Introduction to Python

**A Readable, Dynamic, Pleasant,
Flexible and Powerful Language**

Amit Pathak

Institute of High Energy Physics

Chinese Academy of Sciences

January 25, 2019

Contents:

Why Use Python?

- Python is object-oriented

  Structure supports such concepts as polymorphism, operation overloading and multiple inheritance.

- It's Free (Open Source)

  Downlaoding and Installing Python is free and easy

  Source code is easily accessible

  Free doesn't mean unsupported! Online python is community is huge.

- It's portable

  Python runs virtually every major platform used today.

  As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform.

- It's Powerful

  Dynamic typing

  Built-in types and tools

  Library utilities

  Third party utilities (e.g. Numeric, NumPy, Scipy)

  Automatic memory management

Why Use Python?

- It's Mixable

  Python can be linked to components written in other languages easily

  Linked to fast, compiled code is useful to computationally intensive problems.

  Python is good for code steering and for merging multiple programs in otherwise conflicting languages.

  Python/C integration is quite common.

- It's Easy to use

  Rapid turnaround: no intermediate compile and link and steps as in C or C++

  Python programms are compiled automatically to an intermediate form called **bytecode**, which is interpreter then reads

  this gives python the development speed of an interpreter without the performance loss inherent in purely interpreted languages.

- It's easy to learn

  Structure and syntax are pretty intuitive and easy to grasp

Running Python

- How to call up a Python interpreter will vary a bit depending on your platform, but in a system with a terminal interface, all you need to do is type “python” into your command line.

  Example:

  $ python

  >>> print ‘Amit Pathak’

  Amit Pathak

  >>> x = [0,1,2]

  >>> x

  [0,1,2]

  >>> 2 +3

  5

  Python scripts can be written in text files with the suffix .py. The scrips can be read into the interprter in several ways.

Running Python

- Examples:

  $ python script.py

  #This will simply execute thescript and return to the terminal afterword.

  $ python -i script.py

  #The -i flag keeps the interpreter open after the script is finished running.

  $ python

  >>>execfile('script.py')

  $python

  >>> import script # DO NOT add the .py suffix. Script is amodule here.

Types and Operators: Types of Numbers (1)

- Python supports several different numeric types

Integers:

- Examples: 0, 1, 231, -4252
- Integeres are Implemented as C longs
- Note: dividing an integer by another integer will return only integer part of the quotient, e.g. typing 7/2 will yield 3.

Long Integers

- Examples: 9999999999999L
- Must end in either 1 of L
- Can be arbitrary long

Floating Point Numbers

- Examples: 0.0, 2.6, 1e10, 3.14e-12, 6.999E4
- Implemented as C doubles
- Division works normally for floating point numbers: 7./2. = 3.5
- Operations invloving both floats and integers will yield floats: 6.4 – 2 = 4.4

Types and Operators: Types of Numbers (2)

Octal Constants

- Examples: 0177, -01234
- Must start with leading 0

Hex Constants

- Examples: 0x9ff, 0X7AE
- Must start with a leading 0x or 0X

Complex Numbers

- Examples: 3+4j, 3.0+ 4.0j, 2J
- Must end in j or J
- Typing in the imaginary part first will return the complex number in the order Re + Im J

Types and Operators: Operations on List(1)

- Indexing: L1 [i] , L2 [i] [j]
- Slicing: L3 [i:j]
- Concatenation:

  >>> L1 = [0,1,2] ; L2 = [3,4,5]

  L1 + L2

  [0,1,2,3,4,5]
- Repetition:

  >>> L1*3

  [0,1,2,0,1,2,0,1,2]
- Appending:

  >>> L1.append (3)

  [0,1,2,3]
- Sorting:

  >>> L3 = [2,1,4,3]

  L3.sort ()

  [1,2,3,4]

Types and Operators: Operations on List(2)

- Reversal:

```
>>> L4 = [4,3,2,1]
>>> L4.reverse ()
>>> L4
[1,2,3,4]
```

- Shrinking:

```
>>>del  L4 [2]
>>> Lx [ i : j] = []
```

- Index and Slice assignment:

```
>>> L1 [1] = 1
>>> L2 [1:4] = [4,5,6]
```

- Making a list of Integers:

```
>>> range (4)
[0,1,2,3]
>>> range (1,5)
[1,2,3,4]
```

Types and Operators: Tuples

- Tuples are contained in parentheses ()
- Tuples can contain numbers, strings, nested sub-tuples, or nothing.
- Examples: t1 = (0,1,2,3) , t2 = (‘zero’, ‘one’) , t3 = (0,1, (2,3), ‘three’, (‘four, one’)) , t4 = ()
- Tuple indexing works just like string and list indeing.
- Tuples are immutable: individual elements can not be reassigned in place.
- Concatenation:

  >>> t1 = (0,1,2,3); t2 = (4,5,6)

  >>> t1+t2

  (0,1,2,3,4,5,6)
- Repetition:

  >>> t1*2

  (0,1,2,3,0,1,2,3)

Basic Statements: The if Statement

- If statements have the following basic structure:

```
# inside the Interpreter
>>> if condition:
...       action
...
>>>
```

```
# inside a script
if condition:
    action
```

Subsequent indented lines are assumed to be part of the if statement. The same is true for most other types of python statements. A statement typed into an interpreter ends once an empty line is entered, and a statement in a script ends once an unindented line appears. The same is true for defining functions.

If statements can be combined with else if (elif) and else statements as follows:

```
if condition1:              # if condition1 is true, execute action1
   action1
if condition2:              # if condition1 is not true, but condition2 is, execute
   action2                  #action2
else:                       # if neither condition1 nor condition2 is true
```

Basic Statements: The if Statement

Conditions in if statements may be combined using and & or statements

```
If condition1 and condition2:
    action1
# if both condition1 and condition2 are true, execute action1
if condition1 or condition2:
    action2
# if either condition1 or condition2 is true, execute action2
```

conditions may be expressed using the following operations:

< , <= , > , >= , ++ , != , in

Somewhat unrealistic example:

```
>>> x = 2; y = 3; L = [0,1,2]
>>> if (1<x<=3 and 4>y>=2) or (1==1 or 0!=1) or 1 in L:
... print 'Amit Pathak'
...
Amit Pathak
```

Basic Statements: The for Statements

for statements have the following basic structure:

- for elements in t.p4trk:

```
    print('>>>>>', t.p4trk[5])
    print('0:electron, 1:muon, 2:pion, 3:kaon, 4:proton')
```

The item i is often used to refer to an index in a list, tuple, or array

Example:

```
>>> L = [0,1,2,3]          #or, equivalently, range(4)
>>> for i in range (len(L)):
...         L(i) = L(i)**2
...
>>>L
[0,1,4,9]
```

Basic Statements: The for Statements

The User may combine statements in a myriad of ways

Example:

```
>>> L = [0,1,2,3]          #0r, equivalently, range(4)
>>> for i in range (len(L)):
...         j = i/2
...         if j – int(j) == 0.0:
...             L[i] = L[i]  + 1
...         else: L[i] = -i**2
...
>>> L
[1,-1,3,-9]
```

Scope Rules

- Python employs the following scoping hierachy, in decreasing order of breadth:
- Buil-in (Python)

  Predefined names(len, open, execfile, etc.) and types
- Global (Module)

  Names assigned at the top level of a module, or directly in the interpreter

  Names declared global in a function
- Local(Function)

  Names assigned inside a function definition or loop

  Example:

```
>>> a = 2          #a is assigned in the interpreter, so it's global.
>>> def f(x):      # x is in the function's argument list, so it's a local.
...     y = x +a   #y is only assigned inside the function, so it's local
...     return y   #using the sa
...
```

Some Useful Packages and Resources

- Histograms
- Tables
- Expressions
- Files and directories
- Drawing
  - Units and types
  - Canvases
  - Renderers
  - Layouts
- Plotting
- Lorentz geometry and kinematics
  - Reference Frames
  - vectors
  - Transformations and Frames
- Particle properties

source: http://indetermi.net/pyhep/dl/0.8.1/pyhep-0.8.1.pdf

Thank You