Course: Introduction to Data Science (DS2006) - Laboratory 03

Task 1:
When the code in Figure 1 is run it logs with only one space. Because we are using string concatenation, python treats the log as a single argument, thus not adding an extra space in contrast to Figure 1 where it adds extra spaces between each argument. The str() function used on the player roll variables ensures type safety during the concatenation as long as the player1_roll and player2_roll variables are string convertible data types such as integers or lists.

Task 2:
Figure 3 demonstrates the importance of ensuring type safety in string concatenation. When the function is run it results in a TypeError. This occurs because string concatenation expects the values it concatenates to already be strings. This can be resolved by wrapping the player roll variables in str() function in the way Figure 2 demonstrated.

Task 3:

```
53
54  if player1_roll > player2_roll:
55      print("Player 1 wins this round!")
56      print("Because ", player1_roll," is greater than ", player2_roll)
57  elif player1_roll < player2_roll:
58      print("Player 2 wins this round!")
59      print("Because ", player2_roll," is greater than ", player1_roll
60  else:
61      print("Amaaazzinng! This round has a tie!")
```

Task 4:

```python
import re

def roll_x_dices(x):
    rolls = []
    for i in range(x): rolls.append(random.randint(1, 6))
    return rolls

[player1_roll, player2_roll] = roll_x_dices(2)

players = ["Player 1", "Player 2"]
scores = {}

for player in players:
    scores[player] = 0

winner = ""
round = 0

if player1_roll > player2_roll:
    winner = players[0]
    scores[players[0]] += 1
elif player1_roll < player2_roll:
    winner = players[1]
    scores[players[1]] += 1
else: winner = "Tie"

round += 1

round_str = 'round ' + str(round)

print(f"{ winner + ' wins ' + round_str + '!' if winner != 'Tie' else 'Amaaazzinng! ' + round_str + ' is a tie!' }")
if winner != "Tie": print(f"{ 'Because ' + str(player1_roll) + ' is greater than ' + str(player2_roll) }")
print("Current score:", re.sub(r"{|}|'", "", str(scores)))

# Example Outputs:
# Player 2 wins round 1!
# Because 1 is greater than 4
# Current score: Player 1: 0, Player 2: 1

# Amaaazzinng! round 1 is a tie!
# Current score: Player 1: 0, Player 2: 0
```

✓ 0.0s Python

Task 5:

The execution remains largely the same. This is due to variable += 1 being interpreted the exact same way under the hood in this context as writing variable = variable + 1, with the only real difference being that variable += 1 is faster for a developer to write.

Task 6:

```
51
52  base_win_msg = "Unbelievable! Player X has without a doubt earned the title of Champion of the Battle of Dices! "
53
54  if player1_wins == 3:
55      print(base_win_msg.replace("X", "Player 1"))
56  elif player2_wins == 3:
57      print(base_win_msg.replace("X", "Player 2"))
58  else:
59      print("This heated Battle of Dices is still going on! Who will claim the title of Champion of the Battle of Dices? ")
```

Task 7 & 8: battle-of-dices-bad.py submitted

Task 9:

By creating a reusable function that takes x amount of dice rolls and x sides on the dice, it abstracts away all of the calculations and allows the user to just call the function with those two arguments and get back x amount of roll results where all the rolls use a dice with x amount of sides. This keeps the rolls fair yet allows the user to not need to call the same function x amount of times manually.

At the end of the round, by getting user input on whether they want to play another round, the game can continue on indefinitely until the user explicitly declines another round or terminates the program.

Task 10-11: battle-of-dices-better.py submitted

Task 12-13: Modified dice.py and battle-of-dices-better.py submitted as files

Task 14: battle-of-dices-cooler.py submitted