

Technical Questions — Python Internship Assessment

1. What are Python decorators and give a use-case? (5 Marks)

Python decorators are functions that modify the behavior of other functions or methods without changing their original code. They wrap another function to extend or alter its behavior in a clean and reusable way

```
Users > vinayakajith > a.py > ...
1
2 import time
3
4 def timer(func):
5     def wrapper(*args, **kwargs):
6         start = time.time()
7         result = func(*args, **kwargs)
8         end = time.time()
9         print(f"{func.__name__} took {end - start:.4f} seconds")
10        return result
11    return wrapper
12
13 @timer
14 def compute():
15     time.sleep(1)
16
17 compute()
```

OUTPUT PROBLEMS DEBUG CONSOLE **TERMINAL** PORTS

```
source /Users/vinayakajith/Dellizo_intership/ToDo_ist_intership/venv/bin/activate
/Users/vinayakajith/Dellizo_intership/ToDo_ist_intership/venv/bin/python /Users/vinayakajith/a.py
(base) vinayakajith@vinayaks-MacBook-Air ToDo_ist_intership % source /Users/vinayakajith/Dellizo_intership/ToDo_ist_intership/venv/bin/activate
(venv) (base) vinayakajith@vinayaks-MacBook-Air ToDo_ist_intership % /Users/vinayakajith/Dellizo_intership/ToDo_ist_intership/venv/bin/python /Users/vinayakajith/a.py
compute took 1.0003 seconds
```

2. Explain the difference between deepcopy and copy in Python. (5 Marks)

Shallow copy means copying the object but not the things inside it. So if the object has other objects inside (like lists inside a list), both the original and the copy still share those inner objects. If you change something inside those nested objects, both will see the change.

Deep copy means copying the object and everything inside it, recursively. So the copy is fully independent from the original. Changing anything in the copy won't affect the original at all.

Example:

```
Users > vinayakajith > a.py > ...
1  import copy
2
3  original = [[1, 2], [3, 4]]
4
5  shallow = copy.copy(original) # shallow copy
6  deep = copy.deepcopy(original) # deep copy
7
8  shallow[0][0] = 'a' # change inside shallow copy
9  print(original)     # original also changes because nested objects are shared
10
11 deep[0][0] = 'b'    # change inside deep copy
12 print(original)     # original does NOT change because deep copy has its own nested objects
13
```

OUTPUT PROBLEMS DEBUG CONSOLE TERMINAL PORTS

```
• (venv) (base) vinayakajith@vinayaks-MacBook-Air Todo_ist_internship % /Users/vinayakajith/Dellizo_internship/ToDo_i
st_internship/venv/bin/python /Users/vinayakajith/a.py
[['a', 2], [3, 4]]
[['a', 2], [3, 4]]
```

3. What is a generator? How is it different from a list? (5 Marks)

A generator is a special kind of iterator that yields items one by one, generating values on the fly and using memory efficiently.

In contrast, a list stores all values in memory at once.

Generators are useful for large datasets or streams because they don't require storing the entire sequence in memory.

```
Users > vinayakajith > a.py > ...
1  def gen():
2      yield 1
3      yield 2
4      yield 3
5
6  for value in gen():
7      print(value)
8
```

OUTPUT PROBLEMS DEBUG CONSOLE TERMINAL PORTS

```
• (venv) (base) vinayakajith@vinayaks-MacBook-Air Todo_ist_int
/Users/vinayakajith/a.py
1
2
3
```

4. What are Python's GIL and its implications for multithreading? (5 Marks)

The Global Interpreter Lock (GIL) is a mutex in the CPython interpreter that allows only one thread to execute Python bytecode at a time.

This means:

- CPU-bound Python threads do **not** run in true parallelism, limiting multithreaded performance on multi-core processors.
- I/O-bound tasks (like network or file operations) can still benefit from multithreading.
- For CPU-bound parallelism, multiprocessing or external libraries (e.g., NumPy) are used.

5. Write a list comprehension to flatten a 2D list `[[1, 2], [3, 4], [5, 6]]` into `[1, 2, 3, 4, 5, 6]`. (2 Marks)

List comprehension is a quick way to create a new list by doing something to each item in another list. It's like a shortcut instead of writing a loop.

```
Users > vinayakajith > a.py > ...
1 flat_list = [item for sublist in [[1, 2], [3, 4], [5, 6]] for item in sublist]
2 print(flat_list) # Output: [1, 2, 3, 4, 5, 6]
3

OUTPUT PROBLEMS DEBUG CONSOLE TERMINAL PORTS
• (venv) (base) vinayakajith@vinayaks-MacBook-Air Todo_ist_intership % /Users/vinayakajith/Dellizo_intership/ToDo_ist_intership/venv/bin/python
/Users/vinayakajith/a.py
[1, 2, 3, 4, 5, 6]
```

6. What is the difference between @staticmethod, @classmethod, and normal methods? (4 Marks)

Normal methods: receive the instance (self) as the first argument. They can access and modify instance and class attributes.

@staticmethod: does not receive self or cls. Acts like a regular function inside the class namespace. It cannot access instance or class data unless explicitly passed.

@classmethod: receives the class (cls) as the first argument. Can access and modify class state, but not instance state.

```
Users > vinayakajith > a.py > ...
1  class MyClass:
2      def normal_method(self):
3          print(f"Called normal_method of {self}")
4
5      @staticmethod
6      def static_method():
7          print("Called static_method")
8
9      @classmethod
10     def class_method(cls):
11         print(f"Called class_method of {cls}")
12
13 obj = MyClass()
14 obj.normal_method()      # Works with instance
15 MyClass.static_method() # No instance needed
16 MyClass.class_method()  # Works with class
17
```

OUTPUT PROBLEMS DEBUG CONSOLE TERMINAL PORTS

```
• (venv) (base) vinayakajith@vinayaks-MacBook-Air Todo_ist_intership % /Users/vinayakajith/Dellizo_intership/ToDo_ist_int
/Users/vinayakajith/a.py
Called normal_method of <__main__.MyClass object at 0x10349ea50>
Called static_method
Called class_method of <class '__main__.MyClass'>
```

7. Explain the concept of context managers and write a custom one. (4 Marks)

Answer:

Context managers provide a way to allocate and release resources precisely when you want, typically using the with statement.

They handle setup and cleanup actions automatically.

Example custom context manager:

```
Users > vinayakajith > a.py > ...
1 class MyContext:
2     def __enter__(self):
3         print("Enter")
4         return self
5
6     def __exit__(self, exc_type, exc_val, exc_tb):
7         print("Exit")
8
9 with MyContext():
10     print("Inside context")
```

OUTPUT PROBLEMS DEBUG CONSOLE TERMINAL PORTS

```
• (venv) (base) vinayakajith@vinayaks-MacBook-Air Todo_ist_intership % /Users/vinayakajith/Dellizo_intership/ToDo_ist_int
/Users/vinayakajith/a.py
Enter
Inside context
Exit
```

X-----end-----X