forums enable you to interact with other Python programmers and get your Python questions answered. Popular Python and general programming forums include:

- python-forum.io
- StackOverflow.com
- https://www.dreamincode.net/forums/forum/29-python/
- Colab Notebook for lab

Thank You