Assignment-1 Solutions

M.Vivekananda , 240628

Question No.1

Sol: When we define a function with a mutable default argument like a list , dict or a set like these are mutable, the default value is created only once when the function is defined .

When we define something like this

```
def add_item(item ,box=[])
```

The list with the name box=[ ] is created once in the memory .And every time we call add_item( ) without providing a box argument , the function reuses that same single object. Now when we append an item to this list we are modifying the persistent list object.

And it persists because the default value is evaluated and stored once as a single object in the function’s definition. Since we know that lists are mutable the changes made to that single object during one function call are seen in the next call ..

Why am i considering this as the bug because when we generally call a function without providing a specidfic arg for box , we expect the function to act as if it is fresh for the new arg and we know that a function which is said to be well-designed if it is isolated and predictable.

Fix

we should use an immutable value like **None.**

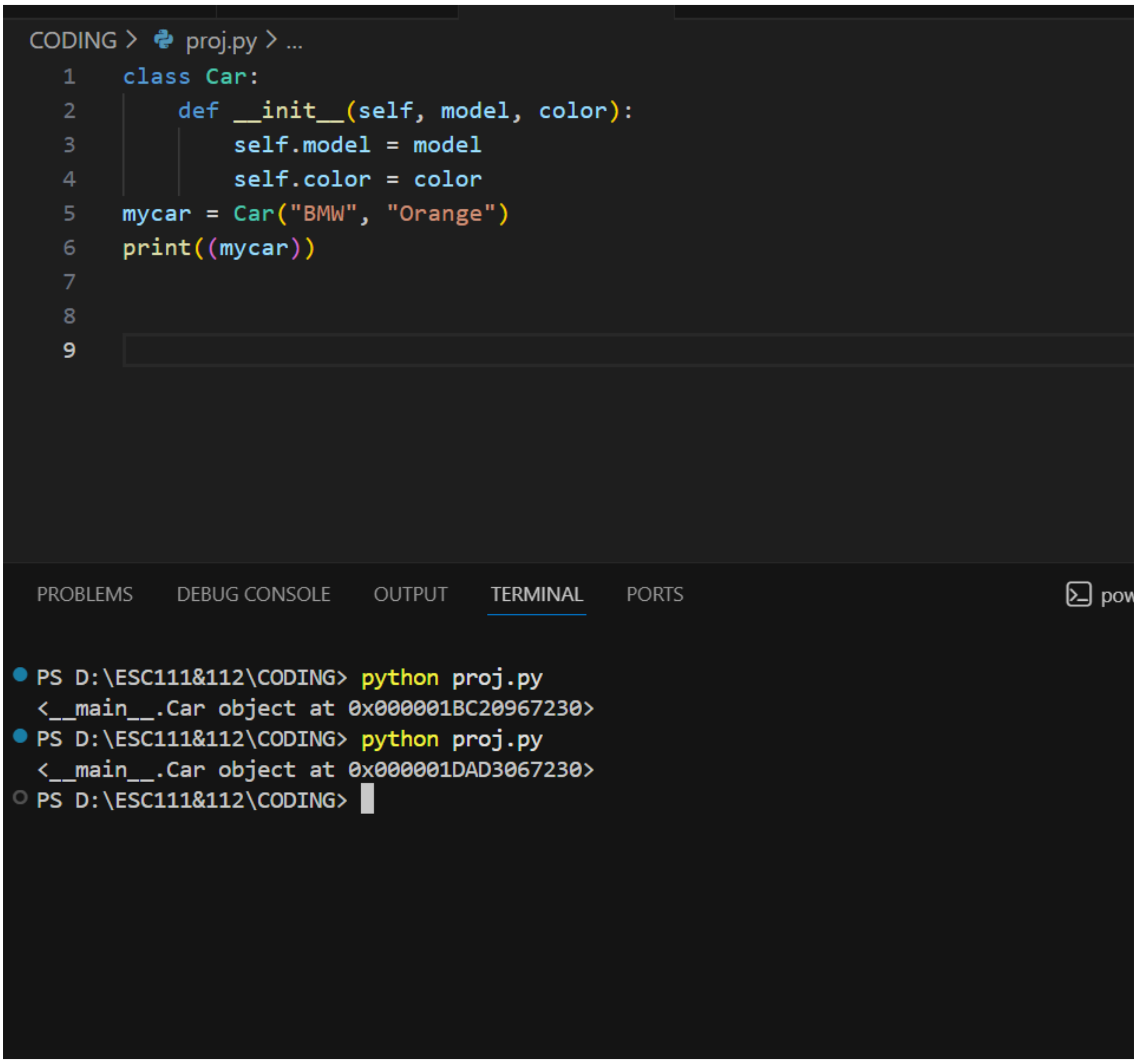
```
def add_item_new(item,box=None)
```

Question No .2

Ans :

S.No	Method	Meaning and Usage
1.	_ _str _ _	It is used for creating a human readable string representation of an object , we use this when we print or log somthing. Python falls back to _ _repr _ _ if _ _str _ _ is not defined.
2.	_ _repr_ _	since the str thing is used for the readability , for to be an unambigous official representation ideally allowing the object to be reconstructed .Like it is used in the interpeter. if _ _repr_ _ is missing python falls back to the generic memory address where the object resides in the computer's RAM. for example

```
class Car:
    def __init__ (self,model,color):
                self.model=model
                self.color=color
mycar=Car("BMW",Orange)
print(repr(mycar))
```



```python
class Car:
    def __init__(self, model, color):
        self.model = model
        self.color = color
mycar = Car("BMW", "Orange")
print((mycar))
```

```
PS D:\ESC111&112\CODING> python proj.py
<__main__.Car object at 0x000001BC20967230>
PS D:\ESC111&112\CODING> python proj.py
<__main__.Car object at 0x000001DAD3067230>
PS D:\ESC111&112\CODING>
```

Question No.3

Instance variable is defined inside **__init__** using self.var=value. Here the varibale is stored aeperatelu in the memory of each individual object.And each obj or instance gets its own copy so changing it on one instance doesn’t affect others.

whereas Class variable is defined directly under the class name .Here the variable is stored once in the memory of the class itself and all the instances share the access to this single memory location.

Changing a calss variable via ClassName.var=new_value the effect is the single shared value stored in the class itself is updated and all existing instances will immediately see and use the new_value when they access the variable.

If we are changing a class variable via instance.var=new_value this does not change teh class variable . It creates a new instance variable with the same name var on that specific instance. So now the instance has its own copy , which overrides the class variable for that specific instance . All other instances still use the original class variable value.