Course: Introduction to Data Science (DS2006) - Laboratory 06

Task 1:
The player count is 2 and the scores dictionary only accounts for 2 players. This implementation would not scale as more players are added in the game. Adding a user prompt for single or multiplayer as well as player count would go a long way in optimizing the Battle of Dices for multiplayer.

Task 2:
The main limitation is that the players are hard coded as a list and all the logic is specifically written to handle and account for just 2 players. The dices are rolled for 2 players and the scores are compared between the two players, which is not ideal for a multiplayer mode that could have more than 2 players. Most things are hardcoded specifically for 2 players and they need to be changed to be more dynamic.

Task 3: multiplayer-battle-of-dices.py submitted

Task 4: cooler-multiplayer-battle-of-dices.py submitted

Task 5 - 8: calc.py submitted

Task 9:

```
1   places = ["Malmo", "Halmstad"]
2   places.append("Gotenburg")
3   print(places)
4
5   # Output
6   # ['Malmo', 'Halmstad', 'Gotenburg']
```

The output of figure 1 is a string representation of the places list. All 3 cities are printed out because they are defined before the print function is called.

Task 10:

```
1   places = ["Malmo", "Halmstad"]
2   places.append("Gotenburg")
3   places.insert(0, "Curitiba")
4   print(places)
5
6   # Output
7   # ['Curitiba', 'Malmo', 'Halmstad', 'Gotenburg']
```

Since Curitiba is inserted at index 0 it becomes the first element in the list, which results in Curitiba being the first city that is printed out while Gothenburg is the last. The .append method adds the new element to the end of the list, but using .insert makes it possible to insert the element at the start of the list which is what is shown in figure 2.

Task 11:

```
1   places = ["Malmo", "Halmstad"]
2   places.append("Gotenburg")
3   places.insert(0, "Curitiba")
4   places.append("Curitiba")
5   print(places)
6
7   # Output
8   # ['Curitiba', 'Malmo', 'Halmstad', 'Gotenburg', 'Curitiba']
```

Not much changes in figure 3. The elements largely remain the same except for a duplicate Curitiba that is inserted at the end of the list. The result is an output that shows a list where Curitiba is both the first and last with Malmo, Halmstad and Gothenburg in between.

Task 12:

```
1   places = ["Malmo", "Halmstad"]
2   places.append("Gotenburg")
3   places.insert(0, "Curitiba")
4   places.append("Curitiba")
5   places.remove("Curitiba")
6   print(places)
7
8   # Output
9   # ['Malmo', 'Halmstad', 'Gotenburg', 'Curitiba']
```

The .remove method removes the first occurrence of the element “Curitiba”. The output is therefore the mostly same output as i figure 3 with the only difference being that Curitiba is now only at the end of the list rather than both start and end.

Task 13:

```
1   places = ["Malmo", "Halmstad", "Gotenburg", "Curitiba"]
2   print(places.count("Rio de Janeiro"))
3
4   # Output
5   # 0
```

Figure 5’s intention is to print out the number of times “Rio de Janeiro” is in the list, but because there is not a single element in the list that matches that name, the output shows 0 occurrences.

Task 14:

```
places = ["Malmo", "Halmstad", "Gotenburg", "Curitiba"]
places.clear()
print(places)

# Output
# []
```

Because .clear() is called right before the list is printed the result becomes an empty list since .clear() removed all the elements.

Task 15:

```
places = ["Malmo", "Halmstad", "Gotenburg", "Curitiba"]
places.sort()
print(places)

# Output
# ['Curitiba', 'Gotenburg', 'Halmstad', 'Malmo']
```

The .sort() method in this context is sorting the elements alphabetically, meaning because Curitiba starts with C and C alphabetically comes before H, G and M it is placed at the start of the list. The same logic is applied for each element in the list, and if two elements start with the same letter it checks the next character in the string to determine which comes first. When the sorting is complete the output should look like the above image if the list contains the same elements as in figure 7.

Task 16:visited_places.py submitted