"Important Questions and Lessons on Python – Mod 07" _Mohiuddin Khan

Below are the answers to each question with definitions and short code examples. I cited the following class notes, reference textbook, and authoritative online references for completeness:

(a) **Course Module Notes** (Mod06_Notes)

(b) **Textbook** (*Python Programming for the Absolute Beginner, 3rd ed.*)

(c) **Official Docs / Reputable Guides:** Built-in types, expressions/operators, exceptions, built-in functions (incl. type()), PEP 8 style

Questions and Answers:

1) What are the differences between statements, functions, and classes?

Statements

A *statement* is a single instruction that Python can execute. Examples include assignments, print commands, conditionals, and loops. Statements are the smallest executable unit.

Example:

```
x = 10      # assignment statement
print(x)    # function call statement
```

As described early in Dawson's text (Chapter 1–2), statements are the foundation of any Python program because they tell the computer what to do step-by-step.

Functions

A *function* is a reusable block of code that performs a specific task. It can take input (parameters) and may return output.

Example:

```
def add(a, b):
    return a + b
```

Your Module 06 and 07 notes emphasize that functions support *abstraction*, *encapsulation*, and *separation of concerns*: instead of repeating code, we bundle logic inside a named unit.

Classes

A *class* is a blueprint for creating *objects*. Classes define both **attributes** (data) and **methods** (functions inside a class).

Example:

```
class Student:
    def __init__(self, first, last):
        self.first = first
        self.last = last
```

Dawson introduces classes in Chapter 8, explaining how they let programmers model "things" using data and behavior.

2) What is the difference between a data class, a presentation class, and a processing class?

This follows the **Separation of Concerns (SoC)** design pattern emphasized in Module 07.

Data Class

Purpose: Holds and validates data.
Contains: attributes, properties, simple validation.

Example from Assignment07:

```
class Student:
    def __init__(self, first, last, course):
        self.first = first
        self.last = last
        self.course = course
```

Presentation Class

Purpose: Handles all user interaction (input/output).
Example:

```
class IO:
    @staticmethod
    def output_menu(menu):
        print(menu)
```

Processing Class

Purpose: Handles the logic of reading, writing, or transforming data.

Example:

```
class FileProcessor:
    @staticmethod
    def write_data_to_file(file_name, data):
        json.dump(data, open(file_name, "w"))
```

3) What is a constructor?

constructor is a special method named __init__ that runs automatically whenever a new object is created. It sets up initial attribute values.

From Dawson (Chapter 8, "Using Constructors"): constructors prepare each object with the attributes it needs.

Example:

```
class Student:
    def __init__(self, name):
        self.name = name
```

4) What is an attribute?

An *attribute* is a variable stored inside a class instance. It represents data that belongs to an object.

Example:

```
student.age = 18   # age is an attribute of student
```

Attributes are discussed in Dawson's "Using Attributes" section in Chapter 8.

5) What is a property?

A *property* allows controlled access to an attribute using getter and setter methods while still using simple dot notation.

Important in Module 07:
Properties allow **validation**, **encapsulation**, and **cleaner syntax**.

Example:

```
class Student:
    @property
    def first_name(self):
        return self._first_name


    @first_name.setter
    def first_name(self, value):
        if value.strip() == "":
            raise ValueError("First name cannot be empty")
        self._first_name = value
```

Dawson covers properties in “Controlling Attribute Access.”

6) What is class inheritance?

Inheritance allows one class (child/derived class) to reuse and extend another class (parent/base class).

Module 07 Lab 03 and Dawson Chapter 9 both demonstrate this.

Example:

```
class Person:
    pass


class Student(Person):
    pass
```

This means Student inherits all attributes and methods from Person by default.

7) What is an overridden method?

An *overridden method* redefines a method from the parent class inside the child class to change its behavior.

This is discussed in Module 07 and in Dawson’s chapter on inheritance.

Example:

```
class Person:
    def describe(self):
        return "I am a person"


class Student(Person):
    def describe(self):
        return "I am a student"
```

Here, Student.describe() overrides Person.describe().

8) What is the difference between Git and GitHub Desktop?

Git-

- A version control system.
- Runs in the command line.
- Manages repositories locally and remotely.
- Used by programmers for branching, merging, and tracking changes.

GitHub Desktop-

- A graphical user interface (GUI) application.
- Allows users to use Git without the terminal.
- Helps with commits, branching, cloning, and pushing in a simplified visual way.
- Requires a GitHub account and integrates with github.com.