Computing

Lesson 2: Playlist

Python Programming with Sequences of Data

Rebecca Franks

[1] *Materials from the Teach Computing Curriculum created by the National Centre for Computing Education*

The solar system

Operations on lists

Add and remove items

list`.append(`item`)` e.g. `numbers.append(42)`	Add an item to the end of the list.
list`.insert(`index, item`)` e.g. `cities.insert(2, "Oslo")`	Insert an item at a given position.
list`.pop(`index`)` e.g. `last = values.pop()`	Remove the item at the given position in the list, and return it. If no index is specified, remove and return the last item in the list.
list`.remove(`item`)` e.g. `countries.remove("Japan")`	Remove the first item from the list with a particular value. Raises a `ValueError` if there is no such item.

Operations on lists

Find and count items

list`.index(`item`)` e.g. `pos = planets.index("Mars")`	Search for the first occurrence of an item in the list and return its (zero-based) index. Raises a `ValueError` if there is no such item.
list`.count(`item`)` e.g. `nb_the = words.count("the")`	Return the number of times an item appears in the list.

Operations on lists

Modify list

list`.reverse()` e.g. `values.reverse()`	Reverse the items of the list.
list`.sort()` e.g. `names.sort()` e.g. `numbers.sort(reverse=True)`	Sort the items of the list. Use the `reverse=True` argument to sort in **descending order**.

Worked Example Third rock from the sun

This program searches for the index of Earth in the list of planets.

```
planets = ["Mercury", "Venus",
           "Earth", "Mars",
           "Jupiter", "Saturn",
           "Uranus", "Neptune"]
position = planets.index("Earth") + 1
print("Earth is planet number", position)
```

Here are the contents of the planets list, with an index next to each item:

0	1	2	3	4	5	6	7
"Mercury"	"Venus"	"Earth"	"Mars"	"Jupiter"	"Saturn"	"Uranus"	"Neptune"

Task 1 Planets

Mercury, Venus, Mars, Jupiter, and Saturn are visible from Earth. These planets have been known since antiquity.

By 1930, astronomers had added Uranus (1781), Neptune (1846), and Pluto (1930) to the list of known planets.

A score of discoveries in the early 2000s led to the demotion of Pluto; since 2006 it is no longer considered a planet and is classified instead as a dwarf planet.

In this task, you will create a program that recounts this short story about our view of the planets in our solar system.

Task 1   Planets

Step 1

Open this incomplete program **oaknat.uk/comp-py-solar-1** in Repl.it:

```
planets = ["Mercury", "Venus",
           "Earth", "Mars",
           "Jupiter", "Saturn"]
print("Planets in antiquity:")
print(planets)
                       # Add Uranus to the list
                       # Add Neptune to the list
                       # Add Pluto to the list
print("Planets by 1930:")
print(planets)
                       # Remove Pluto from the list
print("Planets after 2006:")
print(planets)
```

Challenge   Steps 2 to 5

Step 2

Complete line 6, so that "`Uranus`" is added to the end of the list of planets.

Tip: Consult the 'Operations on lists' section at the beginning of this worksheet to find out how to add an item to the end of a list.

Step 3

Run the program to make sure that `Uranus` is included in the list of planets "`known by 1930`"

Example

Note: Use this example to check your program.

This should be part of the program's output: check that the items and their order are correct.

```
Planets by 1930:
['Mercury', 'Venus', 'Earth', 'Mars',
'Jupiter', 'Saturn', 'Uranus']
```

Challenge Steps 2 to 5

Step 4

Complete line 7 and 8, so that "`Neptune`" and "`Pluto`"are also added to the end of the list of planets.

Step 5

Run the program, to make sure that Neptune and Pluto are included in the list of planets "`known by 1930`".

Example

Note: Use this example to check your program.

This should be part of the program's output: check that the items and their order are correct.	`Planets by 1930:`
	`['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune', 'Pluto']`

Challenge Steps 6 to 7

Step 6

Complete line 11, so that "`Pluto`" is removed from the list of planets.

Tip: Consult the ‘Operations on lists’ section at the beginning of this worksheet to find out how to remove an item from a list.

Step 7

Run the program, to make sure that Pluto is not included in the list of planets “`after 2006`”.

Example

Note: Use this example to check your program.

This should be part of the program’s output: check that the items and their order are correct.

```
Planets after 2006:
['Mercury', 'Venus', 'Earth', 'Mars',
'Jupiter', 'Saturn', 'Uranus',
'Neptune']
```

Task 2 Dwarf planets

In 2006, astronomers set out rules that would classify certain objects in the solar system as dwarf planets. At present there are five such objects (but this list is subject to change): Ceres, Pluto, Haumea, Eris, and Makemake, ordered according to their date of discovery.

There was controversy around which team the discovery of Haumea should be attributed to. The name originally proposed for it was Ataecina.

In this task, you will complete a program that displays the list of dwarf planets.

Task 2 .Dwarf planets

Step 1

Open this incomplete program **oaknat.uk/comp-py-solar-2** in Repl.it:

```
dwarves =                     # Create list of dwarves
                              # Change 2nd item to Haumea
                              # Add Pluto as 2nd item
print("Dwarf planets:")
print(dwarves)
```

Challenge   Steps 2 and 3

Step 2

Complete line 1, so that the list of dwarf planets comprises Ceres, Ataecina, Eris, and Makemake, in that order.

Tip: Check how the list of planets is created in the first task.

Step 3

Run the program to make sure that the list of dwarf planets is correct.

Example

Note: Use this example to check your program.

This should be part of the program's output: check that the items and their order are correct.

```
Dwarf planets:
['Ceres', 'Ataecina', 'Eris',
'Makemake']
```

Challenge   Steps 4 and 5

Step 4

Complete line 2, so that the second item in the list is assigned the value of "`Haumea`".

Tip: You will need to assign the new value to the second item of the dwarves list.

Tip: Item numbering starts from 0, so the second item has an index of 1.

Step 5

Run the program to make sure that the second item in the list is Haumea, instead of Ataecina.

Example

Note: Use this example to check your program.

This should be part of the program's output: check that the items and their order are correct.	`Dwarf planets:`
	`['Ceres', 'Haumea', 'Eris', 'Makemake']`

Challenge Steps 6 and 7

Step 6

Complete line 3, so that "`Pluto`" is added to the list, as its second item.

Tip: Item numbering starts from 0, so the second item has an index of 1.

Step 7

Run the program to make sure that Pluto is now the second item in the list, preceded by Ceres and followed by Haumea, Eris, and Makemake, in that order.

Example

Note: Use this example to check your program.

This should be part of the program's output: check that the items and their order are correct.	`Dwarf planets:`
	`['Ceres', 'Pluto', 'Haumea', 'Eris',`
	`'Makemake']`

Dice battle game

Dice battle   Introduction

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

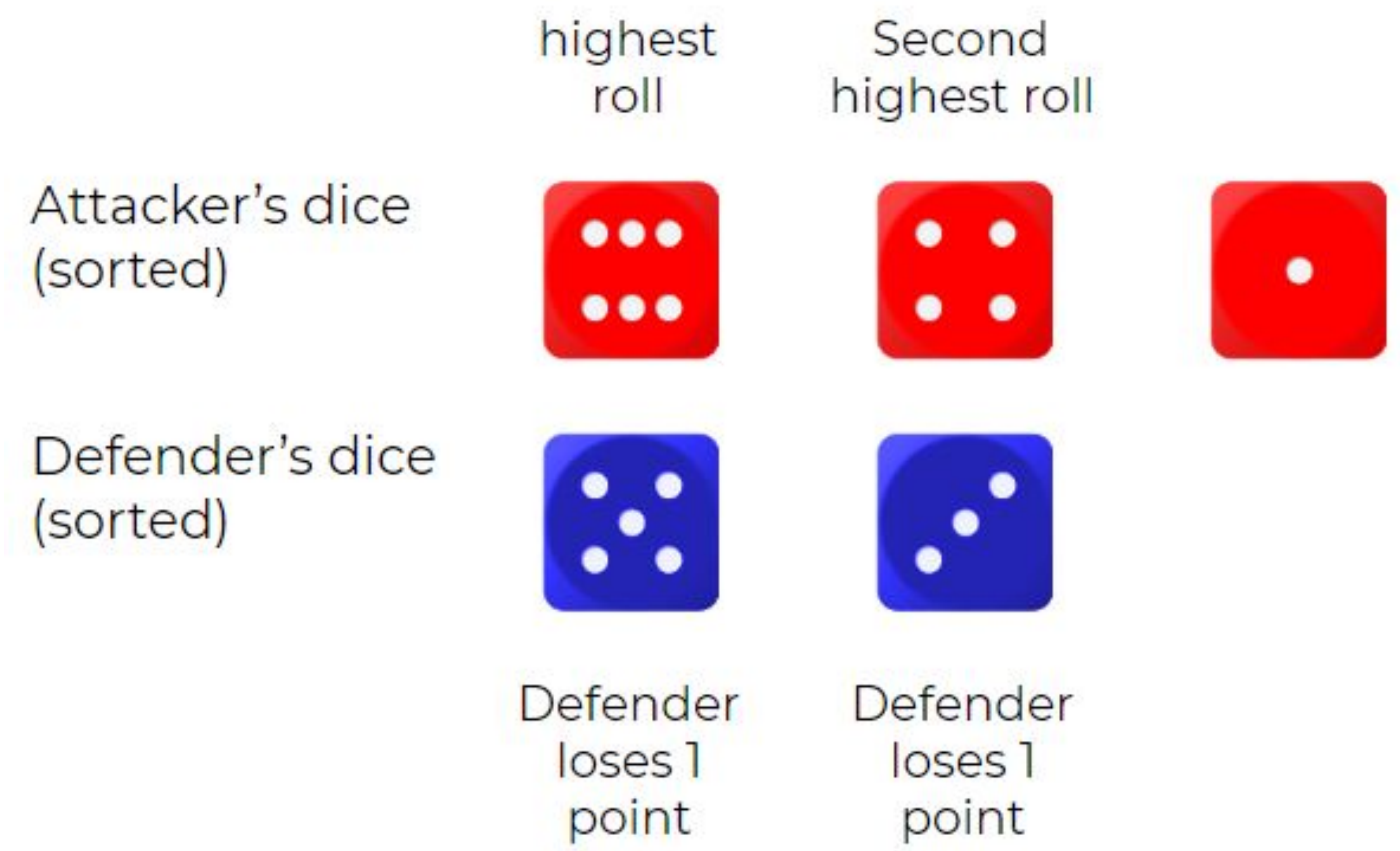
Roll The attacker rolls three dice. The defender rolls two dice.

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

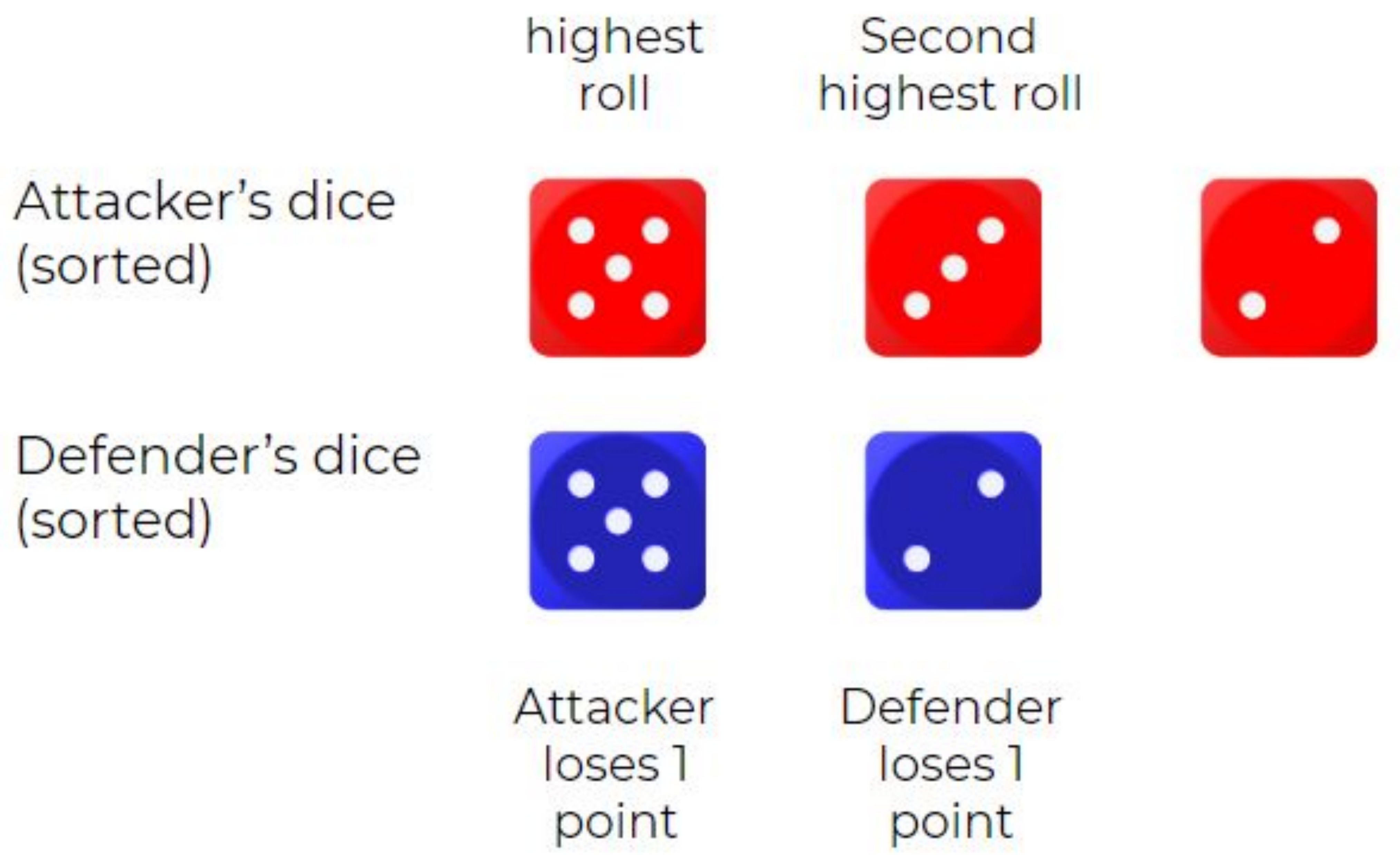
Check The attacker's highest roll is compared to the defender's highest roll. The player with the smallest roll loses a point. If the two rolls are equal, the attacker loses the point.

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.

Example 1

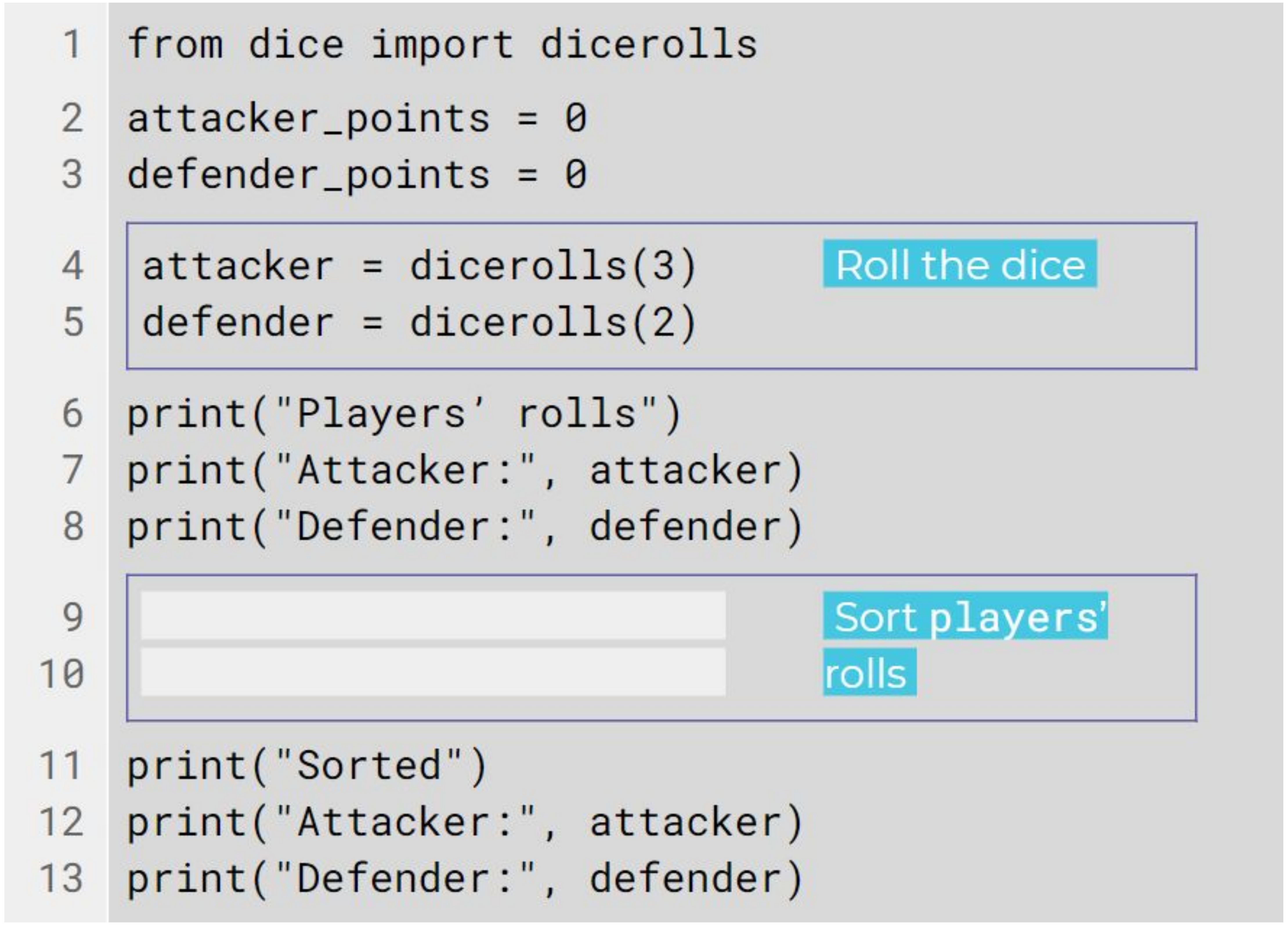

highest roll
Second highest roll
Attacker's dice (sorted)
Defender's dice (sorted)
Defender loses 1 point
Defender loses 1 point

Example 2


highest roll
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 **oaknat.uk/comp-py-battle-1** in Repl.it:



```
from dice import dicerolls
attacker_points = 0
defender_points = 0
attacker = dicerolls(3)
defender = dicerolls(2)
print("Players' rolls")
print("Attacker:", attacker)
print("Defender:", defender)


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

Task

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.

Task

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' rolls
Attacker: [3, 2, 5]
Defender: [2, 5]
```

The program displays the items of the `attacker` and `defender` lists again, after they have been sorted.

```
Sorted
Attacker: [5, 3, 2]
Defender: [5, 2]
```

Task

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.

Task

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 inputs and outputs can be seen on the next slide.

Task

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’ rolls
Attacker: [3, 2, 5]
Defender: [2, 5]
```

The program displays the items of the `attacker` and `defender` lists again, after they have been sorted.

```
Sorted
Attacker: [5, 3, 2]
Defender: [5, 2]
```

The program displays each player’s points after the game.

```
Attacker points: -1
Defender points: -1
```

Explorer Task (optional)

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.