Python 2 or Python 3Installation

Python 2 is installed by default on MacOS

To Install Python 3, download the latest version of Python from python.org website and

run the setup to install python 3. To run python 3 , type python3 on terminal

Text Editor

We will use Sublime Text Editor for learning python

Variable

In python we are declaring variable with data type

Message = “this is a string variable”;

Simple Data types

Strings

Anything in single or double quotes are consider as string

String methods

String\_variable.upper();

String\_variable.lower();

String\_variable.title();

String\_variable.strip();

String Concatenation

We are using + to join to string values.

Numbers

Integers

2, 4 are integer numbers.

Float

All fractions number are called float like 22.10 etc

Comments

Comments start with # in python

Lists

Declaring list

list\_emptys = []

list\_names = [‘list1’, ‘list2’,’list3’]

Accessing List Elements

list\_names[0]

list\_names[1]

Changing / Modifying elements values in List

list\_name[1] = “New Value”

Adding new elements to List

We can add elements to list by two method

1. Append this function add element to end of list
2. Insert we can add element at any position in list

list\_append(‘additional\_value’);

list\_insert(0, ‘additional\_value\_insert’); this will add value at 0 index and move existing value to 1 index

Removing Elements from list

We can remove / delete elements from list by three methods

1. Pop remove element from end of list
2. del with del we can remove element from any index
3. Remove

del list\_name[0]

list\_name.pop[];

list\_name.remove[‘value’]

**Organizing List**

Sorting List Permanently

We can sort the list permanently using sort function. Once sort, not

revertible

list\_names.sort();

list\_name.sort(reverse=True) this will sort list in reverse order

Sorting List Temporarily

We can sort the list temporarily using sorted function.

sorted(list\_names)

displaying list in reverse order

list\_names.reverse()

**Looping Through entire List**

magicians = [‘alice’, ‘david’, ‘carolina’];

for magician in magicians:

print(magician);

**Making Numerical List**

We can create numerical list using range function

num\_lists = range(1,10);

Working with Part of List

Slicing List

players = [‘abid’, ‘babar’, ‘ismail’, ‘yousaf’, ‘usman’,’kamran’];

print(players[0-3];

Copying List

lists1 = list2[:] these are two different lists

list1 = list2 this is pointing to same list

Tuples

Tuples are same like List but not changeable and used () bracket not big brackets

Declaring Tuples

Dimension = (200,50);

Modifying

Not allowed

Looping through Tuples

Same like list

**IF Statements**

#1

If condition:

Do this;

#2

If condition:

Do this:

else:

do this:

#3

If condition:

Do this;

elif condition:

Do this;

Else:

Do this;

Relational Operators

== check equality

= assignment

!= not equal

> greater then

< less then

>= greater or equal

<= less or equal

Logical Operator

and

or

in check values in List, ‘value’ in list\_names:

not in check values not in list value’ not in list\_names:

Boolean

True

False

**Dictionary**

Dictionary in python is Key-Value pairs. Each key is connected to values. We can access

element using key.

Declaration or creation of Dictionary

dictionary\_names = {‘color’: ’green’, ‘points’:5}

dictionary\_names = {} this is empty dictionary example

Accessing Dictionary Elements

dictionary\_names[‘color’];

adding new elements to Dictionary

dictionary\_names[‘speed] = ‘medium’;

modify element in Dictionary

dictionary\_names[‘color’] = ‘yellow’;

removing element from Dictionary

del dictionary\_names[‘points’];

Looping Through Dictionary

for key, value in dictionary\_names.items():

print(key);

print(value);

for key in dictionary\_name.keys():

print(key)

for value in dictionary\_names.values():

print(value)

sorting dictionary keys

for key in sorted(dictionary\_names.keys()):

print(key);

list of dictionaries

alien\_0 = {‘color’:’green’, ‘points’: 5};

alien\_1 = {‘color’:’yellow’, ‘points’: 10};

alien\_2 = {‘color’:’red’, ‘points’: 15};

aliens = [alien\_0, alien\_1, alien\_2];

for alien in aliens:

print(alien);

**list inside dictionary**

pizza = {

‘crust’:’thick’,

‘toppings’: [‘mushroom’, ‘extra cheese’]

}

print(“you order “ + pizza[‘crust’]+”crust pizza with following topping”);

for topping in pizza[‘toppings’]:

print(topping);

**Dictionary inside Dictionary**

users = {

‘aeinstein’: {

‘first’: ‘albert’,

‘last’: ‘einstein’,

‘location’: ‘USA’

},

‘mcurie’ : {

‘first’: ‘marie’,

‘last’: curie’,

‘location’: ‘paris’

}

}

for username, user\_info in users.items:

print(username)

full\_name = user\_info[‘fist’] + “ ” + user\_inf[“last”];

location = user\_info[‘location’];

print(full\_name);

print(location);

User Inputs and While Loops

To take input from user, we use input function in python. If you we are using python2

then raw\_input() function is used to take input from user.

user\_name = input(“please enter your name. “);

print(user\_name)

note : input function return values in string.

**Looping in Python**

For Loop

for n in list\_names:

print(n)

While Loop

current\_number = 1;

while current\_number <= 5:

print(current\_number);

current\_number += 1;

**Functions**

Functions are name block of code that perform specific task.

Modules

If we have a function in separate file, that is called module.

Function Definition

def function\_name():

statement 1

statement 2

function\_name()

Passing Information to Function

def greet\_user(username):

print(“Hello “+ username)

greet\_user(“Rizwan”)

Arguments & Parameter

They are same thing but in python value passed at time of function calling is call

arguments and at time of function definition is called parameter.

Arguments Passing

We can pass arguments to function by multiple ways

1. Positional arguments

Python match the argument pass to function with parameter and use regardless of the order. This can some time make logical problem. Order matter in positional arguments

1. Keyword Arguments

It’s name value pair passed to function. In this case position of argument is not necessary, python automatically take care of it

1. Default Argument or Optional Parameter

If we assign a value to parameter at time of function definition, then it’s work like default value. Then if we not give the argument at time of calling system will take the default values. The default value parameter always be at end of parameter list.

1. Passing Arbitrary number of arguments to Function

Sometime we don’t know the number of parameter to function. Use the \* in front of parameter. Arbitrary parameter must be placed at last in parameter list.

def make\_pizza(\*toppings):

print(toppings)

make\_pizza(‘one argument’)

make\_pizza(‘one argument’,’ two argument’,’ three argument’)

1. Arbitrary Keyword Arguments

To use arbitrary parameter with key-values pair in function then we are using \*\* (double strict in front of parameter name in function definition)

Returning Values from Function

In python to return value from function, use the keywork return. We can return

dictionary from the function as well

passing List as a parameter to Function

it’s normal like other parameter

Store Function in modules

Write function in a file then save the file with meaning full name with .py

Extension

Importing Entire Modules

Using import keyword to import the module in program

Import module name

Import pizza

To call the function in module use below notation

pizza.function\_name()

Importing specific Function from modules

from modules \_name import function\_name

Giving Alias to function name

from modules \_name import function\_name as TST

import pizza.py as P

Importing All Functions from Module

from pizza import \*

Class

Class is blueprint for object

Creating and Using Class

class Dog():

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def sit(self):

print(self.name + “ is sitting now”)

def roll\_over(self):

print(self.name +” is roll over”)

Creating Instance of Class

my\_dog = Dog(‘jacky’, 5)

my\_dog.sit()

my\_dog.roll\_over()

Accessing Class Attributes

We will access class attribute / variable using object name dot attribute name

but best practice to access using method

my\_dog.name

Calling Class method / Function

We will access class method / function using object name dot method name

my\_dog.sit()

Setting default values for attributes

We are setting default values for attributes in constructor of class, here we set

the default color to red

def \_\_init(self, name, age):

self.name = name

self.age = age

self.color = ‘red’

Inheritance

In inheritance we inherit all the attribute and method from parent class means the child

class will have all the functionality of parent class.

# this is class definition

class Car():

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0

def get\_desciptive\_name(self):

self.long\_name = str(self.year)+ " "+ self.make+ " "+ self.model

return self.long\_name

def read\_odometer(self):

print("This car is "+ str(self.odometer\_reading) +" miles on it")

def update\_odometer(self, milage):

self.odometer\_reading = milage

def increment\_odometer(self, milage):

self.odometer\_reading += milage

class ElectricCar(Car):

def \_\_init\_\_(self, make, model, year):

super().\_\_init\_\_(make,model,year)

super(ElectricCar, self).\_\_init\_\_(make,model,year) # Python 2.7 line

my\_tesla = ElectricCar(‘tasla’, ‘model s’, 2016)

print(my\_tesla.get\_description\_name())

# class instance creation

my\_new\_car = Car('audi', 'A4', 2006)

print(my\_new\_car.get\_desciptive\_name())

my\_new\_car.update\_odometer(100)

print(my\_new\_car.read\_odometer())

my\_new\_car.increment\_odometer(20)

print(my\_new\_car.read\_odometer())

\_\_init\_\_() method for child class

The construction function is child class use the super() function to call the construction

function from parent class, like line from above example

super().\_\_init\_\_(make, model, year)

Defining Attributes and Method for Child Class

class Car():

--snip—

Class ElectricCar(Car):

def \_\_init\_\_(self, make, model, year):

super().\_\_init(make,mode,year)

self.battery\_size = 70

def describe\_battery(self):

print(“this Card has a “+ self.battery\_size+ “ kwh battery”)

Overriding Method from Parent Class

The method to override from parent class, we must use the same name of method in

child class

Instance as a Class Attribute

We can declare class instance inside another class and it will be consider as attribute of

that class

class Car():

--snip—

class Battery():

def \_\_init\_\_(self, battery\_size=70):

self.bettery\_size = battery\_size

def describe\_battery(self):

p print(“this Card has a “+ self.battery\_size+ “ kwh battery”)

Class ElectricCar(Car):

def \_\_init\_\_(self, make, model, year):

super().\_\_init(make,mode,year)

self.battery = Battery() # this is instance of above battery class

my\_tesla = ElectricCar(‘tasla’, ‘model s’, 2016)

print(my\_tesla.get\_description\_name())

my\_tesla.battery.describe\_battery()

Importing Class

Write code for Class say Car and save in a file say car.py with .py extension. Like

car.py

now create another file say my\_car.py and enter below code

from car import Car

my\_new\_car = Car(‘Toyota’, ‘Yaris’, 2014)

my\_new\_car.get\_description\_name()

importing single class from module

from car import Car

importing multiple classes from module

from car import Car, ElectricCar

importing entire module

import car

importing all classes from modules

from car import \*

Files & Exceptions Handling

Reading from File

Create a file with some content like 3.1415 and save as file.txt

Now create another file file\_ready.py and add following code

with open(‘file.txt’) as file\_obj:

Contents = file\_obj.read()

Print(contents)

Reading file line by line

We use for loop to read line by line content

with open(‘file.txt’) as file\_obj:

for line in file\_obj:

Print(line)