Unit 3

1-Mark Questions

Functions

Q: Which keyword is used to define a function in Python?
A: `def`.

1. **Q:** What does a Python function return if no return statement is used?
   A: `None`.
2. **Q:** What is the keyword used to return a value from a function?
   A: `return`.
3. **Q:** What is the default return type of a Python function?
   A: `NoneType`.
4. **Q:** Can a function in Python return multiple values?
   A: Yes, as a tuple.
5. **Q:** What are the two main parts of a function in Python?
   A: Function definition and function call.
6. **Q:** Which type of argument has a predefined value in a function?
   A: Default argument.
7. **Q:** Write the syntax of defining a function in Python.
   A: def function_name(parameters):

   statements

   8 . Q: Can functions be called inside other functions in Python?
   A: Yes.

   9. Q: What is a parameter in a function?
   A: A variable defined in the function header to accept input.

   10. Q: Which keyword is used to define a function in Python?
   A: `def`.

Types of Arguments

1. **Q:** Which type of argument is passed in the same order as defined?
   A: Positional argument.

2. **Q:** What is a keyword argument?
   A: An argument passed using parameter name = value format.
3. **Q:** Which type of argument provides a predefined value if no value is passed?
   A: Default argument.
4. **Q:** What symbol is used for variable-length positional arguments in a function?
   A: `*args`.
5. **Q:** What symbol is used for variable-length keyword arguments?
   A: `**kwargs`.
6. **Q:** Can positional and keyword arguments be mixed in a function call?
   A: Yes, but positional arguments must come first.
7. **Q:** Which type of argument is used when the number of inputs is unknown?
   A: Variable-length arguments (`*args`, `**kwargs`).
8. **Q:** In function definition, where must default arguments be placed?
   A: After positional arguments.

Scope of Variables

1. **Q:** What is a local variable?
   A: A variable declared inside a function, accessible only within it.
2. **Q:** What is a global variable?
   A: A variable declared outside all functions, accessible throughout the program.
3. **Q:** Which keyword is used to modify a global variable inside a function?
   A: `global`.
4. **Q:** What is the scope of a variable?
   A: The region of the program where the variable is accessible.
5. **Q:** If a local and global variable have the same name, which one is preferred inside the function?
   A: The local variable.
6. **Q:** What happens if you try to access a local variable outside the function?
   A: An error occurs (`NameError`).
7. **Q:** Which type of scope does Python follow?
   A: LEGB Rule (Local, Enclosing, Global, Built-in).
8. **Q:** Can a global variable be accessed inside a function without the `global` keyword?
   A: Yes, but only for reading, not modification.

Lambda Functions

1. **Q:** What keyword is used to define a lambda function in Python?
   A: `lambda`.
2. **Q:** What is the return type of a lambda function?
   A: A function object.
3. **Q:** Can a lambda function have multiple expressions?
   A: No, only one expression.
4. **Q:** Write a lambda function to square a number.
   A: `lambda x: x**2`.
5. **Q:** Is a lambda function always anonymous?
   A: Yes, unless assigned to a variable.

Map Function

1. **Q:** What does the `map()` function return?
   A: A map object (iterator).
2. **Q:** How many arguments does `map()` take?
   A: Two – a function and an iterable.
3. **Q:** Convert `[1, 2, 3]` into their squares using `map()`.
   A: `map(lambda x: x**2, [1, 2, 3])`.
4. **Q:** Can `map()` work with multiple iterables?
   A: Yes, with multiple iterables of the same length.

Filter Function

1. **Q:** What does the `filter()` function do?
   A: Filters elements from an iterable based on a condition.
2. **Q:** What does `filter()` return?
   A: A filter object (iterator).
3. **Q:** Write a `filter()` to extract even numbers from `[1,2,3,4]`.
   A: `filter(lambda x: x%2==0, [1,2,3,4])`.
4. **Q:** Can `filter()` return more elements than the original iterable?
   A: No.

Reduce Function

1. **Q:** Which module provides the `reduce()` function?
   A: `functools`.
2. **Q:** What does `reduce()` do?
   A: Applies a function cumulatively to reduce iterable to a single value.
3. **Q:** Find the sum of `[1,2,3,4]` using `reduce()`.
   A: `reduce(lambda x,y: x+y, [1,2,3,4])`.
4. **Q:** What is returned by `reduce()`?
   A: A single aggregated value.

Recursion in Python

1. **Q:** What is recursion?
   A: A function calling itself.
2. **Q:** What is the essential part of a recursive function?
   A: Base case (termination condition).
3. **Q:** Write a simple recursive function to calculate factorial.
   A: def fact(n): return 1 if n==0 else n*fact(n-1)
4. Q: What error occurs if recursion has no base case?
   A: `RecursionError` (maximum recursion depth exceeded).

Modules: Importing

1. **Q:** What is a module in Python?
   A: A module is a Python file containing functions, classes, and variables.
2. **Q:** How do you import a module in Python?
   A: Using the `import` statement, e.g., `import math`.
3. **Q:** What is the use of the `from` keyword in imports?
   A: It allows importing specific functions or variables from a module, e.g.,

   `from math import sqrt`.

Creating User-defined Modules

1 Q: How do you create a user-defined module?
A: By writing Python code in a `.py` file and importing it in another program.

2 Q: How do you access a function from a user-defined module?
A: By using the syntax `module_name.function_name()`.

Standard Modules (math, random, datetime)

1. **Q:** Which function from the `math` module is used to find square root?
   A: `math.sqrt()`.
2. **Q:** Which function from the `random` module generates a random integer?
   A: `random.randint(a, b)`.
3. **Q:** Which class from the `datetime` module represents current date and time?
   A: `datetime.datetime`.

Packages in Python

1. **Q:** What is a package in Python?
   A: A package is a collection of modules organized in directories with an `__init__.py` file.
2. **Q:** Give an example of a Python package.
   A: `numpy`, `pandas`, and `matplotlib` are Python packages.

2-Mark Question and Answers:

Functions:

1. **Q:** Differentiate between defining and calling a function in Python.
   A:
   - **Defining:** Writing a function with `def` keyword, e.g., `def add(a,b): return a+b`.
   - **Calling:** Executing the function, e.g., `add(2,3)`.
2. **Q:** What is the purpose of a `return` statement in a function?
   A: It sends a value back to the caller and terminates the function execution.

Types of Arguments:

1. **Q:** Differentiate between positional and keyword arguments with examples.
   A:
   - **Positional:** Based on order → `add(2,3)`.
   - **Keyword:** Based on names → `add(a=2, b=3)`.
2. **Q:** What are default arguments? Give an example.
   A: Arguments that take a default value if not provided.

def greet(name="Guest"): print("Hello", name)

3. **Q: What are variable-length arguments?**
   A: `*args` → accepts multiple positional arguments as a tuple.

   `**kwargs` → accepts multiple keyword arguments as a dictionary.

Scope of Variables:

1. **Q:** What is the difference between local and global variables?
   A:
   - **Local:** Declared inside a function; accessible only within it.
   - **Global:** Declared outside; accessible anywhere in the program.
2. **Q:** What is the use of the `global` keyword?
   A: It allows modification of a global variable inside a function.

Lambda Functions:

1. **Q:** Write a lambda function to add 10 to a given number.
   A: `lambda x: x + 10`.
2. **Q:** Give one difference between `def` function and a lambda function.
   A:
   - `def` can have multiple statements.
   - `lambda` can only have one expression.

Recursive Functions:

1. **Q:** Write a recursive function to calculate the sum of first `n` natural numbers.
   A: def sumn(n):

   return 0 if n==0 else n+sumn(n-1)
2. **Q: Why must a recursive function have a base case?**
   A: To stop infinite recursion and prevent `RecursionError`.

Modules:

1. **Q:** Differentiate between `import math` and `from math import sqrt`.
   A: `import math` imports the entire module, and we must use `math.sqrt( )`.
   `from math import sqrt` imports only the `sqrt` function, so we can directly call `sqrt( )`.

2. **Q:** What is the advantage of using modules in Python?
   A: Modules promote code reuse, organization, and make programs easier to maintain by grouping related functions and classes.

Creating User-defined Modules:

3. **Q:** Write the steps to create and use a user-defined module.
   A:
   - Create a file `mymodule.py` containing functions.
   - Import it in another program using `import mymodule`.
   - Call its functions with `mymodule.function_name()`.
4. **Q:** How is a user-defined module different from a standard module?
   A: A user-defined module is written by the programmer, while a standard module is pre-defined and comes with Python (e.g., `math`, `random`).

Standard Modules (math, random, datetime):

5. **Q:** Give two examples of functions from the `math` module with their use.
   A: `math.sqrt(16)` → returns `4.0` (square root).
   `math.factorial(5)` → returns `120` (factorial).
6. **Q:** Write two uses of the `random` module.
   A: `random.randint(1, 10)` generates a random integer between 1 and 10.
   `random.choice(['a', 'b', 'c'])` selects a random element from a list.
7. **Q:** What is the use of the `datetime` module? Give an example.
   A: It is used to work with dates and times.
   Example: import datetime

```
print(datetime.datetime.now())  # Prints current date and time
```

Packages in Python:

8. **Q:** How is a package different from a module?
   A: A module is a single Python file, whereas a package is a collection of related modules organized in a directory with an `__init__.py` file.
9. **Q:** Give an example of using the `numpy` package.
   A:

```
import numpy as np
arr = np.array([1, 2, 3])
print(arr)  # [1 2 3]
```

10. **Q: Why are packages important in Python?**
    A: Packages help in organizing large programs into smaller, manageable modules and provide powerful pre-built functionality (e.g., `numpy` for arrays, `pandas` for data analysis).

5-Mark Questions with Answers

Functions:

Q1. Explain the difference between defining a function with parameters and without parameters, with examples.
Answer:

- **Without Parameters:** Function does not accept inputs.

  ```
  def greet():
      return "Hello!"
  print(greet())
  ```

- **With Parameters: Function takes inputs and processes them.**

  ```
  def add(a, b):
      return a + b
  print(add(5, 3))
  ```

Difference:

- With parameters → More flexible (works with different inputs).
- Without parameters → Always performs a fixed task.

Types of Arguments:

Q2. Explain positional, keyword, default, and variable-length arguments with examples.
Answer:

```
# Positional
def add(a, b): return a+b
print(add(2, 3))  # order matters

# Keyword
```

print(add(b=5, a=10)) # order doesn’t matter

Default

def greet(name="Guest"): return "Hello " + name

print(greet())

print(greet("Vijaya"))

Variable-length

def show(*args): print(args)

show(1, 2, 3)

def details(**kwargs): print(kwargs)

details(name="Ravi", age=20)

- **Positional: Based on order.**
- **Keyword:** Based on name.
- **Default:** Uses default value if not passed.
- **Variable-length:** Accepts multiple arguments (`*args`, `**kwargs`).

Scope of Variables:

Q3. Differentiate between local and global variables with a Python program.
Answer:

```
x = 10  # global
def test():
    x = 5  # local
    print("Inside function:", x)

test()
print("Outside function:", x)
```

Output:

Inside function: 5

Outside function: 10

- Local variable → Accessible only inside the function.
- Global variable → Accessible throughout the program.

Using `global` keyword:

```
x = 10
def modify():
    global x
    x = 50
modify()
print(x)  # 50
```

Lambda Functions + Map, Filter, Reduce:

Q4. Explain `map()`, `filter()`, and `reduce()` functions with examples using lambda.
Answer:

```
from functools import reduce

nums = [1, 2, 3, 4, 5]
# map → applies function to each element
squares = list(map(lambda x: x**2, nums))
print(squares)  # [1, 4, 9, 16, 25]

# filter → selects elements satisfying condition
evens = list(filter(lambda x: x%2==0, nums))
print(evens)  # [2, 4]

# reduce → reduces list to single value
sum_all = reduce(lambda a,b: a+b, nums)
print(sum_all)  # 15
```

- **map()** → transforms all elements.

- **filter()** → selects specific elements.
- **reduce()** → combines elements into a single result.

Recursive Functions:

Q5. Explain recursion with an example of factorial calculation. Why is a base case necessary?
Answer:

- **Recursion:** Function calling itself.
- **Base case:** Stops infinite recursion.

```
def factorial(n):
    if n == 0 or n == 1:  # base case
        return 1
    else:
        return n * factorial(n-1)


print(factorial(5))  # 120
```

- `factorial(5)` → `5 * factorial(4)`
- `factorial(4)` → `4 * factorial(3)`
- Stops at `factorial(1)` = 1.

Without base case → infinite recursion → `RecursionError`.

Modules:

Q6. Explain with an example how to import a module in Python. Show the difference between `import` and `from ... import ...`.
A:

- A module is a Python file containing functions, classes, and variables.
- To use a module, we import it.

```
import math
print(math.sqrt(25))   # Using math module → Output: 5.0


from math import sqrt
```

print(sqrt(36)) # Direct use without prefix → Output: 6.0

- `import module` → imports the whole module.
- `from module import function` → imports only the required function.

Q7: List the types of modules. Explain any two types in detail.

Q8: What are the two ways of importing a module? Which one is beneficial? Explain.

Q9: Illustrate the internal structure of a typical python module with a suitable example.

User-defined Modules:

Q10. Write the steps and an example program to create and use a user-defined module.
A:
Steps:

1. Create a file `mymodule.py` with functions.
2. Import this module into another Python program.
3. Use the functions with the module name.

Standard Modules:

Q11. Explain with examples the uses of `math`, `random`, and `datetime` modules.
A:

1. **math module** → Provides mathematical functions.

```
import math
print(math.sqrt(16))    # 4.0
print(math.factorial(5))
```

2. **random module** → Used to generate random numbers.

```
import random
print(random.randint(1, 6))   # Random number between 1 and 6
print(random.choice(['A','B','C']))  # Random choice
```

3. **datetime module** → Deals with dates and times.

```
import datetime
print(datetime.datetime.now())
```

Packages:

Q12. What is a package in Python? Explain with an example of `numpy`.
A:

- A **package** is a collection of related modules stored in a directory with an `__init__.py` file.
- Example: `numpy` is a package for numerical computations.

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr)

print(arr.mean())
```

Thus, `numpy` provides powerful tools for handling arrays and mathematical operations.

Q13. Differentiate between a Module and a Package in Python with an example.
A:

- **Module:** A Python file containing functions, classes, variables.
- **Package:** Collection of modules in a directory with `__init__.py`.

Feature	Module	Package
Definition	A single `.py` file containing Python code	A collection of related modules in a directory
Structure	One file	Directory with `__init__.py`
Example	`math.py`	`numpy` package
Import Example	`import math`	`import numpy as np`

Q14. Illustrate the internal structure of a typical python module with a suitable example.

Q15. Illustrate the internal structure of a typical python package. How can you use a package in your program.

Q16. Write a program to create a user-defined module for mathematical operations (add, subtract, multiply, divide) and import it into another file.