AnIn211 - Practical programming in Python

PW05 - OOP - Basics(2)

PW05

Object Oriented Programming

Basics (2)

Module AnIn211 — 2nd year

Contents

1. Objectives and Reminders

1.1 Objectives

✓ Distinguish between instance attributes and class attributes and know how to manipulate them. ✓ Understand the role and limitations of a destructor (`__del__`). ✓ Read a simple UML class diagram

and translate it into Python code.

1.2 Quick reminders

• **Class attribute**: variable shared by all instances of the class. It is accessed and modified via the class name, for example: `ClassName.class_attribute`. • **Destructor (`__del__`)**: method called when the object is destroyed, used to free resources. • **UML notation**: graphical representation of a class as a rectangle divided into three compartments: name, attributes, and methods.

1.3 Course material (studied at home)

• Python*A2*C05_Objet1.pdf

1.4 Deliverables

✓ Submit all Python (.py) files produced during this lab on Moodle.

---

2. Guided exercises

2.1 Definitions and concepts

a. What is the difference between a class attribute and an instance attribute? b. In which case is it relevant to use a class attribute? Provide a concrete example. c. What is the role of a destructor `__del__` in Python? Give a reasonable use case. d. Name the three compartments of a UML class diagram.

2.2 Predicting the code output

Guess the output of the code **without executing**, then check your answer by running it.

Python

```python
class Counter:
    total = 0

    def __init__(self, name):
        self.name = name
        Counter.total += 1
        print(f"created: {self.name} / total={Counter.total}")

    def __del__(self):
        Counter.total -= 1
        print(f"destroyed: {self.name} / total={Counter.total}")

c1 = Counter("A")
c2 = Counter("B")
del c1
print(c2.total)
```

3. PW – Practical exercises

3.1 The Book class

In a file named `book.py`, implement the class `Book`, using the UML diagram below.

Figure 1 – UML diagram - Book class

• The method `display()`: displays the book's information. • Outside the class, create an instance of `Book` and call the `display()` method.

3.2 The Point class

In a file `point.py`, define a class `Point` representing a point in the 2D plane. Each `Point` object has two attributes `x` and `y` of type float.

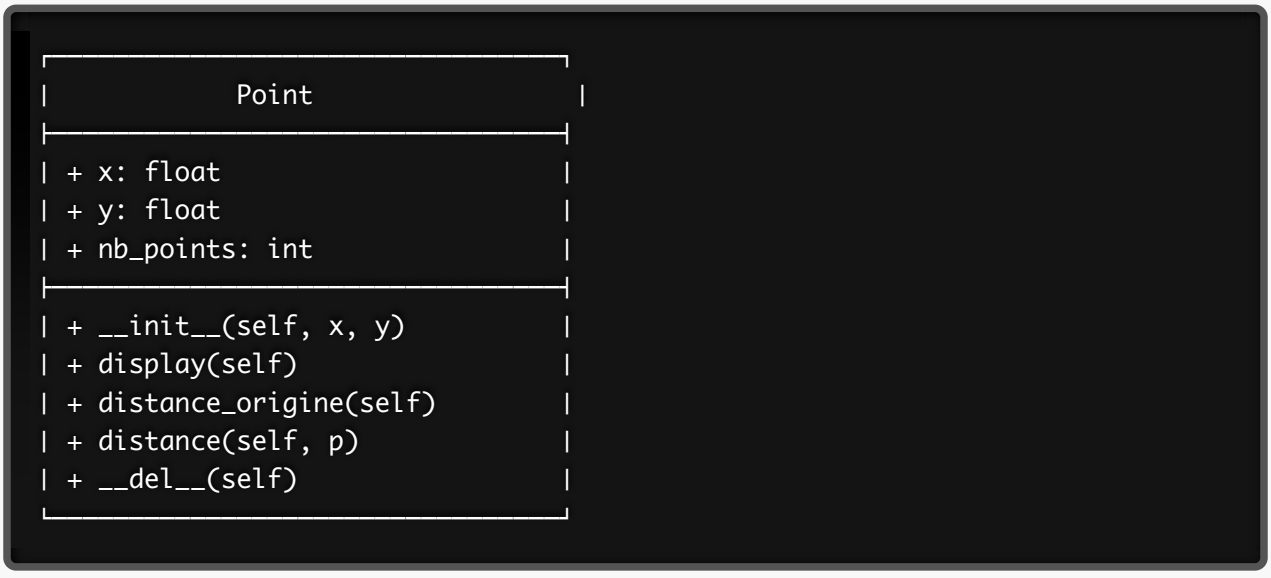

Figure 2 – UML diagram of class Point

To implement in the Point class:

a. Constructor without parameters

• Add to the `Point` class a constructor (`__init__(self)`) without parameters. • Initialize the attributes `x` and `y` to `0.0`. • Test the class by creating an object `p1`, then modifying its attributes.

b. Constructor with parameters

• Modify the constructor by adding two parameters (in addition to `self`) to initialize `x` and `y`. • ⚠️ Provide default values `0.0` to avoid errors during instantiation without arguments: `__init__(self, x=0, y=0)`. • Add the method `display(self)` which displays the coordinates in the format `(x, y)`. • Test your class by creating an object `p2` with random coordinates using `random.random()*10`, and calling the `display` method.

c. Calculation methods

• Add the method `distance_origine(self)`: this method calculates and returns the distance to the origin (0, 0). - Use the Euclidean distance formula: $d = \sqrt{(x^2 + y^2)}$ - Display the distance of `p2` to the origin with `distance_origine`.

• Add the method `distance(self, p)`: which takes another `Point` as argument and returns the distance between the two points: - $d = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$ - Calculate and display the distance between `p1` and `p2` using the `distance` method.

d. Class attribute

• Add to the class the class attribute `nb_points` initialized to 0 (counter of the number of instances). • In the `__init__`, increment the class attribute `nb_points` at each object creation. • Add a destructor `__del__(self)` which decrements `nb_points` when an object is destroyed.

e. Final test:

• Create several `Point` objects and display the value of the class attribute `nb_points`. • Delete one or more references with `del` and observe the counter update.

Note: depending on the execution environment, the destruction of an object occurs when all its references are deleted. In a simple script, the counter update should be visible immediately after `del`.

---

4. Mini-Project – Weather Station (To finish at home)

Objective: Evolve the `CapteurTemperature` class (`capteur.py`) by adding class attributes and a destructor `__del__`, then model the class with a simple UML diagram.

a. Class attribute and destructor:

• **Class attributes to add in `CapteurTemperature`:** - `nbr_capteurs`: of type int and initialized to 0 (instance counter, total number of sensors) - `PLAGE`: a tuple containing 2 values initialized to (-50.0, 60.0) (canonical valid range for temperature).

• Modify the `__init__` constructor to increment the total number of sensors at each instance creation. • Add a destructor `__del__` that decrements the counter and displays a message when the object is destroyed, for example "Sensor [name] has been removed".

b. Advanced calculations

• Add a method `calculer_ecart_type` to calculate the standard deviation of values recorded in `historique`. - Handle the case of less than 2 measurements: return `None` (or raise a documented exception).

c. Practice

• After the 5 measurement simulations, display the `nbr_capteurs` to verify proper incrementation. • Display the standard deviation of each sensor using the new method.

d. UML Diagram

• On paper or with a modeling tool, draw a UML diagram of the `CapteurTemperature` class showing its attributes (with their visibility and type) and its methods.