Let’s continue then with the control of programme execution. This aspect is a bit long, so to make the explanation easier to follow, I have divided it in 3 parts.

[SLIDE 2] Let’s start with Part 1 then.

[SLIDE 3] For this part, I need to explain how to use files with names different from main.py in replit. Please, notice that if you are using another developing environment, this explanation might not be useful.

[SLIDE 4] So far, all our Python files have been called main.py in replit.

But there are cases where we would like to have more than one Python file in our project. How do we execute such file in replit?

[SLIDE 5] Well, let’s first create that file by clicking here.

[SLIDE 6] Next, let’s call that file hello.py,

[SLIDE 7] write that single line of code on it and click on the button Run.

[SLIDE 8] You will see that the file that gets executed is not hello.py, as the message that has appeared on screen is different.

[SLIDE 9] In fact, in this example, the file that got executed was main.py. To execute hello.py you need to

[SLIDE 10] click on the Shell tab, that will take you to a Unix environment, and

[SLIDE 11] execute the file from there by writing “python hello.py”. You will see now that the file hello.py has been executed. This is the end of part 1.

[SLIDE 12] Let’s now move to part 2.

[SLIDE 13] Before the interpreter of Python executes the code, it sets up a few special variables.

[SLIDE 14] One of those variables is “double underscore name double underscore”. Although the name of the variable is “double underscore name double underscore”, calling it like that it is a bit tedious. So, from now on I will call it simply name. But when you use it, please remember that you must write the double underscore before and after name.

[SLIDE 15] The value of the variable name depends on how the Python file is executed. Let’s see the details.

[SLIDE 16] First, please create a file called file_one.py, write

[SLIDE 17] this line of code on it and

[SLIDE 18] execute it. You will see that the variable name has been initialized to “double underscore main double underscore”. This means that the code being executed is in the same file we called when we wrote “python file_one.py” in the shell. This has been the case with all Pythn programmes we have seen so far.

[SLIDE 19] Now, let’s create a second file called file_two.py, and write almost the same line of code we wrote in file_one.py, but now the line of code must say “file two” instead of “file one”.

[SLIDE 20] If we execute file_two.py, as expected, the variable name will take the value “double underscore main double underscore”, because the code being executed is in the same file we called from the shell.

[SLIDE 21] Now, please modify file_one.py by importing file_two on it. You do that by writing “import file_two” in the first line of file_one.py. By doing so, the code of file_two will be included in file_one. This is similar to what you do in C when you include a library. Here, you are including another Python file.

Please, execute this new version of file_one.py now.

[SLIDE 22] You will see that the value of the variable name for file_one remains to be “double underscore main double underscore”, as the code is in the file being called with “python file_one.py”. However, the variable name in file two has changed. Since the file being called was file_one and not file_two, the variable name in file two is now equal to file_two, meaning that the code of file two is executed by being included in another file.

[SLIDE 23] That is, the variable name of the file being called is equal to “double underscore main double underscore”, but the variable name of the files being imported is equal to the file’s name.

The fact that the variable name takes different values depending on how the code is executed is useful to control what parts of imported modules are executed. We will see a couple of examples in Part 3.

[SLIDE 24] Finally, we get to the final part to understand how to use what we have learnt before.

[SLIDE 25] Here you can see file_one.py and file_two.py with a bit more of code. Let’s have a look at this first.

[SLIDE 26] As before, file_one.py imports file_two.py

[SLIDE 27] Second, both files include the instruction that prints the value of variable name on screen.

[SLIDE 28] Next, the files define functions. Function A on file_one.py and functions B and C in file_two.py

[SLIDE 29] And finally, we get to the interesting part. Here, depending on the value of the variable name, we can control what functions of the files are executed.

Notice how, depending on whether file_two is imported or not, different functions will be executed.

Now, assume we execute file_one.py. What will be printed on screen? Please, stop the video while you think about this and come back when you have an answer.

[SLIDE 30] First, the value of the variable name in file_two will be printed. Since file_two has been imported, the variable will be equal to the file name. That is, file_two.

[SLIDE 31] Second, only the part of file two that is executed when the file is imported will be executed. That is, the message that file two has been imported and the execution of function C.

[SLIDE 32] Next, the value of the variable name for file_one is printed on screen.

[SLIDE 33] And finally, the part of the code corresponding to the case of file_one being called for execution will be printed.

[SLIDE 34] If you want to read more details about this, you might go these links.

[SLIDE 35] Ok, let’s move now to how to enter arguments to your Python programme using the command line, as you did with argc and argv in C.

[SLIDE 36] As in C, you can enter input parameters to your Python programme using the command line, as shown here.

[SLIDE 37] To do so, we need to import the module sys,

[SLIDE 38] and use argv.

[SLIDE 39] To check that the user has entered the number of parameters expected by the programme, you use sys.argv. This is an array with as many elements as the number of parameters plus one. The function len returns the number of elements in the array argv. Thus, this is equivalent to argc in C.

[SLIDE 40] To retrieve the data entered by the user in the command line, you access the elements of that array. As in C, the element of the array in position 0 is the name of the file being executed. The parameters are stored from position 1 and onwards.

[SLIDE 41] Here, you can see what happens when the user writes an incorrect number of parameters.

[SLIDE 42] And here, what happens when the user writes the number of parameters expected by the programme.

[SLIDE 43] With this, we finish the videos of Week 1. I hope they helped you learn the basics of Python and get started with the first assessment.