**Python**

Python is a general-purpose, interpreted and object-oriented language. It’s weakly-typed and hence we don’t specify the type of variable before declaring any variable. All variables in Python are passed by reference.

Sample function:

def name():

fname = input("Enter your first name: ")

lname = input("Enter your last name: ")

fullname = fname + " " + lname

print("Your name is: ", fullname)

Sample if-else statement

z = 0

if z > 0:

print("z is positive")

elif z < 0:

print("z is negative")

else:

print("z must be 0")

**Loops:**

A sample for loop:

def countdown1():

for ct in range(9,0,-1):

print(ct,end=' ')

# or we can just pass an integer that will make index loop from 0 to that integer

for i in range(0, 4):

article = random.choice(articles)

A sample while loop:

def count\_down():

ct = 10

while ct>=1:

print(ct,end=' ')

ct = ct-1

**Operators:**

The operators in Python are similar to what are in C/C++ except that it doesn’t contain increment operators. We have to use ‘//’ operator to carry out integer division

'==' operator performs equivalence check while 'is' operator check for identity whether 2 variable refers to same data.

>>> L1 = [1, ('a', 3)]

>>> L2 = [1, ('a', 3)]

>>> L1 == L2, L1 is L2 # Equivalent? Same object?

(True, False)

The case is different in case of strings because internally python caches strings

>>> S1 = 'spam'

>>> S2 = 'spam'

>>> S1 == S2, S1 is S2

(True, True)

**Basic data types in Python:**

**Integers – int, float, complex,**

Following are the basic data types available in python

int, str, float, complex, bool (Can only be True or False)

**Following is the list of built-in objects in Python**

Numbers, Strings, Lists, Dictionaries, Tuples, File, Sets, Boolean, Modules, Classes, Functions

In terms of the core types, numbers, strings, and tuples are immutable; lists, dictionaries, and sets are not.

**List**

List is a compound data type that can contain different type of data in a single variable. Lists are usually enclosed in square brackets as:

fam = [ “go”, 2, “po”, 2.34 ]

To check for an item in list:

**if “<Item>” in “<List>”**

To check for absence of item

**if “<item>” not in “<List>”**

Looping over list:

def count\_a(alist):

ct = 0

for let in alist:

if let == 'a':

ct = ct + 1

print("There are",ct,"letter a's in the list.")

**List Slicing**

We can carve out a part of list using range specifiers:

name[start : last] #start and end are indices in list, with start included and end isn’t

Skipping start index will start including from first element as in name[:5]. While skipping end index will include everything starting from the given vertex name[2:]

**Assignment**

Let’s say we have a list x and we do y = x; then ‘y’ will contain just the reference of list and not the copy of list using equal sign to assign one list to another will just copy the reference and not the complete list. To realize that, you can assign list as:

y = x[:]

y = list(x)

**Deletion in list**

You can also remove elements from list using del() function as in:

**x = ["a", "b", "c", "d"]**

**del(x[1])**

Notes:

* A list can contain more lists.
* We can use negative indices to refer to the elements inside the list, -1 will correspond to last element, -2 to the second-last elemenet and so on.
* In Python, every tool that scans an object from left to right uses the iteration protocol.
* To pre-allocate some space in list, you can do:

l = 100\*[None]

**Methods:**

**list.index(<element>)**

**list.count(<element>) returns count of object**

**list.append(<element>)**

**list.remove(<element>)**

**list.reverse()**

# To remove an element from list

**list.pop(<index\_to\_remove>) or del list[<index\_to\_remove>]**

**list.sort(key=) # Sorts the list key = str.lower makes case insensitive sorting**

**Strings**

We can use both single quote and double quote to represent a string. Strings are immutable. Hence, assignment, such as: s[0] = “w” will fail. Strings also support raw strings wherein special meaning of '\n', '\t' etc. are ignore. It's specially useful in directory paths such as C:\node\temp.

Methods available with strings:

**str.capitalize()**

**str.replace(“s”, “sa”)**

**str\_up = <string>.upper();**

**str.lower()**

**str.find(<substring>)**

**str.replace(<old\_str>, <new\_str>)**

**str.split(<delimiter>)**

**str.upper()**

**str.lower()**

**str.isalpha/isdigit/isspace/islower/isupper/isdigit/isdecimal/istitle/isalnum/isnumeric() # returns a boolean value**

**str.rstrip() # removes whitespace characters from right**

**'{m} {m+1} … '.format(str1, str2...)**

**str.encode('utf8'/'utf16')**

**Formatting inside a print statement**

**{0:^25} # Middle-align with 25 spaces**

**{0:<25} # Left-align with 25 spaces**

**{0:>25} # Right-align with 25 spaces**

**{0: 25} # Left-align(String) or Right-align (An integer) with 25 spaces**

**Tuple**

It is like a list only and it’s just that it is immutable. Also, 'tuple' object has no attribute 'append'

**tuple.count(<element>) # Count number of entry in tuple**

**tuple.index(<element>) # Tells the index of element in tuple**

**Dictionary:**

It is an associative container just like an unordered\_map in C++. Each dictionary entry is just a tuple

Dictionaries can be appended to. Items cannot be retrieved by item number, because the items have no inherent order. The values can be retrieved by using their keys. Dictionaries have no notion of order and hence cannot be sliced ([3:7] doesn't mean anything) and they cannot be strided -- it might appear that you can, but the order is unpredictable.

To loop over dictionary:

**for key,value in d.items():**

**print(key, "--> ", value)**

**for item in d.items():**

**print(item)**

# To retrieve the keys:

**d.keys()**

# To retrieve the items:

**d.items()**

# To retrieve the values:

**d.values()**

# To create dictionary in one go:

**d = dict( name='bob', job='dev', age=40)**

# zipping

**d = dict(zip(['name', 'job', 'age'], ['Bob', 'dev', 40]))**

**Sets:**

**Set is an unordered and immutable data structure. You can make a set out of a sequence (such as string) as**

**X = set(‘spam’)**

**# Make a set with set literals**

**Y = { ‘a’, ‘b’, ‘c’ }**

**We can carry out the following operations with sets:**

**Intersection (X&Y), Union (X|Y), Difference (X-Y), Superset (X>Y)**

**Finding out differences:**

**set('spam') - set('ham') 🡺 {‘p’, ‘s’ }**

**Filtering out duplicates:**

**list(set([1, 2, 1, 3, 1])) 🡺 { 1, 2, 3}**

**Built-in functions:**

To know the type of variable

**type(<variable\_name>);**

To convert types of variable into another

**str() , int() , float()** etc

# To return the list of integer starting from start\_index up to, but not including, end\_index

**range(start\_index, end\_index); -> returns list**

#To round up the given number

**round(<float>, <precision>);**

# To return the max/min/sum of the list

**max/min/sum(list);**

# To round to closest integer

**round(<number>);**

# To get info regarding a function

**help(<function\_name>);**

# To get the memory address of object

**id(<object>)**

# To sort a list

**sorted(<list>)**

# To list down all the attributes in a filename/of an object

**dir(<module\_name>)**

# To retrieve the integer-ordinal value of character

**ord(<char>)**

# To return binary representation of an integer

**bin(integer)**

# To convert source to equivalent Python built-in data type

**eval(<source>)**

**Packages**

Packages are the backbone of Python because not everything is available as a core language feature

**import math** will import the complete math package

**from math import pi** will just import pi function from math package

**Random:**

***import random***

**random.seed()** # Seeds the random number generator

**random.random()** # Generates random number between 0 and 1

**random.choice(<list>)** # returns one of the element in list

**random.randint(<start>, <end>)** # Generates random number between start and end

**string str = random.choice(<list>)** # Returns the string from a list

**statistics**

**statistics.mean()**

**statistics.median()**

**statistics.mode()** # Most often repeated element

**statistics.stddev()**

**statistics.variance()**

**Math**

**math.pi # An attribute**

**math.sqrt(<number>)**

**import Fraction**

Tfrom fractions import Fraction

f = Fraction(2, 3)

f+1 🡺 Fraction(5, 3)

f + Fraction(1, 2) 🡺 Fraction(7, 6)

**import pickle**

The pickle module is a more advanced tool that allows us to store almost any Python

object in a file directly, with no to- or from-string conversion requirement on our part.

It’s like a super-general data formatting and parsing utility. To store a dictionary in a

file, for instance, we pickle it directly:

F = open('datafile.pkl', 'rb')

E = pickle.load(F)

E will contain {'a': 1, 'b': 2}

**import json**

Python dictionary is almost equivalent to JSON.

>>> import json

>>> rec{'job': ['dev', 'mgr'], 'name': {'last': 'Smith', 'first': 'Bob'}, 'age': 40.5}

'{"job": ["dev", "mgr"], "name": {"last": "Smith", "first": "Bob"}, "age": 40.5}'

>>> json.dump(rec, fp=open('testjson.txt', 'w'), indent=4)

>>> P = json.load(open('testjson.txt'))

>>> P

{'job': ['dev', 'mgr'], 'name': {'last': 'Smith', 'first': 'Bob'}, 'age': 40.5}

**import struct**

struct module knows how to both compose and parse packed binary data. In a sense,

this is another data-conversion tool that interprets strings in files as binary data.

>>> F = open('data.bin', 'wb') # Open binary output file

>>> import struct

>>> data = struct.pack('>i4sh', 7, b'spam', 8) # Make packed binary data

>>> data

b'\x00\x00\x00\x07spam\x00\x08'

>>> F.write(data) # Write byte string

To read data:

>>> F = open('data.bin', 'rb')

>>> data = F.read() # Get packed binary data

>>> data

b'\x00\x00\x00\x07spam\x00\x08'

>>> values = struct.unpack('>i4sh', data) # Convert to Python objects , '>' means Big-Endian, '<' refers to little

# endian '@' and '=' refers to native order

>>> values

(7, b'spam', 8)

**import copy**

Normal copy operations won't work in case of nested data structures. So, use deep copy instead of normal copy.

X = copy.deepcopy(Y)

**NumPy**

***import numpy***

NumPy advantage is that you can do efficient and good operations over the entire list in one go using their package such as finding out BMI(Body Mass Index) using arrays of weight and height.

Numpy arrays cannot contain elements of different types.

We can create 2-D NumPy arrays too. Each element of a 2-D NumPy array can be accessed as arr[i][j] or arr[i, j]

<array\_name>. Shape returns the dimensions of 2-D array

**Exception Handling:**

**try:**

**<code>**

**except <error> as e:**

**<code>**

**Working with files**

Python provides concise and utility functions to do file handling. Some of them are:

**file\_handle = open(<Filename>[, ‘w’)**

**file\_handle.close()**

You can iterate over the file using a simple loop as:

**for row in file\_handle**

**for line in open('myfile.txt'): # Use file iterators, not reads**

**print(line, end='')**

# You can specify the files while executing the python script. To access that functionality you need to:

**import sys**

Then , sys.argv[1] will be the first argument passed along with the file.

Preopened file objects in the sys module, such as sys.stdout

**Working with CSV files**

To work with CSV files, you need to:

**import csv**

csv.reader(<file\_handle>) return rows of list

**f = open(filename)**

**for row in csv.reader(f):**

**print(row[0], row[1], row[2])**

**f.close()**

To write CSV data to csv file

**f = open(filename, 'w', newline='')**

**for item in L:**

**csv.writer(f).writerow(item)**

**f.close()**

**Comprehension Expressions:**

A powerful technique to extract data from matrix. These are of the form:

[ <expression> for <element> in <list> if <conditional\_expression> ]

M = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]

[row[1] + 1 for row in M] will result in [3, 6, 9]

**Notes:**

Use ‘#’ as leading character to add comments in python code

Add end=’ ’ o make print not add new line.

We can write import statement inside the function also

String is left aligned and number is right aligned by default

[] is an empty list; () is an empty tuple; and {} is an empty dictionary.

> Only mutable objects—lists, dictionaries, and sets—may be changed in place; you cannot change numbers, strings, or tuples in place. 'bytearray' string type is mutable.

>>>

\* Lists, dictionaries, and tuples can hold any kind of object.

\* Sets can contain any type of immutable object.

\* Lists, dictionaries, and tuples can be arbitrarily nested.

\* Lists, dictionaries, and sets can dynamically grow and shrink

> Syntactically, nested objects are internally represented as pointers to separate pieces of memory.

> By default, assignment creates a reference to already existing object. To create a new copy, either call copy() method of

data type or call in-built functions as list(<orig\_list>), dict(<orig\_dict>),set<orig\_set>

> A = L[:] will create a copy of L and assign that to A

> Comparison between mixed/unrelated types is forbidden in Python

> Comparison between dictionaries is not allowed in Python 3.X

>