Loops

Lesson Duration: 60 minutes

Learning Objectives

- Be able to use loops to control behaviour
- Be able to use a for loop and a while loop

What Are Loops

Loops in Python are used to execute a block of code a specified number of times. One of the principles of writing clean code is Don't Repeat Yourself, or DRY. Writing loops is one way to help us write more DRY code. Instead of repeating a block of code, we can tell Python to run it multiple times.

We're going to look at two types of loops: `while` and `for` .

Let's create a file called `loops.py` in our `session_2` folder!

`while` Loop

A while loop executes code while a specified condition is true. Once the condition is not true the loop ends.

In a way, we can treat it similarly to an `if` statement - `if` a condition is met, keep repeating code, if it's not met anymore, stop it!

```
counter = 0
my_number = 5

while (counter < my_number):
   print(f"counter is {counter}")
   counter += 1
```

> String formatting using the `f""` strings are really cool - you can create a formatted string by adding `f` before a `""` (note, it must be double quotes!) and then you can inject any variable inside `{}` symbols!

> Note that the formatting of the string as seen above is optional - you can use `"counter is " + str(counter)`, although formatted strings tend to be more elegant - please refer to the video!

If we do not create an exit condition, we will keep looping forever - until our computer runs out of RAM and crashes it eventually! Be careful with `while` loops. If you get stuck with one, you can try the following - click into the terminal where the output should be, and press `CTRL` + `C`, for both Windows and Macs!

Playing with Loops

Let's write a program that asks a user to guess the answer to a question, and loops until they get it right!

> Note that this is for training purposes mostly, interaction with the terminal is a rare way of interfacing with the user of the app! If you get stuck in a loop, remember: `CTRL` + `C`!

```
my_number = 5
user_input = input("What number am I thinking of?")

while (value != my_number):
    user_input = input("nope! try again... ")

print("yes!")
```

Even if we guess right, however, it still doesn't work - why?

If you remember from last week, the `input()` function's value will be stored from the user as a `str`, but we try to check if it's equal to an `int`! We need to wrap the input in an `int()` function, so the datatypes will match!

```
my_number = 5
user_input = int(input("What number am I thinking of?"))

while (value != my_number):
    user_input = int(input("nope! try again... "))

print("yes!")
```

> Optional task: Try to give prompts to the user is their guess was too high or too low!

```
my_number = 5
value = int(input("What number am I thinking of? "))

while (value != my_number):
    if (value > my_number):
        print("too high")
    else:
        print("too low")
    value = int(input("try again... "))

print("yes!")
```

`for` Loop

A more common way of using loops is a `for loop`. These work on the principle that you need a collection to loop through - for each iteration, a different element inside the list will be grabbed, a predefined codeblock gets executed with that element, and once the codeblock finished running, it will keep repeating with the next element - until the for loop reaches the end of the list.

Let's start with a simple example using some numbers:

```
# loops.py

numbers = [1, 2, 3, 4, 5]

for number in numbers:
    print(number * 3)
```

You can also add up the total value of all integers stored inside the list:

```
# loops.py
numbers = [1, 2, 3, 4, 5]

total = 0

for number in numbers:
    total = total + number

print(total)
```

Student's names can also be called out from a list of strings:

```
# loops.py

students  = ["Anna", "Colin", "Ed", "Zsolt"]

for student in students:
    print(student.upper())
```

The loop will execute like so:

- first time, the value of `student` == "Anna"
- second time `student` == "Colin"
- third time `student` == "Ed" and so on...

If we add more students we don't have to tell it to loop more times, it will just do it automatically!

Looping Through a List of Dictionaries

One thing we might see is a list of dictionaries. So a collection of information about an item is stored in a dictionary, and then several of these items are stored in a list. This can be really useful to loop through.

```
# loops.py

students = [
    {"name": "Anna", "result": 92},
    {"name": "Colin", "result": 93},
    {"name": "Ed", "result": 89},
    {"name": "Zsolt", "result": 45},
]

for student in students:
    print(f"{student['name']} got {student['result']} on their exam")
```

Please be aware that the student's names and exam results are hidden inside the `student` variable - if we are looping through a list of dictionaries, each iteration the `student` will represent a more complex type of data - so we have to use `student["name"]` or `student["result"]` to get the desired value out.

Task

How would we calculate the mean/average result of the exams?

Solution:

We need to add up the total of the results, then divide it by the number of students to get the mean of results. First, we create a running total, then at the end, using the `len()` function, we will figure out how many people took the exam:

```
running_total = 0

students = [
    {"name": "Anna", "result": 92},
    {"name": "Colin", "result": 93},
    {"name": "Ed", "result": 89},
    {"name": "Zsolt", "result": 45},
]

# Adding up the running total of results
for student in students:
    running_total += student["result"]

# Dividing by the number of students in the list
mean_result = running_total / len(students)

print(mean_result)
```

Conclusion

We've seen how can we execute larger codeblocks multiple times. When using a `while` loop, we do it until a condition becomes no longer `True`. With a `for` loop, we can execute a codeblock as many times as many elements we have in a list. This automates our code, making it more efficient!