Dice battle

In this activity, you will develop a two-player dice game. One player is the attacker and the other is the defender.

**Roll** The attacker rolls three dice. The defender rolls two dice.

**Sort** Each player’s dice are sorted in descending order (highest first).

**Check** The attacker’s highest roll is compared to the defender’s highest roll. The player with the smallest roll loses a point. If the two rolls are equal, the attacker loses the point.

**Check** Then, the attacker’s second highest roll is compared to the defender’s second highest roll. The player with the smallest of the two loses a point. If the two rolls are equal, the attacker loses the point.

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | highestroll | Second highest roll |  |  |
| Attacker’s dice (sorted) |  |  |  |  |
| Defender’s dice(sorted) |  |  |  |  |
|  | Defender loses 1 point | Defender loses 1 point |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | highestroll | Second highest roll |  |  |
| Attacker’s dice(sorted) |  |  |  |  |
| Defender’s dice (sorted) |  |  |  |  |
|  | Attacker loses 1 point | Defender loses 1 point |  |  |

 Task .

**Step 1**

**Open** this incomplete program on Bourne to Code, just below where you downloaded this file:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12345678910111213 | from dice import dicerollsattacker\_points = 0defender\_points = 0

|  |  |
| --- | --- |
| attacker = dicerolls(3)defender = dicerolls(2) |  Roll the dice . |

print("Players’ rolls")print("Attacker:", attacker)print("Defender:", defender)

|  |  |
| --- | --- |
|  . . |  Sort players’ rolls . |

print("Sorted")print("Attacker:", attacker)print("Defender:", defender) |

Line 1 imports the dicerolls function from the dice module. This is **not a standard Python component**. It has been created specifically for this activity.

The dicerolls function returns a list containing a specified number of dice rolls. Three dice are rolled for the attacker (line 4) and two for the defender (line 5).

**Step 2**

**Run** the program 2-3 times, to see how different lists of dice rolls are generated each time. The lists will not be sorted yet.

**Step 3**

**Complete line 9** so that the attacker list, containing the attacker’s dice rolls, is sorted.

**Note**: There are a couple of alternative ways to achieve this. For one of them, you may even need an additional line of code.

Make sure you **run** the program and **check** that the attacker’s list of dice rolls is now indeed sorted.

**Step 4**

**Complete line 10** so that the defender list, containing the defender’s dice rolls, is also sorted.

Make sure you **run** the program and **check** that the defender’s list of dice rolls is now indeed sorted.

|  |  |
| --- | --- |
| **Example**  |  |
| Note: This example illustrates how your program should work. The actual output of your program is generated randomly, so the numbers will be different every time you execute it. |
| The program displays the items of the attacker and defender lists, containing their respective dice rolls. | Players’ rollsAttacker: [3, 2, 5]Defender: [2, 5] |
| The program displays the items of the attacker and defender lists again, after they have been sorted. | SortedAttacker: [5, 3, 2]Defender: [5, 2] |

**Step 5**

**Add** an if-statement to your program that compares the highest dice roll in the attacker list to the highest dice roll in the defender list.

If the attacker wins, decrease defender\_points by 1. Otherwise, decrease attacker\_points by 1. Remember that if the highest rolls are equal, the defender wins.

**Tip**: Since the attacker and defender lists have been sorted, you know exactly which item in each list holds the highest dice roll.

**Step 6**

**Add** another if-statement to your program that compares the second highest dice roll in the attacker list to the second highest dice roll in the defender list.

If the attacker wins, decrease defender\_points by 1. Otherwise, decrease attacker\_points by 1. Remember that if the highest rolls are equal, the defender wins.

**Step 7**

**Add** these lines **at the end of your program** to display how the players’ points have been modified after the game.

|  |  |
| --- | --- |
| ++ | print("Attacker points:", attacker\_points)print("Defender points:", defender\_points) |

Make sure you **run** the program and **check** that the points displayed are consistent with the dice rolls.

|  |  |
| --- | --- |
| **Example**  |  |
| Note: This example illustrates how your program should work. The actual output of your program is generated randomly, so the numbers will be different every time you execute it. |
| The program displays the items of the attacker and defender lists, containing their respective dice rolls. | Players’ rollsAttacker: [3, 2, 5]Defender: [2, 5] |
| The program displays the items of the attacker and defender lists again, after they have been sorted. | SortedAttacker: [5, 3, 2]Defender: [5, 2] |
| The program displays each player’s points after the game. | Attacker points: -1Defender points: -1 |

 Explorer task . Again

**Extend** your program so that the attacker and defender start with an initial, positive number of points. The game is repeated for **multiple rounds**, for as long as **both** players have a positive number of points.

When the game ends, your program should check which of the two players still has some points remaining and declare them as the winner.

This resource is licensed under the Open Government Licence, version 3. For more information on this licence, see [ncce.io/ogl](http://ncce.io/ogl).