Booleans and Loops

◎ Objectives

At the end of this lesson, you should be able to:

- **explain** the concept of boolean / binary
- **use** loops to build more complex scripts
- **build** a simple game

1. Checks - Motivation

For the moment, we have seen how to write very simple scripts. These scripts consist of a **single flow** of actions. However when we looked at flowcharts, we saw that a major part of representing problems is to check the state of the current environment before executing an action.

Example: Checking states

- is the water boiling?
- is the pasta cooked?
- does the number of hours exceed 12?

The point of these checks in programming is to be able to adapt the behaviour of a program according to the state of the "environment". Without surprise, it is possible in Python[1].

2. Booleans

Checking something is the same as verifying a statement. This implies we need to have a way of representing true and false statements.

Idea

The easiest way to think about this is to think about **binary**[2] statements such as:

- `Yes` *vs.* `No`
- `True` *vs.* `False`

In most programming languages, choice statements are represented by a **boolean** type. This type can usually only take two different values:

- `True`
- `False`

A simple example of the use of boolean variables is shown here:

[1] as well as in any programming language, obviously

[2] there is a strong correlation with the word, you can check wikipedia for more information on this subject

</> Code

```
adult = True
student = False
```

Where we instantiate two boolean variables to different values.

3. Conditions

Conditions are **boolean statements** that allow **branching** to happen in the code's flow.

Idea

They are the translation of plain language questions such as:

- Are you an adult?
- Is this value correct?
- Is the water boiling? Is the pasta cooked?[3]

The main concept is to think of "What should happen if this is true? What if it's false?". We then want different code snippets to be executed according to the result of the check.

Conditions use boolean variables and comparisons to establish a fact. The available comparisons are:

- Strictly greater: `a > b`
- Greater or equal: `a >= b`
- Stricly smaller: `a < b`
- Smaller or equal: `a <= b`
- Equal: `a == b`
- Different: `a != b`

These comparisons operators allow us to write **conditional code**. In Python, the syntax is based on 3 keywords: **`if`**, **`elif`**, **`else`**.

</> Code

```
if condition:
  code to execute
elif second\_condition:
  code to execute
else:
  code to execute
```

[3] see `https://irwin.sh/documents/teaching/rubika/algorithms.pdf` for reference ☺

Information

Note the **indentation** level for the lines between the keywords. Indentation is the way Python distinguishes between different code blocks.
The indentation you use should be the same all around your files. It is highly recommended to use 4 spaces[4].

Code

```
alive = True
nb_lives = 1

if alive:
    print("I'm alive")
    if nb_lives > 1:
        print(f"I have {nb_lives} lives left")
    elif nb_lives == 1:
        print("On my last life!")
    else:
        print("I'm dead")
```

Exercise

Write down what the program outputs.
What happens if you change `nb_lives` to `3`?

4. Loops

As we have seen before, writing code is good, but writing the *same* code multiple times is bad. However, sometimes we have to repeat actions on different elements (elements of a list, for a certain number of times, etc...).

In programming languages, we do this with **loops**. Two main types of loops exist: **`for`** loops and **`while`** loops.

For loops

Sometimes you want to execute a portion of code a given number of times, or on every element in a list. You can use **`for`** loops to do this.

[4] it is easier to configure VSCode to replace a tab character by 4 spaces so you just have to use the tab key

Idea

for loops allow us to translate steps such as these:

Example:

I have to go to school **for all** my classes!

They are best used when you know the exact number of times the code portion should be executed.

In Python, the syntax is quite simple:

Code

```
for i in range(10):
    print(i, sep=" ")
```

Output

```
0 1 2 3 4 5 6 7 8 9
```

While loops

Sometimes, you must execute a portion of code without knowing the exact number of repetitions. This is usually the case when dealing with user input, waiting for an action to be finished, etc...

Idea

while loops allow to translate steps such as these:

Example:

While it's sunny, I'll play outside

The syntax is similar than the one for **for** loops.

Code

```
while alive:
    if is_hit:
        alive = False
     else:
        print("yay")
```

Exercise - the ~~final~~ first countdown

Write code that prints a countdown for a firework show:

>_ Output

```
3
2
1
wouhou !
```

Exercise

Write code that asks the user for a word in input and prints the number of characters in the word.

Exercise

Write code that prints the number of spaces in a sentence. You can ask the user for the sentence.