Exercise

1

Create a Car class with two instance attributes:

1. `.color`, which stores the name of the car's color as a string
2. `.mileage`, which stores the number of miles on the car as an integer

Then instantiate two Car objects—a blue car with 20,000 miles and a red car with 30,000 miles—and print out their colors and mileage. Your output should look like this:

```
The blue car has 20,000 miles.
The red car has 30,000 miles.
```

In [ ]:

2

Define a Student class with four instance attribures:

1. `.name`, which holds the full name of the student as a string, e.g., 'John Downy'
2. `.sex`, which holds either `'male'` or `'female'`,
3. `.birthdate`, which holds the birthdate as an integer in the format of YYYYMMDD, e.g., 19950204 for "4th December 1995",
4. `.gpa`, which holds the student GPA as a float with 2 decimal points, e.g., 2.75.

Consequently, creates a Student object, and print out its data in the following format,

```
Mr. John Downy/ 19950204/ 2.75
Ms. Susann Katzyan/ 20081125/ 3.22
```

In [ ]:

3

From the Student class defined above, write a class method `.status()`, which prints a string showing the status from the following conditions

GPA	Status
< 1.00	Dismissal
>= 1.00 and < 2.00	Probation
>= 2.00 and < 2.75	Normal
>= 2.75	Good

In [ ]:

4

Define a Book class with 3 instance atrributes:

1. `.title`, which stores the book title as a string,
2. `.author`, which stores the author's name as a string,
3. `.year`, which stores the publishing year.

When print its object, the output shoud be like:

```
Mathematical Analysis written by Walter Rudin was published in
1992.
```

In [ ]:

5

Define a class called `BankAccount` that has three attributes: `balance`, `owner`, and `transactions`.

- The `balance` attribute should be initialized to 0, and the transactions attribute should be initialized to an empty list.
- Define the following methods for the BankAccount class:
  1. `deposit(amount)`: adds the amount to the balance and records the transaction in the transactions list.
  2. `withdraw(amount)`: subtracts the amount from the balance and records the transaction in the transactions list. If the amount is greater than the balance, print an error message.
  3. `get_balance()`: returns the current balance.
  4. `get_transactions()`: returns a list of all the transactions.
- Define a constructor method that initializes the `owner` attribute.

In [ ]:

Create an instance of the BankAccount class with your own name as the owner.

Test the methods by making several deposits and withdrawals, and print the balance and transaction history using the get_balance and get_transactions methods.

For example,

```
account = BankAccount("John")
account.deposit(100)
account.withdraw(50)
account.deposit(500)
print(account.get_balance())  # Output: 550
print(account.get_transactions())  # Output: ['Deposited 100',
'Withdrew 50', 'Deposited 500']
```

In [ ]:

6

Define a class called `Circle` that has two attributes: `radius` and `color`.

Define the following methods for the Circle class:

- `get_radius()`: returns the radius of the circle.
- `get_color()`: returns the color of the circle.
- `set_radius(radius)`: sets the radius of the circle to the specified value.
- `set_color(color)`: sets the color of the circle to the specified value.
- `get_area()`: returns the area of the circle.
- `get_circumference()`: returns the circumference of the circle.

Also define a constructor method that initializes the `radius` and `color` attributes.

In [ ]:

Create an instance of the Circle class with your own preferred radius and color. Test the methods by printing the radius, color, area, and circumference of the circle.

For example,

```
circle = Circle(5, "red")
print(circle.get_radius())  # Output: 5
print(circle.get_color())  # Output: "red"
print(circle.get_area())  # Output: 78.53981633974483
print(circle.get_circumference())  # Output: 31.41592653589793
```

In [ ]:

7

Here is an exercise for creating a class that represents a character in an RPG game

1. Define a class called `Character` that has three attributes: `name`, `health`, and `power`.
2. Define the following methods for the Character class:
   - `attack(other_character)`: reduces the other_character's health by the attacking character's power.
   - `is_alive()`: returns `True` if the character's health is greater than 0, and `False` otherwise.
   - `get_health()`: returns the character's current health.
3. Define a constructor method that initializes the `name`, `health`, and `power` attributes.

In [ ]:

Create two instances of the Character class with different names, health, and power values. Test the attack method by making one character attack the other, and print the health of each character using the get_health method to verify that the attack was successful.

For example,

```
hero = Character("Hero", 10, 5)
villain = Character("Villain", 5, 10)
hero.attack(villain)
print(villain.get_health())  # Output: 0
print(villain.is_alive())  # Output: False
```

In [ ]:
