

SOFTWARE 1 PRACTICAL

SELECTION

Week 2 – Practical 01c

Flow-Control Structures

The programs that we can achieve so far are not very interesting. Every command is executed exactly once. This is very limited. We are now going to introduce **control structures** so statements might not be executed, executed once or many times.

The first control structure we will look at will enable us to choose which statement to execute depending on a condition. Let look at our latest program. For our output we have to print cake (s) in case we have one or more cakes. It will be better if we could decide to print cake or cakes depending on the number of cakes entered by the user.

In plain English, we could write:

IF the number of cake is one, print cake, OTHERWISE/ELSE print cakes

You can notice that the underline part of the sentence is a condition, e.g. is True or False, and depending on the value we will be doing two different things. If it is **True**, we will print cake (singular), if it is **False** we will print cakes (plural). So depending on the value of the condition, one statement will be ignored and the other will be executed, it is sometime called branching.

How is it done in Python?

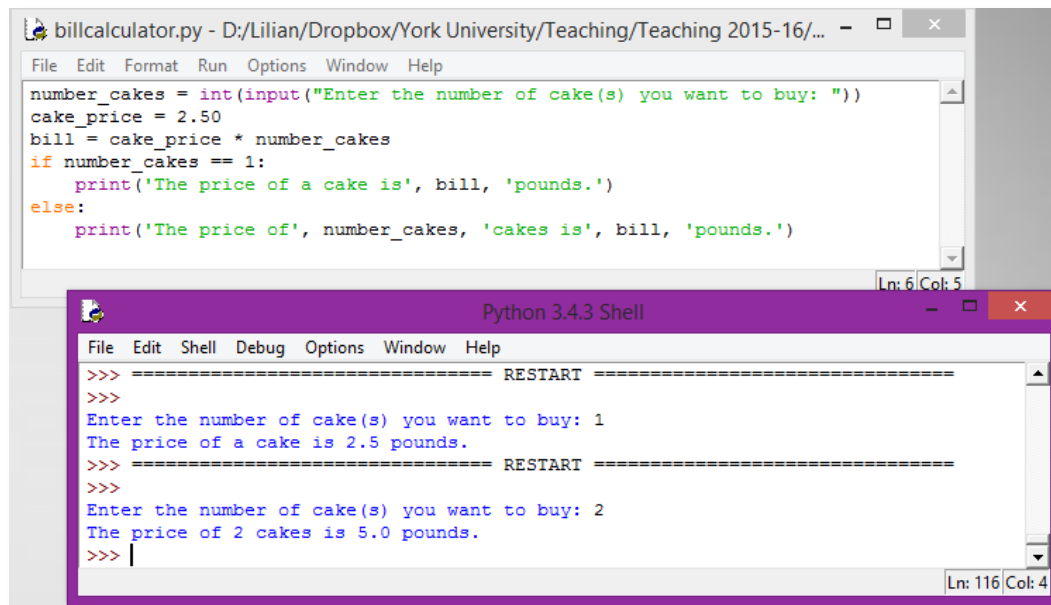
We have the `if-else` control structure.

```
if condition :  
    Statement A1  
    ...  
    Statement An  
else :  
    Statement B1  
    ...  
    Statement Bk
```

First thing to remember are the two keywords, `if` and `else` with the colon at the end of the line. The second is the condition, an expression that can be evaluated to **True** or **False**. Such expressions are called **Boolean expression** (can return only the value `True` or `False`). Finally a series of statements after the `if` and the `else`. Note, and this is important, that the statements are **indented to the right** compared to the `if` and the `else`. Indentation is Python's way of representing blocks of code. The expression above means, if condition is `True` then execute block of code A₁ to A_n, else execute block of code B₁ to B_k.

Example

Figure 18 shows of an if-else statement can be used to improve our program. The condition is `number_cakes == 1`. Note the use of **double equal operator**; it is to inform the interpreter that it is not an assignment but an equality comparison (you will make this error often). In our case the blocks of statements contains only one statement each, but they could have had more than one.



The image shows two overlapping windows. The top window is a text editor titled 'billcalculator.py' showing the following Python code:

```
number_cakes = int(input("Enter the number of cake(s) you want to buy: "))
cake_price = 2.50
bill = cake_price * number_cakes
if number_cakes == 1:
    print('The price of a cake is', bill, 'pounds.')
else:
    print('The price of', number_cakes, 'cakes is', bill, 'pounds.')
```

The bottom window is a 'Python 3.4.3 Shell' showing the program's execution. It displays two runs separated by 'RESTART' lines. In the first run, the user enters '1' and the output is 'The price of a cake is 2.5 pounds.' In the second run, the user enters '2' and the output is 'The price of 2 cakes is 5.0 pounds.'

Figure 1

We can see in Figure 18 the result of running the program twice using two different inputs. The results obtained are the expected ones. The introduction of control structures, and especially branching, presents a new problem: TESTING!

To ensure that our program works properly, we must design a series of tests such that all branches of our program are traversed. We will look at testing in more details later in the term; nonetheless you should keep that problem in mind for the time being.

Be wary of the mighty user

When you are interacting with a user, you can be sure it will input something you did not expect. How can I protect my program? In our example we assumed that the user will input a value greater or equal to 1. What if the input is -1? The program will execute without errors/crashing, however the result will be incorrect/not making sense.

We can use another control structure: `if-elif-else`.

```

if conditionA :
    Statement A1
    ...
    Statement An
elif conditionB :
    Statement B1
    ...
    Statement Bm
else :
    Statement C1
    ...
    Statement Ck

```

An example on how to use the `if-elif-else` control structure for our problem is shown in Figure 19. You can use as many `elif` between the `if` and `else` as you need (almost). Now we can consider that our program is protected (to a certain extent) against user inputs. If the user enters an incorrect number we are displaying an error message, otherwise we provide a correct answer. Note again that our test case is made of three inputs in order to go through all branches of our program, e.g. the `if` branch, the `elif` branch, and the `else` branch.

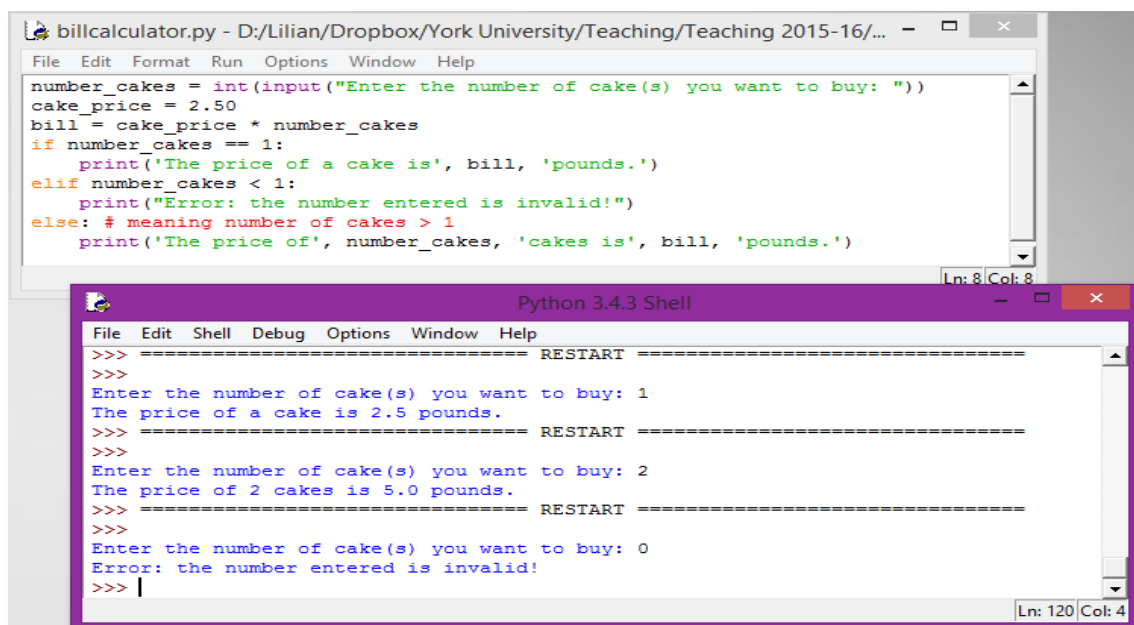


Figure 2

NOTE: what is the statement in red after the `else`? In Python, as in all languages, we can have comments. A comment is just plain text that will be ignored by the interpreter, e.g. not executed. They are mainly there to help understanding the code, especially if you are reading someone else's code. The symbol `#` is used in Python (it differs between languages, e.g. Java, C++, etc.). Everything that follows `#` and is on the same line of code is considered a comment and will be ignored by the interpreter. In **Idle** comments are highlighted in **red**.

- Search how blocks of comments are represented in Python.