Python for AI

Anshu Pandey

Contents

- Introduction to Python
- Installing python
- Numbers, variables & basic operations
- Data Types – Lists, tuples & dictionary
- Various operators
- Defining functions
- Defining Classes
- Inheritance

Popular

Quora

DISQUS

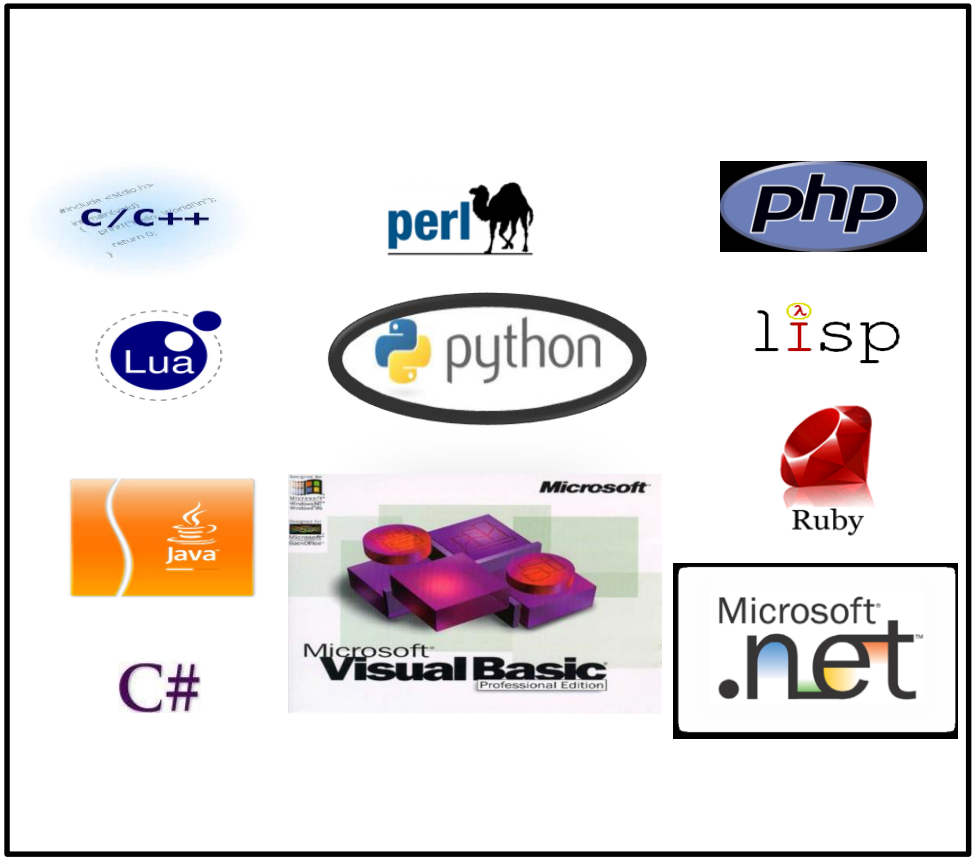
Google

Pinterest

What is Python ?

A dynamic, **open source programming language** with a focus on **simplicity and productivity**. It has an elegant syntax that is natural to read and easy to write.

There are so many programming Languages. Why Python ?



Why Python?

- **Python is *Simple & Beautiful***

- **Python is Object-Oriented**
  - Structure supports concepts as *Polymorphism*, *Operation overloading & Multiple Inheritance*

- **Python is Free (Open Source Language)**
  - Downloading & Installing Python is *Free & Easy*
  - *Source Code* is easily *Accessible*

- **Python is Portable & Powerful**
  - Runs virtually on every major Platforms used today
  - *Dynamic Typing, Built-in types & tools, Library Utilities, Automatic Memory Management*

Java:

```
public class HelloWorld {
    public static void main(String[], args) {
        System.out.println("Hello, World!");
    }
}
```

Python:

```
print "Hello, World"
```

There are two versions of Python currently available –

Python 2.x

Python 3.x

Which version to use

Python2 or **Python3?**

Lets see the basic differences-

Python 2.x is legacy, Python 3.x is the present and future of the language.

The most visible difference is probably the way the “print” statement works.
In python2 it’s a statement

```
print “Hello World!”
```

In Python3 it’s a function

```
print(“Hello World!”)
```

Getting Started with Python

Type the following text at the Python prompt and press the Enter:

```
>>> print "Hello, Python!“
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in

```
print ("Hello, Python!")
```

However in Python version 2.4.3, this produces the following result:

```
Hello, Python!
```

Python Strings:

Strings in Python are identified as a contiguous set of characters in between quotation marks.

```
str = 'Hello World!'

Print(str) # Prints complete string
print(str[0]) # Prints first character of the string
Print(str[2:5]) # Prints characters starting from 3rd to 6th
print(str[2:]) # Prints string starting from 3rd character
print(str * 2) # Prints string two times
print(str + "TEST“) # Prints concatenated string
```

Python Lists:

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).

```
list1 = [ ‘hello', 786 , 4.1, ‘Ram', 702 ]
list2 = [123, ‘Victor']

print(list1) # Prints complete list
print(list1[0]) # Prints first element of the list
print(list1[1:3]) # Prints elements starting from 2nd to 4th
print(list1[2:]) # Prints elements starting from 3rd element
print(list2 * 2) # Prints list two times
print(list1 + list2) # Prints concatenated lists
```

Positive and negative indices

```
>>> t = [23, ‘abc’, 4.56, [2,3], ‘def’]
```

Positive index: count from the left, starting with 0

```
>>> t[1]
‘abc’
```

Negative index: count from right, starting with –1

```
>>> t[-3]
4.56
```

Python Tuples:

A tuple is another sequence data type that is similar to the list. Unlike lists, however, tuples are enclosed within parentheses.

```
tuple1 = ( ‘hello', 786 , 2.23, ‘victor', 70.2 )
tuple2 = (123, ‘jay')
print(tuple1) # Prints complete list
print(tuple1[0]) # Prints first element of the list
print(tuple1[1:3]) # Prints elements starting from 2nd to 4th
print(tuple1[2:]) # Prints elements starting from 3rd element
print(tuple2 * 2) # Prints list two times
print(tuple1 + tuple2) # Prints concatenated lists
```

Python Dictionary:

Python 's dictionaries consist of key-value pairs.

```
dict1 = {'name': 'john','code':6734, 'dept': 'sales'}

print(dict1['name’]) # Prints value for 'one' key
print(dict['code’]) # Prints value for 2 key
print(dict1) # Prints complete dictionary
print(dict1.keys()) # Prints all the keys
Print(dict1.values()) # Prints all the values
```

The ‘in’ Operator

- Boolean test whether a value is inside a container:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- For strings, tests for substrings

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

- Be careful: the *in* keyword is also used in the syntax of *for loops* and *list comprehensions*

Lists are mutable

```
>>> li = [‘abc’, 23, 4.34, 23]
>>> li[1] = 45
>>> li
    [‘abc’, 45, 4.34, 23]
```

- We can change lists *in place.*
- Name *li* still points to the same memory reference when we’re done.

Tuples are immutable

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
>>> t[2] = 3.14

Traceback (most recent call last):
  File "<pyshell#75>", line 1, in -toplevel-
    tu[2] = 3.14
TypeError: object doesn't support item assignment
```

- You can't change a tuple.
- You can make a fresh tuple and assign its reference to a previously used name.

```
>>> t = (23, 'abc', 3.14, (2,3), 'def')
```

- *The immutability of tuples means they're faster than lists.*

Operations on Lists Only

```
>>> li = [1, 11, 3, 4, 5]

>>> li.append('a')     # Note the method syntax
>>> li
[1, 11, 3, 4, 5, 'a']

>>> li.insert(2, 'i')
>>>li
[1, 11, 'i', 3, 4, 5, 'a']
```

Operations on Lists Only

Lists have many methods, including index, count, remove, reverse, sort

```
>>> li = ['a', 'b', 'c', 'b']
>>> li.index('b')  # index of 1st occurrence
1
>>> li.count('b')  # number of occurrences
2
>>> li.remove('b') # remove 1st occurrence
>>> li
  ['a', 'c', 'b']
```

If else structure

```
if expression1:
        statement(s)
elif expression2:
        statement(s)
elif expression3:
        statement(s)
else:
        statement(s)
```

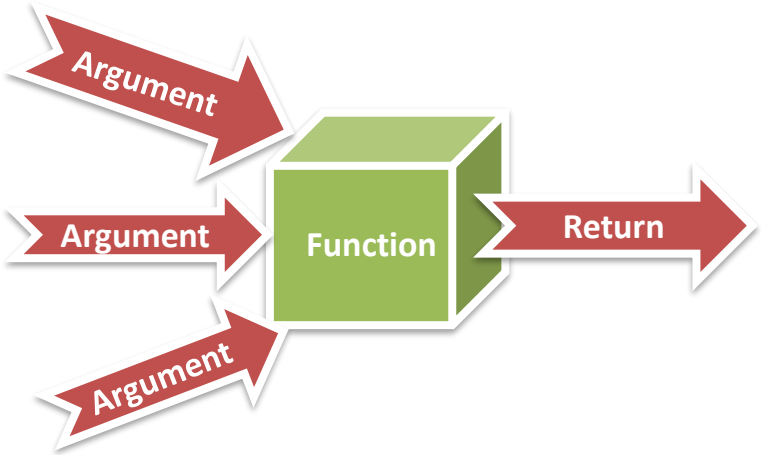
For loop

```
for iterating_var in sequence:
        statements(s)
```

While loop

```
while expression:
        statements(s)
```

Defining Functions



Defining Functions

```
def functionname( parameters ):
        "function_docstring"
        function_suite
        return [expression]
```

Example:

```
def printme( str ):
        "This prints a passed string into this function"
        print(str)
        return
```

Class

```
>>> class person:
        def sayhi(self):
            print('Hello, How are you?')
>>> p=person()
>>> p.sayhi()
Hello, How are you?
```

Inheritance :

```
>>> class A:
        def hello(self):
                print('I am Super Class')
>>> class B(A):
        def bye(self):
                print('I am Sub Class')

>>> p=B()
>>> p.hello()
I am Super Class
>>> p.bye()
I am Sub Class
```

Using in-built Modules

```
>>> import time
>>> print('The sleep started')
The Sleep started
>>> time.sleep(3)
>>> print('The sleep Finished')
The sleep Finished
```