Homework Problems

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 [1]:
```
class Restaurant():
    def __init__(self, restaurant_name, cuisine_type):
        self.restaurant_name = str(restaurant_name).title()
        self.cuisine_type = str(cuisine_type).title()

    def describe_restaurant(self):
        print(f"{self.restaurant_name} makes {self.cuisine_type} food")

    def open_restaurant(self):
        print(f"{self.restaurant_name} is open!")

restaurant = Restaurant('chipotle', 'mexican')
print(restaurant.restaurant_name)
print(restaurant.cuisine_type)
restaurant.describe_restaurant()
restaurant.open_restaurant()
```

```
Chipotle
Mexican
Chipotle makes Mexican food
Chipotle is open!
```

9-2. Three Restaurants: Start with your class from above. Create three different instances of the class, and call `describe_restaurant()` for each instance.

In [2]:
```
restaurant1 = Restaurant('iron rooster', 'american')
restaurant1.describe_restaurant()

restaurant2 = Restaurant('pussers', 'caribbean')
restaurant2.describe_restaurant()

restaurant3 = Restaurant('moes', 'mexican')
restaurant3.describe_restaurant()
```

```
Iron Rooster makes American food
Pussers makes Caribbean food
Moes makes Mexican food
```

9-3. Users: Make a class called `User`. Create two attributes called `first_name` and `last_name`, and then create at least three 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 three instances representing different users, and call both methods for each user.

In [3]:
```
class User():
    def __init__(self, first_name, last_name, age, race, gender):
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.age = age
        self.race = race.title()
        self.gender = gender.title()

    def describe_user(self):
        print(f"{self.first_name} {self.last_name} \nage: {self.age} \nrace:

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

andrew = User('andrew', 'bernas', 20, 'asian', 'male')
andrew.describe_user()
andrew.greet_user()
print("\n")

dug = User('dug', 'moloney', 21, 'caucasian', 'male')
dug.describe_user()
dug.greet_user()
print("\n")

maeve = User('maeve', 'carrigg', 19, 'caucasian', 'female')
maeve.describe_user()
maeve.greet_user()
print("\n")
```

```
Andrew Bernas
age: 20
race: Asian
gender: Male
Hello Andrew!


Dug Moloney
age: 21
race: Caucasian
gender: Male
Hello Dug!


Maeve Carrigg
age: 19
race: Caucasian
gender: Female
Hello Maeve!
```

9-4. Number Served: Start with your program from 9-1. 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 [4]:
```
class Restaurant():
    def __init__(self, restaurant_name, cuisine_type):
        self.restaurant_name = str(restaurant_name).title()
        self.cuisine_type = str(cuisine_type).title()
        self.number_served = 0

    def describe_restaurant(self):
        print(f"{self.restaurant_name} makes {self.cuisine_type} food")

    def open_restaurant(self):
        print(f"{self.restaurant_name} is open!")

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

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

restaurant = Restaurant('chipotle', 'mexican')
print(restaurant.number_served)
restaurant.number_served = 2
print(restaurant.number_served)
restaurant.set_number_served(4)
print(restaurant.number_served)
restaurant.increment_number_served(5)
print(restaurant.number_served)
```

```
0
2
4
9
```

9-5. Login Attempts: Add an attribute called `login_attempts` to your `User` class from 9-3. 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 propertly, and then call `reset_login_attempts()`. Print `login_attempts` again to make sure it was reset to 0.

In [5]:
```
class User():
    def __init__(self, first_name, last_name, age, race, gender, login_attemp
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.age = age
        self.race = race.title()
        self.gender = gender.title()
        self.login_attempts = login_attempts

    def describe_user(self):
        print(f"{self.first_name} {self.last_name} \nage: {self.age} \nrace:

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

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

    def reset_login_attempts(self):
        self.login_attempts = 0

andrew = User('andrew', 'bernas', 20, 'asian', 'male', 0)
print(andrew.login_attempts)
andrew.increment_login_attempts()
andrew.increment_login_attempts()
andrew.increment_login_attempts()
print(andrew.login_attempts)
andrew.reset_login_attempts()
print(andrew.login_attempts)
```

```
0
3
0
```

9-6. Ice Cream Stand: An ice cream stand is a specific kind of restaurant. Write a class caled `IceCreamStand` that inherits from the `Restaurant` class you wrote in 9-1 or 9-4. Either version of the clas 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 [6]:
```
class IceCreamStand(Restaurant):
    def __init__(self, restaurant_name, *flavors):
        super().__init__(restaurant_name, cuisine_type = 'ice cream')
        self.flavors = list(flavors)

    def display_flavors(self):
        print(f"{self.restaurant_name} sells: ")
        for flavor in self.flavors:
            print(f"- {flavor.title()}")

baskin_robbins = IceCreamStand('Baskin-Robbins', 'chocolate', 'vanilla', 'roc
baskin_robbins.display_flavors()
```

```
Baskin-Robbins sells:
- Chocolate
- Vanilla
- Rocky Road
```

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 9-3 or 9-5. 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 [7]:
```
class Admin(User):
    def __init__(self, first_name, last_name, age, race, gender, login_attemp
        super().__init__(first_name, last_name, age, race, gender, login_atte
        self.privileges = ['can add post', 'can delete post', 'can ban user',

    def show_privileges(self):
        print("You have the following privileges: ")
        for privilege in self.privileges:
            print(f"- {privilege}")

admin = Admin('andrew', 'bernas', '20', 'asian', 'male', 0)
admin.show_privileges()
```

```
You have the following privileges:
- can add post
- can delete post
- can ban user
- can create user
```

9-8. Privileges: Write a separate `Privileges` class. The class should have one attribute, `privileges`, that stores a list of strings as described in 9-7. Move the `show_privileges()` method to this class. Make a `Privileges` instance as an attribute in the `Admin` class. Create a new instance of `Admin` and use your method to show its privileges.

In [8]:
```
class Privileges():
    def __init__(self):
        self.privileges = ['can add post', 'can delete post', 'can ban user',

    def show_privileges(self):
        print("You have the following privileges: ")
        for privilege in self.privileges:
            print(f"- {privilege}")

class Admin(User):
    def __init__(self, first_name, last_name, age, race, gender, login_attemp
        super().__init__(first_name, last_name, age, race, gender, login_atte
        self.privileges = Privileges()

admin = Admin('andrew', 'bernas', '20', 'asian', 'male', 0)
admin.privileges.show_privileges()
```

```
You have the following privileges:
- can add post
- can delete post
- can ban user
- can create user
```

9-9. Battery Upgrade: Use the final verion of the `electric_car.py` from this lecture. 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 the call `get_range()` a second time after upgrading the battery. You should see an increase in the car's range.

In [9]:

```
class Car():
    """A simple attempt to represent a car."""

    def __init__(self, manufacturer, model, year):
        """Initialize attributes to describe a car."""
        self.manufacturer = manufacturer
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = str(self.year) + ' ' + self.manufacturer + ' ' + self.mod
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles

class Battery():
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=60):
        """Initialize the batteery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 60:
            range = 140
        elif self.battery_size == 85:
            range = 185
        elif self.battery_size == 100:
            range = 240

        message = "This car can go approximately " + str(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):
    """Models aspects of a car, specific to electric vehicles."""

    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery = Battery()

tesla = ElectricCar('tesla', 'model s', '2023')
tesla.battery.get_range()
tesla.battery.upgrade_battery()
tesla.battery.get_range()
```

```
This car can go approximately 140 miles on a full charge.
This car can go approximately 240 miles on a full charge.
```

9-10. Imported Restaurant: Using your latest `Restaurant` class, store it in a module. Import `Restaurant` and make an instance of the class. Then call one of its methods to show that the `import` statement is working properly.

In [10]:
```
import restaurant

chipotle = restaurant.Restaurant('chipotle', 'mexican')
chipotle.describe_restaurant()
```

```
Chipotle makes Mexican food
```

9-11. Imported Admin: Start with your work form 9-8. Store the classes `User`, `Privileges`, and `Admin` in one module. Import just the `Admin` class and make an instance of it. Then call `show_privileges()` to show that everything is working properly.

In [11]:
```
from user import Admin

admin = Admin('andrew', 'bernas', '20', 'asian', 'male', 0)
admin.privileges.show_privileges()
```

```
You have the following privileges:
- can add post
- can delete post
- can ban user
- can create user
```

9-12. Multiple Modules: Store the `User` class in one module, and store the `Privileges` and `Admin` classes in a a seperate module. Set up the import statements so that you can create an instance of an `Admin` below. Then call `show_privileges()` to show that

everything is working properly.

In [12]:
```
import admin

admin = Admin('andrew', 'bernas', '20', 'asian', 'male', 0)
admin.privileges.show_privileges()
```

```
You have the following privileges:
- can add post
- can delete post
- can ban user
- can create user
```

9-13. Dice: Make a class `Die` with one attribute called `sides`, which has a default value of 6. Write a method called `roll_die()` that prints a random number between 1 and the number of sides the die has. Make a 6-sided die and roll it 10 times.

Make a 10-sided die and a 20-sided die. Roll each die 10 times.

In [13]:

```python
from random import randint

class Die():
    def __init__(self, sides=6):
        self.sides = sides

    def roll_die(self):
        print(f"You rolled a {randint(1,self.sides)}")

six_sided_die = Die()
for x in range(10):
    six_sided_die.roll_die()

print("\n")

ten_sided_die = Die(10)
for x in range(10):
    ten_sided_die.roll_die()

print("\n")

twenty_sided_die = Die(20)
for x in range(10):
    twenty_sided_die.roll_die()
```

```
You rolled a 2
You rolled a 6
You rolled a 1
You rolled a 2
You rolled a 6
You rolled a 3
You rolled a 3
You rolled a 2
You rolled a 1
You rolled a 6


You rolled a 8
You rolled a 9
You rolled a 2
You rolled a 3
You rolled a 4
You rolled a 1
You rolled a 5
You rolled a 3
You rolled a 10
You rolled a 7


You rolled a 7
You rolled a 4
You rolled a 4
You rolled a 1
You rolled a 4
You rolled a 13
You rolled a 7
You rolled a 15
You rolled a 13
You rolled a 9
```

9-14. Lottery: Make a list or tuple containing a series of 10 numbers and five letters. Randomly select four numbers or letters from the list and print a message saying that any ticket matching these four numbers or letters wins a prize.

In [14]:
```
from random import choice

my_list = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'b', 'c', 'd', 'e')

ticket = []
for x in range(4):
    ticket.append(str(choice(my_list)))

print(ticket)
```

```
['c', '2', '1', '5']
```

9-15. Lottery Analysis: You can use a loop to see how hard it might be to win the kind of lottery you just modeled. Make a list or tuple called `my_ticket` . Write a loop that keeps pulling numbers until your ticket wins. Print a message reporting how many times the loop had

to run to give you a winning ticket.

In [15]:
```
from random import choice

my_list = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'b', 'c', 'd', 'e')
my_ticket = ['6', 'e', '5', '1']
i = 0

while True:
    ticket = []
    for x in range(4):
        ticket.append(str(choice(my_list)))

    if ticket != my_ticket:
        i += 1
    else:
        print(f"Looped {i} times.")
        break
```

```
Looped 50531 times.
```

9-16. Python Module of the Week: One excellent resource for exploring the Python standard library is a site called `Python Module of the Week` . Go to https://pymotw.com/ and look at the table of contents. Find a module that looks interesting to you and read about it, perhaps starting with the `random` module. Write three things you learned about the module below.

hashlib:

- Allows you easy access to cryptographic hashing algorithms
- It is “backed” by OpenSSL
- Some algorithms are available on all platforms, and some depend on the underlying libraries