Task 8

Exercise 9.1: Restaurant.

Make a class called `Restaurant`. The `__init__()` method for `Restaurant` should store two attributes: a restaurant_name and a cuisine_type. Make a method called `describe_restaurant()` that prints these two pieces of information, and a method called `open_restaurant()` that prints a message indicating that the restaurant is open. Make an instance called `restaurant` from your class. Print the two attributes individually, and then call both methods.

In [ ]:
```
class Restaurant():
    def __init__(self, name, cuisine_type):
        self.name = name
        self.cuisine_type = cuisine_type
    def describe_restaurant(self):
        print(f"{self.name} serves wonderful {self.cuisine_type}.")
    def open_restaurant(self):
        print(f"{self.name} is open. Come on in!")

restaurant=Restaurant('Half Michelin Star','Chinese')
print(restaurant.name)
print(restaurant.cuisine_type)
restaurant.describe_restaurant()
restaurant.open_restaurant()
```

```
Half Michelin Star
Chinese
Half Michelin Star serves wonderful Chinese.
Half Michelin Star is open. Come on in!
```

Exercise 9.2: Three Restaurants.

Start with your class from Exercise 9.1. Create three different instances from the class, and call `describe_restaurant()` for each instance.

In [ ]:
```
class Restaurant():
    def __init__(self, name, cuisine_type):
        self.name = name
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        print(f"{self.name} serves wonderful {self.cuisine_type}.")

    def open_restaurant(self):
        print(f"{self.name} is open. Come on in!")

restaurant1=Restaurant('Half Michelin Star','Chinese')
```

```
restaurant2=Restaurant('Full Michelin Star','French')
restaurant3=Restaurant('No Michelin Star','American')

restaurant1.describe_restaurant()
restaurant2.describe_restaurant()
restaurant3.describe_restaurant()
```

```
Half Michelin Star serves wonderful Chinese.
Full Michelin Star serves wonderful French.
No Michelin Star serves wonderful American.
```

Exercise 9.3: Users.

Make a class called `User`. Create two attributes called `first_name` and `last_name`, and then create several other attributes that are typically stored in a user profile. Make a method called `describe_user()` that prints a summary of the user's information. Make another method called `greet_user()` that prints a personalized greeting to the user. Create several instances representing different users, and call both methods for each user.

In [ ]:
```
class user():
    def __init__(self, first_name, last_name, age, gender):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.gender = gender
    def describe_user(self):
        print(f"{self.first_name} {self.last_name}, of age {self.age}, is a {self.g
    def greet_user(self):
        print(f"Hello, {self.first_name} {self.last_name}!")

user1=user('Muid', 'Bin Aslam', '22', 'male')
user2=user('Muhammad', 'Ali', '24', 'male')
user3=user('Muhammad', 'Usman', '23', 'male')

user1.describe_user()
user1.greet_user()
user2.describe_user()
user2.greet_user()
user3.describe_user()
user3.greet_user()
```

```
Muid Bin Aslam, of age 22, is a male
Hello, Muid Bin Aslam!
Muhammad Ali, of age 24, is a male
Hello, Muhammad Ali!
Muhammad Usman, of age 23, is a male
Hello, Muhammad Usman!
```

Exercise 9.4: Number Served.

Start with your program from Exercise 9.1 (page 166). Add an attribute called `number_served` with a default value of `0`. Create an instance

called `restaurant` from this class. Print the number of customers the restaurant has served, and then change this value and print it again.
Add a method called `set_number_served()` that lets you set the number of customers that have been served. Call this method with a new number and print the value again.
Add a method called `increment_number_served()` that lets you increment the number of customers who've been served. Call this method with any number you like that could represent how many customers were served in, say, a day of business.

In [ ]:
```
class Resuaurant():
    def __init__(self, name, cuisine_type):
        self.name = name
        self.cuisine_type = cuisine_type
        self.number_served = 0
    def describe_restaurant(self):
        print(f"{self.name} serves wonderful {self.cuisine_type}.")
    def open_restaurant(self):
        print(f"{self.name} is open. Come on in!")
    def set_number_served(self, number):
        self.number_served = number
    def increment_number_served(self, number):
        self.number_served += number

restaurant=Resuaurant('Half Michelin Star','Chinese')
restaurant.describe_restaurant()
print(f"Number served: {restaurant.number_served}")
restaurant.number_served = 23
print(f"Number served: {restaurant.number_served}")
restaurant.set_number_served(46)
print(f"Number served: {restaurant.number_served}")
restaurant.increment_number_served(23)
print(f"Number served: {restaurant.number_served}")
```

```
Half Michelin Star serves wonderful Chinese.
Number served: 0
Number served: 23
Number served: 46
Number served: 69
```

Exercise 9.5: Login Attempts.

Add an attribute called `login_attempts` to your `User` class from Exercise 9.3 (page 166). Write a method called `increment_login_attempts()` that increments the value of `login_attempts` by 1. Write another method called `reset_login_attempts()` that resets the value of `login_attempts` to 0.
Make an instance of the `User` class and call `increment_login_attempts()` several times. Print the value of `login_attempts` to make sure it was incremented properly, and then call `reset_login_attempts()` . Print `login_attempts` again to make sure it was reset to 0.

In [ ]:
```
class user():
    def __init__(self, first_name, last_name, username, email, location):
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        print(f"{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        print(f"Hello, {self.first_name} {self.last_name}!")

    def increment_login_attempts(self):
        self.login_attempts += 1

    def reset_login_attempts(self):
        self.login_attempts = 0

user=user('muid', 'bin aslam', 'mxid', 'abdulmuid300@gmail.com', 'islamabad')
user.describe_user()
user.greet_user()

print(f"Login attempts: {user.login_attempts}")
user.increment_login_attempts()
user.increment_login_attempts()
user.increment_login_attempts()
print(f"Login attempts: {user.login_attempts}")

print("Resetting login attempts...")
user.reset_login_attempts()
print(f"Login attempts: {user.login_attempts}")
```

```
Muid Bin Aslam
  Username: mxid
  Email: abdulmuid300@gmail.com
  Location: Islamabad
Hello, Muid Bin Aslam!
Login attempts: 0
Login attempts: 3
Resetting login attempts...
Login attempts: 0
```

Exercise 9.6: Ice Cream Stand.

An ice cream stand is a specific kind of restaurant. Write a class called `IceCreamStand` that inherits from the `Restaurant` class you wrote in Exercise 9.1 (page 166) or Exercise 9.4 (page 171). Either version of the class will work; just pick the one you like better. Add an attribute called `flavors` that stores a list of ice cream flavors. Write a method that displays these flavors. Create an instance of `IceCreamStand`, and call this method.

In [ ]:
```python
class Resuaurant():
    def __init__(self, name, cuisine_type):
        self.name = name
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        print(f"{self.name} serves wonderful {self.cuisine_type}.")

    def open_restaurant(self):
        print(f"{self.name} is open. Come on in!")

    def set_number_served(self, number):
        self.number_served = number

    def increment_number_served(self, number):
        self.number_served += number

class IceCreamStand(Resuaurant):
    def __init__(self, name, cuisine_type='ice cream'):
        super().__init__(name, cuisine_type)
        self.flavors = []

    def show_flavors(self):
        print("Flavours available are:")
        for flavor in self.flavors:
            print(f"- {flavor.title()}")

stand=IceCreamStand('Soft Swirl')
stand.flavors = ['cookies n cream', 'chocolate', 'brownie']

stand.describe_restaurant()
stand.show_flavors()
```

```
Soft Swirl serves wonderful ice cream.
Flavours available are:
- Cookies N Cream
- Chocolate
- Brownie
```

Exercise 9.7: Admin.

An administrator is a special kind of user. Write a class called `Admin` that inherits from the `User` class you wrote in Exercise 9.3 (page 166) or Exercise 9.5 (page 171). Add an attribute, `privileges`, that stores a list of strings like `"can add post"`, `"can delete post"`, `"can ban user"`, and so on. Write a method called `show_privileges()` that lists the administrator's set of privileges. Create an instance of `Admin`, and call your method.

In [ ]:
```python
class user():
    def __init__(self, first_name, last_name, username, email, location):
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        print(f"{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        print(f"Hello, {self.first_name} {self.last_name}!")

    def increment_login_attempts(self):
        self.login_attempts += 1

    def reset_login_attempts(self):
        self.login_attempts = 0

class admin(user):
    def __init__(self, first_name, last_name, username, email, location):
        super().__init__(first_name, last_name, username, email, location)
        self.privileges = []

    def show_privileges(self):
        print("Privileges:")
        for privilege in self.privileges:
            print(f"- {privilege}")

admin=admin('muid', 'bin aslam', 'mxid', 'abdulmuid300@gmail.com', 'islamabad')
admin.describe_user()
```

```
admin.privileges = [
    'can ban accounts',
    'can mute accounts',
    'can view logs'
    ]

admin.show_privileges()
```

```
Muid Bin Aslam
  Username: mxid
  Email: abdulmuid300@gmail.com
  Location: Islamabad
Privileges:
- can ban accounts
- can mute accounts
- can view logs
```

In [ ]:

```
class user():
    def __init__(self, first_name, last_name, username, email, location):
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        print(f"{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        print(f"Hello, {self.first_name} {self.last_name}!")

    def increment_login_attempts(self):
        self.login_attempts += 1

    def reset_login_attempts(self):
        self.login_attempts = 0


class Privileges():
    def __init__(self, privileges=[]):
        self.privileges = privileges

    def show_privileges(self):
        print("Privileges:")
        if self.privileges:
            for privilege in self.privileges:
                print(f"- {privilege}")
        else:
            print("- This user has no privileges.")

class admin(user):
    def __init__(self, first_name, last_name, username, email, location):
```

```
        super().__init__(first_name, last_name, username, email, location)
        self.privileges = Privileges()

admin=admin('muid', 'bin aslam', 'mxid', 'abdulmuid300@gmail.com', 'islamabad')
admin.describe_user()

admin.privileges.show_privileges()

print("Adding privileges...")
admin_privileges = [
    'can ban accounts',
    'can mute accounts',
    'can view logs'
]
admin.privileges.privileges = admin_privileges
admin.privileges.show_privileges()
```

```
Muid Bin Aslam
  Username: mxid
  Email: abdulmuid300@gmail.com
  Location: Islamabad
Privileges:
- This user has no privileges.
Adding privileges...
Privileges:
- can ban accounts
- can mute accounts
- can view logs
```

Exercise 9.9: Battery Upgrade.

Use the final version of `electric_car.py` from this section. Add a method to the `Battery` class called `upgrade_battery()`. This method should check the battery size and set the capacity to 100 if it isn't already. Make an electric car with a default battery size, call `get_range()` once, and then call `get_range()` a second time after upgrading the battery. You should see an increase in the car's range.

In [ ]:
```
class car():
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
```

```
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles

class battery():
    def __init__(self, battery_size=75):
        self.battery_size = battery_size

    def describe_battery(self):
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315

        message = f"This car can go approximately {range}"
        message += " miles on a full charge."
        print(message)

    def upgrade_battery(self):
        if self.battery_size != 100:
            self.battery_size = 100

class electricCar(car):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.battery = battery()

rivian=electricCar('rivian', 'r1t', 2021)
print(rivian.get_descriptive_name())
rivian.battery.get_range()

print("Upgrading battery...")
rivian.battery.upgrade_battery()
rivian.battery.get_range()
```

```
2021 Rivian R1T
This car can go approximately 260 miles on a full charge.
Upgrading battery...
This car can go approximately 315 miles on a full charge.
```