

CS: Objects in Python

Encapsulation

Lab - Encapsulation

Encapsulation Lab 1

Lab 1

This lab will focus on building a journal to rate different coffees. The journal will use encapsulation. Users enter the name of the roaster, the country of origin, the region and how many stars (*) they rate the coffee. Entries are saved to a CSV file. Users can display coffees already in the journal, or they can add new coffees to the journal. There are two main parts to this project, the `CoffeeJournal` class and the command line interface.

The `CoffeeJournal` Class

Since this project will be reading and writing to a CSV file, we need to first import the CSV module. After that we are going to create the constructor for the `CoffeeJournal` class. Notice, all of the attributes use the single leading underscore convention to indicate being private.

```

import csv

# *****

# code for the CoffeeJournal class

# *****

class CoffeeJournal:
    def __init__(self, file):
        self._file = file
        self._roaster = ""
        self._country = ""
        self._region = ""
        self._stars = ""
        self._new_coffee = []
        self._old_coffee = self.load_coffee()

```

The `_file` attribute represents the CSV file that will be used. The next four attributes are information about the coffee: the roaster, the country of origin, the region, and how many stars the coffee earned. The `_new_coffee` is a list of the new coffees entered. These will be saved to the CSV file when quitting the script. The `_old_coffee` attribute is a 2D list of all of the coffees saved in the CSV file. The `load_coffee` returns a 2D list of all of the information in the CSV file. At the very least, there will be the headers Roaster, Country, Region, and Stars.

```

def load_coffee(self):
    coffee = []
    with open(self._file) as f:
        reader = csv.reader(f, delimiter=',')
        for row in reader:
            coffee.append(row)
    return coffee

```

This method opens the CSV file stored in `self._file` in read mode. Each line of the file is read as a list. This list is appended to `coffee`, which makes it a 2D list. The `coffee` attribute is returned to the `_old_coffee` attribute.

Testing Your Code

Let's make sure your class is working as expected. Enter the following code to create an instance of the `CoffeeJournal` class and print out the `_old_coffee` attribute after reading from the CSV file.

```
# *****  
  
# code for testing your script  
  
# *****  
  
test_object = CoffeeJournal("code/encapsulation/test_journal1.csv")  
test_object.load_coffee()  
print(test_object._old_coffee)
```

You should see the following output:

```
[['Roaster', 'Country', 'Region', 'Stars']]
```

▼ Self-Check

This is what your code should look like.

```

import csv

# *****

# code for the coffee object
# *****

class CoffeeJournal:
    def __init__(self, file):
        self._file = file
        self._roaster = ""
        self._country = ""
        self._region = ""
        self._stars = ""
        self._old_coffee = self.load_coffee()
        self._new_coffee = []

    def load_coffee(self):
        coffee = []
        with open(self._file) as f:
            reader = csv.reader(f, delimiter=',')
            for row in reader:
                coffee.append(row)
        return coffee

# *****

# code for testing your script
# *****

test_object = CoffeeJournal("code/encapsulation/test_journal1.csv")
test_object.load_coffee()
print(test_object._old_coffee)

```

Encapsulation Lab 2

Lab 2

The next step is to add the getters and setters for the different categories coffee categories: roaster, country, region, and stars. Use the `@property` decorator when creating the getters and setters. The name of the getter and setter is the attribute minus the leading underscore.

```

@property
def roaster(self):
    return self._roaster

@roaster.setter
def roaster(self, new_roaster):
    self._roaster = new_roaster

@property
def country(self):
    return self._country

@country.setter
def country(self, new_country):
    self._country = new_country

@property
def region(self):
    return self._region

@region.setter
def region(self, new_region):
    self._region = new_region

@property
def stars(self):
    return self._stars

@stars.setter
def stars(self, new_stars):
    self._stars = new_stars

```

Testing Your Code

Let's make sure your class is working as expected. Enter the following code to create an instance of the `CoffeeJournal` class, set values for some of the attributes, and then print these attributes.

```
# *****
# code for testing your script
# *****

test_object = CoffeeJournal("code/encapsulation/test_journal2.csv")
test_object.roaster = "Peace River"
test_object.country = "Rawanda"
test_object.region = "Remera"
test_object.stars = "***"
print(test_object.roaster)
print(test_object.country)
print(test_object.region)
print(test_object.stars)
```

You should see the following output:

```
Peace River
Rawanda
Remera
***
```

▼ Self-Check

This is what your code should look like.

```
import csv

# *****
# code for the coffee object
# *****

class CoffeeJournal:
    def __init__(self, file):
        self._file = file
        self._roaster = ""
        self._country = ""
        self._region = ""
        self._stars = ""
        self._old_coffee = self.load_coffee()
```

```
        self._new_coffee = []

    def load_coffee(self):
        coffee = []
        with open(self._file) as f:
            reader = csv.reader(f, delimiter=',')
            for row in reader:
                coffee.append(row)
        return coffee

    @property
    def roaster(self):
        return self._roaster

    @roaster.setter
    def roaster(self, new_roaster):
        self._roaster = new_roaster

    @property
    def country(self):
        return self._country

    @country.setter
    def country(self, new_country):
        self._country = new_country

    @property
    def region(self):
        return self._region

    @region.setter
    def region(self, new_region):
        self._region = new_region

    @property
    def stars(self):
        return self._stars

    @stars.setter
    def stars(self, new_stars):
        self._stars = new_stars
```



```
# *****  
  
# code for testing your script  
  
# *****  
  
test_object = CoffeeJournal("code/encapsulation/test_journal2.csv")  
test_object.roaster = "Peace River"  
test_object.country = "Rawanda"  
test_object.region = "Remera"  
test_object.stars = "****"  
print(test_object.roaster)  
print(test_object.country)  
print(test_object.region)  
print(test_object.stars)
```

Encapsulation Lab 3

Lab 3

We will finish up the `CoffeeJournal` class with some methods that add a coffee to the journal, prints the journal, and saves the updated journal to the CSV file. Printing the journal will print any information stored in the CSV file as well as any new information entered by the user. Only when the user quits the program will the new information be saved to the CSV file.

Methods

The `add_coffee` method stores the `_roaster`, `_country`, `_region`, and `_stars` attributes in a list and then appends it to the `_new_coffee` list.

```
def add_coffee(self):  
    self._new_coffee.append([self._roaster, self._country, self._region,
```

The `save` method opens the CSV file in append mode. The 2D list stored in `_new_coffee` is added to the end of the CSV file. **Note**, eventually the `save` method will be linked to stopping the script. So saving should be the last thing you do before exiting the program.

```
def save(self):  
    with open(self._file, 'a') as f:  
        writer = csv.writer(f)  
        writer.writerows(self._new_coffee)
```

The `show_coffee` method takes into account three cases. One, there is no information about a coffee in either the `_old_coffee` or in the `_new_coffee` attributes. Print a message to the user to add a coffee. Two, there is information about coffee in only the `_old_coffee` attribute. Print only the contents of `_old_coffee`. Three, in all other cases print the contents of `_old_coffee` followed by the contents of `_new_coffee`.

```

def show_coffee(self):
    print()
    # if there is no information on any coffee, tell the user to add a

    if len(self._old_coffee) < 2 and len(self._new_coffee) == 0:
        print("Enter a coffee first")
        # if there is information in the CSV but not new coffee print the

    elif len(self._old_coffee) > 2 and len(self._new_coffee) == 0:
        for row in self._old_coffee:
            print(f"{row[0]:13} {row[1]:13} {row[2]:13} {row[3]:13}")
        # print both the old coffee and the new coffee
    else:
        for row in self._old_coffee:
            print(f"{row[0]:13} {row[1]:13} {row[2]:13} {row[3]:13}")
        for row in self._new_coffee:
            print(f"{row[0]:13} {row[1]:13} {row[2]:13} {row[3]:13}")
    print()

```

Testing Your Code

Test 1

Let's make sure your class is working as expected. The first test is going to try and print the coffee journal with no information in it.

```

# *****
# code for testing your script
# *****

test_object = CoffeeJournal("code/encapsulation/test_journal3.csv")
test_object.show_coffee()

```

You should see the following output:

```
Enter a coffee first
```

Test 2

Change your testing code to look like the code below. The second test is going to add a coffee to the journal and save it. Then it will open the CSV file and print its contents.

```
# *****
# code for testing your script
# *****

test_object = CoffeeJournal("code/encapsulation/test_journal3.csv")
test_object.roaster = "Peace River"
test_object.country = "Rawanda"
test_object.region = "Remera"
test_object.stars = "***"
test_object.add_coffee()
test_object.save()
test_object._old_coffee = test_object.load_coffee()
test_object._roaster = ""
test_object._country = ""
test_object._region = ""
test_object._stars = ""
test_object._new_coffee = []
test_object.show_coffee()
```

You should see the following output:

| Roaster | Country | Region | Stars |
|-------------|---------|--------|-------|
| Peace River | Rawanda | Remera | *** |

Test 3

Change your testing code to look like the code below. The third test is going to add a coffee to the journal and print the coffee information already stored in the CSV file plus the newly entered coffee.

```
# *****

# code for testing your script

# *****

test_object = CoffeeJournal("code/encapsulation/test_journal3.csv")
test_object.roaster = "Peace River"
test_object.country = "Ethiopia"
test_object.region = "Sidoma"
test_object.stars = "****"
test_object.add_coffee()
test_object.show_coffee()
```

You should see the following output:

| Roaster | Country | Region | Stars |
|-------------|----------|--------|-------|
| Peace River | Rawanda | Remera | *** |
| Peace River | Ethiopia | Sidoma | **** |

▼ Self-Check

This is what your code should look like.

```
import csv

# *****

# code for the coffee object

# *****

class CoffeeJournal:
    def __init__(self, file):
        self._file = file
        self._roaster = ""
        self._country = ""
        self._region = ""
        self._stars = ""
        self._old_coffee = self.load_coffee()
        self._new_coffee = []

    def load_coffee(self):
```

```

        coffee = []
        with open(self._file) as f:
            reader = csv.reader(f, delimiter=',')
            for row in reader:
                coffee.append(row)
        return coffee

    @property
    def roaster(self):
        return self._roaster

    @roaster.setter
    def roaster(self, new_roaster):
        self._roaster = new_roaster

    @property
    def country(self):
        return self._country

    @country.setter
    def country(self, new_country):
        self._country = new_country

    @property
    def region(self):
        return self._region

    @region.setter
    def region(self, new_region):
        self._region = new_region

    @property
    def stars(self):
        return self._stars

    @stars.setter
    def stars(self, new_stars):
        self._stars = new_stars

    def save(self):
        with open(self._file, 'a') as f:
            writer = csv.writer(f)
            writer.writerow([self._name, self._roaster, self._country, self._region, self._stars])

```

```

        writer = csv.writer(f)
        writer.writerow(self._new_coffee)

def show_coffee(self):
    print()
    if len(self._old_coffee) < 2 and len(self._new_coffee) == 0:
        print("Enter a coffee first")
    elif len(self._old_coffee) < 2 and len(self._new_coffee) == 0:
        for row in self._old_coffee:
            print(f"{row[0]:15} {row[1]:15} {row[2]:15} {row[3]:15}")
        else:
            for row in self._old_coffee:
                print(f"{row[0]:15} {row[1]:15} {row[2]:15} {row[3]:15}")
            for row in self._new_coffee:
                print(f"{row[0]:15} {row[1]:15} {row[2]:15} {row[3]:15}")
    print()

def add_coffee(self):
    self._new_coffee.append([self._roaster, self._country, self._region, self._stars])

# *****
# code for testing your script
# *****

test_object = CoffeeJournal("code/encapsulation/test_journal3.csv")

# print journal with no coffee information
test_object.show_coffee()

# save information to the CSV file and then print
test_object.roaster = "Peace River"
test_object.country = "Rawanda"
test_object.region = "Remera"
test_object.stars = "****"
test_object.add_coffee()
test_object.save()
test_object = CoffeeJournal("code/encapsulation/test_journal3.csv")
test_object.show_coffee()

# print from both the CSV and from `_new_coffee`
test_object.roaster = "Peace River"

```

```
test_object.roaster = "Peace River"  
test_object.country = "Ethiopia"  
test_object.region = "Sidoma"  
test_object.stars = "****"  
test_object.add_coffee()  
test_object.show_coffee()
```


Encapsulation Lab 4

Lab 4

The CoffeeJournal is now complete. This lab focuses on how using an object from this class and building a command line interface around it. The following code should be added after the CoffeeJournal class.

Command Line Interface

We are going to start by creating some helper functions. We want a menu-driven interface. The user will enter a loop and be presented with a list of choices, which is the `main_menu` function. The `perform_action` function takes the user choice and performs the appropriate action. **Note**, the code examples uses the parameter `coffee` which is an instance of the `CoffeeJournal` class.

```
# *****  
  
# code for the command line application  
  
# *****  
  
def main_menu():  
    print("Coffess of the World")  
    print("\t1. Show Coffee")  
    print("\t2. Add Coffee")  
    print("\t3. Save and Quit")  
    choice = int(input("Enter the number of your selection: "))  
    return choice  
  
def perform_action(choice, coffee):  
    if choice == 1:  
        coffee.show_coffee()  
    elif choice == 2:  
        enter_coffee(coffee)  
    elif choice == 3:  
        quit(coffee)
```

The menu provides three options. The first one can be handled by the `coffee` object. The other two options require the `coffee` object but need some additional assistance. The `enter_coffee` function prompts the user to enter information about the new coffee, while the `quit` function saves the new information and exits the loop. **Note**, `run_loop` is a boolean variable that controls the loop.

```
def enter_coffee(coffee):
    print()
    coffee.roaster = input("Enter the name of the roaster: ")
    coffee.country = input("Enter the country of origin: ")
    coffee.region = input("Enter the region: ")
    coffee.stars = int(input("Enter the number of stars '*' (1-4): ")) *

    print()
    coffee.add_coffee()

def quit(coffee):
    global run_loop
    coffee.save()
    run_loop = False
```

Testing Your Code

To test our code, we are going to set up a loop that controls the command line application. The loop should run as long as `run_loop` is `True`. Present the user with the menu of options. Finally perform the desired action.

```
# *****

# code for testing your script

# *****

run_loop = True
file = "code/encapsulation/coffee_journal.csv"
my_coffee = CoffeeJournal(file)

while run_loop:
    choice = main_menu()
    perform_action(choice, my_coffee)
```

Use the following information to enter two new coffees.

| Roaster | Country | Origin | Stars |
|-----------|-----------|----------|-------|
| : | — | : | — |
| : | — | : | — |
| Ritual | Guatemala | Antigua | *** |
| Oak Cliff | Peru | Oxapampa | ** |

▼ Using the Terminal

When using the input command, you must type something in the terminal. That is why there is a terminal below the IDE.

You should see the following output:

| Roaster | Country | Region | Stars |
|-------------|-----------|----------|-------|
| Peace River | Rawanda | Remera | *** |
| Ritual | Guatemala | Antigua | *** |
| Oak Cliff | Peru | Oxapampa | ** |

▼ Self-Check

This is what your code should look like.

```
import csv

# *****

# code for the coffee object

# *****
```

```

class CoffeeJournal:
    def __init__(self, file):
        self._file = file
        self._roaster = ""
        self._country = ""
        self._region = ""
        self._stars = ""
        self._old_coffee = self.load_coffee()
        self._new_coffee = []

    def load_coffee(self):
        coffee = []
        with open(self._file) as f:
            reader = csv.reader(f, delimiter=',')
            for row in reader:
                coffee.append(row)
        return coffee

    @property
    def roaster(self):
        return self._roaster

    @roaster.setter
    def roaster(self, new_roaster):
        self._roaster = new_roaster

    @property
    def country(self):
        return self._country

    @country.setter
    def country(self, new_country):
        self._country = new_country

    @property
    def region(self):
        return self._region

    @region.setter
    def region(self, new_region):
        self._region = new_region

```

```

        self._region = new_region

@property
def stars(self):
    return self._stars

@stars.setter
def stars(self, new_stars):
    self._stars = new_stars

def save(self):
    with open(self._file, 'a') as f:
        writer = csv.writer(f)
        writer.writerows(self._new_coffee)

def show_coffee(self):
    print()
    if len(self._old_coffee) < 2 and len(self._new_coffee) == 0:
        print("Enter a coffee first")
    elif len(self._old_coffee) < 2 and len(self._new_coffee) == 0:
        for row in self._old_coffee:
            print(f"{row[0]:15} {row[1]:15} {row[2]:15} {row[3]:15}")
        else:
            for row in self._old_coffee:
                print(f"{row[0]:15} {row[1]:15} {row[2]:15} {row[3]:15}")
            for row in self._new_coffee:
                print(f"{row[0]:15} {row[1]:15} {row[2]:15} {row[3]:15}")
    print()

def add_coffee(self):
    self._new_coffee.append([self._roaster, self._country, self._region])

# *****
# code for the command line application
# *****

def main_menu():
    print("Coffee of the World")
    print("\t1. Show Coffee")
    print("\t2. Add Coffee")
    print("\t3. Save and Quit")

```

```

    print( "\n5. Save and quit ")
    choice = int(input("Enter the number of your selection: "))
    return choice

def perform_action(choice, coffee):
    if choice == 1:
        coffee.show_coffee()
    elif choice == 2:
        enter_coffee(coffee)
    elif choice == 3:
        quit(coffee)

def enter_coffee(coffee):
    print()
    coffee.roaster = input("Enter the name of the roaster: ")
    coffee.country = input("Enter the country of origin: ")
    coffee.region = input("Enter the region: ")
    coffee.stars = input("Enter the number of stars '*' (1-4): ")
    print()
    coffee.add_coffee()

def quit(coffee):
    global run_loop
    coffee.save()
    run_loop = False

# *****
# code for testing your script
# *****

run_loop = True
file = "code/encapsulation/coffee_journal.csv"
my_coffee = CoffeeJournal(file)

while run_loop:
    choice = main_menu()
    perform_action(choice, my_coffee)

```

Lab Challenge

Lab Challenge

Problem

Write a class named `Person` that has attributes for name, age, and occupation. These attributes should follow the Python convention for private attributes. Create getters and setters for each attribute. Name your getters and setters as `get_` followed by the attribute name or `set_` followed by the attribute name. Do not use the property decorator or function for the getters and setters.

Expected Output

- * Declare the instance `my_person = Person("Citra Curie", 16, "student")`
- * The method `get_name()` returns `Citra Curie`
- * The method `set_name("Rowan Faraday")` changes the name attribute to "Rowan Faraday"
- * The method `get_age()` returns `16`
- * The method `set_age(18)` changes the age attribute to `18`
- * The method `get_occupation()` returns `student`
- * The method `set_occupation("plumber")` changes the occupation attribute to "plumber"

Important

Use the Python convention for designating private attributes, and do not use the property decorator or function.

Testing Your Code

Use the button below to test your code before submitting it for evaluation.