TMTplus Introduction to Scientific Programming

Mahdi Farnaghi, Mahdi Khodadadzadeh & Robert Ohuru

March 2021

Note from the teaching staff

The materials that we use for this course are somewhat new and have been heavily reworked. We will likely have made mistakes in this work or have glossed over issues that deserved better or different treatment. Where you find issues worth noting please do report these to us as it will allow to repair and improve.

By the way, welcome to the Exercise book for Introduction to Scientific Programming! We believe this book is self-explanatory, so go ahead and practice.

Exercise 3

Conditionals

Computer programs can be coded to make decisions on the basis of data available to them. The prototypical command for this, in many languages, is the branching command also known as IF-THEN-ELSE. This exercise set is about this construct and related commands that apply conditions.

The exercises

The main goal of each exercise is to use the Python programming language to review concepts you have learned during the lectures. In particular, you will work with *condtions and statements that make use of these* in this exercise.

Once you have executed an exercise satisfactorily, we encourage you to continue a little bit and experiment with the code that you have in your PyCharm project. Every now and then, pose yourself the question what else you could try to deepen your understanding. Typical questions to pose will start with "what will happen if I change . . . ?"

3.1 A simple start

Ex 3.1 The main goal of the exercises is to learn how to solve problems with conditionals. Please try to understand the exercises and *don't forget to use the correct indentation*. First enter the script below and save to file. Do not run yet.

```
def is_even(x):
    return x % 2 == 0
```

```
a=is_even(22)
print (a)

b=is_even(21)
print(b)
```

Before executing the code, try to predict its output. Compare your prediction with the results and explain what each script line does. Ex 3.2

code lines	predicted output	explanation
`def is_even(x):` `  return x % 2 == 0`		
`a = is_even(22)` `print(a)`		
`b = is_even(21)` `print(b)`		

Ex 3.3

The function `is_even()` has a return statement. Describe what a return statement is and does.

3.2 Comparisons

During the lecture, you learned that there are different comparison operators:

==	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

A result of a comparison will be `true` or `false`. In this exercise, we address how to decide a student passes or fails a taught module. If the module mark is less than 55, the student fails the module, otherwise, she passes. This decision process is captured in the below flowchart of Figure 3.1.

Please develop a Python script that corresponds with the above flow diagram, and that reports a pass or fail depending on given mark. Ex 3.4

Nested `if` statements are often used to test different possibilities. In the next example, illustrated in Figure 3.2, if the module mark is less than 55 the student will fail, otherwise the student will pass the module. In addition, if the mark is Ex 3.5

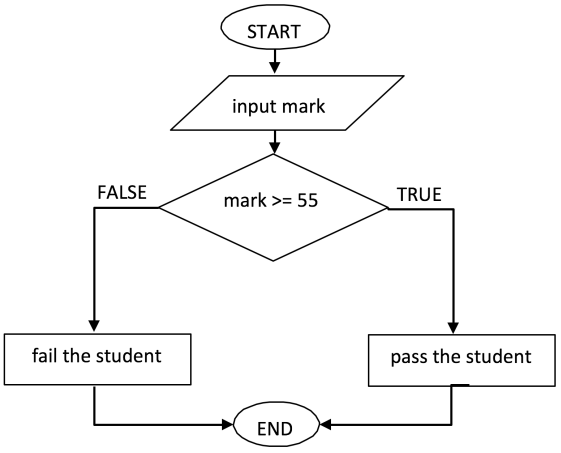

Figure 3.1: Flowchart to pass or fail a student

equal to or more than 80, the student will be considered marked as excellent.

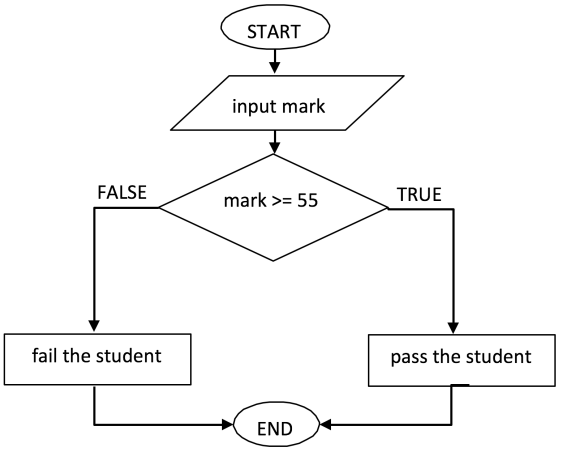

Figure 3.2: Flowchart to pass, fail or declare as excellent a student

Develop the script for this situation also, now reporting on one of three cases.

3.3 Mapping the days

Ex 3.6 Suppose we have numbered the days of the week from 0 to 6, and we want to map the day number to a string as follows: 0 – Monday, . . . , 6 – Sunday.

Develop the Python script as a function called `day_of_week()` that maps the number of the week to a string using nested conditionals. *Note*: you can check the slides on conditionals. Run a few tests to verify for code correctness. Ensure you use proper indentation.

We are making a bet that your code initially did not cover the case for an input with value 12. Can you repair for this, and provide some type of useful feedback? This is an issue of creating *robust* code. One way to resolve this matter is to simply return a string that is something different.

Another way to repair, more a good practice, is to ensure that you generate a proper Python runtime error when you (that is, your code) detect the problem. The phrasing here is that you need to "raise an exception." Read up on Pythonic exceptions, and try to code for raising a ValueError exception.

Ex 3.7

Although nested if statements are useful, they are cumbersome when many possible combinations exist. Then, *chained conditionals* can be easier to implement. Change the above code from nested conditionals into chained conditionals. Check the slides on conditionals once more, run again and test.