You can have another `if` statement inside the if-block of an `if` statement and so on - this is called a nested `if` statement.

Remember that the `elif` and `else` parts are optional. A minimal valid `if` statement is:

```
if True:
    print('Yes, it is true')
```

After Python has finished executing the complete `if` statement along with the associated `elif` and `else` clauses, it moves on to the next statement in the block containing the `if` statement. In this case, it is the main block (where execution of the program starts), and the next statement is the `print('Done')` statement. After this, Python sees the ends of the program and simply finishes up.

Even though this is a very simple program, I have been pointing out a lot of things that you should notice. All these are pretty straightforward (and surprisingly simple for those of you from C/C++ backgrounds). You will need to become aware of all these things initially, but after some practice you will become comfortable with them, and it will all feel 'natural' to you.

> **Note for C/C++ Programmers**
>
> There is no `switch` statement in Python. You can use an `if..elif..else` statement to do the same thing (and in some cases, use a dictionary to do it quickly)

The while Statement

The `while` statement allows you to repeatedly execute a block of statements as long as a condition is true. A `while` statement is an example of what is called a *looping* statement. A `while` statement can have an optional `else` clause.

Example (save as `while.py`):

```
number = 23
running = True

while running:
    guess = int(input('Enter an integer : '))

    if guess == number:
        print('Congratulations, you guessed it.')
        # this causes the while loop to stop
        running = False
    elif guess < number:
        print('No, it is a little higher than that.')
    else:
        print('No, it is a little lower than that.')
else:
    print('The while loop is over.')
    # Do anything else you want to do here

print('Done')
```

Output:

```
$ python while.py
Enter an integer : 50
No, it is a little lower than that.
Enter an integer : 22
No, it is a little higher than that.
Enter an integer : 23
Congratulations, you guessed it.
The while loop is over.
Done
```

How It Works

In this program, we are still playing the guessing game, but the advantage is that the user is allowed to keep guessing until he guesses correctly - there is no need to repeatedly run the program for each guess, as we have done in the previous section. This aptly demonstrates the use of the `while` statement.

We move the `input` and `if` statements to inside the `while` loop and set the variable `running` to `True` before the while loop. First, we check if the variable `running` is `True` and then proceed to execute the corresponding *while-block*. After this block is executed, the condition is again checked which in this case is the `running` variable. If it is true, we execute the while-block again, else we continue to execute the optional else-block and then continue to the next statement.

The `else` block is executed when the `while` loop condition becomes `False` - this may even be the first time that the condition is checked. If there is an `else` clause for a `while` loop, it is always executed unless you break out of the loop with a `break` statement.

The `True` and `False` are called Boolean types and you can consider them to be equivalent to the value `1` and `0` respectively.

> **Note for C/C++ Programmers**
>
> Remember that you can have an `else` clause for the `while` loop.

The `for` loop

The `for..in` statement is another looping statement which *iterates* over a sequence of objects i.e. go through each item in a sequence. We will see more about sequences in detail in later chapters. What you need to know right now is that a sequence is just an ordered collection of items.

Example (save as `for.py`):

```
for i in range(1, 5):
    print(i)
else:
    print('The for loop is over')
```

Output:

```
$ python for.py
1
2
3
4
The for loop is over
```

How It Works

In this program, we are printing a *sequence* of numbers. We generate this sequence of numbers using the built-in `range` function.

What we do here is supply it two numbers and `range` returns a sequence of numbers starting from the first number and up to the second number. For example, `range(1,5)` gives the sequence `[1, 2, 3, 4]`. By default, `range` takes a step count of 1. If we supply a third

number to `range`, then that becomes the step count. For example, `range(1,5,2)` gives `[1,3]`. Remember that the range extends *up to* the second number i.e. it does *not* include the second number.

Note that `range()` generates only one number at a time, if you want the full list of numbers, call `list()` on the `range()`, for example, `list(range(5))` will result in `[0, 1, 2, 3, 4]`. Lists are explained in the data structures chapter.

The `for` loop then iterates over this range - `for i in range(1,5)` is equivalent to `for i in [1, 2, 3, 4]` which is like assigning each number (or object) in the sequence to i, one at a time, and then executing the block of statements for each value of `i`. In this case, we just print the value in the block of statements.

Remember that the `else` part is optional. When included, it is always executed once after the `for` loop is over unless a break statement is encountered.

Remember that the `for..in` loop works for any sequence. Here, we have a list of numbers generated by the built-in `range` function, but in general we can use any kind of sequence of any kind of objects! We will explore this idea in detail in later chapters.

> **Note for C/C++/Java/C# Programmers**
>
> The Python `for` loop is radically different from the C/C++ `for` loop. C# programmers will note that the `for` loop in Python is similar to the `foreach` loop in C#. Java programmers will note that the same is similar to `for (int i : IntArray)` in Java 1.5.
>
> In C/C++, if you want to write `for (int i = 0; i < 5; i++)`, then in Python you write just `for i in range(0,5)`. As you can see, the `for` loop is simpler, more expressive and less error prone in Python.

The break Statement

The `break` statement is used to *break* out of a loop statement i.e. stop the execution of a looping statement, even if the loop condition has not become `False` or the sequence of items has not been completely iterated over.

An important note is that if you *break* out of a `for` or `while` loop, any corresponding loop `else` block is **not** executed.

Example (save as `break.py`):

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    print('Length of the string is', len(s))
print('Done')
```

Output:

```
$ python break.py
Enter something : Programming is fun
Length of the string is 18
Enter something : When the work is done
Length of the string is 21
Enter something : if you wanna make your work also fun:
Length of the string is 37
Enter something : use Python!
Length of the string is 11
Enter something : quit
Done
```

How It Works

In this program, we repeatedly take the user's input and print the length of each input each time. We are providing a special condition to stop the program by checking if the user input is `'quit'`. We stop the program by *breaking* out of the loop and reach the end of the program.

The length of the input string can be found out using the built-in `len` function.

Remember that the `break` statement can be used with the `for` loop as well.

Swaroop's Poetic Python

The input I have used here is a mini poem I have written:

```
Programming is fun
When the work is done
if you wanna make your work also fun:
    use Python!
```

The `continue` Statement

The `continue` statement is used to tell Python to skip the rest of the statements in the current loop block and to *continue* to the next iteration of the loop.

Example (save as `continue.py`):

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    if len(s) < 3:
        print('Too small')
        continue
    print('Input is of sufficient length')
    # Do other kinds of processing here...
```

Output:

```
$ python continue.py
Enter something : a
Too small
Enter something : 12
Too small
Enter something : abc
Input is of sufficient length
Enter something : quit
```

How It Works

In this program, we accept input from the user, but we process the input string only if it is at least 3 characters long. So, we use the built-in `len` function to get the length and if the length is less than 3, we skip the rest of the statements in the block by using the `continue` statement. Otherwise, the rest of the statements in the loop are executed, doing any kind of processing we want to do here.

Note that the `continue` statement works with the `for` loop as well.

Summary

We have seen how to use the three control flow statements - `if` , `while` and `for` along with their associated `break` and `continue` statements. These are some of the most commonly used parts of Python and hence, becoming comfortable with them is essential.

Next, we will see how to create and use functions.