Foundation

Advanced Python

Weeks 2 & 3

Module 2 Exercises

Version	Date	Author	Comments
1.0	08/ 06 / 20	Kevin Wright	First draft
2.0	05/01/2022	Nikola Ignjatovic	Second Draft

Module 2A: Classes and Objects

1. a) Write an empty Python class named Trainee and display its type and namespace. Additionally, display only the keys of the Trainee's namespace.

   b) Create an object of class Trainee, and display its type, namespace and its class name.

   c) Write an empty Python class named Trainer and create an object of this class. Check whether this object and the object of class Trainee created in question 1b are instances of the said classes or not. Also, check whether the classes Trainee and Trainer are subclasses of the built-in object class or not.

   d) Add two class attributes: academy and trainee_name to the class Trainee. Modify the attribute values of the class Trainee and print the original and modified values of the attributes.

   e) Add a class method to print the values of the two class attributes: academy and trainee_name. Use this method to print the values of the two class attributes. Check that the two attributes and their values are included to the Trainee's class namespace.

   f) Add an instance method to print the values of the two class attributes: academy and trainee_name from an object. Create an object of class Trainee and use the method to print the values of the two class attributes. Using the object's namespace check that the object has no instance attributes. Access each class attribute from the object to print its value.

   g) Add another instance method to the Trainee class to print the values of its instance attributes (from an object, as instance attributes are not accessible from the class).
   Create another object of class Trainee.
   Print its class attributes:
   - using the class method created in question 1e
   - using the instance method created in question 1f

- using the class namespace
- directly from the object.

Assign new values to the attributes academy and trainee_name directly from the object.

Print class attributes and their values
- using the class method created in question 1e
- using the instance method created in question 1f
- using the class namespace.

Then print the values of the object's instance attributes
- using the instance method created in this question
- using the object's namespace
- directly from the object.

Explain what has happened.

Tip: use the __dict__ attribute to display the namespaces.

2. Rewrite the class Trainee so that it has 2 class attributes: company - initialised as 'FDM Group', and count – initialised to 0, to count the number of trainees. Also add two instance attributes: academy and trainee_name. Keep the three methods created in your solution for question 1 and add two more methods: the constructor to set up the values for the two instance attributes when creating a new object and to increase the trainee counter, as well as the destructor to decrease the counter every time a trainee is removed. Once the class has been created, write the client code to do the following:
   - print the values of class attributes as soon as the class has been created
   - print the Trainee class namespace (using the __dict__ attribute)
   - create three trainees and for each one print its class and instance attributes, and its namespace (using the __dict__ attribute)
   - remove the first trainee
   - print the values of Trainee class attributes using the class (not an object)

3. Create the two classes as depicted by the UML diagram below

a. Create a constructor for each class. Ensure that it only initialises the attributes shown in the UML.

b. Except for the constructor, all methods that don't return or change values should simply print out a message containing the name of the method and the value of its argument(s).

   For example, the read_data method returns a String. The String should contain a simple message like: "read_data: reading data from " + file.

c. Ensure that attributes shown in upper case are read-only instance attributes.

d. Use the Python property decorators to get, set and delete all attributes

HardDrive
- MODEL : string -CAPACITY: float - used_space: float
+ __init__(model, capacity) + read_data(file:string):string + write_data(date:string, file:string) + set_used_space(used_space: float)

Memory
- MODEL : string - CAPACITY : float - used_space:float - SPEED : float
+ __init__(model, capacity, speed) + store_data(data: string) + set_used_space(used_space: float)

Client code to test the HardDrive class:

- create a HardDrive object
- display the namespace of both the class and the object
- display the model and capacity directly from the object (ensure the getter gets invoked in both cases)
- set the value of model for the object to 'WD' and the value of capacity to 3072GB (ensure the setter gets invoked in both cases displaying the appropriate message)
- display the values of model and capacity directly from the object (ensure the getter gets invoked in both cases and that the values of both attributes have not been changed)

- use the set_used_space method to change the used space to 0GB and check that it was changed by displaying the used_space attribute directly from the object
- execute the read_data() and write_data() instance methods to check that the appropriate messages are displayed
- set the docstring for the model to 'name of the hard disk model' and for the capacity to 'hard disk capacity', and then display them both
- delete the model, capacity and used space attributes and check their removal by displaying the object's namespace after each attribute is deleted

Client code to test the Memory class:

- create a Memory object
- display the namespace of both the class and the object
- display the model, capacity and speed directly from the object (ensure the getter gets invoked in both cases)
- set the value of model for the object to 'Corsair' and the value of capacity and speed to 8GB and 1333Hz respectively (ensure the setter gets invoked in both cases displaying the appropriate message)
- display the values of model, capacity and speed directly from the object (ensure the getter gets invoked in both cases and that the values of none of the attributes have been changed)
- use the set_used_space method to change the used space to 0.5GB and check that it was changed by displaying the used_space attribute directly from the object
- execute the store_data()instance method to check that the appropriate message is displayed
- set the docstring for the model to 'name of the hard disk model', for the capacity to 'hard disk capacity' and for the speed to "memory speed", and then display them all
- delete the model, capacity, speed and used space attributes and check their removal by displaying the object's namespace after each attribute is deleted

4. Write a Python class which has two methods get_string() and print_string(). get_string() accepts a string from the user and print_string() prints the string in 'proper' case (the first character of each word of the string is upper case and the rest of the characters are lower case).
   Example:
   'tHiS iS My UNTIDY string' -> This Is My Untidy String

5. Write a Python class named Circle consisting of an instance attribute radius and two methods which will compute the area and the perimeter of a circle.
   Note: import and use the constant pi from the math module.

6. a) Write a Python class for the 'Tesla' car manufacturer. It needs to include the following:
   - 2 class attributes: MAKE (a constant set to 'Tesla') and condition, set to value 'new' (by default a car is new)
   - 4 instance variables: model, fuel, max_speed, colour
   - 4 methods: the constructor, drive_car() - which sets the object's condition to 'used', change_colour() - which changes the colour of the car and display() - which prints all car details, using this format:
     "This is a silver Tesla 3 electric with max speed of 140 mph. Condition: new."

   Once the class has been created, write the client code to do the following:
   - create a car object as make: 3, fuel: electric, max_speed: 140, colour: "silver"
   - display the Car class namespace and the namespace of the Car's object (using the __dict__ attribute)
   - execute the object's display() method
   - execute the object's drive_car() method
   - print the value of class attribute condition from the Car's object
   - print the value of class attribute condition from the Car class
   - Explain why the attribute condition of the car object is 'used' but the attribute condition of the Car class remains 'new' (you may want to display the namespace of the Car's object to help you answer the question)
   - create another car object as make: X, fuel: petrol, max_speed: 200, colour: "red"

- execute the object's display() method
- print the value of class attribute condition from the Car's object
- print the value of class attribute condition from the Car class

b) improve the class Car by preventing users to change the class attribute MAKE through an object of class Car in the client code

7. Write a Python class to convert a positive integer to the equivalent roman numeral. The roman numerals and their corresponding decimal values are given below:
   M:1000, CM:900, D:500, CD: 400, C:100,
   XC:90, L:50, XL:40, X:10, IX:9, V:5, IV:4, I:1
   Examples:
   8 -> VIII
   54 -> LIV
   100 -> C
   153 -> CLIII
   2022 -> MMXXII
   4000 -> MMMM

8. Write a Python class to convert a roman numeral to the equivalent integer value.
   Examples:
   VIII -> 8
   XIX -> 19
   XXXIX -> 39
   XLIV -> 44
   LIV -> 54
   C -> 100
   CLIII -> 153
   CM -> 900
   MMXXII -> 2022
   MMMM -> 4000

Module 2B: Inheritance

1. a) Define a simple empty class called Vehicle.
   b) Add an instance method called move() to the Vehicle class. Create an instance (object) of this class and call its move() method through its object.
   c) Convert this class into an abstract class in order to use it as common interface (blueprint) for classes that represent different types of vehicles. What happens when trying to create an instance of this class? Why?
   d) Create an empty concrete child class Aircraft that is derived from the class Vehicle. What happens when trying to create an instance of the class Aircraft? Why?
   e) Include the method move() to the concrete child class Aircraft and that cpntains just one print statement displaying the message 'flying...'.
   Create an instance (object) of the class Aircraft and call its move() method through its object.
   f) Add another abstract method to class Vehicle, called accelerate(). Create an instance of the class Vehicle.
   What happens? Why?
   g) Implement the accelerate() method in Aircraft class by displaying the message: 'reached 150 mph in 30 seconds'
   Create an instance (object) of the class Aircraft and call its accelerate() method through its object.
   h) Add a print statement to the accelerate() method of the Vehicle abstract class, displaying the message 'engine started...'.
   Call the accelerate method of the abstract class from the Aircraft child class before printing the message 'reached 150 mph in 30 seconds'. See what happens and explain why.
   i) Create another concrete child class Car that is derived from the class Vehicle and implements both move() and accelerate() methods. The move() method displays the message 'driving...' and the accelerate method displays the message 'reached 60 mph

in 10 seconds'
Create an object of classes Car and Aircraft, and call both their move and accelerate methods. See what happens and explain the benefit of having the message 'engine started...' displayed in the abstract method accelerate().

j) Introduce an instance attribute called speed to the abstract class Vehicle and include property methods to read and modify its value (getter and setter). Declare these two property methods as abstract in order to enforce their implementation in every subclass of Vehicle. Create an instance of the Aircraft or Car class after adding the abstract property methods to their parent abstract class. What happens and why?

k) Implement the property methods to read and modify the value of instance attribute speed.
Create an instance of the Aircraft and Car class after adding the abstract property methods to their parent abstract class.
Modify the speed of both the aircraft and car objects.

l) Change the constructor to set the initial speed at 0 mph (once created, any vehicle is not moving), ensuring that every vehicle must have its initial speed set to 0 once created.
Set the speed in the accelerate() method of both Aircraft and Car concrete classes to the speed displayed in the message.
Test that it all works correctly by creating an object of both Aircraft and Car and doing the following for each object:
- prints its speed
- call its acccelerate method
- print its speed
- change its speed to some value above the one stated in the accelerate method
- print its speed
- call its move method

2. Add the class Hardware to the classes created in Module 2A question 3 and make it parent of both HardDrive and Memory. Move the common

attributes and methods from the two existing classes to the Hardware class and amend the code in the two setters to make them work correctly.
Then write the client code to do the following:

- All tasks listed under client code to test the HardDrive class in Module 2A question 3. In addition, check that:
    - the private instance attributes model and capacity are name-mangled to _Hardware__model and _Hardware__capacity, instead of _HardDrive__model and _HardDrive__capacity (by printing the object's namespace as soon as the HardDrive object is created)
    - model & capacity properties are now listed in the Hardware namespace instead of HardDrive namespace (by printing both HardDrive and Hardware namespaces as soon as the HardDrive object is created)
- All tasks listed under client code to test the Memory class in Module 2A question 3. In addition, check that:
    - the private instance attributes model and capacity are name-mangled to _Hardware__model and _Hardware__capacity, instead of _Memory__model and _ Memory __capacity but that speed is name-mangled to _Memory__speed (by printing the object's namespace as soon as the Memory object is created)
    - model & capacity properties are now listed in the Hardware namespace instead of Memory namespace, but speed property is still listed in the Memory namespace (by printing both Memory and Hardware namespaces as soon as the Memory object is created)
- create a Hardware object
- display the namespace of both the class and the object
- display the model and capacity directly from the object (ensure the getter gets invoked in both cases)
- set the value of model for the object to 'Dell' and the value of capacity to 4096GB (ensure the setter gets invoked in both cases displaying the appropriate message)

- display the values of model and capacity directly from the object (ensure the getter gets invoked in both cases and that the values of both attributes have not been changed)
- use the set_used_space method to change the used space to 500GB and check that it was changed by displaying the used_space attribute directly from the object
- set the docstring for the model to 'name of the hard disk model' and for the capacity to 'hard disk capacity', and then display them both
- delete the model, capacity and used space attributes and check their removal by displaying the object's namespace after each attribute is deleted

3. The class Hardware created in Module 2B question 2 is generic (relates to any hardware device) and should therefore not be instantiable. Modify the class Hardware to make it abstract and ensure that it does not affect how its subclasses HardDrive and Memory work. Check that an attempt to instantiate the Hardware class throws an error.

4. Improve the classes Hardware, HardDrive and Memory created in Module 2B Question 3 by implementing the following:
    - include the hard drive counter: count_hard_drive, storing the number of created hard drives
    - include the memory counter: count_memory, storing the number of created memory components
    - include the hardware counter: count_hardware, storing the number of created hardware components, where the hardware counter will always show the total of hard drive and memory counters
    - increase the counters count_hard_drive and count_memory whenever a hard disk or a memory has been produced respectively
    - include the increment_counter() method to the Hardware class to increase the class attribute count_hardware whenever a hard disk or a memory has been produced, and display a message: "New hard drive created" or "New memory created", depending on whether a hard drive or memory has been created

- include the print_count() method to print the number of produced Hardware, HardDrive and Memory components.