TMA Assignment 15

Course Name: Programming and basic Data Analysis Using Python

Assignment Topic: Object-Oriented Programming

Number of Questions: 1

Assignment Weight: 2 points

Semester: 2025B

Final Submission Date: 12.6.2025

Please Note:

- **You must strictly follow the function names exactly as written.**
- The program must be internally documented using comments in English only. At the beginning of the program, **include a comment explaining what the program does in general, and throughout the program, explain the code using comments.** These comments should follow the **PEP 8** standard and the examples shown in section 1.9 of the course website.
- **Do not add functions beyond those explicitly required in the assignment, unless it is explicitly stated otherwise. In addition, you may define auxiliary functions only if they are private (i.e., not public!) within your own functions.**
- **Do not use advanced material or content that was not taught in the course.**
- You must work independently and submit individual work.
- You may use external resources wherever possible.
- Be sure to define the department-specified functions exactly as specified in terms of names and parameters, and not using alternative wording.
- **Be precise with spelling (identifiers – variable names, correct capitalization, etc.), and with any required output strings (in English), as specified in the course.**
- You must be precise in formatting the output **exactly as instructed in the question**, including wording, punctuation, quotation marks, parentheses, whitespace, etc.

Question 1 (The only one)

You are asked to develop an information system for a real estate development company. The company is constructing a building that includes several types of apartments:

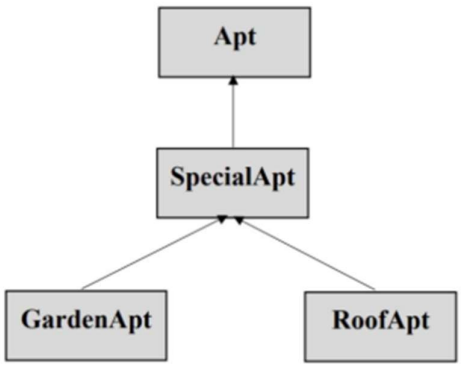
- **Apt (Apartment)** – characterized by the number of floors (floor) and the apartment's area in square meters (area).
- **SpecialApt (Special Apartment)** – this is a unique and upgraded apartment. Additionally, it is defined by whether the apartment has a view (has_view).
- **GardenApt (Garden Apartment)** – this is a unique and upgraded apartment that also includes the garden's area in square meters (garden_area). A garden apartment is always located on the ground floor (floor 0) and never has a view.
- **RoofApt (Rooftop Apartment)** – this is a unique and upgraded apartment that is additionally defined by whether the apartment has a pool (has_pool). A rooftop apartment always has a view.

Note:

Attributes defined in each class **must not be declared public** but rather **protected**, using the underscore convention, e.g.:

self. _attribute_name

The inheritance tree of the classes is as follows:



A. For each of the four classes described above:

1. Define the class (including inheritance, if applicable). You must define at least one constructor.
2. Define a method named matches that checks compatibility based on the required specification. Add **constant values** (constants), as needed, in **uppercase letters**, as required.

3.

Make sure that the values received by the constructors meet the required conditions.

The constructor headers of the classes must match exactly the following format:

 # For the Apt class:

```
def __init__(self, floor, area)
```

For the SpecialApt class:

```
def __init__(self, floor, area, has_view)
```

For the GardenApt class:

```
def __init__(self, area, garden_area)
```

For the RoofApt class:

```
def __init__(self, floor, area, has_pool)
```

4.

Implement a `get` function for each attribute. The function name must follow the pattern:

```
get_attribute_name()
```

For example, in class Apt, the getter function for the attribute `floor` will be:

```
def get_floor(self)
```

5.

Implement the `__eq__` method to compare whether two apartment instances are equal. The method receives another apartment object as a parameter.
The function should return True if all relevant attributes of both apartments are equal. Otherwise, return `False`.

If the object received is not of the same type (class), return `NotImplemented`.

6.

Implement the `__str__` method, which returns a string representing the object in the following format:

```
attribute_name: attribute_value, attribute_name:
attribute_value, ...
```

Use a colon (`:`) after each attribute name and comma (`,`) between pairs.
If the class inherits from a superclass, the subclass's `__str__` should call the superclass's `__str__` accordingly.

Apartment prices are determined according to the following rules:

- The price per square meter is ₪20,000.
  Each additional floor adds ₪5,000 to the apartment's base price.
  (There is no additional floor charge for apartments on the ground floor.)
- A view adds ₪600× the floor number to the apartment price.
  For example, an apartment with a view on the 10th floor would add ₪6,000.
- A rooftop apartment adds ₪40,000 to the apartment price,
  and a pool adds ₪30,000 to the apartment price.

B.

Add the get_price function in **all classes** of the hierarchy in order to perform the required pricing (only in the classes where it is relevant), based on the principles of independent program design.

Note: Do **NOT** add any attributes to the classes.

Add a project file named `mmn15.py` containing the following functions:

C. Write a function named `average_price` that receives a list of apartments apts.
The function should return the average price of the apartments.

D. Write a function named `how_many_rooftop` that receives a list of apartments apts.
The function should return the number of rooftop apartments with a pool in the list.
Apartments without pools are not to be counted.

E. Write a function named `how_many_apt_type` that receives a list of apartments apts.
The function should return a dictionary (dict) where the keys are apartment types (classes Apt, SpecialApt, RoofApt, GardenApt) and the values are the counts of apartments of each type in the list.
Each apartment should be counted under only one key.
If the list is empty, the dictionary should be empty.

F. Write a function named `top_price` that receives a list of apartments apts.
The function should return the apartment with the **highest price** in the list.
If there are multiple apartments with the same highest price, return the first one that appears in the list.
If the list is empty, return None.

G. Write a function named `only_valid_apts` that receives a list of apartments apts.
The function should return a list of apartments **with a view or with a pool**, whose price

does **not exceed ₪1,000,000**.

If no apartment meets the condition, return `None`.

Note:

In all functions from sections G–Z, you may assume that the list `apts` contains objects of the appropriate classes,

and all items in the list are apartments or subclasses of the `Apt` class.

(There is no need to verify that objects belong to the inheritance hierarchy.)

Please Note — for all assignment questions:

- You may **no**t add any attribute that is not mentioned or required by the class.
- You may add private methods, but **not public ones**.
- Do **not** use magic numbers (unnamed numeric values) in your code. You must define constants for all fixed numbers and only use them by name, except in trivial cases (e.g., the number 0).
- Make sure to generally follow the principles of independent program design. Try to write each method in each class so that it performs a single, specific task**. If you find yourself copying code between classes, it's a sign that you should be using inheritance.**
- You must write a **docstring for each method you implement**, **and for each class**. The docstring should follow **Python's formatting conventions**. It must include a concise description of what the method/class does, without excessive verbosity.

Please also note that we have posted test files for the four classes on the course website.

These files allow you to test your implementation against ours. ***They do not replace your own testing!*** *You must test your code thoroughly. The test files are only partial and are meant to help you compare your implementation with ours.* ***If your implementation passes the test file but does not meet the assignment requirements, it will still receive a grade of zero.***

Submission

1. Submission of the assignment is to be done electronically only, via the official assignment submission system.
2. Create separate Python files (.py) for each class (Apt, SpecialApt, GardenApt, RoofApt) as follows:
    - The Apt class should be written in a file named `apt.py`.
    - The SpecialApt class should be written in `special_apt.py`.
    - The GardenApt class should be written in `garden_apt.py`.
    - The RoofApt class should be written in `roof_apt.py`.

    Additionally, the functions from sections (C) to (G) should be written in a file named **`mmn15.py`**.

3. Submit a **single ZIP archive** containing all the above files (**not a .rar file**). Submit only this archive.

4. You may submit the assignment multiple times before the deadline, but only the **last** submission will be graded.
    If the final version is submitted after the deadline, the **previous** version (submitted on time) will be graded.
    If all versions were submitted late, the **last** one will be considered (and marked as late).

Be sure to do this in advance — requests sent close to the deadline will not be processed in time, and late submissions due to this will be considered your responsibility.

Good luck! 😊