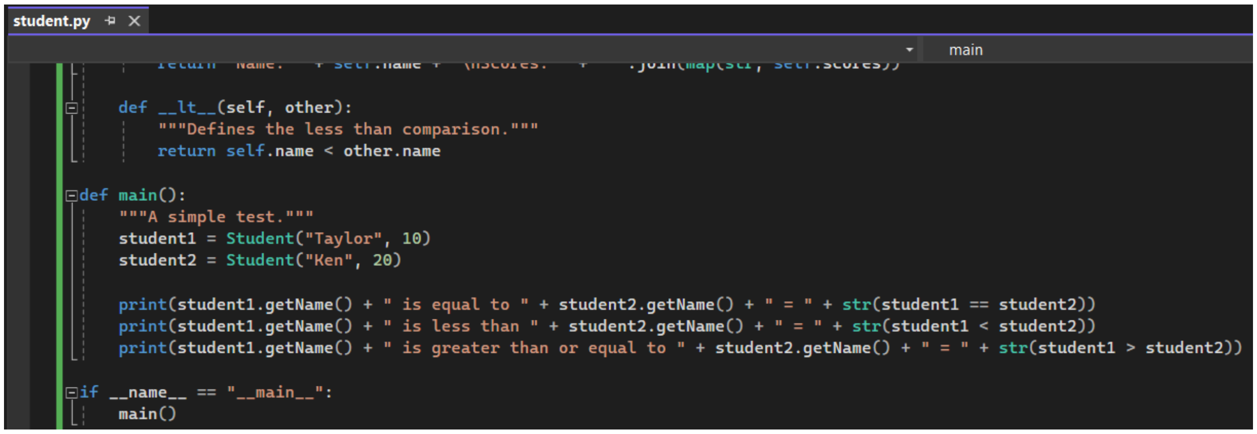
PostLab

Programming Problems

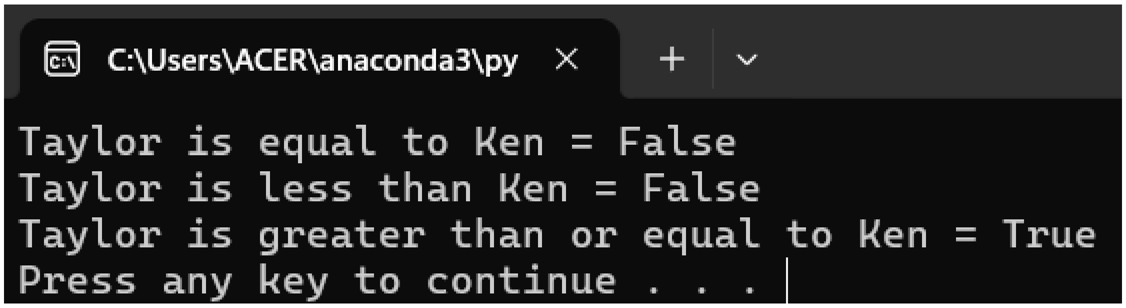
1. Add three methods to the Student class (in the file student.py) that compare two Student objects. One method should test for equality. A second method should test for less than. The third method should test for greater than or equal to. In each case, the method returns the result of the comparison of the two students' names. Include a main function that tests all of the comparison operators. (LO: 10.2). (Torres)



```
        return "Name: " + self.name + "\nScores: " + " ".join(map(str, self.scores))

    def __lt__(self, other):
        """Defines the less than comparison."""
        return self.name < other.name

def main():
    """A simple test."""
    student1 = Student("Taylor", 10)
    student2 = Student("Ken", 20)

    print(student1.getName() + " is equal to " + student2.getName() + " = " + str(student1 == student2))
    print(student1.getName() + " is less than " + student2.getName() + " = " + str(student1 < student2))
    print(student1.getName() + " is greater than or equal to " + student2.getName() + " = " + str(student1 > student2))

if __name__ == "__main__":
    main()
```

Figure 4: Adding Three Methods to Student Class

This code defines a class called Student with a method called __lt__ that compares two Student objects' names alphabetically using the < operator. The primary function creates two Student objects with different names and compares their names using string formatting and comparison operators (==, <, >). The output shows whether the two students have the same name (==), the order of their names (</>), and how to define custom comparison methods for a class based on specific criteria such as alphabetical order by name.

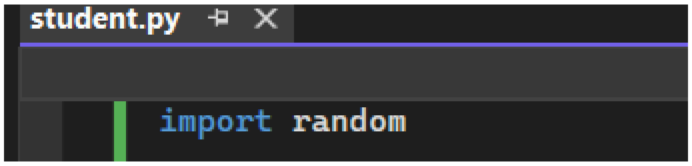


```
Taylor is equal to Ken = False
Taylor is less than Ken = False
Taylor is greater than or equal to Ken = True
Press any key to continue . . .
```

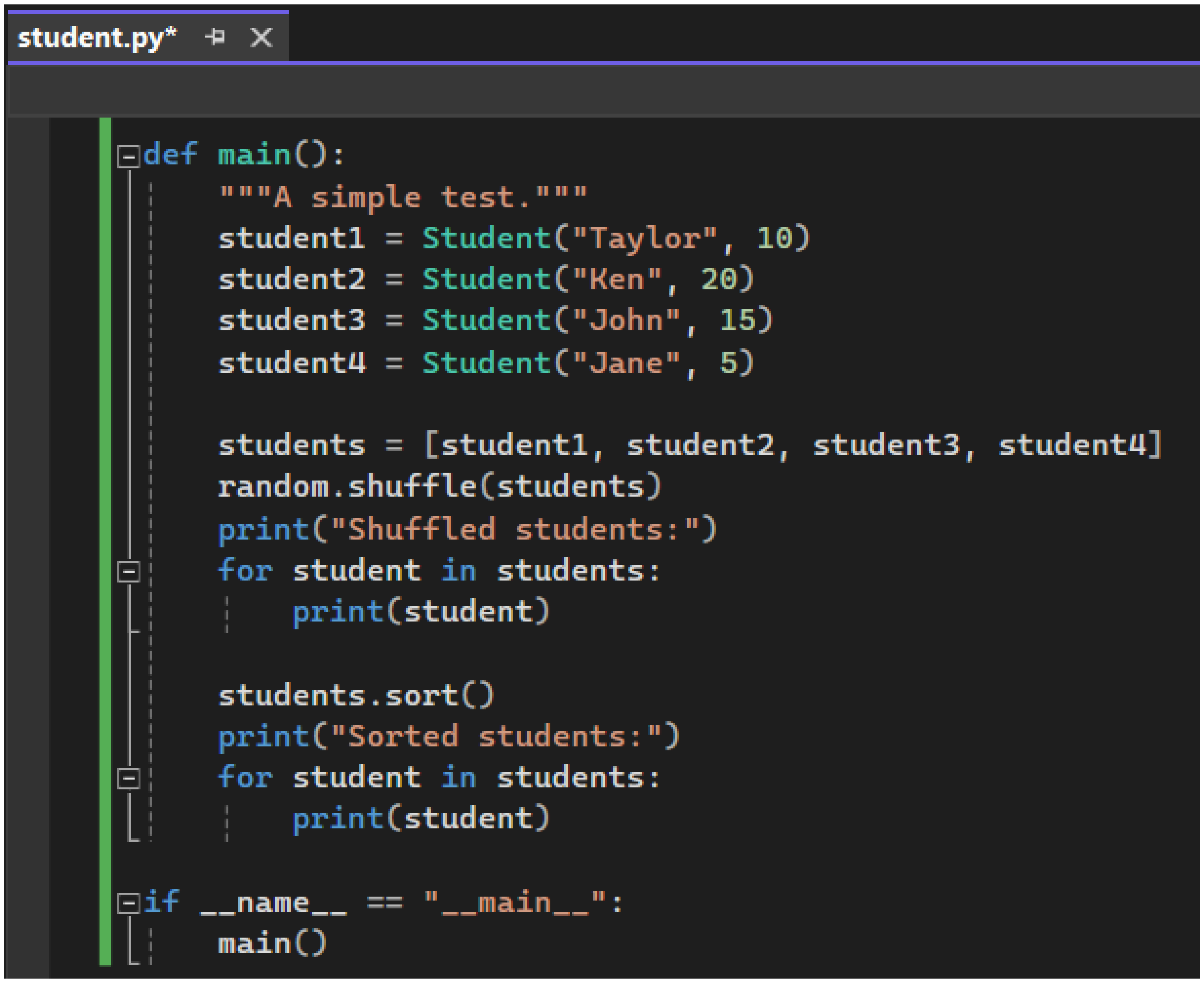
Figure 5: Output of the program

Figure 5 shows that the two names are compared as equal, not equal, and greater than or equal.

2. This exercise assumes that you have completed Programming Exercise 1. Place several Student objects containing different names into a list and shuffle it. Then run the sort method with this list and display all of the students' information. (LO: 10.1, 10.2). (Torres)



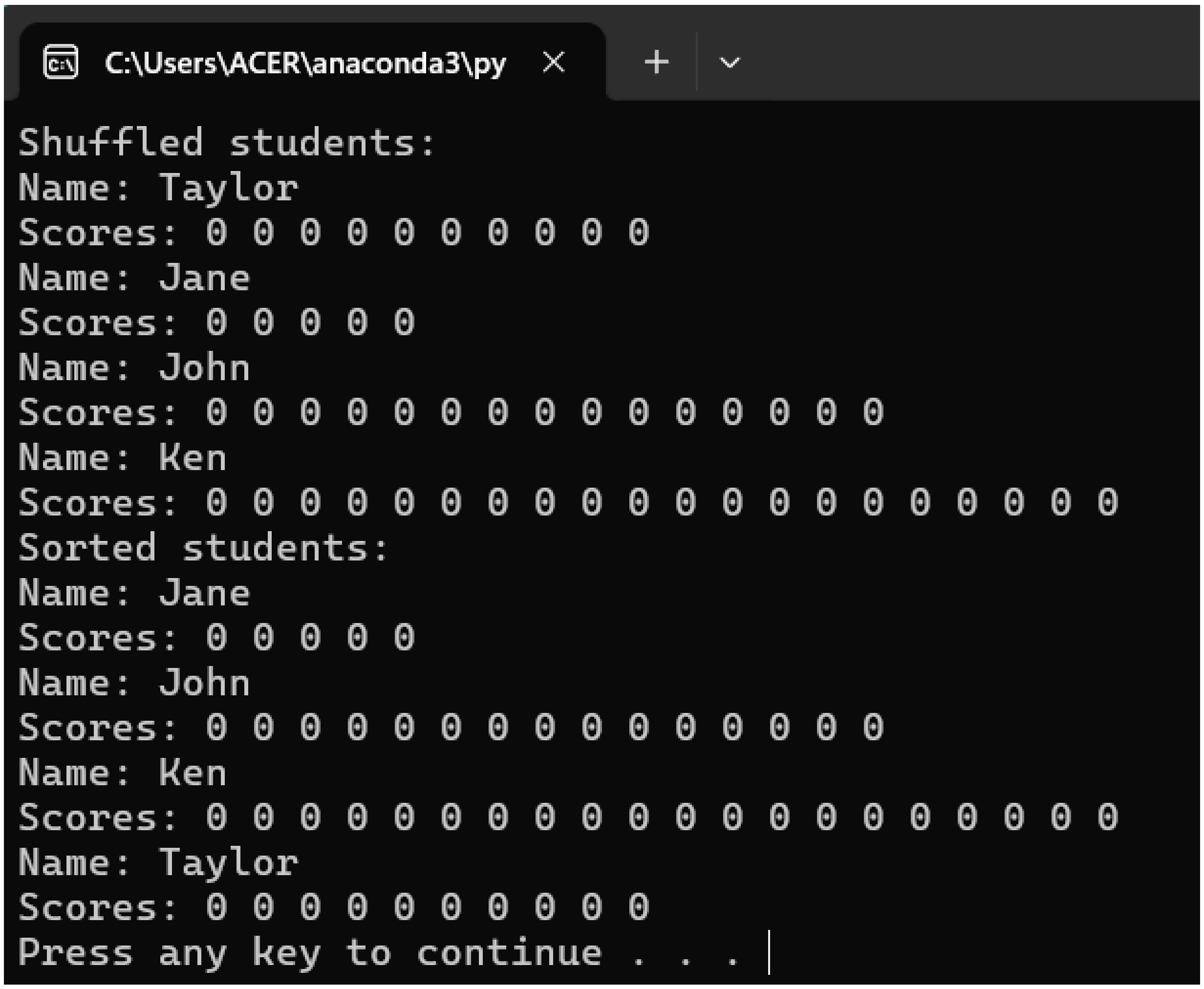
```python
import random
```



```python
def main():
    """A simple test."""
    student1 = Student("Taylor", 10)
    student2 = Student("Ken", 20)
    student3 = Student("John", 15)
    student4 = Student("Jane", 5)

    students = [student1, student2, student3, student4]
    random.shuffle(students)
    print("Shuffled students:")
    for student in students:
        print(student)

    students.sort()
    print("Sorted students:")
    for student in students:
        print(student)

if __name__ == "__main__":
    main()
```

Figure 6: The main function that shuffled and sorted a list of students

The code defines the "main" function to manage a list of four students. I then added the required imports for the random module, enabling shuffling. Then, I created a list called *students, which contains Student objects. Then it uses random.shuffle() to shuffle the list and the default sorting behavior to sort it. It creates instances of the Student class with names and numbers, shuffles the list, sorts it based on student names, and prints both the shuffled and sorted lists. This code helps us learn how to shuffle and sort lists of objects.*



```
Shuffled students:
Name: Taylor
Scores: 0 0 0 0 0 0 0 0 0 0
Name: Jane
Scores: 0 0 0 0 0
Name: John
Scores: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Name: Ken
Scores: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Sorted students:
Name: Jane
Scores: 0 0 0 0 0
Name: John
Scores: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Name: Ken
Scores: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Name: Taylor
Scores: 0 0 0 0 0 0 0 0 0 0
Press any key to continue . . .
```

Figure 6.2: Output of the program

For Figure 6.2, the output shows and proves that the four names are sorted and shuffled.

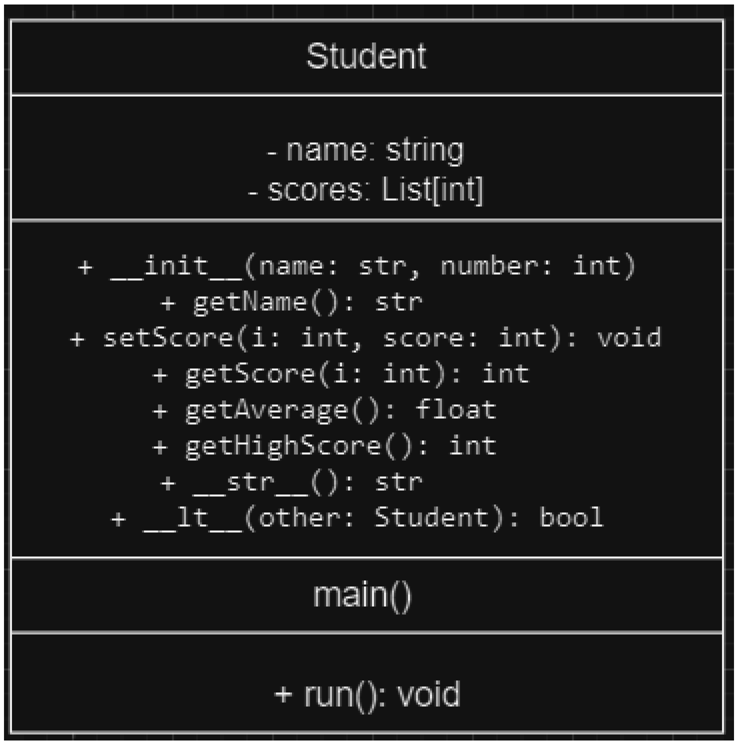


Figure 7: Update UML Diagram with 3 Additional Methods

Figure 7 shows that the student class has been updated, and three methods have been added. The run() method interacts with instances of the Student class (student1, student2, etc.) by creating, shuffling, and sorting them based on their names.

3. The str method of the Bank class (in the file bank.py) returns a string containing the accounts in random order. Design and implement a change that causes the accounts to be placed in the string by order of name. (Hint: You will also have to define some methods in the SavingsAccount class, in the file savingsaccount.py.) (LO: 10.1, 10.2).