Assignment 19

**METACLASSES & DATACLASSES**

**RESEARCH WORK**

**Q1: What are metaclasses and when should you use them?**

**Ans: Metaclasses** are actually the classes of classes. They are a blueprint for creating classes as classes are a blueprint for creating instances. By using metaclasses we can control class creation and its behavior.

**EXAMPLE:**from datetime import datetime

class TrackInstances(type):

  def \_\_call\_\_(cls, \*args, \*\*kwargs):

    instance = super().\_\_call\_\_(\*args, \*\*kwargs)

    if not hasattr(cls, "\_instances"):

      cls.\_instances = []

    cls.\_instances.append(instance)

    instance.creation\_time = datetime.now()

    return instance

class Tracker(metaclass=TrackInstances):

  def \_\_init\_\_(self, name):

    self.name = name

t1 = Tracker("First")

t2 = Tracker("Second")

print(Tracker.\_instances)

for obj in Tracker.\_instances:

  print(f"Object name: {obj.name}, Created at: {obj.creation\_time}")

**Q2: How does @dataclass simplify class creation compared to traditional \_init\_()?**

**Ans: @dataclass** simplifies class creation as compared to traditional \_\_init\_\_() because the code becomes shorter and simpler when using @dataclass. The same result can be obtained by writing shorter code instead of longer code.

**EXAMPLE:**

from dataclasses import dataclass

@dataclass

class Person:

  name: str = "Ahmed"

  age: int = 20

  email: str = "Ahmed@example.com"

  def \_\_post\_init\_\_(self):

    self.valid\_age = self.age > 0

    print("Post init...")

p1 = Person("Ali",23,"Ali@example.com")

print(p1)

**Q3: What impact does frozen=True have in dataclasses?**

**Ans: frozen=True** creates an immutable dataclass in which the attributes can’t be modified after they are created. If we try to change them we, get an error.

**EXAMPLE:**

from dataclasses import FrozenInstanceError, dataclass

@dataclass(frozen=True)

class ImmutablePoint:

  x: int

  y: int

p1 = ImmutablePoint(3, 4)

print(p1)

try:

  p1.x = 4

  print(p1)

except FrozenInstanceError as e:

  print("FrozenInstanceError", ":",  e)

**Q4: Explain how \_post\_init\_() enhances data validation.**

**Ans: \_\_post\_init\_\_()** enhances data validation by implementing conditions inside dataclasses such as age validations. We can implement conditions and can get the desired output.

**EXAMPLE:**from dataclasses import dataclass

@dataclass

class Person:

  name: str = "Ahmed"

  age: int = 20

  email: str = "Ahmed@example.com"

  def \_\_post\_init\_\_(self):

    self.valid\_age = self.age > 0

    print("Post init...")

p1 = Person("Ali",23,"Ali@example.com")

print(p1)

**Q5: Compare metaclasses and decorators in terms of modifying class behavior.**

**Ans: Metaclasses** provide deep class control and control the class behavior. It runs as the class is created. It is used to enforce methods in a class.

**Decorators** add simple modifications to the class but don’t have all the control. It runs after the class is created. It is used to add loging.