**1. Module, Package, Library**

**Q1. What is a module in Python?  
A:** A module is simply a Python file (.py) that contains code (functions, classes, variables). Example: math.py.

**Q2. What is a package in Python?  
A:** A package is a collection of modules in a directory that contains an \_\_init\_\_.py file. Example: numpy is a package with many modules inside.

**Q3. How is a library different from a package?  
A:** A library is a collection of packages and modules that provide ready-to-use functionality. For example, **Pandas library** contains multiple modules and sub-packages.

**Q4.** How do you create your own module and package?  
**A:** Write a .py file as a module. Create a folder with multiple .py files and add \_\_init\_\_.py to make it a package.

**2. Function Overloading in Lambda**

**Q1. Does Python support function overloading like Java or C++?  
A:** No, Python does not support traditional function overloading. If multiple functions with the same name are defined, the last one overrides the earlier ones.

**Q2. How can you achieve function overloading in Python?  
A:** Using default arguments, \*args, \*\*kwargs, or using dispatcher.

**Q3. Can a lambda function be overloaded in Python?  
A:** No. Lambda is just a single expression function, it cannot be overloaded. But you can mimic overloading with conditionals inside the lambda.  
Example:

f = lambda x=None: "No arg" if x is None else f"Arg: {x}"

print(f()) # No arg

print(f(10)) # Arg: 10

**Q4. Why doesn’t Python need explicit function overloading?  
A:** Because Python is dynamically typed, the same function can accept different types of arguments.

**Q7. Example of function overloading with default arguments:**

def greet(name="Guest"):

return f"Hello, {name}!"

**3. Namespaces in Python**

**Q1. What is a namespace in Python?  
A:** A namespace is a container that maps names (identifiers) to objects. It prevents naming conflicts.

**Q2. What are the different types of namespaces in Python?  
A:**

* Local Namespace (inside a function).
* Global Namespace (module-level).
* Built-in Namespace (Python built-ins like len, id).

**Q3. Example showing namespace resolution in Python.**

x = 10 # global

def func():

x = 5 # local

print(x)

func() # 5

print(x) # 10

**Q4.** What is LEGB rule in Python?  
**A:** Python resolves names in order: **Local → Enclosing → Global → Built-in**.

**Q5. How to check available names in a namespace?  
A:** Use locals() for local namespace and globals() for global namespace.

**4. Overloading in Python**

**Q1. Does Python support method overloading?  
A:** Not directly. Defining multiple methods with the same name keeps only the last one. Overloading can be simulated using default parameters or \*args.

**Q2. Example of simulated method overloading using \*args:**

def add(\*args):

return sum(args)

print(add(2, 3))

print(add(1, 2, 3, 4))

**Q3. What about operator overloading in Python?  
A:** Python supports operator overloading using **dunder methods** (\_\_add\_\_, \_\_sub\_\_, etc.).

**Q4. Example of operator overloading:**

class Point:

def \_\_init\_\_(self, x, y): self.x, self.y = x, y

def \_\_add\_\_(self, other): return Point(self.x + other.x, self.y + other.y)

p1, p2 = Point(1,2), Point(3,4)

p3 = p1 + p2

print(p3.x, p3.y) # 4 6

**5. Dunder Methods**

**Q1. What are dunder methods?  
A:** Special methods in Python with double underscores (\_\_init\_\_, \_\_str\_\_, \_\_len\_\_). They allow customization of built-in behaviors.

**Q2. How do you use dunder methods for operator overloading?  
A:** By defining methods like \_\_add\_\_, \_\_mul\_\_, etc.

**Q3. Difference between \_\_new\_\_ and \_\_init\_\_?  
A:** \_\_new\_\_ creates the instance, \_\_init\_\_ initializes it.

**Q4. Why are dunder methods useful?  
A:** They let you integrate user-defined classes seamlessly with Python operators and built-ins.

**6. Closure in Python**

**Q1. What is a closure in Python?  
A:** A closure is a function that remembers variables from its enclosing scope even after the outer function has finished execution.

**Q2.** Example of a closure:

def outer(x):

def inner(y):

return x + y

return inner

add5 = outer(5)

print(add5(10)) # 15

**Q3. When are closures useful?  
A:** When you want to keep some state without using global variables or classes.

**Q4. Difference between closure and lambda?  
A:** A lambda is just a short anonymous function, while a closure remembers the environment where it was created.

**Q5. How to check if a function is a closure?  
A:** Inspect \_\_closure\_\_ attribute. If not None, it’s a closure.

print(add5.\_\_closure\_\_)