ILP 2023 – W3S1
For loops iterations, generators

Matthieu DE MARI – Singapore University of Technology and Design

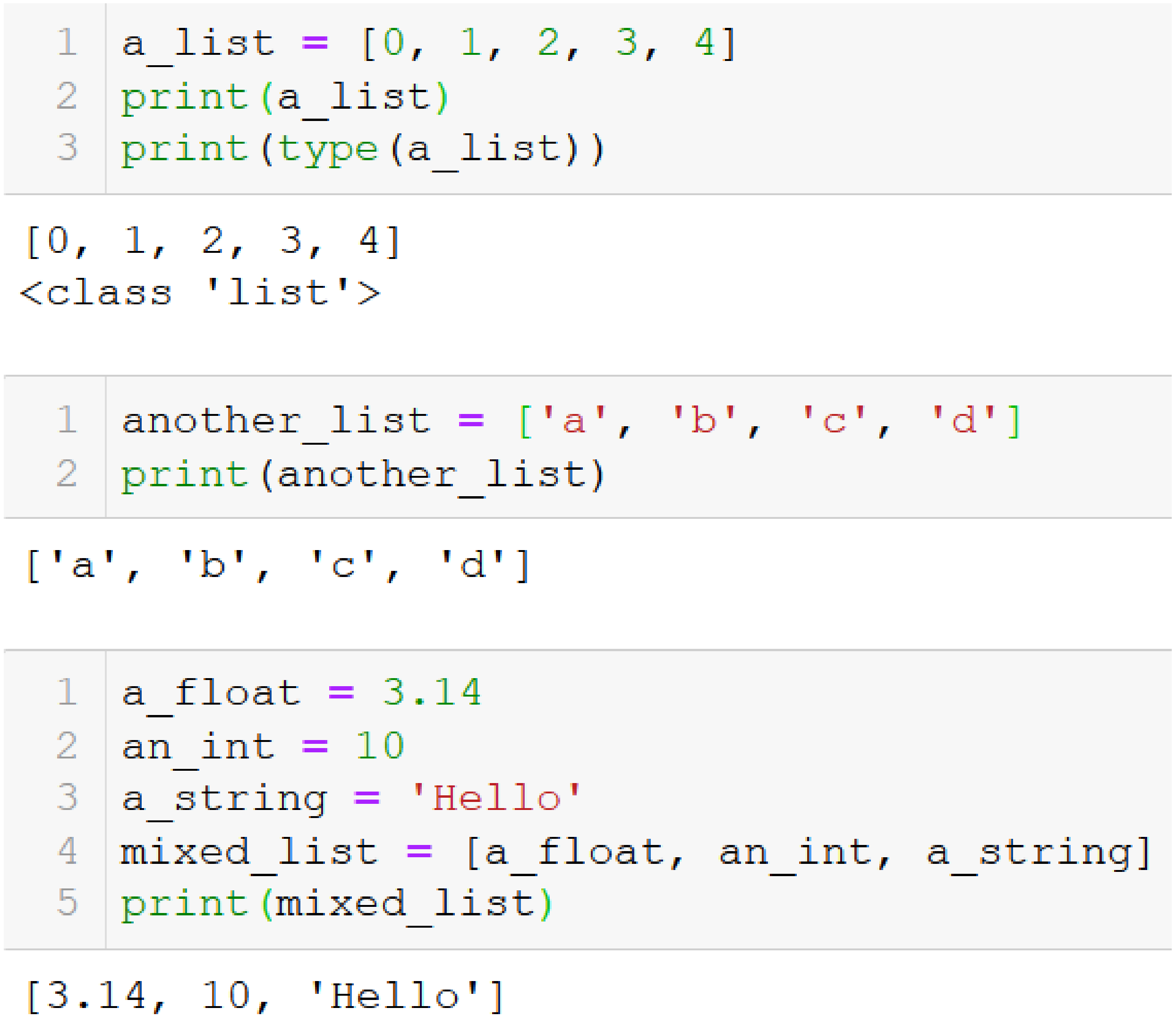
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Outline (Week3, Session1 – W3S1)

- The list type (quick intro, much more to come on W3)
- The for statement
- The range() generator
- The enumerate() generator
- The zip() generator
- Nesting for loops
- Breaking for loops

The list type

- Let us first introduce a new type of objects, called **lists**.
- **Definition (lists):** a list is a **sequence** of several variable elements, listed in order, between **brackets** and separated by **commas**.
- It can contain variables of any types (int, float, string, etc.).
- List can also contain mixed types of variables.



```python
a_list = [0, 1, 2, 3, 4]
print(a_list)
print(type(a_list))
```

```
[0, 1, 2, 3, 4]
<class 'list'>
```

```python
another_list = ['a', 'b', 'c', 'd']
print(another_list)
```

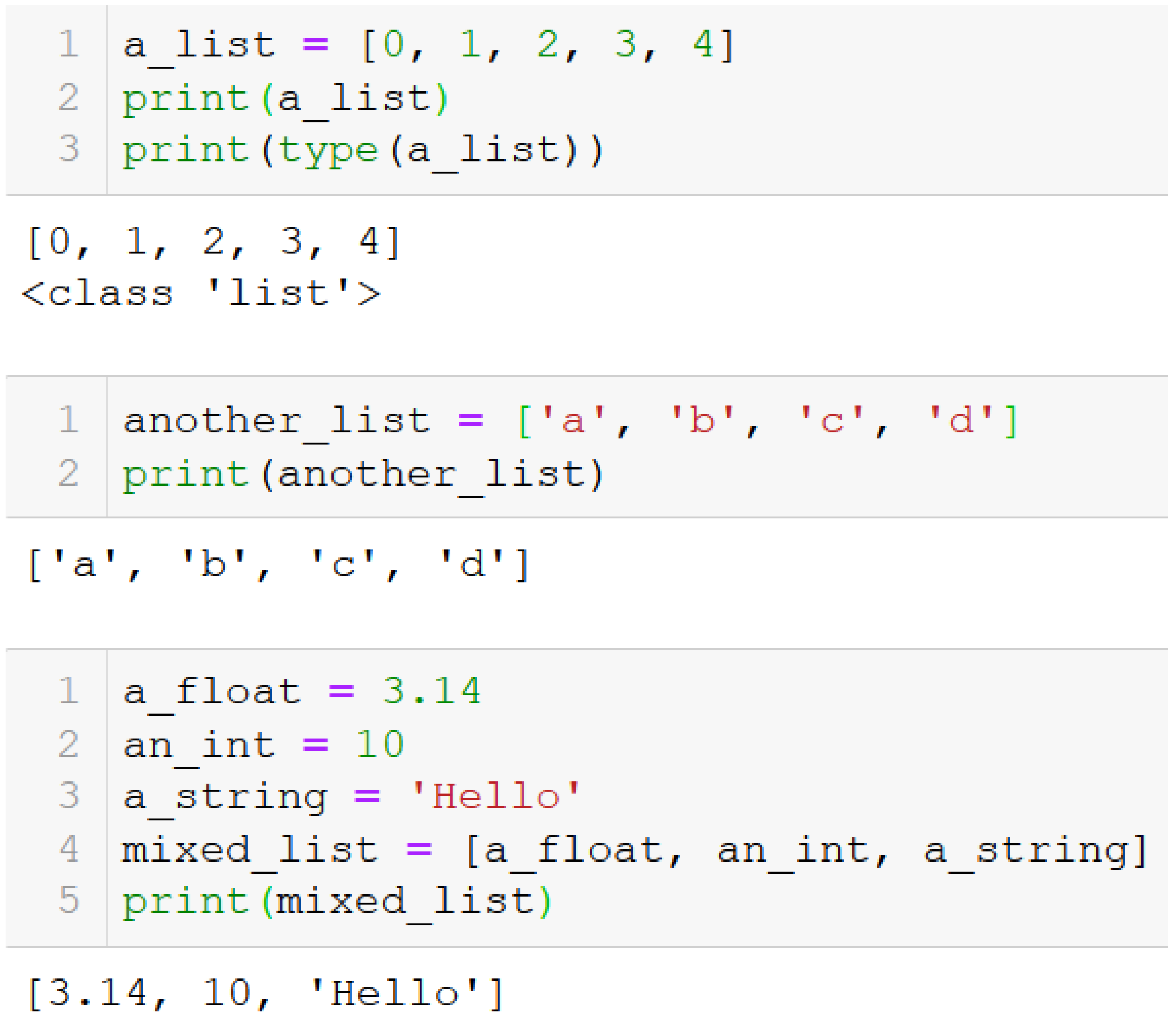
```
['a', 'b', 'c', 'd']
```

```python
a_float = 3.14
an_int = 10
a_string = 'Hello'
mixed_list = [a_float, an_int, a_string]
print(mixed_list)
```

```
[3.14, 10, 'Hello']
```

The list type

We will learn more about lists on the next two sessions!



```python
a_list = [0, 1, 2, 3, 4]
print(a_list)
print(type(a_list))
```

```
[0, 1, 2, 3, 4]
<class 'list'>
```

```python
another_list = ['a', 'b', 'c', 'd']
print(another_list)
```

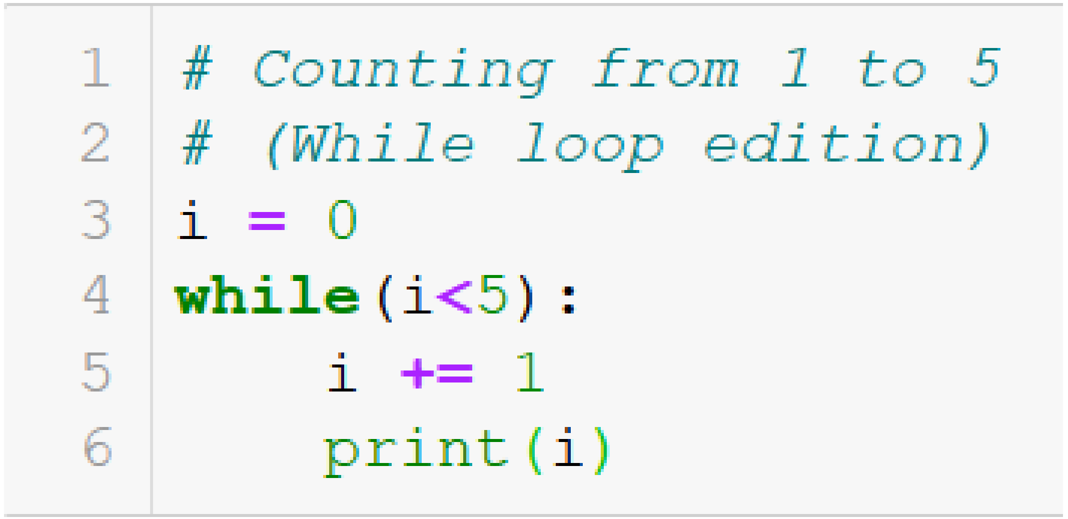
```
['a', 'b', 'c', 'd']
```

```python
a_float = 3.14
an_int = 10
a_string = 'Hello'
mixed_list = [a_float, an_int, a_string]
print(mixed_list)
```

```
[3.14, 10, 'Hello']
```

The for statement

- Sometimes in programming, there is a block of code that you want to repeat **for a fixed number of times**.
- It could be done with a **while** statement, but there is a more convenient way.
- **More convenient way:** the **for** statement is used to repeat a given block of code **for** a given number of times.



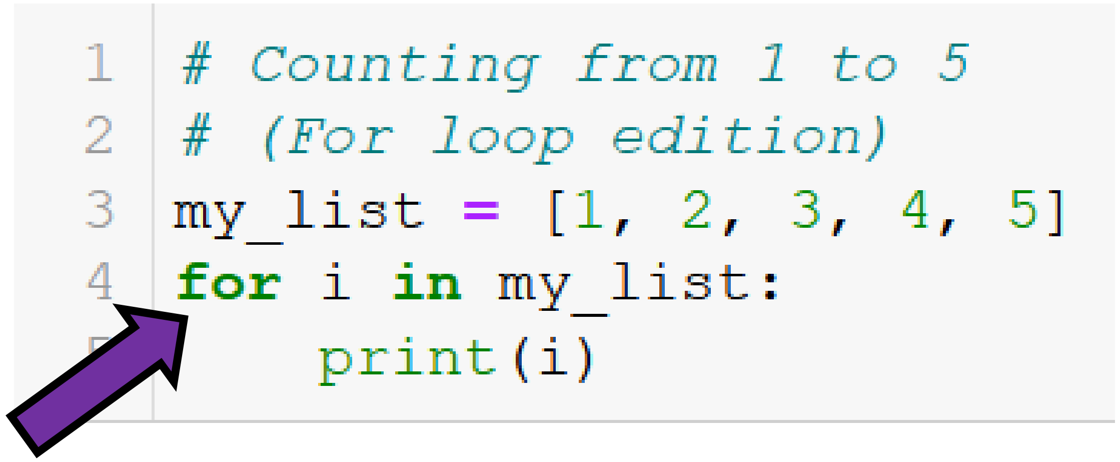
```python
# Counting from 1 to 5
# (While loop edition)
i = 0
while(i<5):
    i += 1
    print(i)
```

```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,



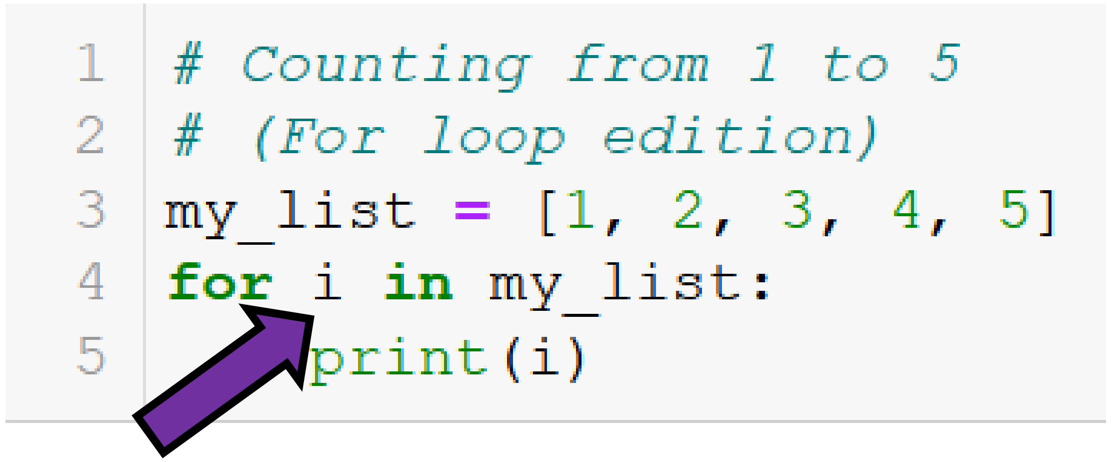
```python
# Counting from 1 to 5
# (For loop edition)
my_list = [1, 2, 3, 4, 5]
for i in my_list:
    print(i)
```

```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,



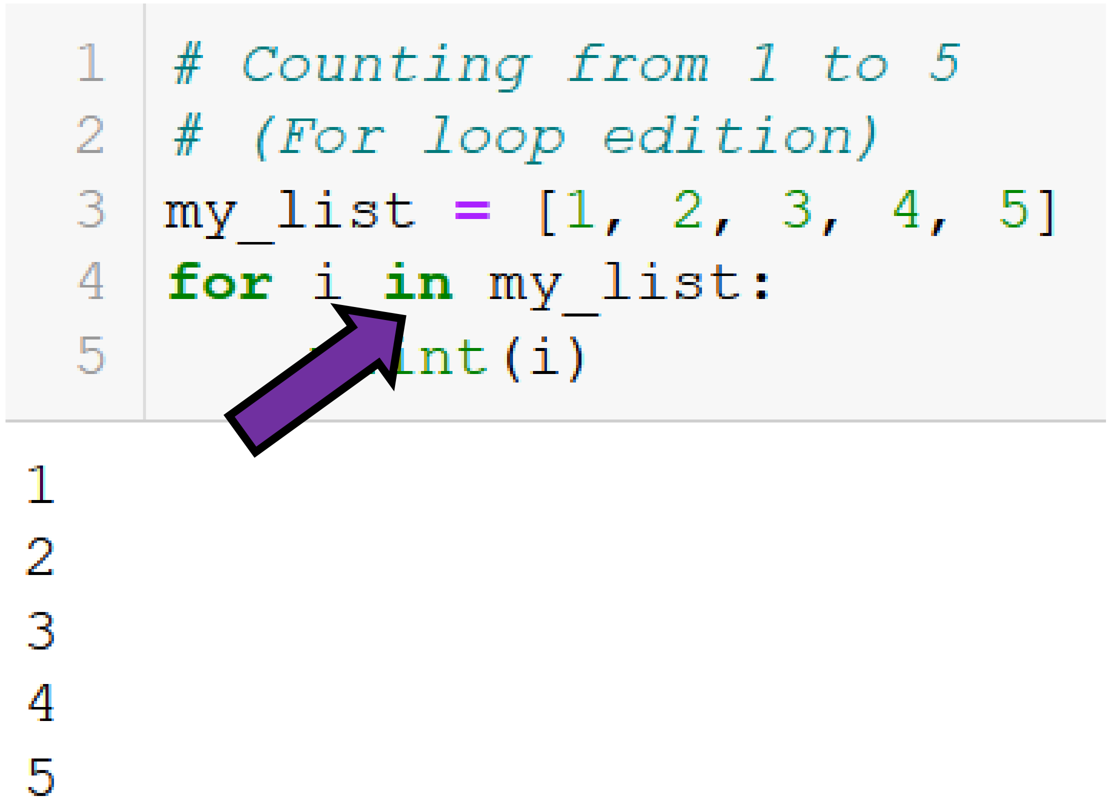
```python
# Counting from 1 to 5
# (For loop edition)
my_list = [1, 2, 3, 4, 5]
for i in my_list:
    print(i)
```

```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,
- Use the in keyword to indicate that the **iteration variable** will take values in a given **list**,



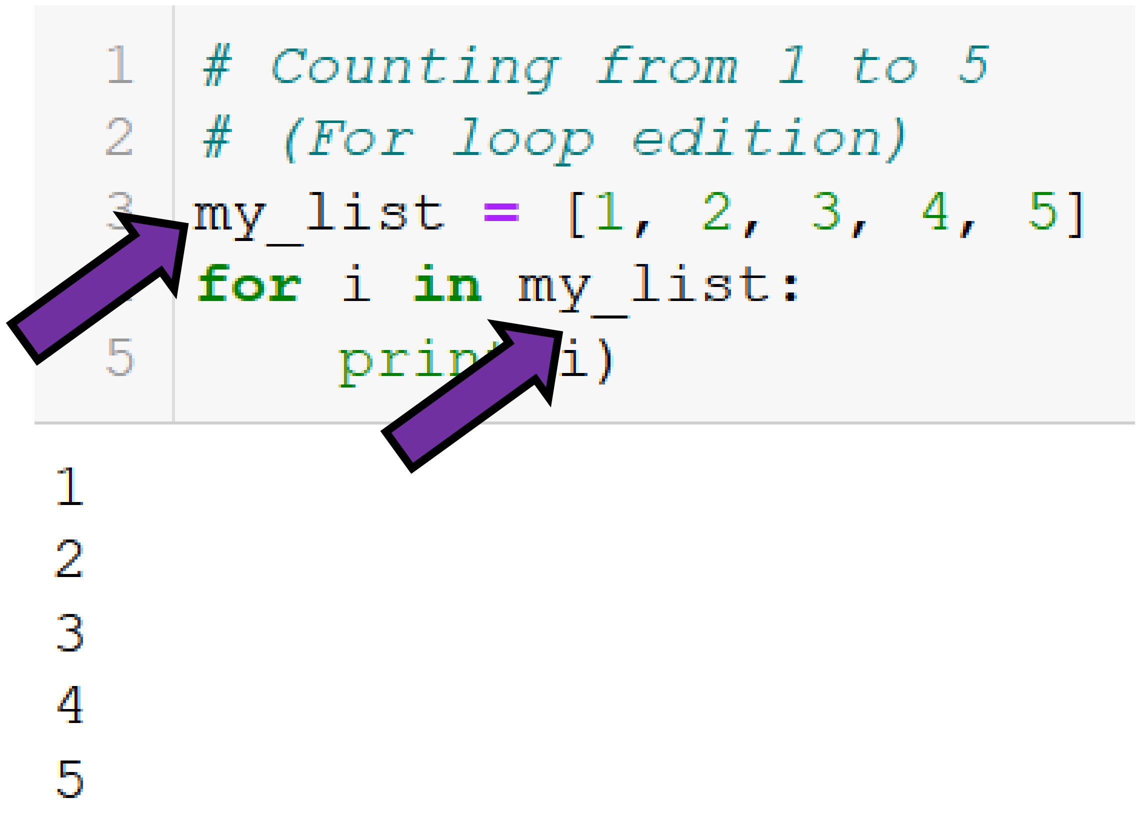
```python
# Counting from 1 to 5
# (For loop edition)
my_list = [1, 2, 3, 4, 5]
for i in my_list:
    print(i)
```

```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,
- Use the in keyword to indicate that the **iteration variable** will take values in a given **list**,
- Provide a **list** object, finish with a **: symbol.**



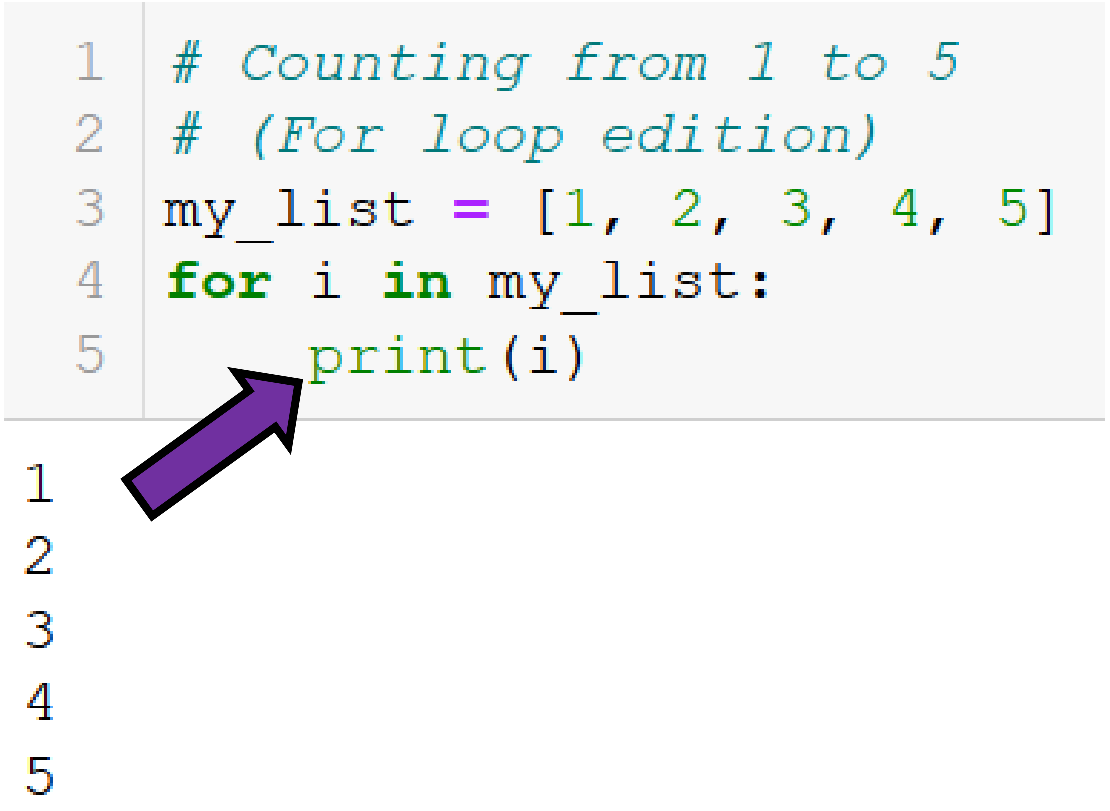
```
# Counting from 1 to 5
# (For loop edition)
my_list = [1, 2, 3, 4, 5]
for i in my_list:
    print(i)
```

```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,
- Use the in keyword to indicate that the **iteration variable** will take values in a given **list**,
- Provide a **list** object, finish with a **: symbol.**
- **Indent some** code to be repeated inside the for, as in if/while.



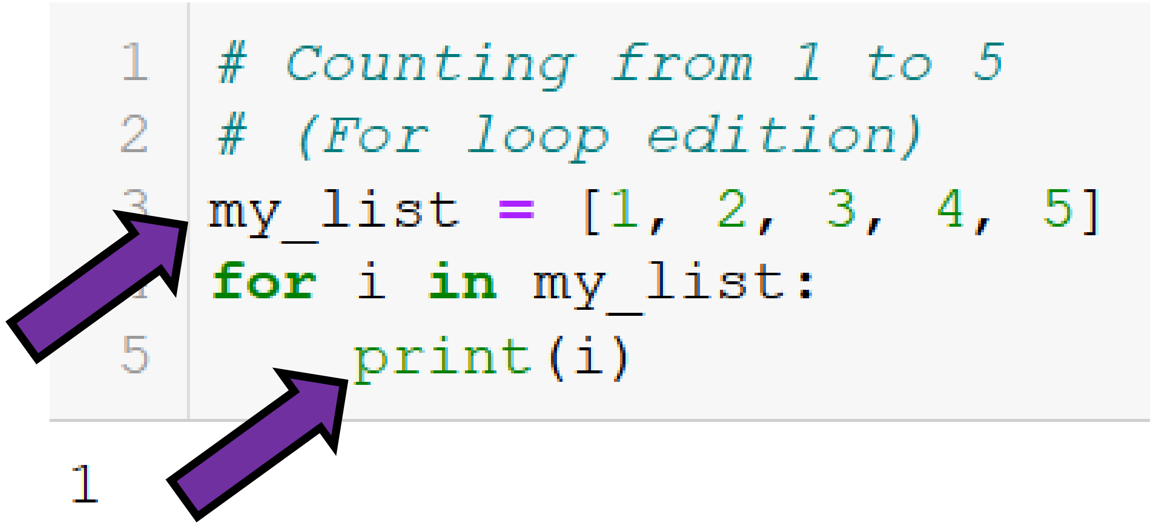
```python
# Counting from 1 to 5
# (For loop edition)
my_list = [1, 2, 3, 4, 5]
for i in my_list:
    print(i)
```

```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,
- Use the in keyword to indicate that the **iteration variable** will take values in a given **list**,
- Provide a **list** object, finish with a **: symbol.**
- **Indent some** code to be repeated inside the for, as in **if/while**.



```python
# Counting from 1 to 5
# (For loop edition)
my_list = [1, 2, 3, 4, 5]
for i in my_list:
    print(i)
```

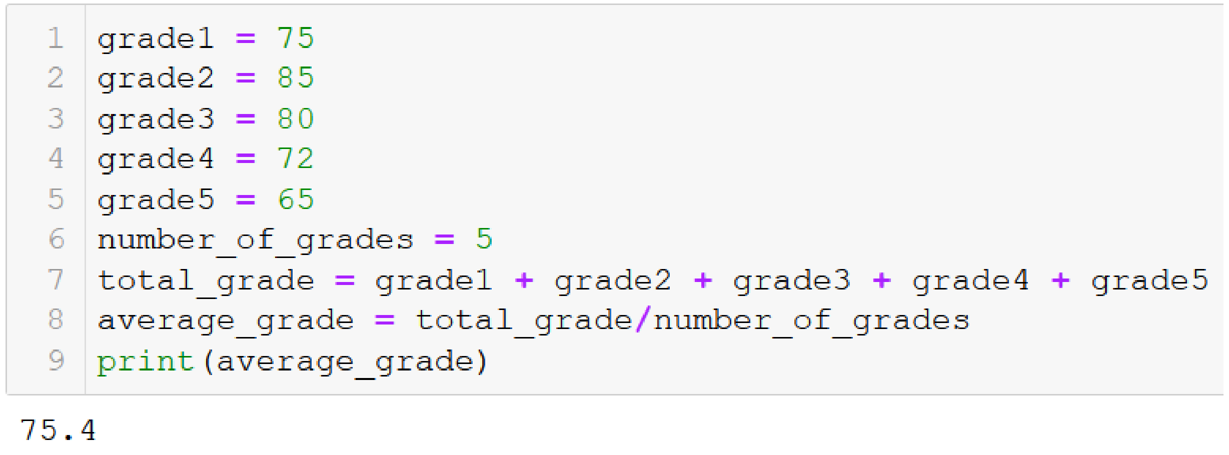
```
1
2
3
4
5
```

Notice how **the iteration variable value changes after each repetition of the code** inside the for loop.

An example: average grade for student

Not using a for loop: lots of variables

- Lots of variable
- Long code

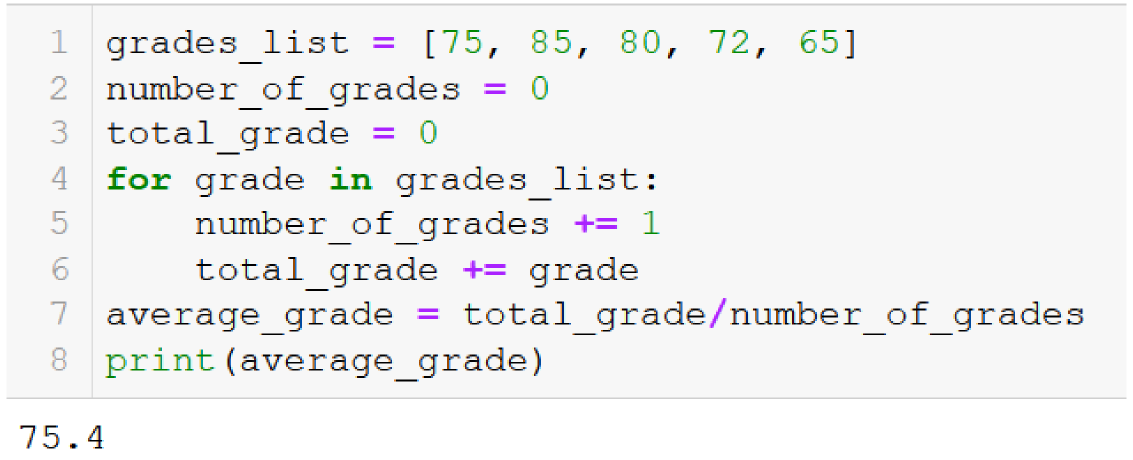


```python
grade1 = 75
grade2 = 85
grade3 = 80
grade4 = 72
grade5 = 65
number_of_grades = 5
total_grade = grade1 + grade2 + grade3 + grade4 + grade5
average_grade = total_grade/number_of_grades
print(average_grade)
```

```
75.4
```

Using a for loop

- Cleaner and shorter code
- Modular (works with any number of grades in list)

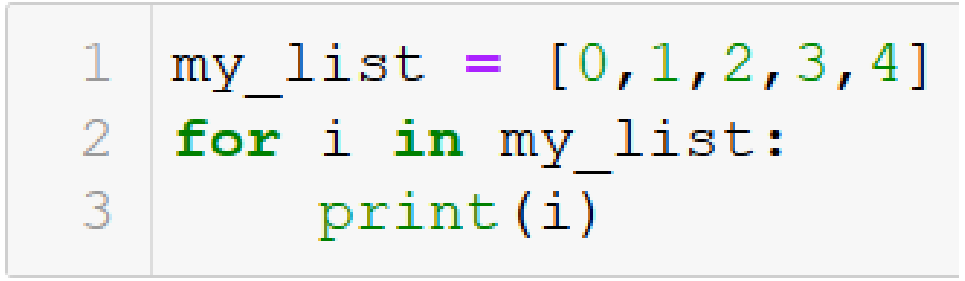


```python
grades_list = [75, 85, 80, 72, 65]
number_of_grades = 0
total_grade = 0
for grade in grades_list:
    number_of_grades += 1
    total_grade += grade
average_grade = total_grade/number_of_grades
print(average_grade)
```

```
75.4
```

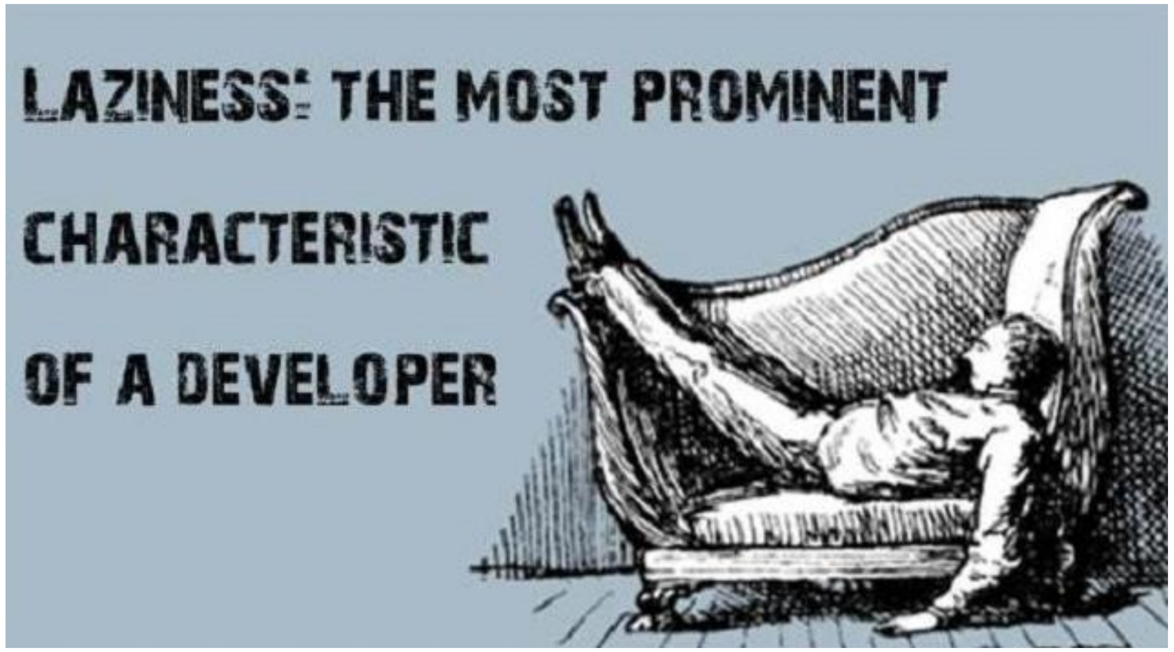
The range() generator

- **Problem:** typing a list of numbers manually is **cumbersome**, especially if it is supposed to contain lots of elements/numbers.



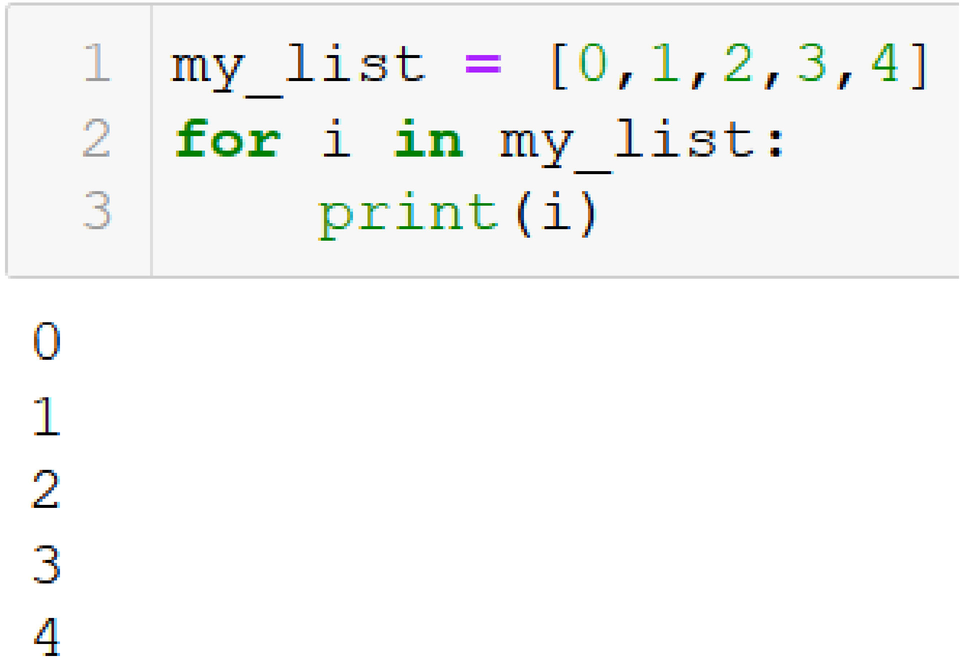
```python
my_list = [0,1,2,3,4]
for i in my_list:
    print(i)
```

```
0
1
2
3
4
```



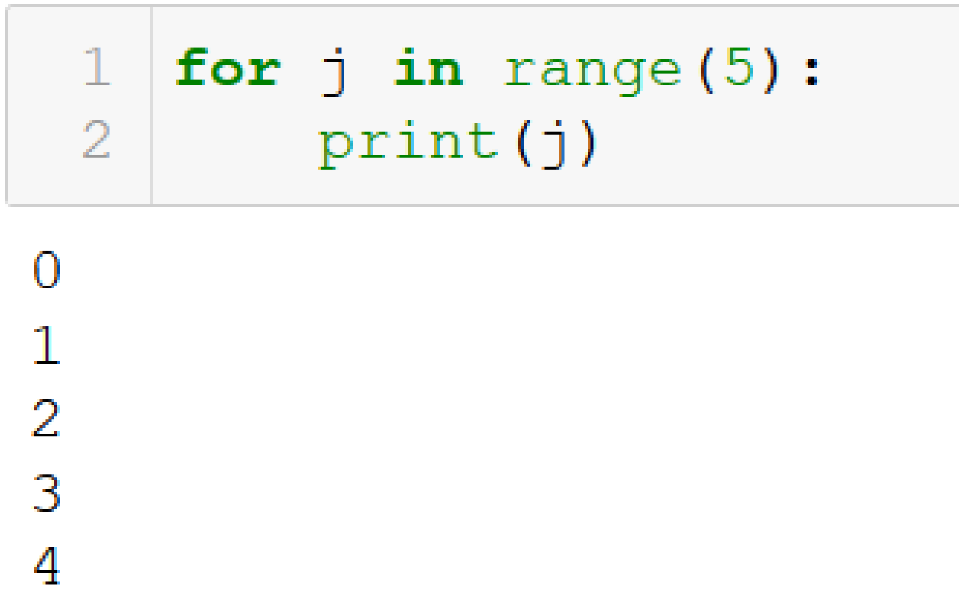
The range() generator

- **Solution:** The range() generator can be used to replace the list object in the for loop definition.



```python
my_list = [0,1,2,3,4]
for i in my_list:
    print(i)
```

```
0
1
2
3
4
```

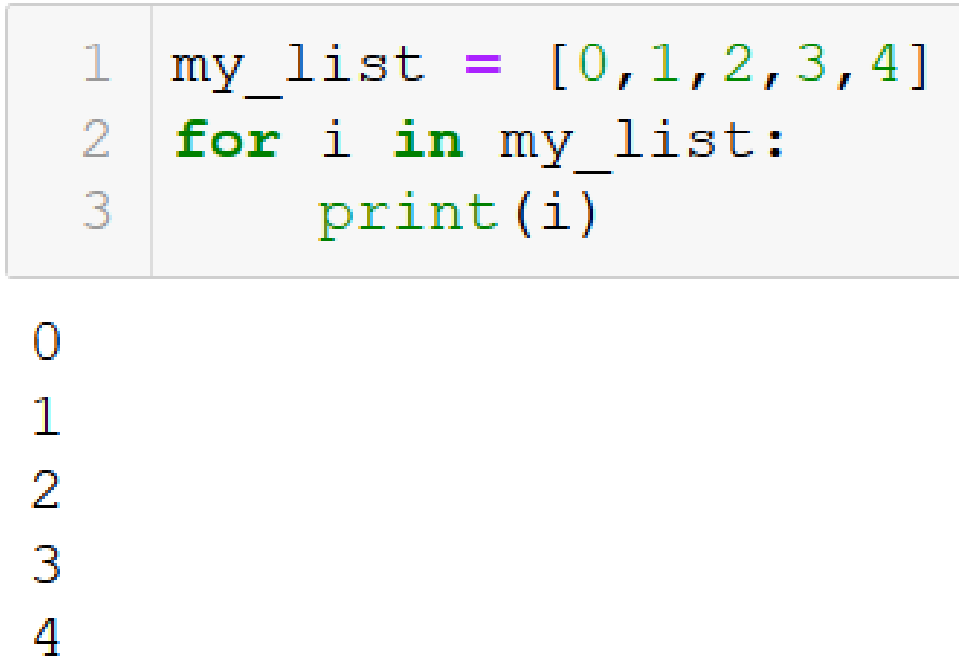


```python
for j in range(5):
    print(j)
```

```
0
1
2
3
4
```

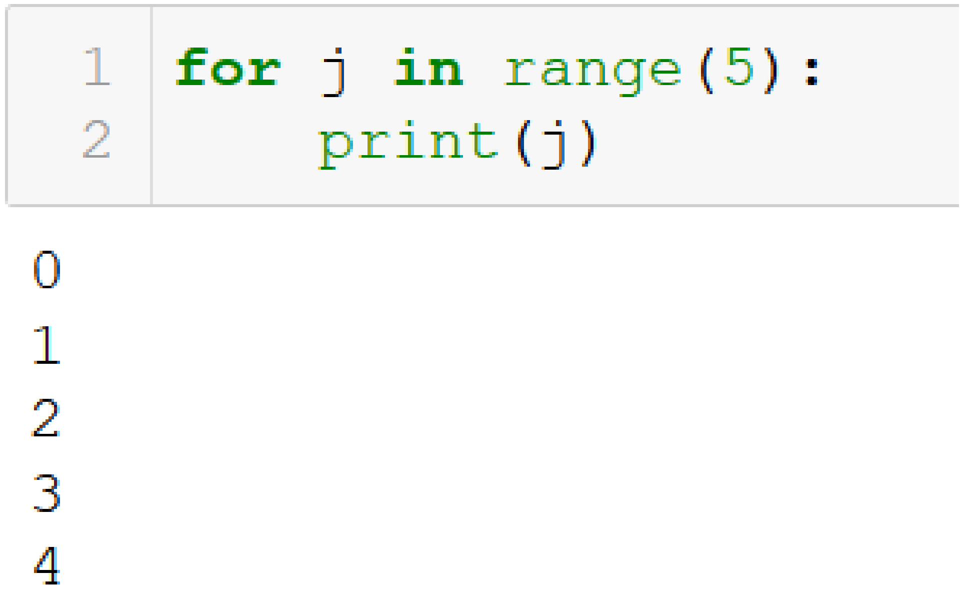
The range() generator

- **Solution:** The **range()** generator can be used to replace the list object in the **for** loop definition.
- It receives an integer **n.**
- Here, **range(n)** means: the **iteration variable** will take **n** successive values, starting from 0 and incrementing by 1 each time.



```
my_list = [0,1,2,3,4]
for i in my_list:
    print(i)
```

```
0
1
2
3
4
```



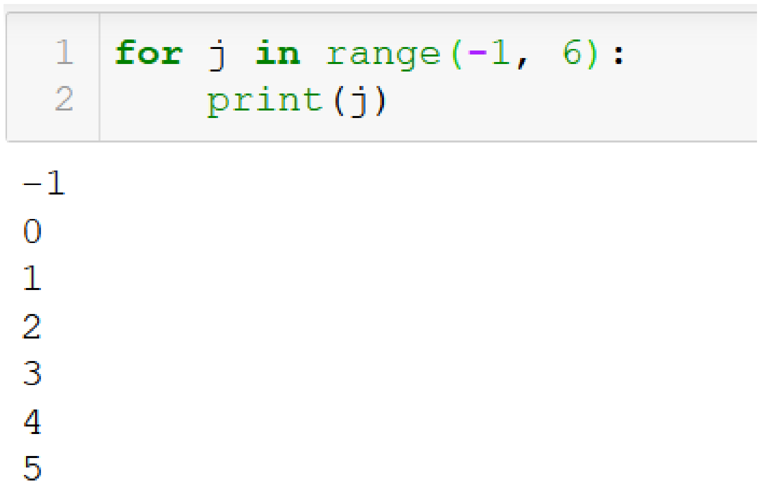
```
for j in range(5):
    print(j)
```

```
0
1
2
3
4
```

The range() generator

Up to three parameters can be given to the **range()** generator.

- **2 parameters: range(m, n)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by 1 each time, until we reach **n (n not included)**.



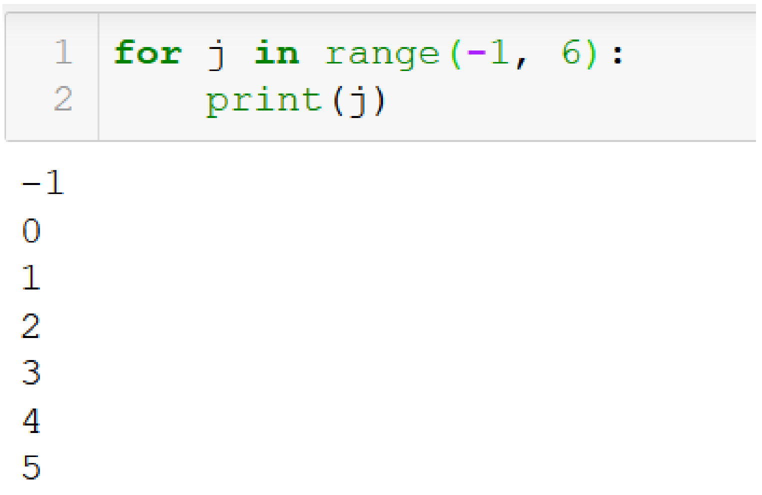
```
for j in range(-1, 6):
    print(j)
```

```
-1
0
1
2
3
4
5
```

The range() generator

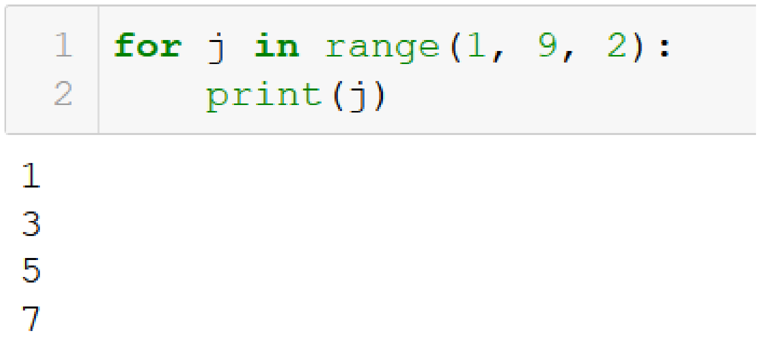
Up to three parameters can be given to the **range()** generator.

- **2 parameters: range(m, n)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by 1 each time, until we reach **n (n not included)**.
- **3 parameters: range(m, n, p)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by **p (instead of 1)** each time, until we reach **n (n not included)**.



```python
for j in range(-1, 6):
    print(j)
```

```
-1
0
1
2
3
4
5
```



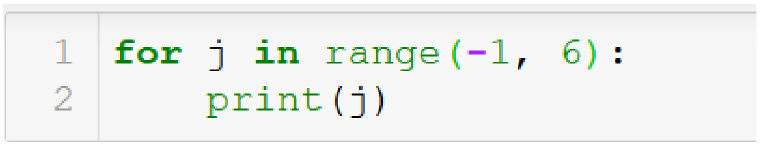
```python
for j in range(1, 9, 2):
    print(j)
```

```
1
3
5
7
```

The range() generator

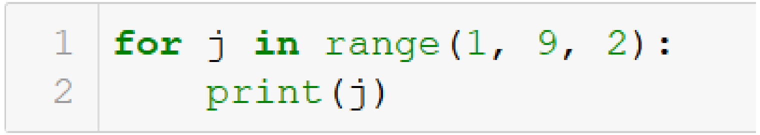
Up to three parameters can be given to the **range()** generator.

- **2 parameters: range(m, n)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by 1 each time, until we reach **n (<u>n not included</u>)**.
- **3 parameters: range(m, n, p)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by **p (instead of 1)** each time, until we reach **n (<u>n not included</u>)**.



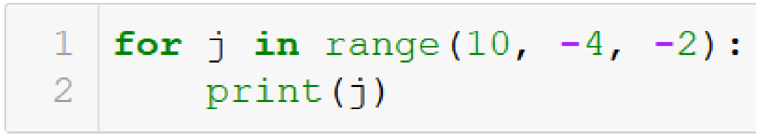
```python
for j in range(-1, 6):
    print(j)
```

```
-1
0
1
2
3
4
5
```



```python
for j in range(1, 9, 2):
    print(j)
```

```
1
3
5
7
```



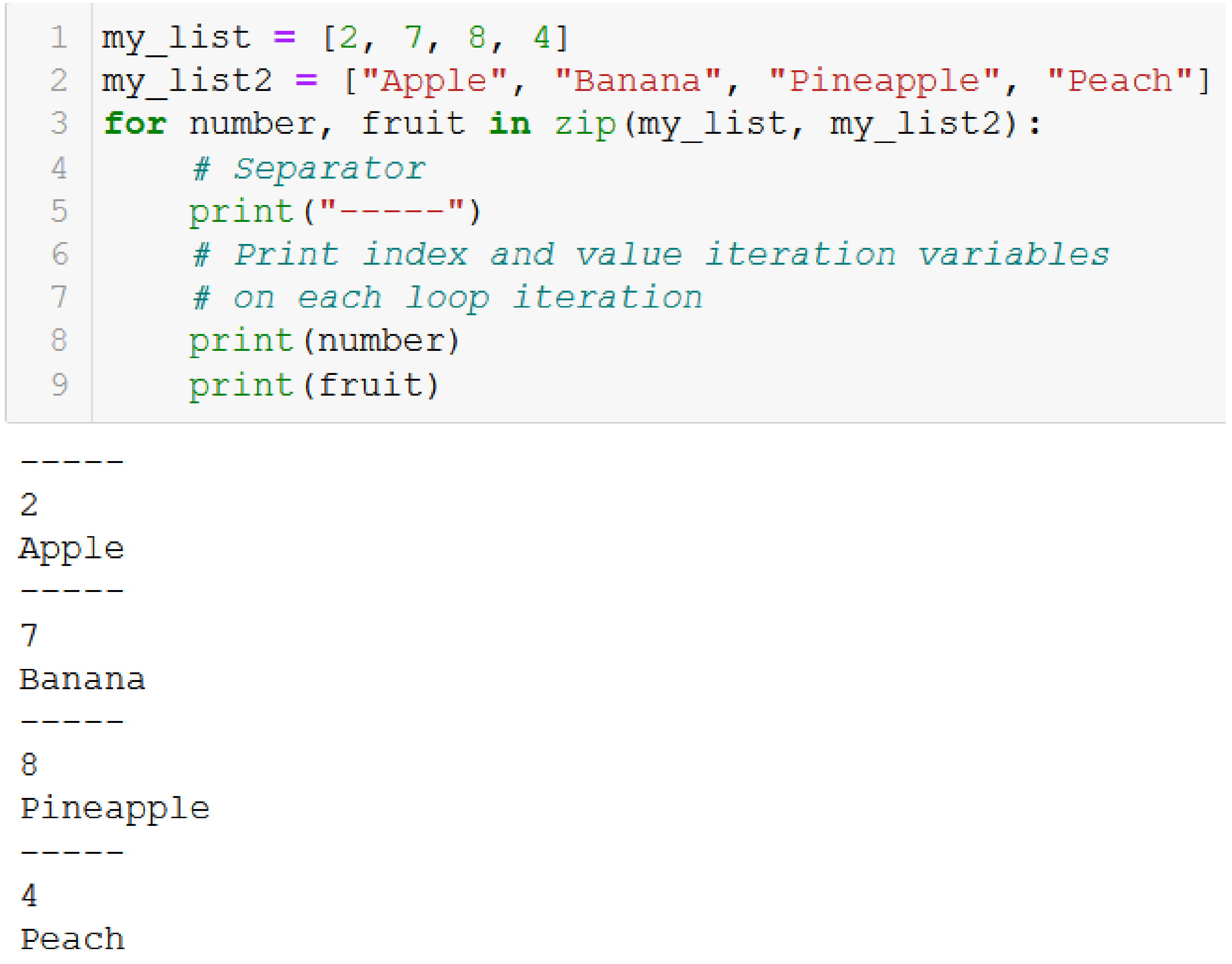
```python
for j in range(10, -4, -2):
    print(j)
```

```
10
8
6
4
2
0
-2
```

Note: if two or more parameters are used, we can play with negative values.

The zip() generator

- Want to browse through the elements of multiple lists at the same time?
- Use the **zip()** generator!

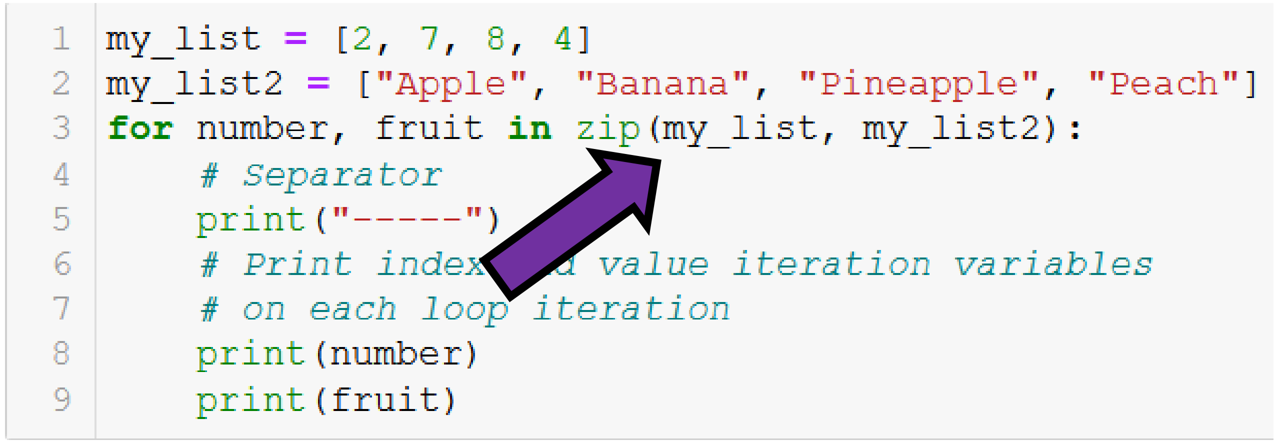


```python
my_list = [2, 7, 8, 4]
my_list2 = ["Apple", "Banana", "Pineapple", "Peach"]
for number, fruit in zip(my_list, my_list2):
    # Separator
    print("-----")
    # Print index and value iteration variables
    # on each loop iteration
    print(number)
    print(fruit)
```

```
-----
2
Apple
-----
7
Banana
-----
8
Pineapple
-----
4
Peach
```

The zip() generator

- Want to browse through the elements of multiple lists at the same time?
- Use the **zip()** generator!
- The **zip()** generator takes **multiple lists** of **<u>equal length</u>** (same number of elements).

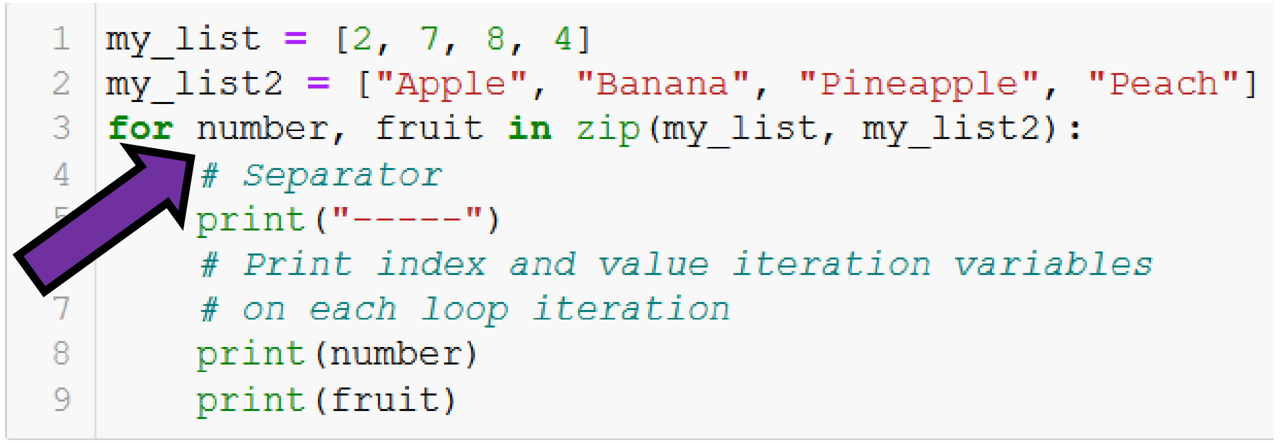


```python
my_list = [2, 7, 8, 4]
my_list2 = ["Apple", "Banana", "Pineapple", "Peach"]
for number, fruit in zip(my_list, my_list2):
    # Separator
    print("-----")
    # Print index and value iteration variables
    # on each loop iteration
    print(number)
    print(fruit)
```

```
-----
2
Apple
-----
7
Banana
-----
8
Pineapple
-----
4
Peach
```

The **zip()** generator

- Want to browse through the elements of multiple lists at the same time?
- Use the **zip()** generator!
- The **zip()** generator takes **multiple lists** of **equal length** (same number of elements).
- Updates that many **iteration variables** on each loop iteration, in a **synchronized manner**.

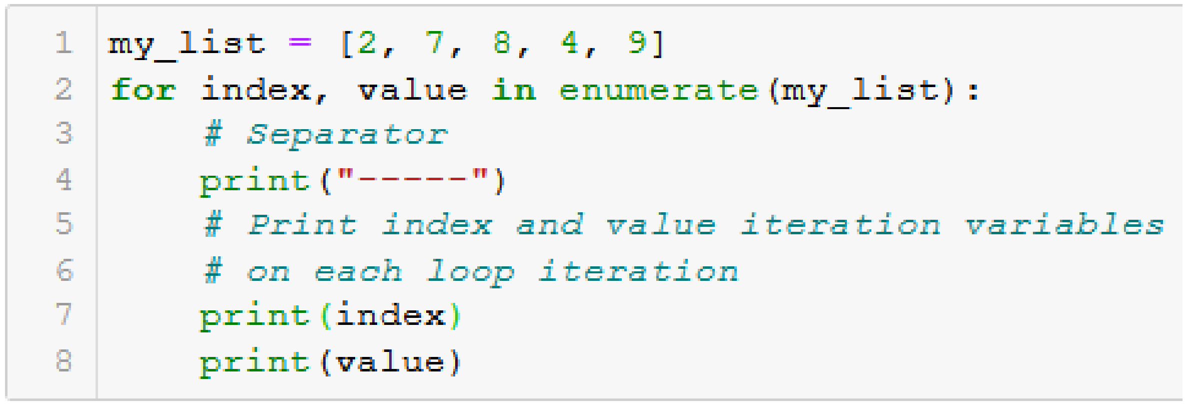


```python
my_list = [2, 7, 8, 4]
my_list2 = ["Apple", "Banana", "Pineapple", "Peach"]
for number, fruit in zip(my_list, my_list2):
    # Separator
    print("-----")
    # Print index and value iteration variables
    # on each loop iteration
    print(number)
    print(fruit)
```

```
-----
2
Apple
-----
7
Banana
-----
8
Pineapple
-----
4
Peach
```

The enumerate() generator

- The **enumerate()** generator can be used to **update two iteration variables** at once.

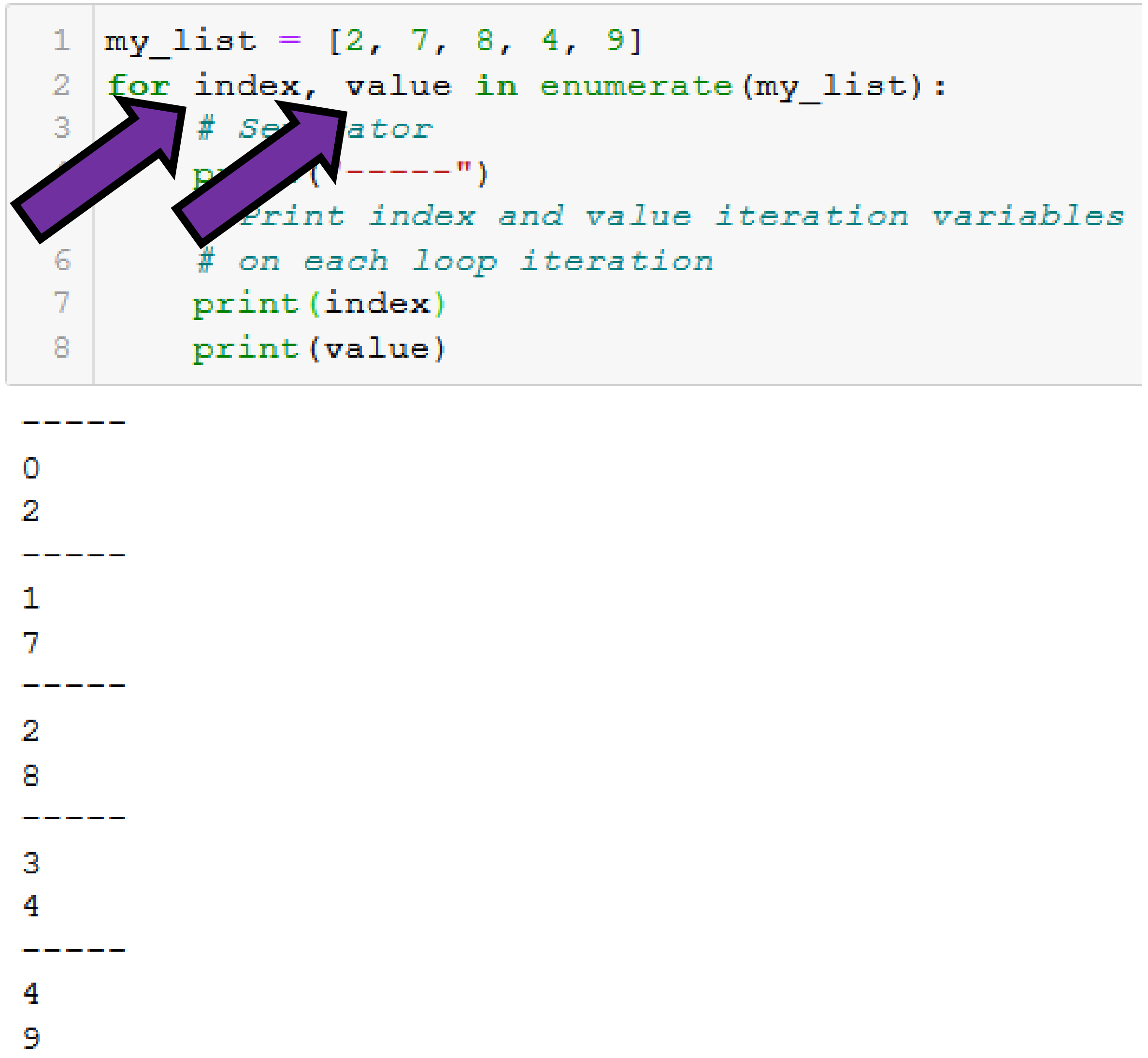


```python
my_list = [2, 7, 8, 4, 9]
for index, value in enumerate(my_list):
    # Separator
    print("-----")
    # Print index and value iteration variables
    # on each loop iteration
    print(index)
    print(value)
```

```
-----
0
2
-----
1
7
-----
2
8
-----
3
4
-----
4
9
```

The enumerate() generator

- The **enumerate()** generator can be used to **update two iteration variables** at once.
  On each loop iteration:
- The first one takes values consisting of the **position index** (1st, 2nd,3rd,… element),
- The second is takes **value** of the element in the list.

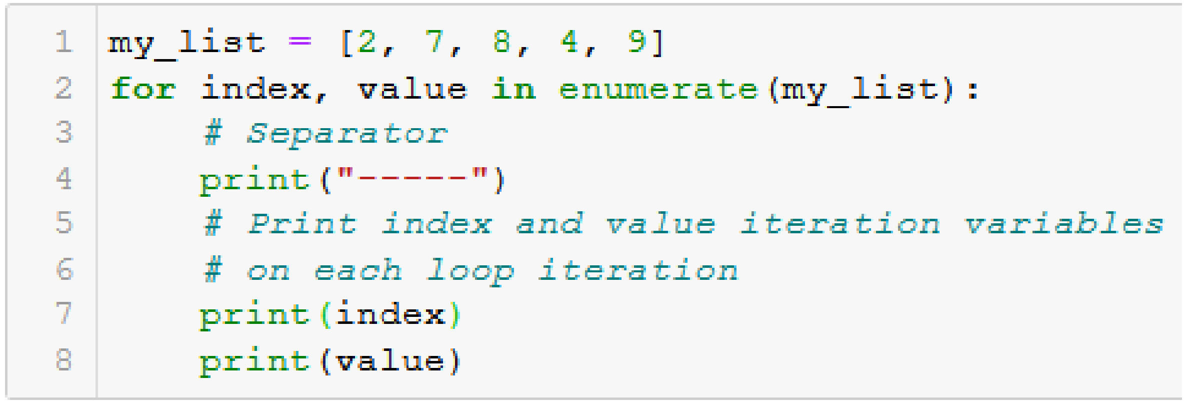


```
my_list = [2, 7, 8, 4, 9]
for index, value in enumerate(my_list):
    # Separator
    print("------")
    # Print index and value iteration variables
    # on each loop iteration
    print(index)
    print(value)
```

```
-----
0
2
-----
1
7
-----
2
8
-----
3
4
-----
4
9
```

The enumerate() generator

- The **enumerate()** generator can be used to **update two iteration variables** at once.
  On each loop iteration:
- The first one takes values consisting of the **position index** (1st, 2nd,3rd,… element),
- The second is takes **value** of the element in the list.



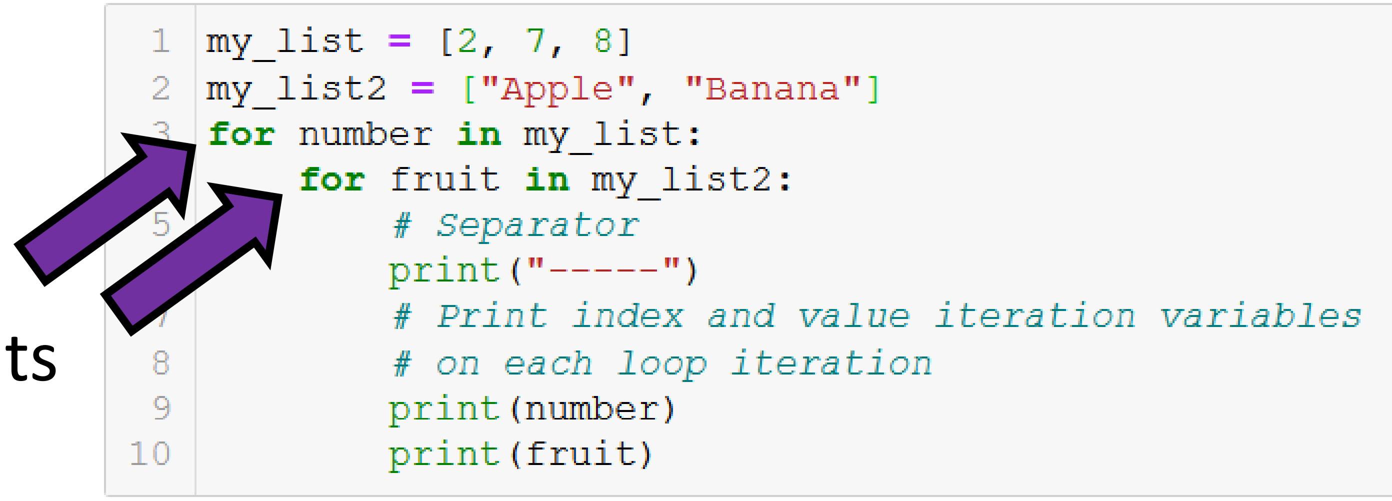
```python
my_list = [2, 7, 8, 4, 9]
for index, value in enumerate(my_list):
    # Separator
    print("-----")
    # Print index and value iteration variables
    # on each loop iteration
    print(index)
    print(value)
```

```
-----
0
2
-----
1
7
-----
2
8
-----
3
4
-----
4
9
```

Note: In Python, **we start counting from 0**. What we call the 1st element of the list in English, is called the 0th element (index = 0) in programming.

Nesting for loops

- Just like we **nested if** statements earlier, we can **nest for** loops.

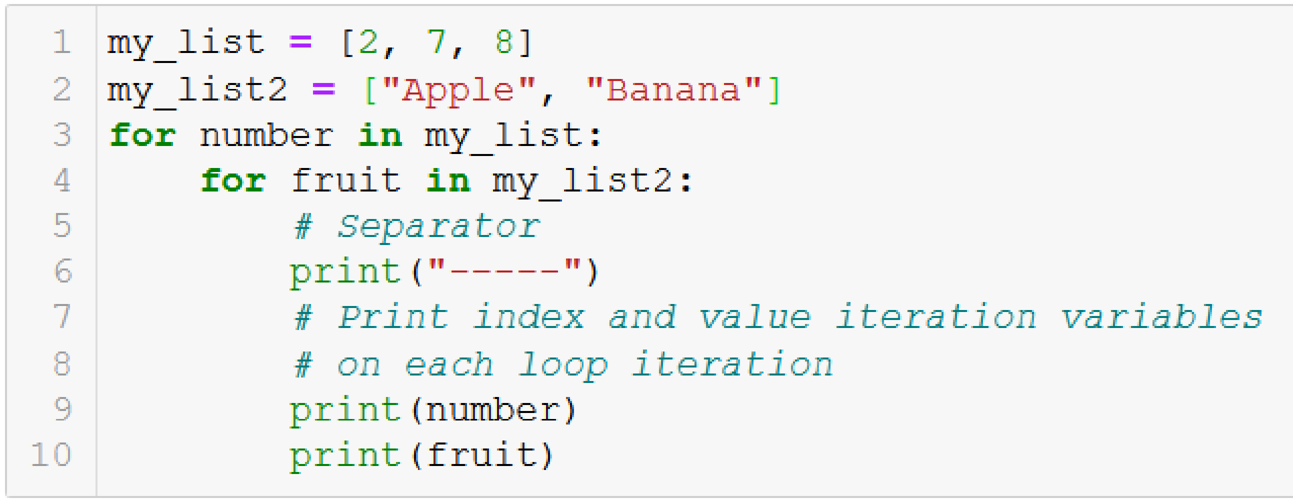


```python
my_list = [2, 7, 8]
my_list2 = ["Apple", "Banana"]
for number in my_list:
    for fruit in my_list2:
        # Separator
        print("-----")
        # Print index and value iteration variables
        # on each loop iteration
        print(number)
        print(fruit)
```

```
-----
2
Apple
-----
2
Banana
-----
7
Apple
-----
7
Banana
-----
8
Apple
-----
8
Banana
```

Nesting for loops

- Just like we **nested if** statements earlier, we can **nest for** loops.
- Works "almost" like the **zip** generator,
- But updates the **iteration variables** in an **<u>unsynchronized</u> manner**.

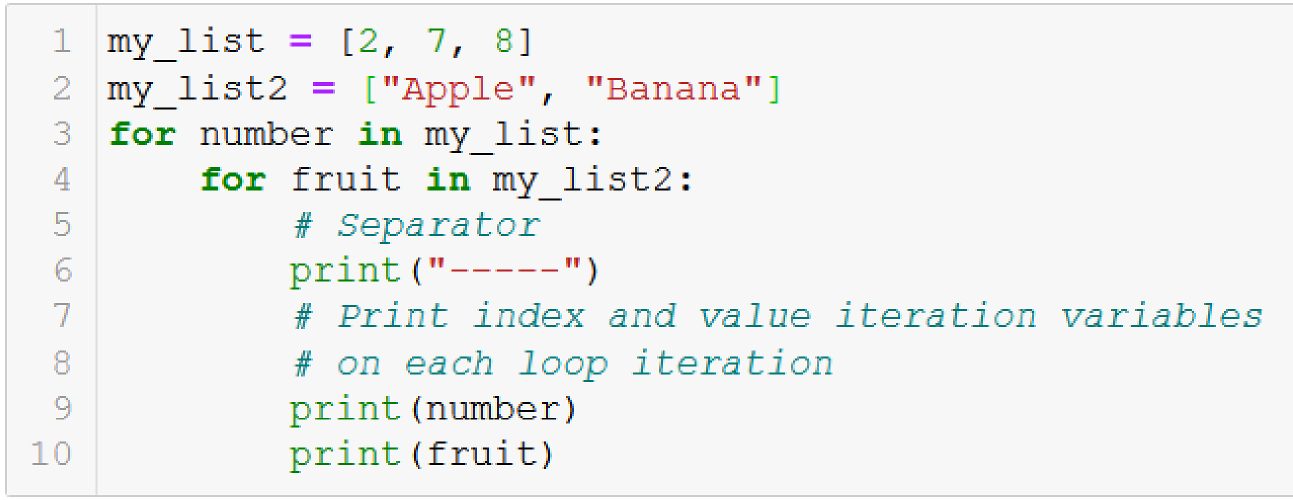


```python
my_list = [2, 7, 8]
my_list2 = ["Apple", "Banana"]
for number in my_list:
    for fruit in my_list2:
        # Separator
        print("-----")
        # Print index and value iteration variables
        # on each loop iteration
        print(number)
        print(fruit)
```

```
-----
2
Apple
-----
2
Banana
-----
7
Apple
-----
7
Banana
-----
8
Apple
-----
8
Banana
```

Nesting for loops

- Just like we **nested if** statements earlier, we can **nest for** loops.
- Works "almost" like the **zip** generator,
- But updates the **iteration variables** in an **<u>unsynchronized</u> manner**.
- Convenient for checking all combinations of values in two given lists!



```python
my_list = [2, 7, 8]
my_list2 = ["Apple", "Banana"]
for number in my_list:
    for fruit in my_list2:
        # Separator
        print("-----")
        # Print index and value iteration variables
        # on each loop iteration
        print(number)
        print(fruit)
```

```
-----
2
Apple
-----
2
Banana
-----
7
Apple
-----
7
Banana
-----
8
Apple
-----
8
Banana
```

The **break** statement (episode 2)

- In W2S3, we have seen how the **break** statement can be used to **interrupt** a **while** loop.
- It also works with **for** loops!

```python
my_list = [1, 2, 3, 4, 5]
for value in my_list:
    # Separator
    print("-----")
    # Print iteration variable value
    print(value)
    # Break if
    if(value == 3):
        print("Breaking for loop")
        break
```

```
-----
1
-----
2
-----
3
Breaking for loop
```

To recap

We have seen multiple ways to make **for** loops work

1. **Pass a list:** easiest way, browse through each element one by one.

To recap

We have seen multiple ways to make **for** loops work

1. **Pass a list:** easiest way, browse through each element one by one.
2. **The range() generator:** Replace a list of regularly spaced values with a range() generator, as to avoid having to type the elements of the list manually.

To recap

We have seen multiple ways to make **for** loops work

1. **Pass a list:** easiest way, browse through each element one by one.
2. **The range() generator:** Replace a list of regularly spaced values with a range() generator, as to avoid having to type the elements of the list manually.
3. **The zip() generator:** Browse through multiple lists elements in a synchronized manner.

To recap

We have seen multiple ways to make **for** loops work

1. **Pass a list:** easiest way, browse through each element one by one.
2. **The range() generator:** Replace a list of regularly spaced values with a range() generator, as to avoid having to type the elements of the list manually.
3. **The zip() generator:** Browse through multiple lists elements in a synchronized manner.
4. **The enumerate() generator:** Updates two iteration variables at once, one being the position index of the element in the list, and the second being the value of said element in list.

To recap

We have seen multiple ways to make for loops work

1. **Pass a list:** easiest way, browse through each element one by one.
2. **The range() generator:** Replace a list of regularly spaced values with a range() generator, as to avoid having to type the elements of the list manually.
3. **The zip() generator:** Browse through multiple lists elements in a synchronized manner.
4. **The enumerate() generator:** Updates two iteration variables at once, one being the position index of the element in the list, and the second being the value of said element in list.
5. **Nesting for loops:** Browse through multiple lists elements in an unsynchronized manner.

Practice activities: basic for loops

Let us practice a bit with for loops, with the following activities

Activity 1 - How many items in my inventory.ipynb

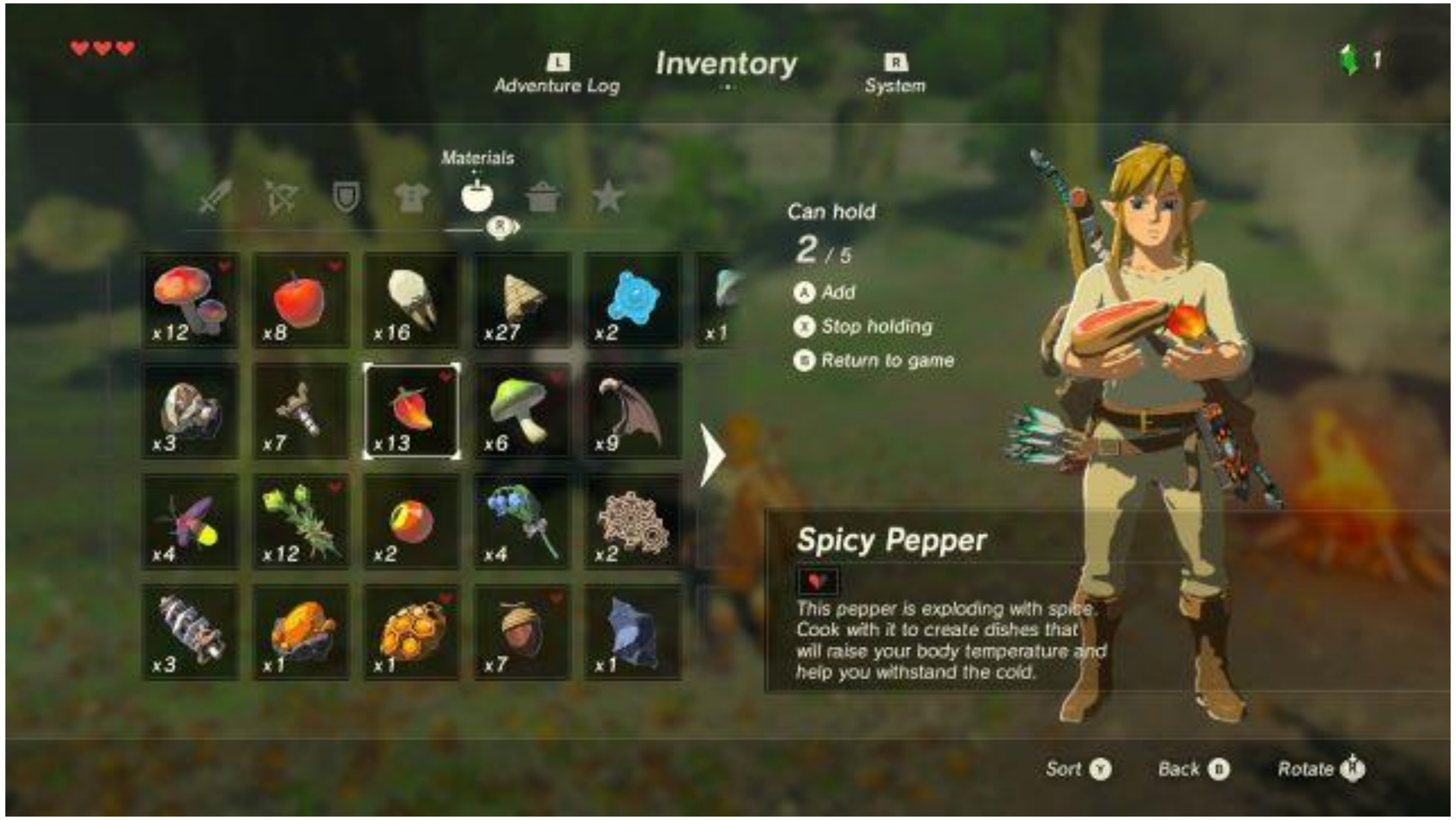
Activity 2 - Best equipment finder.ipynb

Activity 3 - Best equipment finder v2.ipynb

Activity 4 - Find the missing card.ipynb

Activity 1 - How many items in my inventory

- In several video games, the main character will have an **inventory**, i.e. a list of items that he/she is carrying at the moment.



This pepper is exploding with spice. Cook with it to create dishes that will raise your body temperature and help you withstand the cold.

Activity 1 - How many items in my inventory

- In several video games, the main character will have an **inventory**, i.e. a list of items that he/she is carrying at the moment. This **inventory** could be defined as a **list**, as shown below.

inventory = ["Sword", "Armor", "Potion", "Potion", "Torch", "Potion", "Bow", "Potion", "Torch", "Potion"]

Activity 1 - How many items in my inventory

- In several video games, the main character will have an **inventory**, i.e. a list of items that he/she is carrying at the moment. This **inventory** could be defined as a **list**, as shown below.

inventory = ["Sword", "Armor", "Potion", "Potion", "Torch", "Potion", "Bow", "Potion", "Torch", "Potion"]

- Our objective is to write a function **how_many_items()**, which:
    - **receives** an **inventory list**, such as the one above, as its first parameter,
    - **receives** an **item name**, as a second parameter (e.g. item_name = "Torch")
    - and **returns the number of times the item** in question **appears in the inventory**.

Activity 2 - Best equipment finder

- Let us define an **inventory list**, below, which contains a list of weapons that our character has acquired during gameplay.

inventory = ["Dull Sword", "Wooden Branch", "Master Sword", "Iron Sword", "Silver Sword"]

Activity 2 - Best equipment finder

- Let us define an **inventory list**, below, which contains a list of weapons that our character has acquired during gameplay.

inventory = ["Dull Sword", "Wooden Branch", "Master Sword", "Iron Sword", "Silver Sword"]

- Let us also consider we have been given a **second list**, which contains the **attack points for each weapon** currently in inventory, in order:

weapon_stats = [1, 1, 10, 5, 8]

Activity 2 - Best equipment finder

- Let us define an **inventory list**, below, which contains a list of weapons that our character has acquired during gameplay.

inventory = ["Dull Sword", "Wooden Branch", "Master Sword", "Iron Sword", "Silver Sword"]

- Let us also consider we have been given a **second list**, which contains the **attack points for each weapon** currently in inventory, in order:

weapon_stats = [1, 1, 10, 5, 8]

- Write a function **maximal_attack_points()**, which
  - **receives** the **weapon_stats list** as its only parameter,
  - and **returns the maximal attack points** we would have if we were to equip the **best weapon** currently in inventory.

Activity 3 - Best equipment finder v2

- Let us define an **inventory list**, below, which contains a list of weapons that our character has acquired during gameplay.

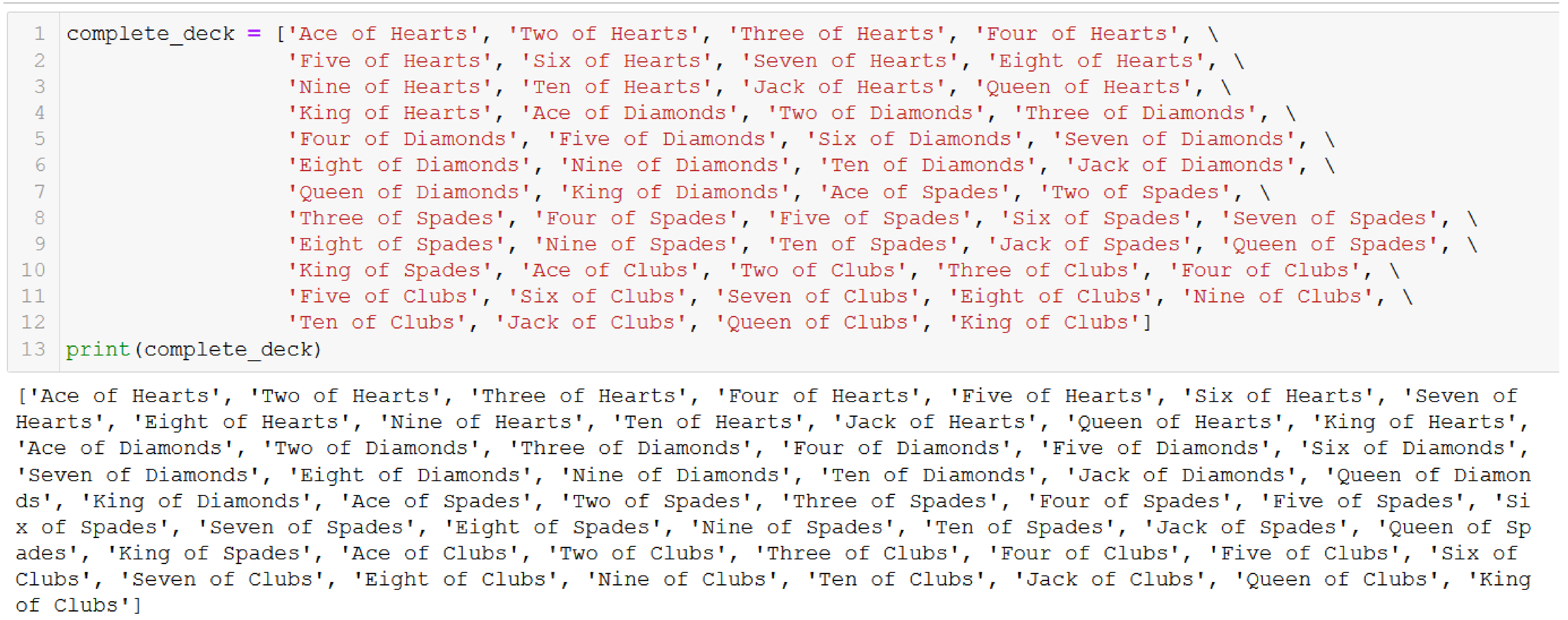
inventory = ["Dull Sword", "Wooden Branch", "Master Sword", "Iron Sword", "Silver Sword"]

- Let us also consider we have been given a **second list**, which contains the **attack points for each weapon** currently in inventory, in order:

weapon_stats = [1, 1, 10, 5, 8]

- **Task:** As in activity 2, but I want **the name of the best weapon to be returned** instead of the maximal attack points I would obtain if I decided to equip it!

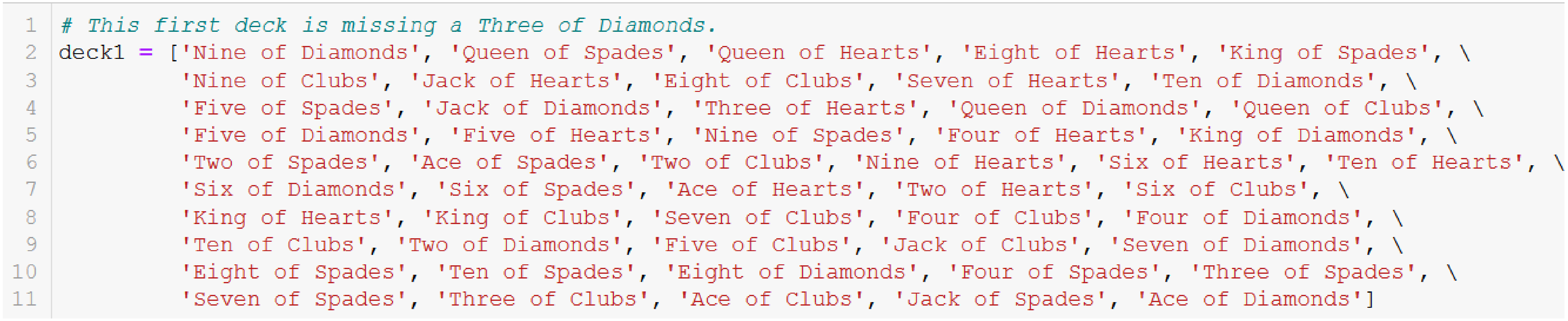
Activity 4 - Find the missing card



```python
complete_deck = ['Ace of Hearts', 'Two of Hearts', 'Three of Hearts', 'Four of Hearts', \
                 'Five of Hearts', 'Six of Hearts', 'Seven of Hearts', 'Eight of Hearts', \
                 'Nine of Hearts', 'Ten of Hearts', 'Jack of Hearts', 'Queen of Hearts', \
                 'King of Hearts', 'Ace of Diamonds', 'Two of Diamonds', 'Three of Diamonds', \
                 'Four of Diamonds', 'Five of Diamonds', 'Six of Diamonds', 'Seven of Diamonds', \
                 'Eight of Diamonds', 'Nine of Diamonds', 'Ten of Diamonds', 'Jack of Diamonds', \
                 'Queen of Diamonds', 'King of Diamonds', 'Ace of Spades', 'Two of Spades', \
                 'Three of Spades', 'Four of Spades', 'Five of Spades', 'Six of Spades', 'Seven of Spades', \
                 'Eight of Spades', 'Nine of Spades', 'Ten of Spades', 'Jack of Spades', 'Queen of Spades', \
                 'King of Spades', 'Ace of Clubs', 'Two of Clubs', 'Three of Clubs', 'Four of Clubs', \
                 'Five of Clubs', 'Six of Clubs', 'Seven of Clubs', 'Eight of Clubs', 'Nine of Clubs', \
                 'Ten of Clubs', 'Jack of Clubs', 'Queen of Clubs', 'King of Clubs']
print(complete_deck)
```

```
['Ace of Hearts', 'Two of Hearts', 'Three of Hearts', 'Four of Hearts', 'Five of Hearts', 'Six of Hearts', 'Seven of
Hearts', 'Eight of Hearts', 'Nine of Hearts', 'Ten of Hearts', 'Jack of Hearts', 'Queen of Hearts', 'King of Hearts',
'Ace of Diamonds', 'Two of Diamonds', 'Three of Diamonds', 'Four of Diamonds', 'Five of Diamonds', 'Six of Diamonds',
'Seven of Diamonds', 'Eight of Diamonds', 'Nine of Diamonds', 'Ten of Diamonds', 'Jack of Diamonds', 'Queen of Diamon
ds', 'King of Diamonds', 'Ace of Spades', 'Two of Spades', 'Three of Spades', 'Four of Spades', 'Five of Spades', 'Si
x of Spades', 'Seven of Spades', 'Eight of Spades', 'Nine of Spades', 'Ten of Spades', 'Jack of Spades', 'Queen of Sp
ades', 'King of Spades', 'Ace of Clubs', 'Two of Clubs', 'Three of Clubs', 'Four of Clubs', 'Five of Clubs', 'Six of
Clubs', 'Seven of Clubs', 'Eight of Clubs', 'Nine of Clubs', 'Ten of Clubs', 'Jack of Clubs', 'Queen of Clubs', 'King
of Clubs']
```

Activity 4 - Find the missing card



```python
# This first deck is missing a Three of Diamonds.
deck1 = ['Nine of Diamonds', 'Queen of Spades', 'Queen of Hearts', 'Eight of Hearts', 'King of Spades', \
         'Nine of Clubs', 'Jack of Hearts', 'Eight of Clubs', 'Seven of Hearts', 'Ten of Diamonds', \
         'Five of Spades', 'Jack of Diamonds', 'Three of Hearts', 'Queen of Diamonds', 'Queen of Clubs', \
         'Five of Diamonds', 'Five of Hearts', 'Nine of Spades', 'Four of Hearts', 'King of Diamonds', \
         'Two of Spades', 'Ace of Spades', 'Two of Clubs', 'Nine of Hearts', 'Six of Hearts', 'Ten of Hearts', \
         'Six of Diamonds', 'Six of Spades', 'Ace of Hearts', 'Two of Hearts', 'Six of Clubs', \
         'King of Hearts', 'King of Clubs', 'Seven of Clubs', 'Four of Clubs', 'Four of Diamonds', \
         'Ten of Clubs', 'Two of Diamonds', 'Five of Clubs', 'Jack of Clubs', 'Seven of Diamonds', \
         'Eight of Spades', 'Ten of Spades', 'Eight of Diamonds', 'Four of Spades', 'Three of Spades', \
         'Seven of Spades', 'Three of Clubs', 'Ace of Clubs', 'Jack of Spades', 'Ace of Diamonds']
```

Activity 4 - Find the missing card

Write a function **find_missing_card()**, which **receives** a **complete deck of cards** as its first parameter, and **receives a second deck**, as its second parameter.

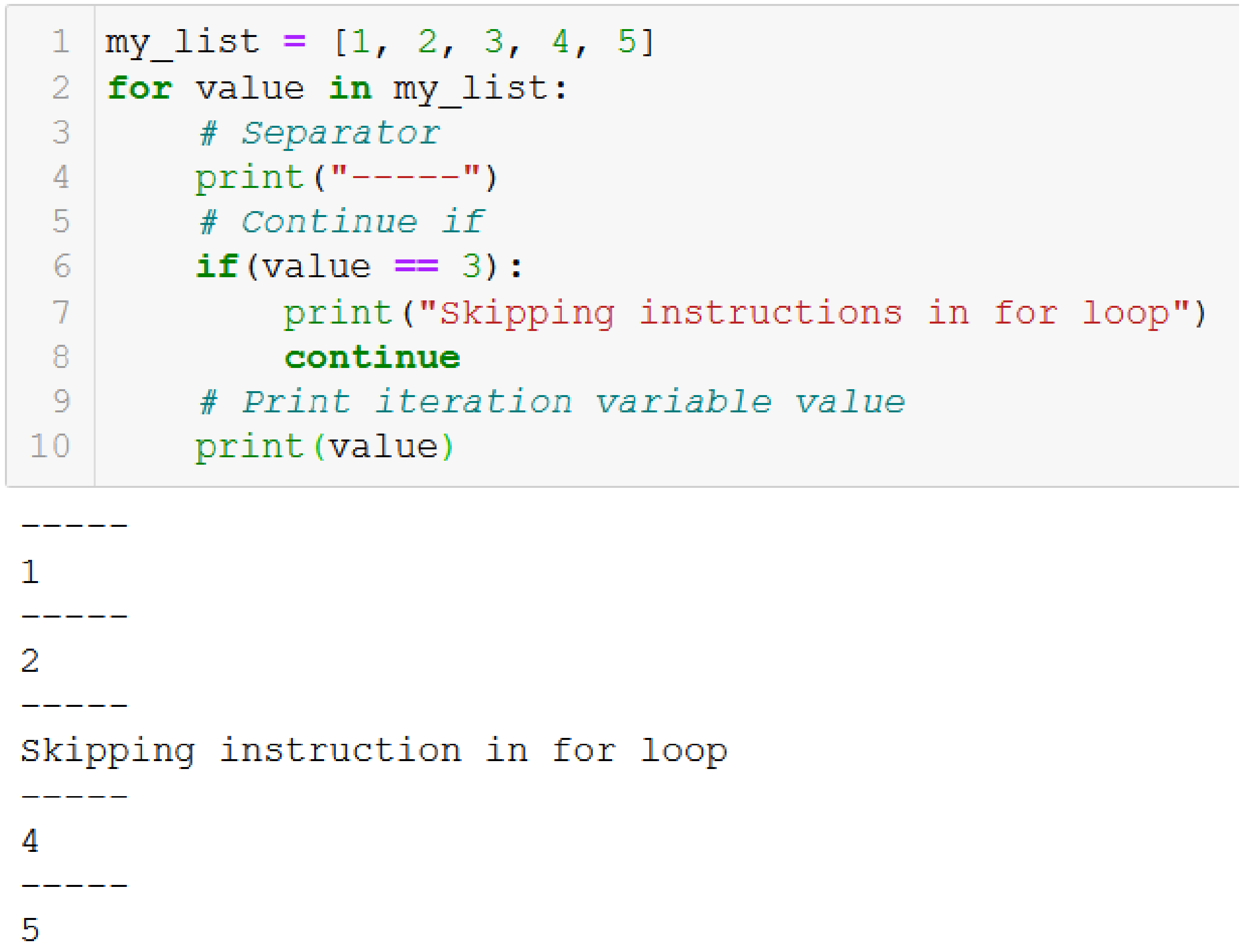
- The second deck is a standard deck that has been shuffled and **may be missing a single card**.
- The function **find_missing_card()** should **return the name of the one card that is missing** in the second deck. It should **return None**, if no card is missing.
- Note that:
  - The decks are missing one card at most,
  - The decks will contain no duplicates.

Conclusion

- The list type (quick intro, more to come on W3S1)
- The for statement
- The range() generator
- The enumerate() generator
- The zip() generator
- Nesting for loops
- Breaking for loops

The continue statement

- Similar to the **break** statement, which was be used to **interrupt** a **while**/**for** loop…
- We can define the **continue** statement!
- When encountered in the indented code inside a loop, it **ends the current iteration** and **moves on to the next one**.

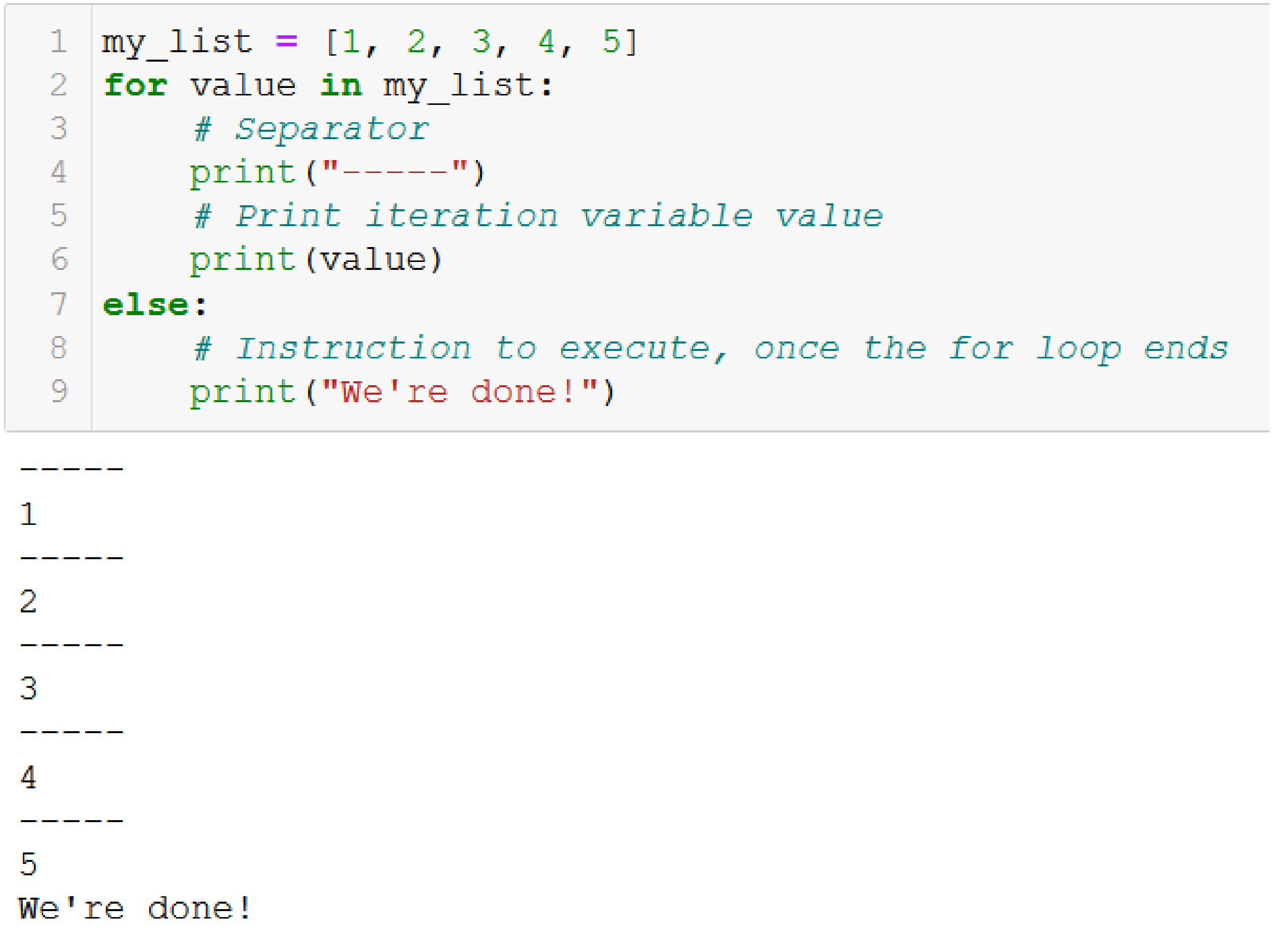


```python
my_list = [1, 2, 3, 4, 5]
for value in my_list:
    # Separator
    print("-----")
    # Continue if
    if(value == 3):
        print("Skipping instructions in for loop")
        continue
    # Print iteration variable value
    print(value)
```

```
-----
1
-----
2
-----
Skipping instruction in for loop
-----
4
-----
5
```

The else statement (episode 2)

- Similar to the **else** statement, which was used in **if** statements…
- We can define the **else** statement in **for** loops.
- It defines a piece of code to be executed when the **for** loop ends **normally**.
- **Normally:** completed all iterations, **not interrupted** by a **break**.

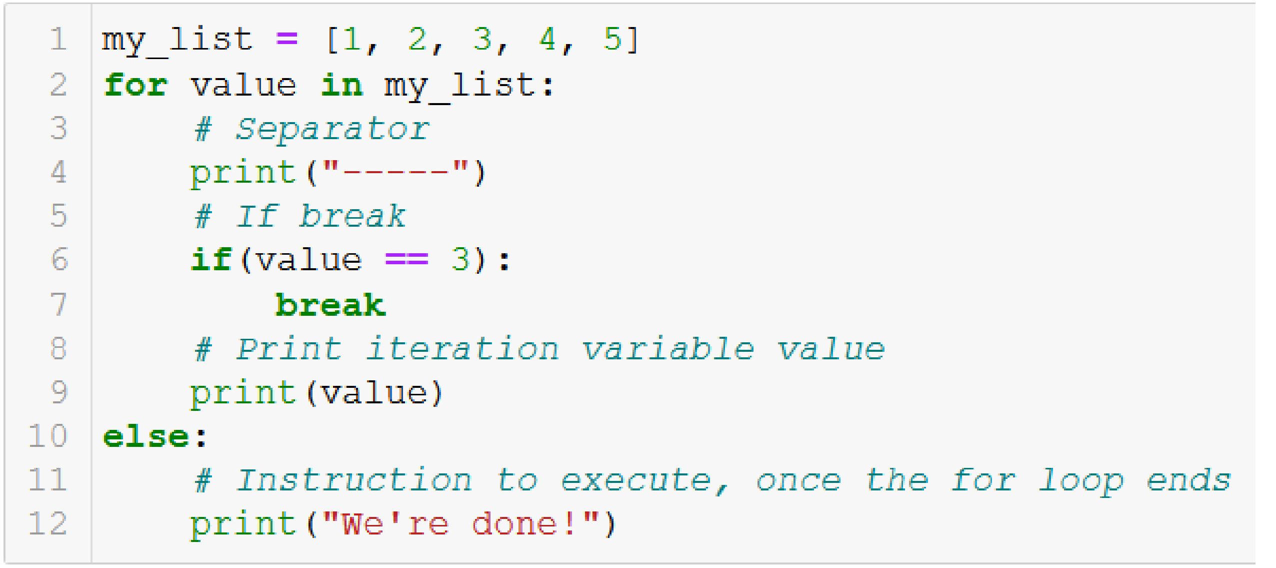


```python
my_list = [1, 2, 3, 4, 5]
for value in my_list:
    # Separator
    print("-----")
    # Print iteration variable value
    print(value)
else:
    # Instruction to execute, once the for loop ends
    print("We're done!")
```

```
-----
1
-----
2
-----
3
-----
4
-----
5
We're done!
```

The else statement (episode 2)

- Similar to the **else** statement, which was used in **if** statements…
- We can define the **else** statement in **for** loops.
- It defines a piece of code to be executed when the **for** loop ends **normally**.
- **Normally:** completed all iterations, **not interrupted** by a **break**.



```
my_list = [1, 2, 3, 4, 5]
for value in my_list:
    # Separator
    print("-----")
    # If break
    if(value == 3):
        break
    # Print iteration variable value
    print(value)
else:
    # Instruction to execute, once the for loop ends
    print("We're done!")
```

```
-----
1
-----
2
-----
```