Chapter 6
Flow of Control Using If Statements

6.1 Introduction

In this chapter we are going to look at the `if` statement in Python. This statement is used to control the flow of execution within a program based on some condition. These conditions represent some choice point that will be evaluated to `True` or `False`. To perform this evaluation it is common to use a comparison operator (for example to check to see if the temperature is greater than some threshold). In many cases these comparisons need to take into account several values and in these situations logical operators can be used to combine two or more comparison expressions together.

This chapter first introduces comparison and logical operators before discussing the `if` statement itself.

6.2 Comparison Operators

Before exploring `if` statements we need to discuss *comparison operators*. These are operators that return Boolean values. They are key to the conditional elements of flow of control statements such as `if`.

A comparison operator is an operator that performs some form of test and returns `True` of `False`.

These are operators that we use in everyday life all the time. For example, do I have enough money to buy lunch, or is this shoe in my size etc.

J. Hunt, *A Beginners Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-20290-3_6

In Python there are a range of comparison operators represented by typically one or two characters. These are:

Operator	Description	Example
`==`	Tests if two values are equal	`3 == 3`
`!=`	Tests that two values are *not* equal to each other	`2 != 3`
`<`	Tests to see if the left-hand value is less than the right-hand value	`2 < 3`
`>`	Tests if the left-hand value is greater than the right-hand value	`3 > 2`
`<=`	Tests if the left-hand value is less than *or* equal to the right-hand value	`3 <= 4`
`>=`	Tests if the left-hand value is greater than or equal to the right-hand value	`5 >= 4`

6.3 Logical Operators

In addition to comparison operators, Python also has logical operators.

Logical operators can be used to combined Boolean expressions together. Typically, they are used with comparison operators to create more complex conditions. Again, we use these every day for example we might consider whether we can afford an ice cream *and* whether we will be having our dinner soon etc.

There are three logical operators in Python these are listed below:

Operator	Description	Example
`and`	Returns True if both left and right are true	`(3 < 4) and (5 > 4)`
`or`	Returns two if either the left or the right is truce	`(3 < 4) or (3 > 5)`
`not`	Returns true if the value being tested is False	`not 3 < 2`

6.4 The If Statement

An `if` statement is used as a form of conditional programming; something you probably do every day in the real world. That is, you need to decide whether you are going to have tea or coffee or to decide if you will have toast or a muffin for breakfast etc. In each of these cases you are making a choice, usually based on some information such as I had coffee yesterday, so I will have tea today.

In Python such choices are represented programmatically by the `if` condition statement.

In this construct if some condition is true some action is performed, optionally if it is not true some other action may be performed instead.

6.4.1 Working with an If Statement

In its most basic from, the `if` statement is

```
if <condition-evaluating-to-boolean>:
  statement
```

Note that the condition must evaluate to `True` or `False` (or an equivalent value—see later in this chapter). If the condition is `True` then we will execute the indented statement.

Note that indentation, this is *very* important in Python; indeed, layout of the code is *very*, *very* important in Python. Indentation is used to determine how one piece of code should be associated with another piece of the code.

Let us look at a simple example,

```
num = int(input('Enter a number: '))
if num < 0:
    print(num, 'is negative')
```

In this example, the user has input a number; if it is less than zero a message noting this will be printed to the user. If the number is positive; then nothing will be output.

For example,

```
Enter a number: -1
-1  is negative
```

If we wish to execute multiple statements when our condition is `True` we can indent several lines; in fact all lines indented to the same level after the `if` statement will automatically be part of the `if` statement. For example;

```
num = int(input('Enter another number: '))
if num > 0:
    print(num, 'is positive')
    print(num, 'squared is ', num * num)

print('Bye')
```

If we now run this program and input 2 then we will see

```
Enter another number: 2
2  is positive
2  squared is  4
Bye
```

However, if we enter the value −1 then we get

```
Enter another number: -1
Bye
```

Note that neither of the indented lines was executed.

This is because the two indented lines are associated with the `if` statement and will only be executed if the Boolean condition evaluates (returns) `True`. However, the statement `print('Bye')` is not part of the `if` statement; it is merely the next statement to executed after the `if` statement (and its associated `print()` statements) have finished.

6.4.2 Else in an If Statement

We can also define an *else* part of an `if` statement; this is an optional element that can be run if the conditional part of the `if` statement returns `False`. For example:

```
num = int(input('Enter yet another number: '))
if num < 0:
    print('Its negative')
else:
    print('Its not negative')
```

Now when this code is executed, if the number entered is less than zero then the first `print()` statement will be run otherwise (else) the second `print()` statement will be run. However, we are *guaranteed* that at least one (and at most one) of the `print()` statements will execute.

For example, in run 1 if we enter the value 1:

```
Enter yet another number: 1
Its not negative
```

And in run 2 if we enter the value −1:

```
Enter yet another number: -1
Its negative
```

6.4.3 The Use of elif

In some cases there may be several conditions you want to test, with each condition being tested if the previous one failed. This *else-if* scenario is supported in Python by the `elif` element of an `if` statement.

The elif element of an if statement follows the if part and comes before any (optional) else part. It has the format:

```
 elif <condition-evaluating-to-boolean>:
   statement
```

For example

```
savings = float(input("Enter how much you have in savings: "))
if savings == 0:
    print("Sorry no savings")
elif savings < 500:
    print('Well done')
elif savings < 1000:
    print('Thats a tidy sum')
elif savings < 10000:
    print('Welcome Sir!')
else:
    print('Thank you')
```

If we run this:

```
 Enter how much you have in savings: 500
 Thats a tidy sum
```

Here we can see that the first if condition failed (as savings is not equal to 0). However, the next elif also must have returned False as savings were greater than 500. In fact it was the second elif statement that returned True and thus the associated print('Thats a tidy sum') statement was executed. Having executed this statement the if statement then terminated (the remaining elif and else parts were ignored).

6.5 Nesting If Statements

It is possible to *nest* one if statement inside another. This term *nesting* indicates that one if statement is located within part of the another if statement and can be used to refine the conditional behaviour of the program.

An example is given below. Note that it allows some behaviour to be performed before and after the nested if statement is executed/run. Also note that indentation is key here as it is how Python works out whether the if statements are nested or not.

```
snowing = True
temp = -1
if temp < 0:
    print('It is freezing')
    if snowing:
        print('Put on boots')
    print('Time for Hot Chocolate')
print('Bye')
```

In this example, if the temperature if less than Zero then we will enter the `if` block of code. If it is not less than zero we will skip the whole `if` statement and jump to the `print('Bye')` statement which is after both If statements.

In this case the temperature is set to `−1` and so we will enter the If statement. We will then print out the 'It is freezing' string. At this point another `if` statement is *nested* within the first `if` statement. A check will now be made to see if it is `snowing`. Notice that `snowing` is already a Boolean value and so will be either `True` or `False` and illustrates that a Boolean value on its own can be used here.

As it is snowing, we will print out 'Put on boots'.

However, the statement printing out 'Time for Hot Chocolate' is not part of the nested `if`. It is part of the outer `if` (this is where indentation is important). If you wanted it to only be printed out if it was snowing then it must be indented to the same level as the first statement in the nested `if` block, for example:

```
snowing = True
temp = -1
if temp < 0:
    print('It is freezing')
    if snowing:
        print('Put on boots')
        print('Time for Hot Chocolate')
print('Bye')
```

This now changes the inner (or nested) if to have two `print` statements associated with it.

This may seem subtle, but it is *key* to how Python uses layout to link together individual statements.

6.6 If Expressions

An if expression is a short hand form of an `if` statement that returns a value. In fact, the difference between an expression and a statement in a programming language is just that; expressions return a value; statements do not.

It is quite common to want to assign a specific value to a variable dependent on some condition. For example, if we wish to decide if someone is a teenager or not then we might check to see if they are over 12 and under 20. We could write this as:

```
age = 15
status = None
if (age > 12) and age < 20:
    status = 'teenager'
else:
    status = 'not teenager'
print(status)
```

If we run this, we get the string 'teenager' printed out.

However, this is quite long and it may not be obvious that the real intent of this code was to assign an appropriate value to `status`.

An alternative is an *if expression*. The format of an if expression is

```
<result1> if <condition-is-met> else <result2>
```

That is the result returned from the if expression is the first value unless the condition fails in which case the result returned will be the value after the else. It may seem confusing at first, but it becomes easier when you see an example.

For example, using the if expression we can perform a test to determine the value to assign to `status` and return it as the result of the if expression. For example:

```
status = ('teenager' if age > 12 and age < 20 else 'not
teenager')
print(status)
```

Again, the result printed out is 'teenager' however now the code is much more concise, and it is clear that the purpose of the test is to determine the result to assign to `status`.

6.7 A Note on True and False

Python is actually quite flexible when it comes to what actually is used to represent `True` and `False`, in fact the following rules apply

- 0, '' (empty strings), `None` equate to `False`
- Non zero, non empty strings, any object equate to `True`.

However, we would recommend sticking to just `True` and `False` as it is often a cleaner and safer approach.

6.8 Hints

One thing to be very careful of in Python is layout.

Unlike language such as Java and C# the layout of your program is *part of your program*. It determines how statements are associated together and how flow of control elements such as if statements effect which statements are executed.

Also, be careful with the `if` statement and its use of the ':' character. This character is key in separating out the conditional part of the `if` statement from the statements that will be executed depending upon whether the condition is `True` or `False`.

6.9 Online Resources

See the Python Standard Library documentation for:

- https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not Boolean Operations.
- https://docs.python.org/3/library/stdtypes.html#comparisons Comparison operators.
- https://docs.python.org/3/tutorial/controlflow.html the online Python flow of control tutorial.

6.10 Exercises

There are three different exercises in this section, you can select which you are interested in or do all three.

6.10.1 Check Input Is Positive or Negative

The aim of this exercise is to write a small program to test if an integer is positive or negative.

Your program should:

1. Prompt the user to input a number (use the input() function). You can assume that the input will be some sort of number.
2. Convert the string into an integer using the int() function.
3. Now check whether the integer is a positive number or a negative number.
4. You could also add a test to see if the number is Zero.

6.10.2 Test if a Number Is Odd or Even

The exercises requires you to write a program to take input from the user and determine if the number is odd or even. Again you can assume that the user will enter a valid integer number.

Print out a message to the user to let them know the result.

To test if a number is even you can use

```
(num % 2) == 0
```

Which will return `True` if the number is even (note the brackets are optional but make it easier to read).

6.10.3 Kilometres to Miles Converter

In this exercise you should return to the kilometres to miles converter you wrote in the last chapter.

We will add several new tests to your program:

1. Modify your program such that it verify that the user has entered a positive distance (i.e. they cannot enter a negative number).
2. Now modify your program to verify that the input is a number; if it is not a number then do nothing; otherwise convert the distance to miles.

To check to see if a string contains only digits use the method `isnumeric()` for example `'42'.isnumeric()`; which returns `True` if the string only contains numbers. Note this method only works for positive integers; but this is sufficient for this example.