Exercises

1) Create a House class with address, bedrooms, and bathrooms attributes and a method that prints out the house's address, number of bedrooms, and number of bathrooms.
2) Create a BankAccount class with a balance attribute and methods to deposit, withdraw, and check the balance.
3) Create a Person class with name and age attributes and methods to change the person's name and age.
4) Create a Circle class with a radius attribute and methods to calculate the area and circumference of the circle.
5) Create a Triangle class with base and height attributes and a method to calculate the area of the triangle.
6) Create a Square class with side attribute and methods to calculate the area and perimeter of the square.
7) Create a Dog class with name and breed attributes and a method that prints out the dog's name and breed.
8) Create a Cat class with name and color attributes and a method that prints out the cat's name and color.
9) Create a Student class with name and gpa attributes and a method to calculate the student's letter grade based on their GPA.
10) Create a Bank class with a list of BankAccount objects and methods to add and remove accounts and to print out the account with the highest balance.

Exercises and solution

1) Create a House class with address, bedrooms, and bathrooms attributes and a method that prints out the house's address, number of bedrooms, and number of bathrooms.

```
class House:
    def __init__(self, address, bedrooms, bathrooms):
        self.address = address
        self.bedrooms = bedrooms
        self.bathrooms = bathrooms

    def print_house_info(self):
        print("Address:", self.address)
        print("Number of bedrooms:", self.bedrooms)
        print("Number of bathrooms:", self.bathrooms)
```

2) Create a BankAccount class with a balance attribute and methods to deposit, withdraw, and check the balance.

```
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Insufficient funds")
```

```
    def check_balance(self):
        print("Current balance:", self.balance)
```

3) Create a Person class with name and age attributes and methods to change the person's name and age.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def change_name(self, new_name):
        self.name = new_name

    def change_age(self, new_age):
        self.age = new_age
```

4) Create a Circle class with a radius attribute and methods to calculate the area and circumference of the circle.

```
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return 3.14 * self.radius ** 2

    def calculate_circumference(self):
        return 2 * 3.14 * self.radius
```

5) Create a Triangle class with base and height attributes and a method to calculate the area of the triangle.

```
class Triangle:
    def __init__(self, base, height):
        self.base = base
        self.height = height
```

```
    def calculate_area(self):
        return 0.5 * self.base * self.height
```

6) Create a Square class with side attribute and methods to calculate the area and perimeter of the square.

```
class Square:
    def __init__(self, side):
        self.side = side

    def calculate_area(self):
        return self.side ** 2

    def calculate_perimeter(self):
        return 4 * self.side
```

7) Create a Dog class with name and breed attributes and a method that prints out the dog's name and breed.

```
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def print_info(self):
        print("Name:", self.name)
        print("Breed:", self.breed)
```

8) Create a Cat class with name and color attributes and a method that prints out the cat's name and color.

```
class Cat:
    def __init__(self, name, color):
        self.name = name
        self.color = color
```

```
    def print_info(self):
        print("Name:", self.name)
        print("Color:", self.color)
```

9) Create a Student class with name and gpa attributes and a method to calculate the student's letter grade based on their GPA.

```
class Student:
    def __init__(self, name, gpa):
        self.name = name
        self.gpa = gpa

    def calculate_letter_grade(self):
        if self.gpa >= 4.0:
            return "A+"
        elif self.gpa >= 3.7:
            return "A"
        elif self.gpa >= 3.3:
            return "A-"
        elif self.gpa >= 3.0:
            return "B+"
        elif self.gpa >= 2.7:
            return "B"
        elif self.gpa >= 2.3:
            return "B-"
        elif self.gpa >= 2.0:
            return "C+"
        elif self.gpa >= 1.7:
            return "C"
        elif self.gpa >= 1.3:
            return "C-"
        elif self.gpa >= 1.0:
            return "D"
        else:
            return "F"
```

The Student class has name and gpa attributes. The calculate_letter_grade method returns a letter grade based on the student's GPA, using the standard 4.0 scale. The letter grades are calculated as follows:

A+: 4.0
A: 3.7
A-: 3.3
B+: 3.0
B: 2.7
B-: 2.3
C+: 2.0
C: 1.7
C-: 1.3
D: 1.0
F: 0.0

To test the Student class, you can create a Student object and call the calculate_letter_grade method:

```
student = Student("Alice", 3.8)
print(student.calculate_letter_grade())  # Output: A-
```

This will output "A-", since a GPA of 3.8 corresponds to an A- grade.

10) Create a Bank class with a list of BankAccount objects and methods to add and remove accounts and to print out the account with the highest balance.

```
class Bank:
    def __init__(self):
        self.accounts = []

    def add_account(self, account):
        self.accounts.append(account)

    def remove_account(self, account):
        if account in self.accounts:
```

```
            self.accounts.remove(account)
        else:
            print("Account not found")

    def print_highest_balance(self):
        if len(self.accounts) == 0:
            print("No accounts found")
        else:
            highest_balance = max(account.balance for account in
self.accounts)
            for account in self.accounts:
                if account.balance == highest_balance:
                    print("Account with the highest balance:", account)
                    break
```

The Bank class has a list of BankAccount objects as an attribute. The add_account and remove_account methods allow you to add and remove BankAccount objects from the list. The print_highest_balance method finds the account with the highest balance and prints its information. If there are no accounts in the list, it prints "No accounts found".

To test the Bank class, you can create some BankAccount objects and add them to a Bank object:

```
account1 = BankAccount(1000)
account2 = BankAccount(5000)
account3 = BankAccount(2000)

bank = Bank()
bank.add_account(account1)
bank.add_account(account2)
bank.add_account(account3)

bank.print_highest_balance()
```

This will output:

Account with the highest balance: <__main__.BankAccount object at 0x7f0f3a0d8d30>

Note that the output shows the memory address of the BankAccount object. If you want to print more information about the account, you can add a __str__ method to the BankAccount class, for example:

```
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Insufficient funds")

    def check_balance(self):
        print("Current balance:", self.balance)

    def __str__(self):
        return f"BankAccount(balance={self.balance})"
```

Now if you run the test code again, it will output:

Account with the highest balance: BankAccount(balance=5000)