Mutable objects

- Definition: Objects whose contents can be changed without changing their identity.
  - Lists (list)
  - Dictionaries (dict)
  - User-defined objects (if their attributes can be modified)

```
numbers = [1, 2, 3]
numbers[0] = 10 # Change an element
print(numbers) # Output: [10, 2, 3]
```

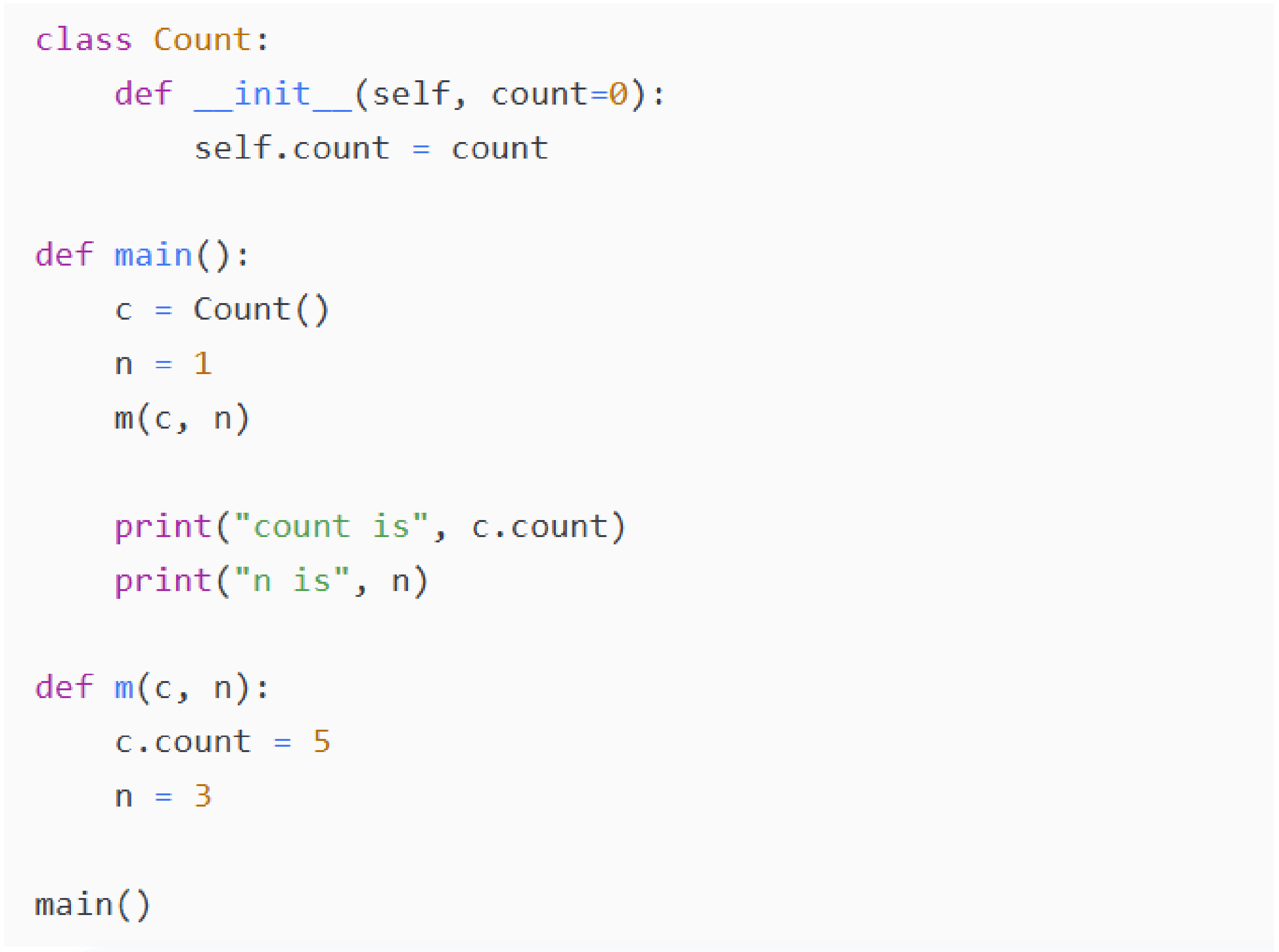
Immutable objects

- Definition: Objects whose contents cannot be changed after creation; any modification creates a new object.
  - Integers (int)
  - Floats (float)
  - Strings (str)
  - Tuples (tuple)

```
name = ″Alice″
name = name.replace(″A″, ″M″)
print(name)
```

Even though it looks like we changed `name`, the original string `″Alice″` remained unchanged, and Python returned a **new string**.

Practice



```
class Count:
    def __init__(self, count=0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c.count = 5
    n = 3

main()
```

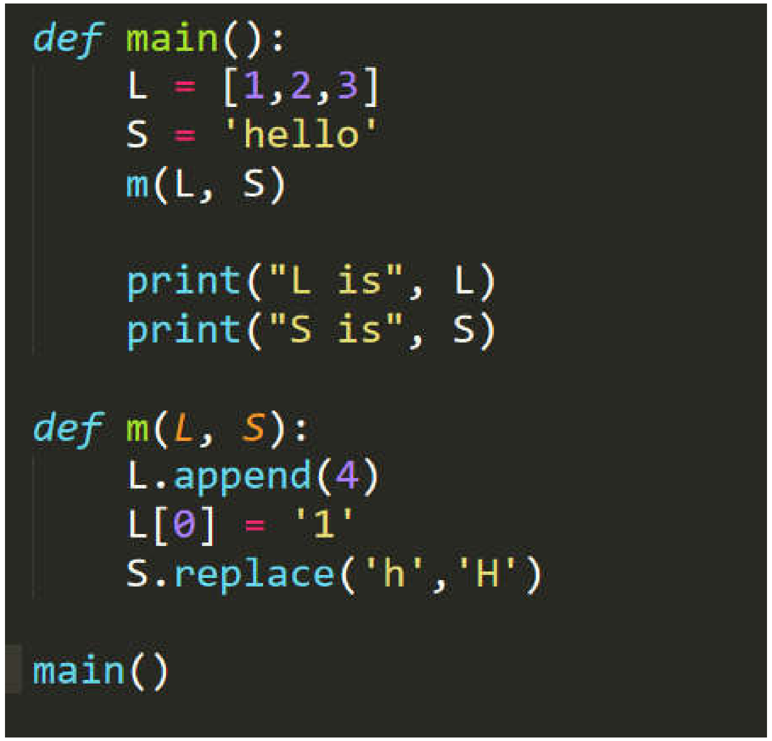
- What would be the output of the above program?

Practice

```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```

- What would be the output of the above program?

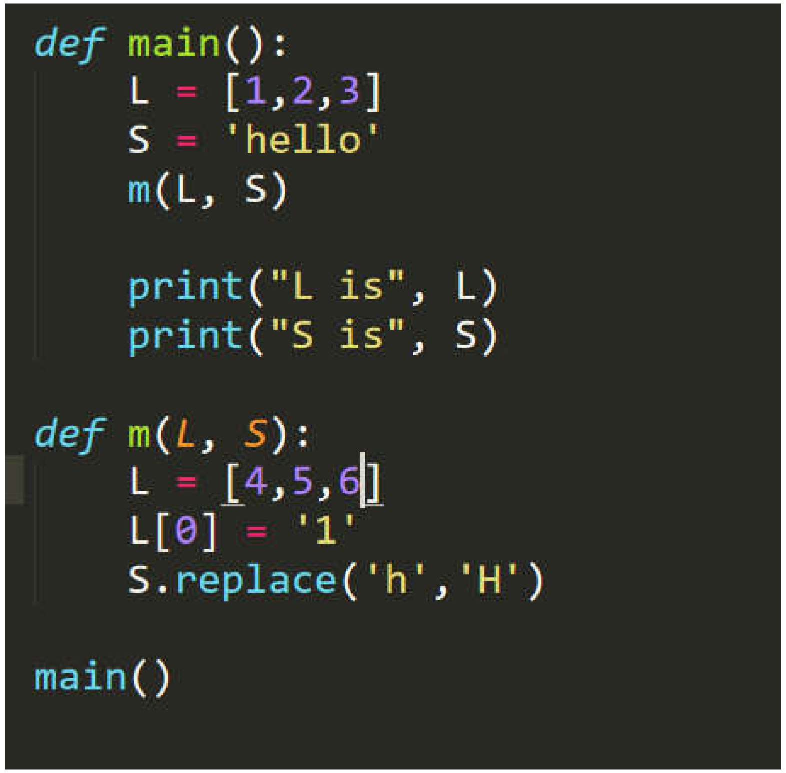
Practice



```python
def main():
    L = [1,2,3]
    S = 'hello'
    m(L, S)

    print("L is", L)
    print("S is", S)

def m(L, S):
    L.append(4)
    L[0] = '1'
    S.replace('h','H')

main()
```

- What would be the output of the above program?

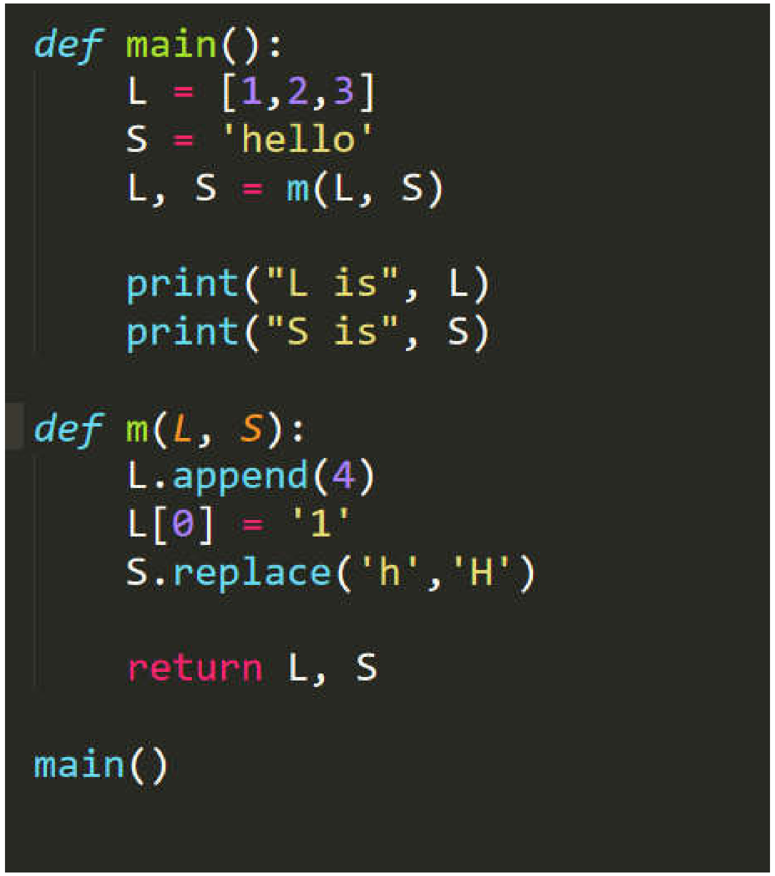
Practice



```
def main():
    L = [1,2,3]
    S = 'hello'
    m(L, S)

    print("L is", L)
    print("S is", S)

def m(L, S):
    L = [4,5,6]
    L[0] = '1'
    S.replace('h','H')

main()
```

- What would be the output of the above program?

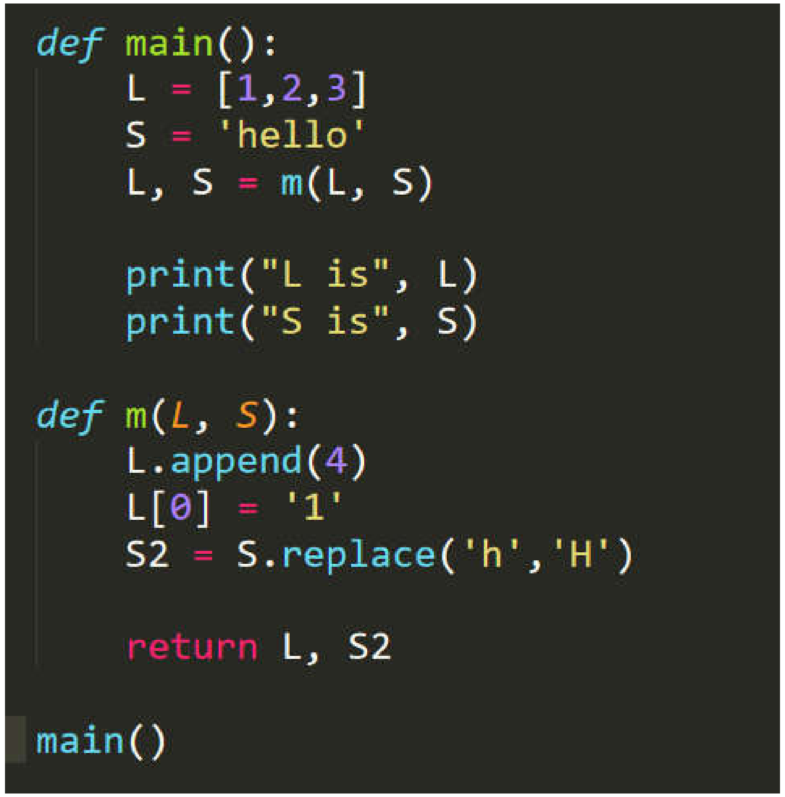
Practice



```
def main():
    L = [1,2,3]
    S = 'hello'
    L, S = m(L, S)

    print("L is", L)
    print("S is", S)

def m(L, S):
    L.append(4)
    L[0] = '1'
    S.replace('h','H')

    return L, S

main()
```

- What would be the output of the above program?

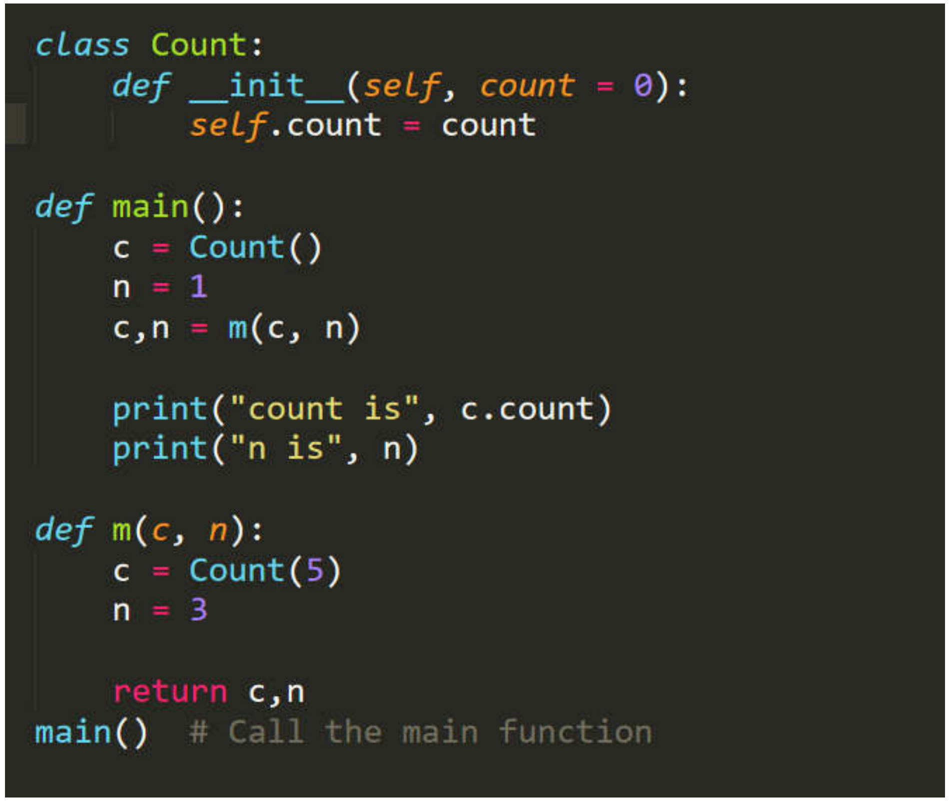
Practice



```python
def main():
    L = [1,2,3]
    S = 'hello'
    L, S = m(L, S)

    print("L is", L)
    print("S is", S)

def m(L, S):
    L.append(4)
    L[0] = '1'
    S2 = S.replace('h','H')

    return L, S2

main()
```

- What would be the output of the above program?

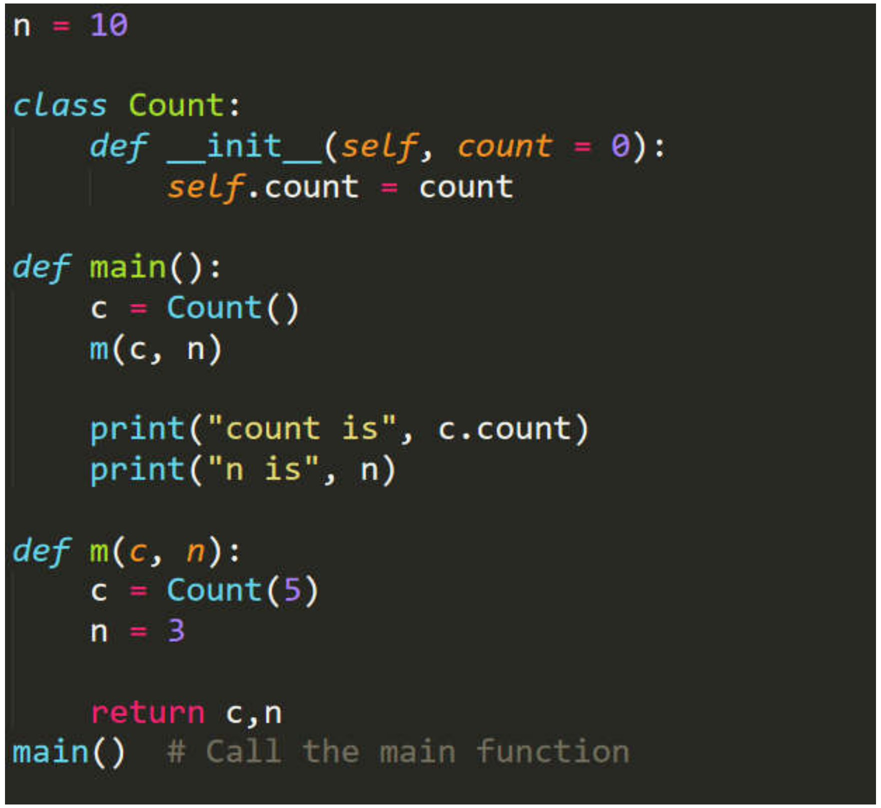
Practice



```python
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    c,n = m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

    return c,n
main()  # Call the main function
```

- What would be the output of the above program?

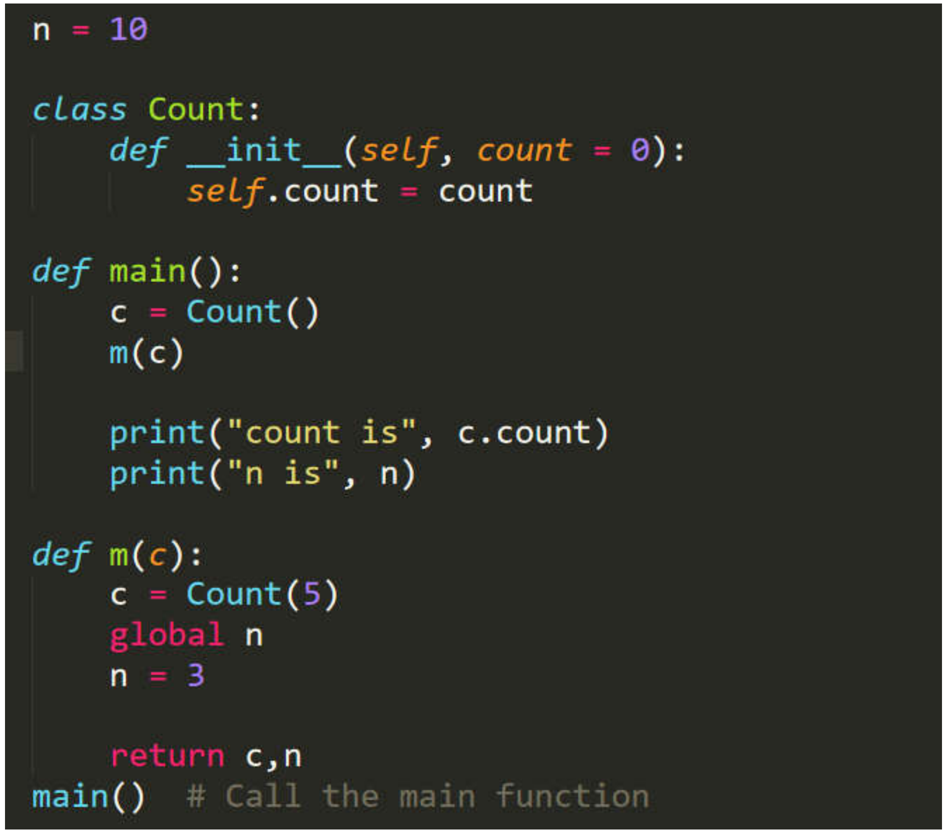
Practice



```python
n = 10

class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

    return c,n
main()  # Call the main function
```

- What would be the output of the above program?

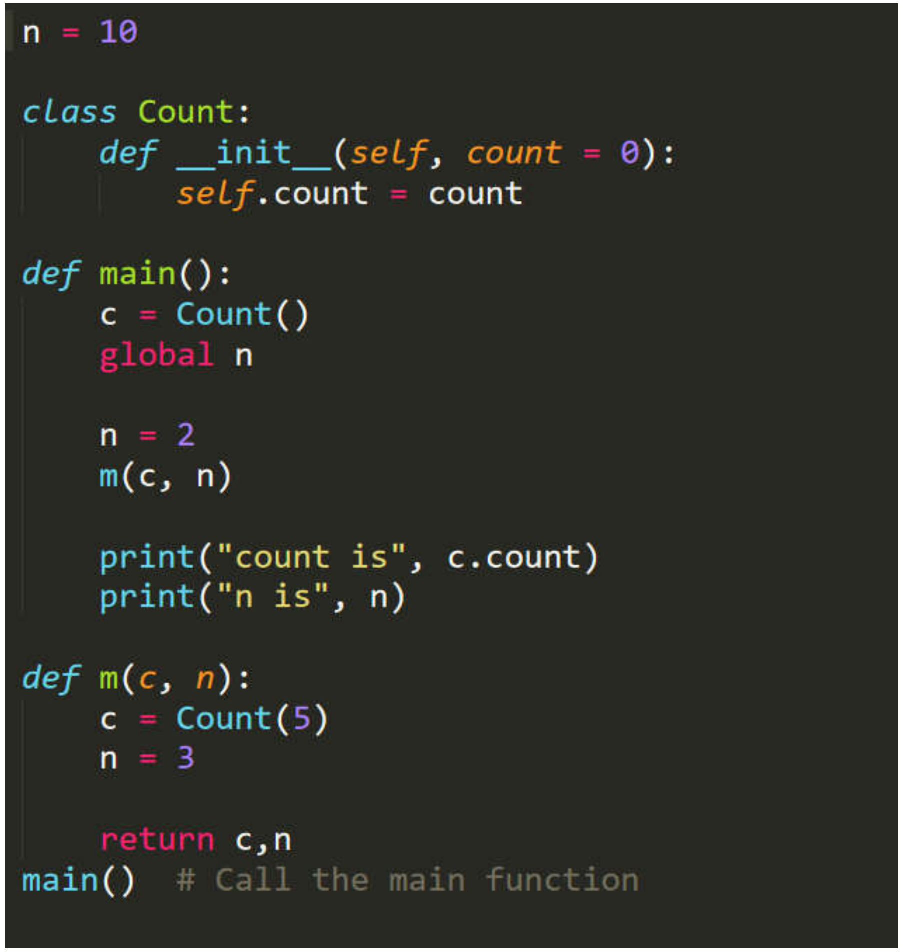
Practice



```python
n = 10

class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    m(c)

    print("count is", c.count)
    print("n is", n)

def m(c):
    c = Count(5)
    global n
    n = 3

    return c,n
main()  # Call the main function
```

- What would be the output of the above program?

Practice



```python
n = 10

class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    global n

    n = 2
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

    return c,n
main()  # Call the main function
```

- What would be the output of the above program?

Practice

(The Rectangle class) Following the example of the Circle class, design a class named Rectangle to represent a rectangle. The class contains:

■ Two data fields named width and height.

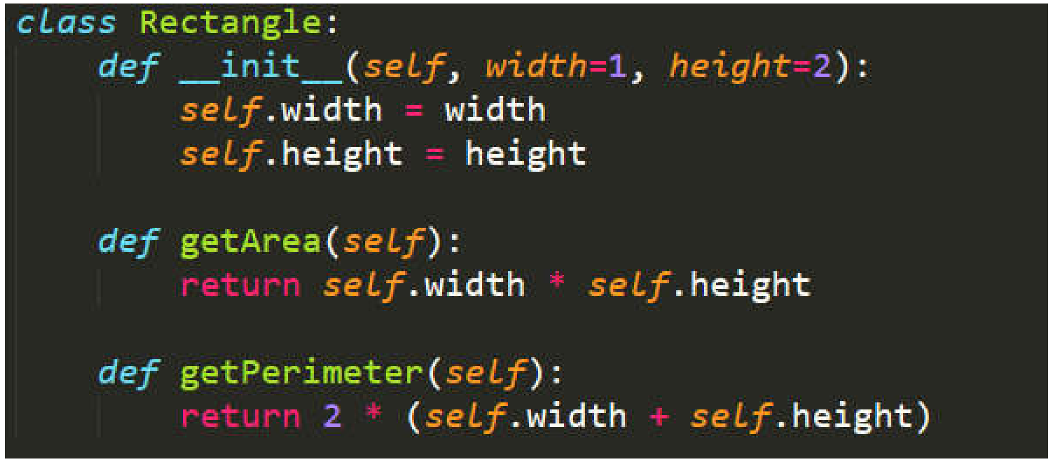
■ A constructor that creates a rectangle with the specified width and height.

The default values are 1 and 2 for the width and height, respectively.

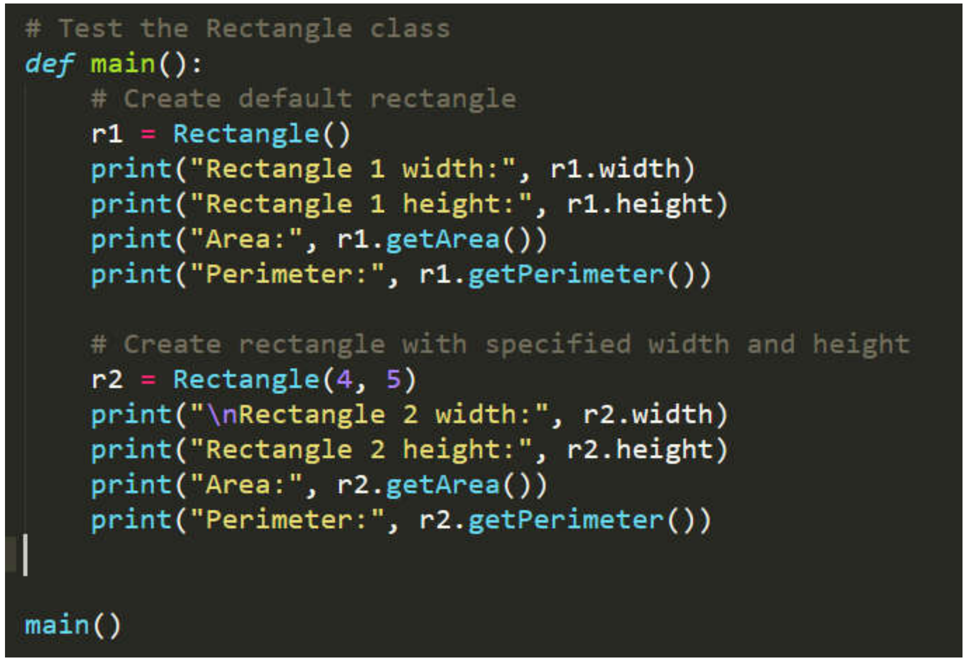
■ A method named getArea() that returns the area of this rectangle.

■ A method named getPerimeter() that returns the perimeter.

Answer



```python
class Rectangle:
    def __init__(self, width=1, height=2):
        self.width = width
        self.height = height

    def getArea(self):
        return self.width * self.height

    def getPerimeter(self):
        return 2 * (self.width + self.height)
```



```python
# Test the Rectangle class
def main():
    # Create default rectangle
    r1 = Rectangle()
    print("Rectangle 1 width:", r1.width)
    print("Rectangle 1 height:", r1.height)
    print("Area:", r1.getArea())
    print("Perimeter:", r1.getPerimeter())

    # Create rectangle with specified width and height
    r2 = Rectangle(4, 5)
    print("\nRectangle 2 width:", r2.width)
    print("Rectangle 2 height:", r2.height)
    print("Area:", r2.getArea())
    print("Perimeter:", r2.getPerimeter())


main()
```

Practice

(The Stock class) Design a class named Stock to represent a company's stock that contains:

■ A private string data field named symbol for the stock's symbol.

■ A private string data field named name for the stock's name.

■ A private float data field named previousClosingPrice that stores the stock price for the previous day.

■ A private float data field named currentPrice that stores the stock price for the current time.

■ A constructor that creates a stock with the specified symbol, name, previous price, and current price.

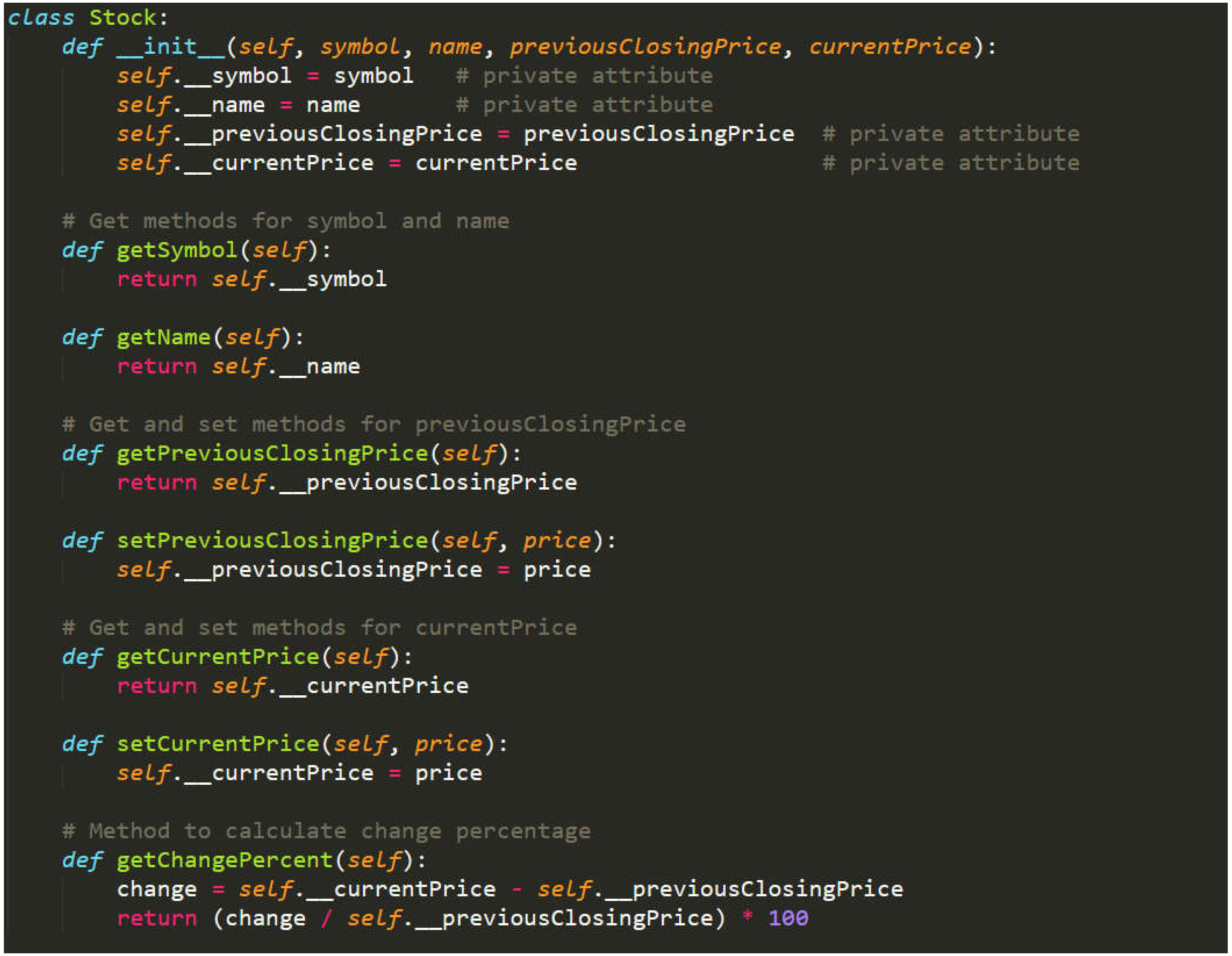
■ A get method for returning the stock name.

■ A get method for returning the stock symbol.

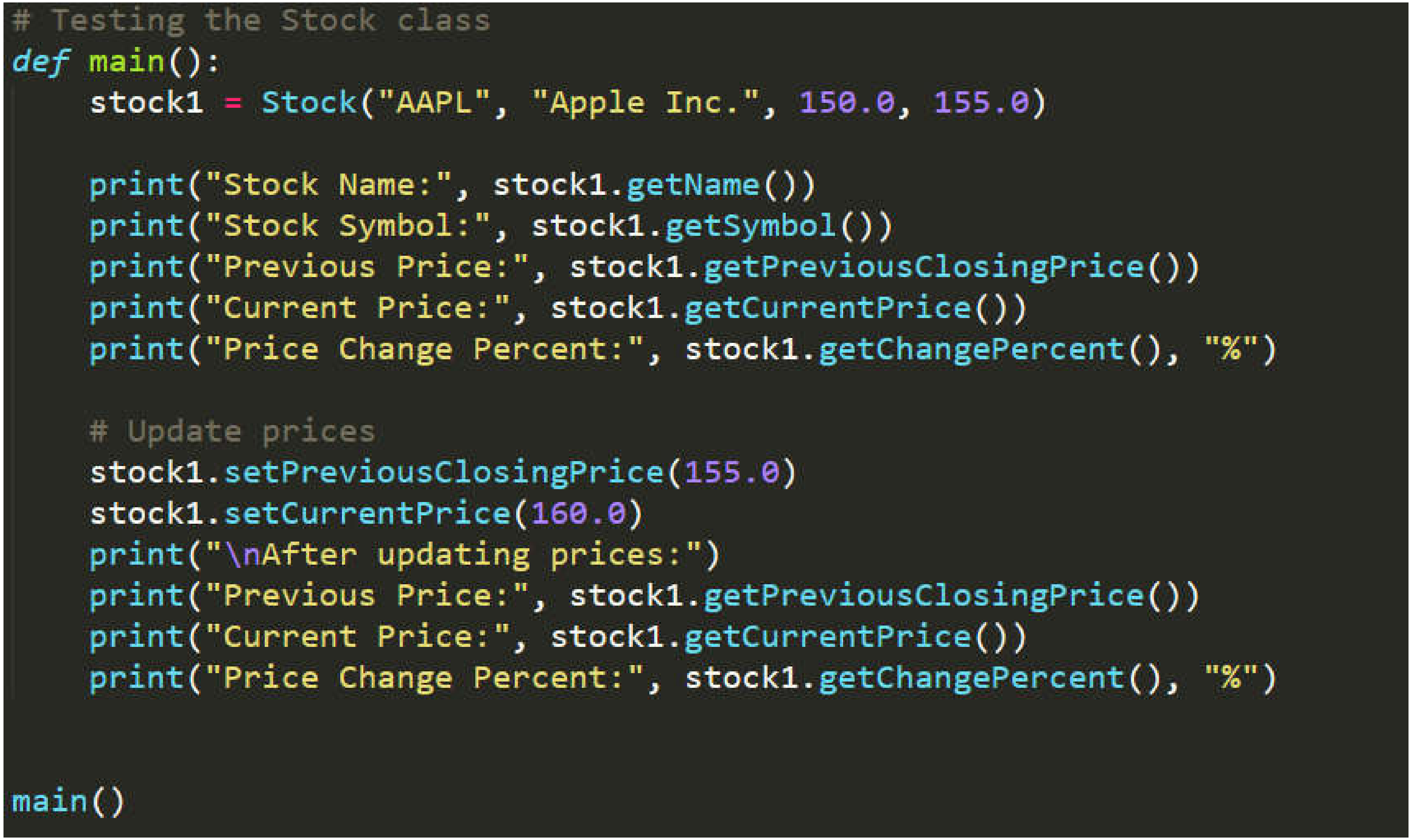
■ Get and set methods for getting/setting the stock's previous price.

■ Get and set methods for getting/setting the stock's current price.

■ A method named getChangePercent() that returns the percentage changed from previousClosingPrice to currentPrice.



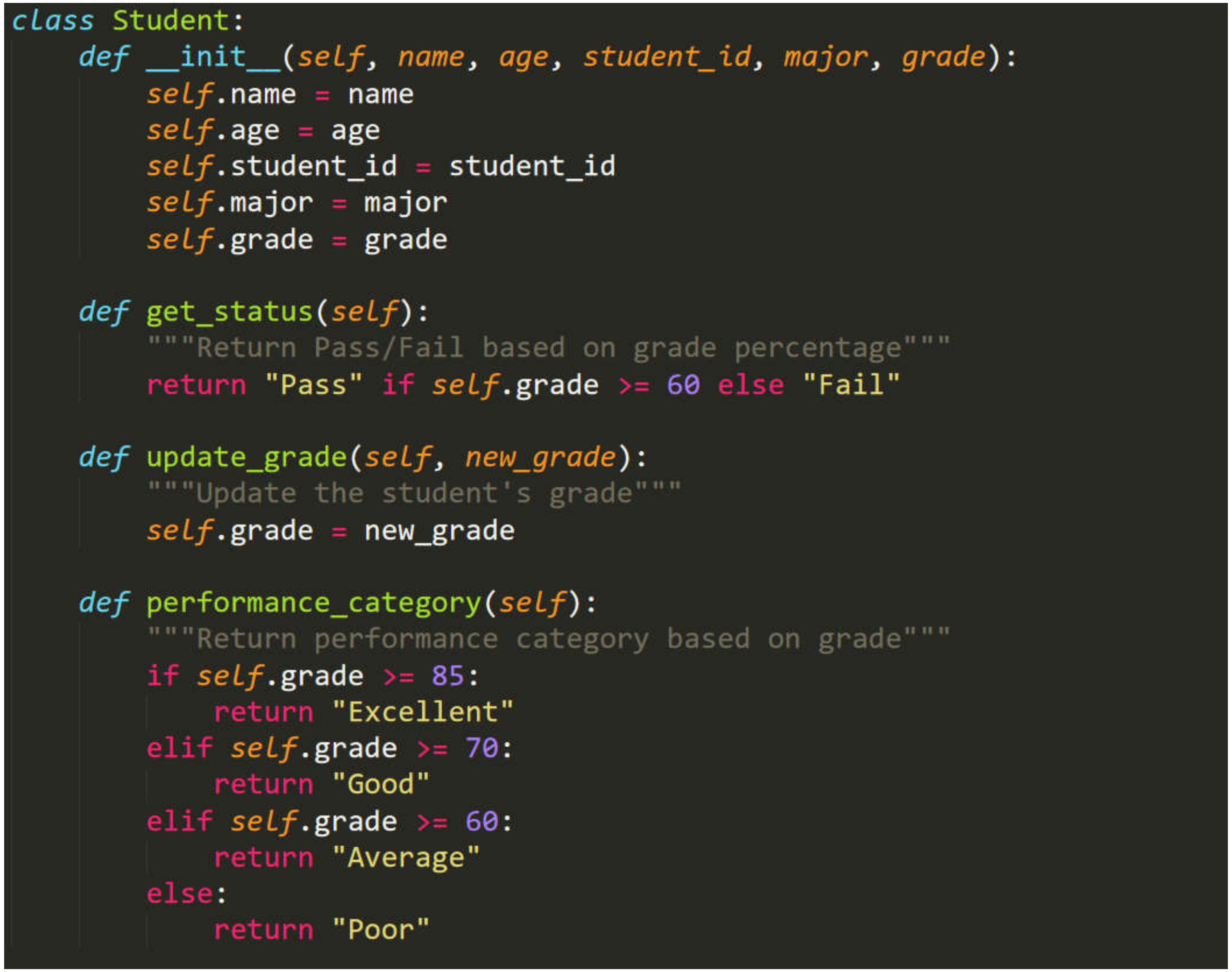
```python
class Stock:
    def __init__(self, symbol, name, previousClosingPrice, currentPrice):
        self.__symbol = symbol    # private attribute
        self.__name = name        # private attribute
        self.__previousClosingPrice = previousClosingPrice  # private attribute
        self.__currentPrice = currentPrice                  # private attribute

    # Get methods for symbol and name
    def getSymbol(self):
        return self.__symbol

    def getName(self):
        return self.__name

    # Get and set methods for previousClosingPrice
    def getPreviousClosingPrice(self):
        return self.__previousClosingPrice

    def setPreviousClosingPrice(self, price):
        self.__previousClosingPrice = price

    # Get and set methods for currentPrice
    def getCurrentPrice(self):
        return self.__currentPrice

    def setCurrentPrice(self, price):
        self.__currentPrice = price

    # Method to calculate change percentage
    def getChangePercent(self):
        change = self.__currentPrice - self.__previousClosingPrice
        return (change / self.__previousClosingPrice) * 100
```



```python
# Testing the Stock class
def main():
    stock1 = Stock("AAPL", "Apple Inc.", 150.0, 155.0)

    print("Stock Name:", stock1.getName())
    print("Stock Symbol:", stock1.getSymbol())
    print("Previous Price:", stock1.getPreviousClosingPrice())
    print("Current Price:", stock1.getCurrentPrice())
    print("Price Change Percent:", stock1.getChangePercent(), "%")

    # Update prices
    stock1.setPreviousClosingPrice(155.0)
    stock1.setCurrentPrice(160.0)
    print("\nAfter updating prices:")
    print("Previous Price:", stock1.getPreviousClosingPrice())
    print("Current Price:", stock1.getCurrentPrice())
    print("Price Change Percent:", stock1.getChangePercent(), "%")


main()
```

Practice

1. Create your own class named Student.
2. Add attributes to store details about each student, such as name, age, grade, ID number, and major.
3. Add methods (operations) to:
    - Check if the student passed or failed based on their grade.
    - Update their grade.
    - Display student details.
    - Classify their performance into categories (Excellent, Good, Average, Poor).



```python
class Student:
    def __init__(self, name, age, student_id, major, grade):
        self.name = name
        self.age = age
        self.student_id = student_id
        self.major = major
        self.grade = grade

    def get_status(self):
        """Return Pass/Fail based on grade percentage"""
        return "Pass" if self.grade >= 60 else "Fail"

    def update_grade(self, new_grade):
        """Update the student's grade"""
        self.grade = new_grade

    def performance_category(self):
        """Return performance category based on grade"""
        if self.grade >= 85:
            return "Excellent"
        elif self.grade >= 70:
            return "Good"
        elif self.grade >= 60:
            return "Average"
        else:
            return "Poor"
```