Chapter 3 - Flow Control

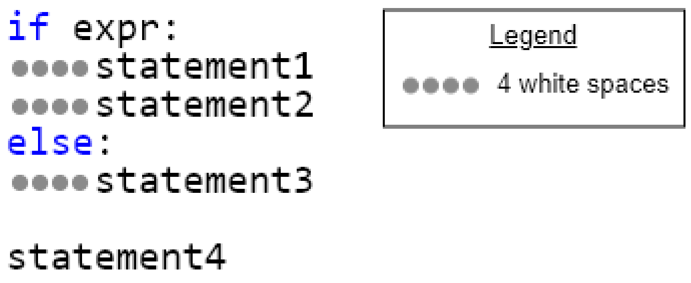
3 Flow Control - Introduction

Flow Control of a program is the order or flow at which the statements are executed or evaluated when the program is running. Python uses a strict indentation rule to group blocks of statements together thus understanding how indentation is structured is very important.

3.1 Indentation

Before we start on flow control of a program, we first have to understand how Python groups statements (codes) together. In the previous chapters, we have been using individual Python statements that can be executed on their own but now we want to use special statements that are able to change the direction of the flow of a program. This involves grouping of statements that are related to each other, together into blocks to perform certain tasks.

Python uses indentation to achieve this grouping of related statements. Indentation is seen as the space to the left of a statement. Statements of the same group have the same level of indentation. This can be seen in figure 1 below where the grey dots shows that there are 4 whitespaces to the left of the statements. The first group of statements consist of `statement1` and `statement2`, and the second group of statements consist of `statement3`.

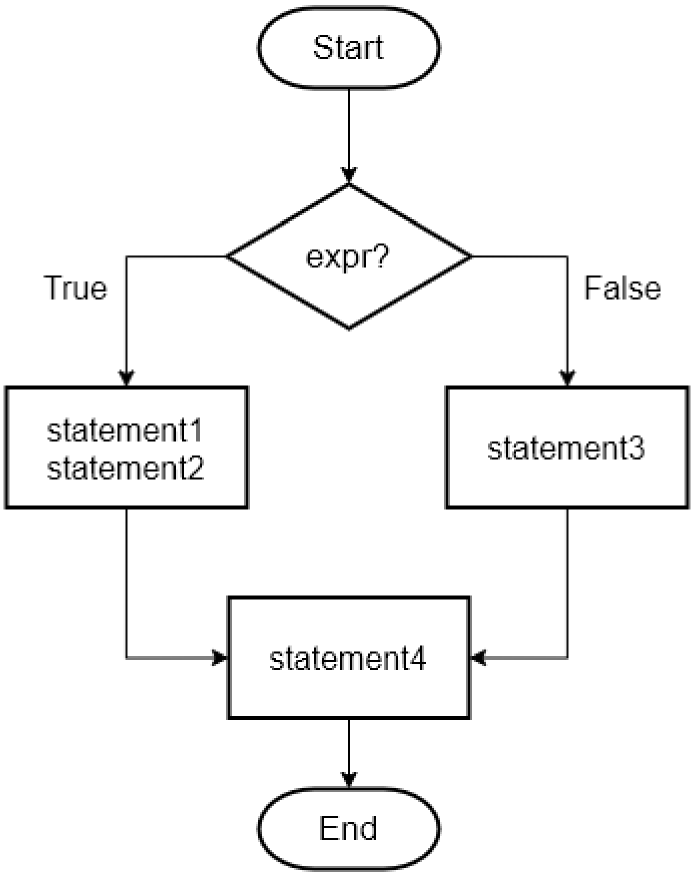


```
if expr:
●●●●statement1
●●●●statement2
else:
●●●●statement3

statement4
```

Figure 1: Showing how indentation is used to group related statements.

From figure 1 above, the first 2 statements (`statement1` and `statement2`) after the `if expr:` and `statement3` after the `else:` belongs to different blocks. The program flow (refer to flowchart 1 on the next page), should the `expr` resolves to `True` will execute the first 2 statements followed by `statement4` then terminate. Should the `expr` resolves to `False`, `statement3` after the `else:` will be executed followed by `statement4` then terminate as well.



Flowchart 1

Important note there is a difference between tab spacing and 4 whitespaces spacing for indentations. When you start writing programs, pick a type of indentation and **stick to it** otherwise the Python interpreter will prompt an indentation error. 4 whitespaces indentation spacing is used in the figure above and on all subsequent blocks of statements.

3.2 Conditional Statements

Conditional statements are statements that specify which action to take based on the evaluation of a single or a group of multiple expressions that will produce a **single** `True` or `False` outcome. Take note that Python, assumes any **non-zero** and **non-null** values as `True` and any **zero** and **null** values as `False`.

Python has 3 types of conditional statements:

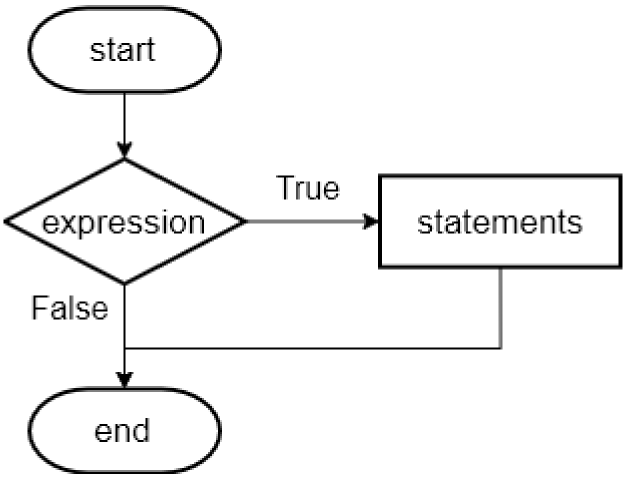
- `if` statements - resolves a/several boolean expression/s followed by the execution of one or more statements.
- `if...else` or `if..elif...else` statements - resolves a/several boolean expression/s followed by the execution of the respective block of statement/s per `True` or `False` value from the expression.
- nested `if` statements - any combination of `if` , `if...else` and `if...elif...else` statements inside a `if` or `if...else` or `if...elif...else` statements.

3.2.1 `if` statements

The syntax for an `if` statement is as follows:

```
if expression:
    statement(s)
```

The expression/s is evaluated in Boolean context (refer back to the chapter on logical, membership and identity operators) and this determines if the statement or block of statements are to be executed (refer to the flowchart 2 below).



Flowchart 2 - `if` statement

A code example is shown below

```python
x = 18
y = 10
z = 5

# single expression (evaluates to False)
if x < y:
    print("x is greater.")

# multiple expressions (evaluates to True)
if y > z and y < x:
    print ("y is greater than z and less than x.")

```

and the output is

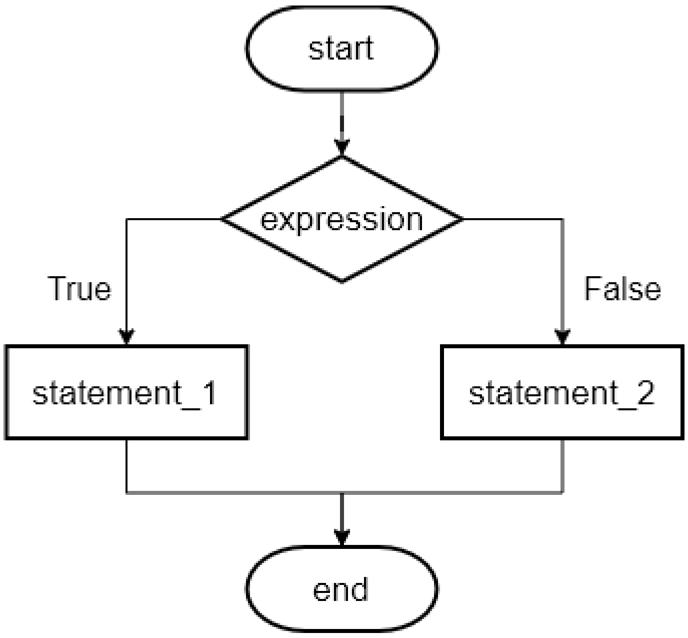
```
y is greater than z and less than x.
```

Note that when an expression evaluates to `False`, the statement is **not** executed. To execute a statement/s with a resulting `False` expression, we need to use the `if...else` statement.

3.2.2 `if...else` or `if..elif...else` statements

When there are alternative statements to execute in the event of a False expression, the `else` clause is combined with the `if` statement. This allows the `if` statement to resolve both True and False conditions of the expression. The syntax and flowchart is shown below.

```
if expression:
    statement_1     # statement_1 to execute when True
else:
    statement_2     # statement_2 to execute when False
```



Flowchart 3 - `if...else` statement

Example:

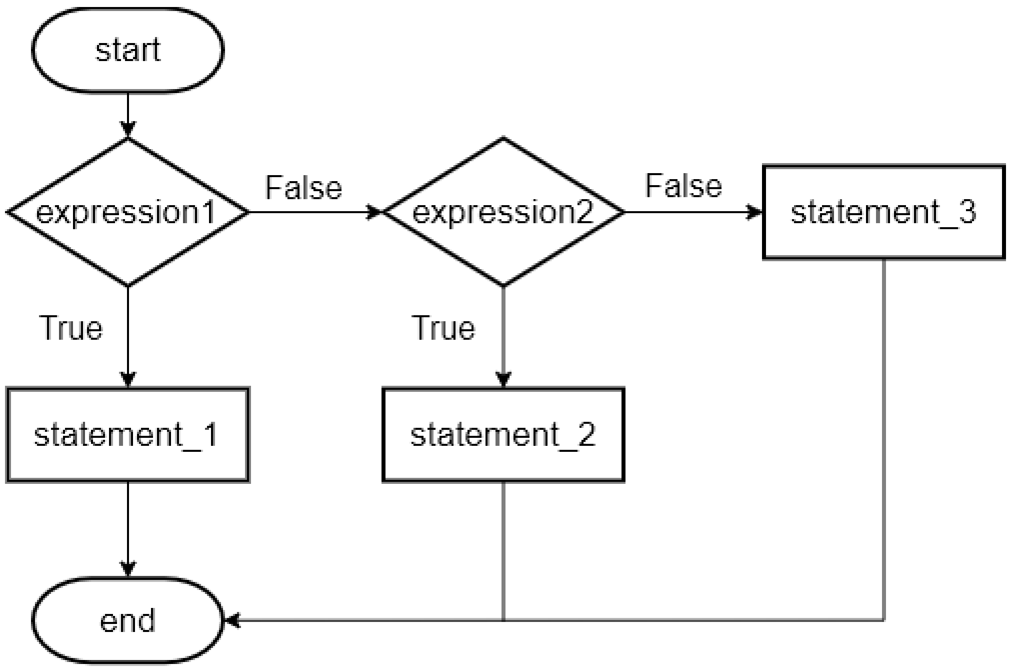
```
cost = 50   # change this value to give a different result
if cost > 80:
    print("10% discount given")
else:
    print("5% discount given")
```

and the output is

```
5% discount given
```

Sometimes there will be cases where you will need to check for multiple expressions and only execute the statement/s as soon as an expression resolves to a `True` value. For these cases, the `elif` clause (short for `else if`) is used to shorten multiple `if...else` blocks. There can be any number of `elif` clauses following an `if` statement. The syntax and flowchart is shown on the next page.

```python
if expression1:
    statement_1     # statement_1 to execute when expression1 is True
elif expression2:
    statement_2     # statement_2 to execute when expression2 is True
else:
    # statement_3 to execute when expression1 and expression2 is False
    statement_3
```



Flowchart 4 - `if...elif...else` **statement**

Example:

```python
grade = 83      # change this value to give a different result
if grade < 50:
    print("F")
elif grade >= 50 and grade < 65:
    print("D")
elif grade >= 65 and grade < 80:
    print("C")
elif grade >= 80 and grade < 95:
    print("B")
else:
    print("A")
```

and the output is

```
B
```

3.2.3 Nested `if` statements

There will be times where we would like to check for other expression/s after a previous expression resolves to a `True`. In this situation, nested `if` statements can be used. The syntax is as follows:

```
if expression1:
   statement(s)
   if expression2:
      statement(s)
   elif expression3:
      statement(s)
   else
      statement(s)
elif expression4:
   statement(s)
else:
   statement(s)
```

Example:

```python
str_val = 'quick fox jumps'
if 'fox' in str_val:
    if len(str_val) >= 15:
        print("The cow jumps over the moon.")
    else:
        print("The cow climbed a tree")
else:
    print("There is no 'fox' in the sentence")
```

and the output is

```
The cow jumps over the moon.
```

3.3 References

1. Python 3 - Decision Making, https://www.tutorialspoint.com/python3/python_decision_making.htm
2. Lubanovic, 2019, 'Chapter 4. Choose with if' in Introducing Python, 2nd Edition, O`Reilly Media, Inc.