**Saroj Banjara**

**WEEKEND ACTIVITY- THEORETICAL JOURNEY**

**1. What is Pickling and Unpickling in Python? Explain with the help of example.**

Pickling - is the process whereby a Python object hierarchy is converted into a byte stream, and Unpickling - is the inverse operation, whereby a byte stream is converted back into an object hierarchy.

Pickling (and unpickling) is alternatively known as serialization, marshalling, or flattening.

import pickle

dictionary = {1: 'App',2: 'Ba', 3: 'Ca', 4: 'Do', 5: 'Eg'}

with open('data\_pickle.pkl', 'wb') as pickle\_file: #pickling

pickle.dump(dictionary, pickle\_file)

with open('data\_pickle.pkl', 'rb') as pickle\_file: #unpickling

new\_data = pickle.load(pickle\_file)

2. **How Memory Management is achieved in Python? Explain with the help of example.**

Memory manager is responsible for memory management in python. Memory manager keeps track of the number of references to each object in the program. Python manages objects by using reference counting and when the reference counting drops to zero, the part of memory manager called garbage collector automatically frees the memory from the particular object. This process of allocation and deallocation is fully automatic.

>>> x = 5

>>> print (x)

5

>>> del (x)

>>> print(x)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'x' is not defined

**3. Write a Program In Python to explain Multithreading in Python.**

Multithreading is a process of executing multiple tasks by running multiple threads at the same time. Multithreading is usually done when the execution of a task involves some waiting.

A simple example of multithreading is written below.

from time import sleep

from threading import \*

class Hello(Thread):

def run(self):

for i in range(5):

print('Hello')

sleep(1)

class Hi(Thread):

def run(self):

for i in range(5):

print('Hi')

sleep(1)

t1 = Hello()

t2 = Hi()

sleep(0.2)

t1.start()

t2.start()

t1.join()

t2.join()

print('Bye')

*# prints Hello -> Hi in a sequence order and after execution, it prints Bye at last. The sleep() gives time for second thread to run when first thread is generating a value.*

**4. What are the Collection in Python. Explain following terms with Example.**

Collections in Python are containers that are used to store collections of data like list, dict, set, tuple, etc. These are built-in collections. There are some modules which provide additional data structure to store collection of data. One such module is th Python collection module.

* [**namedtuple()**](https://docs.python.org/3/library/collections.html#collections.namedtuple)**:**

The namedtuple() returns a tuple with names for each position in the tuple.

from collections import namedtuple

Student = namedtuple('Student', 'fname, lname, age')

s1 = Student('John', 'Clarke', '13')

print(s1)

*#prints Student(fname='John', lname='Clarke', age='13')*

*# using \_make() to create a namedtuple instance with a list*

s2 = Student.\_make(['Adam','joe','18'])

print(s2)

*# prints Student(fname='Adam', lname='joe', age='18')*

* [**Counter**](https://docs.python.org/3/library/collections.html#collections.Counter) **:**

Counter is a subclass of a dictionary object. The Counter() function in collections module takes an iterable or a mapping as the argument and returns a Dictionary. We have to import Counter class before creating counter instance.

From collections import Counter

list = [1,2,3,4,1,2,6,7,3,8,1]

cnt = Counter(list)

print(cnt) *# returns Counter({1: 3, 2: 2, 3: 2, 4: 1, 6: 1, 7: 1, 8: 1})*

print(cnt[1]) # prints 3 - accesing counter item with key

* [**OrderedDict**](https://docs.python.org/3/library/collections.html#collections.OrderedDict)**:**

OrderedDict is a dictionary where keys maintain the order in which they are inserted., which means if you change the value of a key later, it will not change the position of the key.

from collections import OrderedDict

od = OrderedDict()

od['a'] = 1

od['b'] = 2

od['c'] = 3

print(od) *# returns OrderedDict([('a', 1), ('b', 2), ('c', 3)])*

for key, value in od.items(): #accessing each elements

print(key, value) #prints a 1 \n b 2 \n c 3

* **C**[**hainMap**](https://stackabuse.com/introduction-to-pythons-collections-module/#thechainmap)**:**

It is used to combine several dictionaries or mappings. It returns a list of dictionaries.

from collections import ChainMap

dict1 = { 'a' : 1, 'b' : 2 }

dict2 = { 'c' : 3, 'b' : 4 }

chain\_map = ChainMap(dict1, dict2)

print(chain\_map.maps) # prints [{'a': 1, 'b': 2}, {'c': 3, 'b': 4}]

# if we change the value of c from dict2 to 5. ChainMap will modify the change too.

Dict[‘c’] = 5

print(chain\_map.maps) # prints [{'a': 1, 'b': 2}, {'c': 5, 'b': 4}]

print (list(chain\_map.keys())) #prints ['c', 'b', 'a']

print (list(chain\_map.values())) #prints [3, 2, 1]

# to add new dictionary, new\_child() function is used.

dict3 = {'e' : 5, 'f' : 6}

new\_chain\_map = chain\_map.new\_child(dict3)

print(new\_chain\_map) # prints ({'e': 5, 'f': 6}, {'a': 1, 'b': 2}, {'c': 3, 'b': 4})

* [**Deque**](https://stackabuse.com/introduction-to-pythons-collections-module/#thedeque)**:**

Deque is a list optimized for inserting and removing items.

From collections import deque

list = ['a','b','c','d']

deq = deque(list) # using constructor

print(list) # prints deque ['a', 'b', 'c', 'd']

deq.append('d') # to add element to the right

deq.appendleft('e') # to add element to the left

print(deq) # prints deque(['e', 'a', 'b', 'c', 'd', 'd'])

deq.pop() # removes element from the right end

deq.popleft() # removes element from the left end

print(deq) # prints deque(['a', 'b', 'c', 'd'])

deq.clear() # remove all the elements from a deque

**5. Learn about PEP8 - How to write beautiful Python Code?**

PEP 8 is Python's style guide. It's a set of rules for how to format your Python code to maximize its readability. Writing code to a specification helps to make large code bases, with lots of writers, more uniform and predictable, too.

Below are some of the examples of how we can write the code beautifully:

* Funtion - function, my\_function
* Variable - x, var, my\_variable
* Class - Model, MyClass
* Method - class\_method, method

**6. Difference between the pip, virtualenv, conda?**

Pip is a Python Package Manager.

Virtualenv is a Python Environment Manager.

Conda is a Package Manager + Environment Manager + Additional Scientific Libraries.