Day_4 Python

Topics Covered :-

Difference Between is Vs ==

Difference between Shallow Copy Vs Deep Copy

Class Methods And variables

Static Method

OOPs- Multiple Inheritance

Python Eval Function

Concurrent Futures - Launching Parallel tasks

Vulture Library

Python Zip function

Secure hash Algorithm

Python OOPs- Public , Protected and Private

main_welcome(channel_name)

output: Welcome
Anirbet
Please Subscribe

Difference between is Vs ==

== is used to compare for ex
list 1 = list 2
True or false

If we use
list 1 is list 2 * If all elements are same in both
False

- Here one variable is created and by using is that is reffered from that memory location

Difference between shallow Copy Vs Deep Copy

ex list 1 = [1, 2, 3, 4]
list 2 = list 1

list 2[1] = 1000
[1, 1000, 3, 4] * It will also change lst 1

If we use .copy {shallow copy}
list 2 = list 1.copy
If we change in lst 2 it will not effect lst 1
Because this is making different memory location

If we use nested list

Here it is behaving like previous because it is suffering to same object in nested list

Class Methods And class variables

```
Class Car:
    base_price = 10000  # class variable
    def __init__(self, windows, doors, power):
        self.windows = windows
        self.doors = doors
        self.power = power
    def what_base_price(self):
        print("The base price is {}.format(self.base_price)
```

instance variables

```
car.base_price
```

output: 10000

```
# suppose we need to add amount after 1 year or we want to add inflation
@classmethod                                    → w.r.t class
def revise_base_price(cls, inflation):
    cls.base_price = cls.base_price + cls.base_price * inflation
```

```
Car.revise_base_price(0.10)
```

11000 for ex.

- Static Methods. @staticmethod

We cannot create multiple copy of static method. It runs once while application is running

```
@staticmethod
def check_year():
    if now.year == 2024:
        return True
    else
        return False
```

This run faster.

OOPs - Multiple Inheritance

```
class A:
    def method1(self):
        print("A class method is called")

class B(A):
    def method1(self):
        print("B class method is called")

class C(A)
    def method1(self):
        print("C class method is called")

class D(B,C)
    def method1(self):
        print("D class method is called")

d = D()
d.method1()
```

output: D class method is called

```
B.method1(d)
```

out: B class method is called

Python Eval function - Evaluating Expressions Dynamically

eval() {source, globals=None, locals=None}

Evaluates expressions which are written as strings.

for ex: eval(5*5)

=> 25

How does Eval work:

- Parse python Expression
- Compile into a byte code
- Evaluate the python expression
- It will return the result

Concurrent.futures - launching parallel tasks

The module provides a high-level interface for asynchronously executing callables

The asynchronous execution can be performed with threads using ThreadPoolExecutor or seperate process using ProcessPoolExecutor

eg

```
import time
from concurrent.futures import ThreadPoolExecutor
def returnnumber(a):
    return a

returnnumber(12)
```

out:- 12

```
start = time.time()
with ThreadPoolExecutor(max_workers=None) as executor:
    for result in executor.map(returnnumber, range(10)):
        print("Count: {0}".format(result))
print("Total time is: {0}".format(time.time()-start))
```

Vulture library :- used to find dead code.

It will helps us to find unused code in our project

Python Zip function - Parallel Iteration

Ex: lst1 = ["Krish", "Sam", "John"]
lst2 = ["a", 'b', 'c']

zip output = zip(lst1, lst2)

list(output)

output: [("Krish", "a"), ("Sam", 'b'), ("John", 'c')]

Creating a tuple with same index

- You can iterate through multiple iterator

SHA (Secure Hash Algorithms) - Base of Blockchain

- SHA 256
- SHA 384
- SHA 224

} many more are there

import hashlib

Ex: SHA 256 → 64 characters

```
sha1 = "Krish Naik"
# Process of Hashing,
# first step is to encode
# Then apply hashing Algorithms
hashedval = hashlib.sha256(sha1.encode())
print(hashedval)
```

output: <sha256 Hash object @ 0x00025F ---

```
# Convert the value to hexadecimal
hashedval.hexdigest()
```

Python OOPs - Public, Protected And Private

ex
```
Class Car():
    def __init__(self, windows, doors, enginetype):
        self.windows = windows
        self.door = doors
        self.enginetype = enginetype

audi = car(4, 5, "Diesel")
audi
```

output <__main__.Car at 0x1fa423c {memory}

Till Here all class variables are Public {they can be accessed from anywhere}

let's suppose we are creating class variables protected

* we will add underscore before
  ex self._window = windows

* we can over write the public class variables but in protected it only available in sub class.

Private:

self.__windows = windows } these are private and cannot be accessed from outside the class.
or modified.