Control Flow in Python

Guillaume Macneil

October 6, 2020

Abstract

This is the second lesson on the 'Introduction to Python' course. This course is loosely based around the 'MIT Introduction to Computer Science - Fall 2016' course. The course is more focused towards the Python programming side of things.

1 Strings

To begin, let's start with another type of object - *strings*. Strings are a combination of letters, special charaters, spaces, digits, etc all enclosed by either "" or ''.

```
cat = "The cat "
state = "sat on the mat."
cat_state = cat + state
print(cat_state)
# Prints 'The cat sat on the mat.'

laugh = "ha"
laugh_5 = laugh * 5
print(laugh_5)
# Prints 'hahahahaha'
```

As you can see, strings are interesting. As you can see in the 'cat_state' example strings can be concatenated (joined together) by using the '+' operator. Certain operations can also be performed on strings, like using the '*' operator to duplicate strings.

Another useful function is the *input()* function that allows the user to *input* data into a variable in your program. All the user has to do is type something and press enter.

```
string_input = input("What string do you want to enter? ")
type(string_input)
# Returns 'str'
print(string_input)
# Prints whatever the user inputted

integer_input = int(input("What integer do you want to enter? "))
type(integer_input)
# Returns 'int'
print(integer_input)
# Prints whatever the user inputted
```

By default *input()* returns a string. However, if you want to force the user to input an *int* or *float* you can type cast the input using *int()* or *float()* respectively. That said, if the user inputs a string when you cast to an integer, an error occur.

2 Comparison and Logic Operators

Comparison operators compare any two variables, lets say x and y and evaluate to a Boolean (either True or False). The common comparison operators are as follows:

- **x >y -** x is greater than y
- **x >= y -** x is greater than or equal to y
- **x <y -** x is less than y
- **x <= y -** x is less than or equal to y
- **x == y -** x is equal to y
- **x != y -** x is not equal to y

Logic operators compare two variables with Boolean values, lets sat a and b and evaluate to a Boolean. The logic operators are as follows:

- **not a -** returns the opposite Boolean value of a
- **a and b -** returns *True* if a and b are *True*, *False* otherwise
- **a or b -** returns *True* if either of both a and b are *True*, *False* otherwise

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

```
a = 4
b = 7
print(a >= 7)
# Prints 'False'

true_flag = True
complicated = a == b and true_flag
print(complicated)
# Prints 'False'
```

The first of the above examples is quite obvious, converted to english it says '4 is greater than or equal to 7' which is *False*. The second is quite strange, converted to english it says '4 is equal to 7 and True'. This can be simplified further as '4 is equal to 7' is *False*, so the evaluation becomes 'False and True' which as we know, is *False*.

3 Control Flow

What is 'Control Flow' you might be asking? In essence it is a fancy way of saying *if statements* and *loops*. So, let's begin with the former.

3.1 If, Elif and Else

```
a = 9
b = 3
c = 15

if a * b < c:
    print("Statement A reached")
elif a - b == c - a:
    print("Statement B reached")
else:
    print("Statement C reached")
```

This is arguably the most important example so far, as control flow is (in my opinion) one of the most fundamental elements of programming. In the above example, we have three variables a, b and c with values 9, 3 and 15 respectively. Below that we have what I will call an *'if, else block'* which performs comparisons on the variables and prints an output ***if*** a condition is met. The statement becomes fairly easy to read when you realise that ':' means 'then' in english.

In an *'if, else block'* there are 3 main components - 'if', 'elif' and 'else'. 'If' will allow print statement A to be reached *if* the condition evaluates to *True*. 'Elif' will allow print statement B to be reached *if* the prior condition was *False* but the current condition is *True*. 'Else' will allow print statement C to be reached *if* all other conditions were *False*. Based on this, I'll allow you to determine what each condition means in english - just know that the program prints "Statement B reached".

Note also that all of the code that is dependent on an 'if', 'elif' or 'else' statement is *indented* (pushed up a level). Indentation in Python **is not optional**, it is very important that you use it correctly.

3.2 Loops

A loop is a statement that repeats a block of code (also indented), provided that a given condition is met. In my opinion, they are as important as if statements (so pay attention).

The first loop that we'll start with is the *while loop*. As the name suggests, this loop repeats a block of code *while* the given condition is true. Here's an example:

```
count = 0
while count <= 10:
    print(f"Number: {count}")
    count += 1
# Prints "Number: 0", "Number: 1", ... , "Number: 9", "Number: 10"
```

This example can be shortened down with the use of our second kind of loop, the *for loop*. This loop performs a block of code *for* each x in y. This may be confusing, but here's the simplified example:

```
for number in range(0, 11):
    print(f"Number: {number}")
# Prints "Number: 0", "Number: 1", ... , "Number: 9", "Number: 10"
```

The first thing of note in the above example is that the variable 'number' could be called anything - it's just a variable. The second thing of note is the *range()* function, all this function does is *iterate* over all of the integers from the **start** to the **stop** with a given **step** (automatically assumed to be 1). So, in the above example the 'number' variable repeatedly takes each integer from 0 to 11 not inclusive, then for each loop, the 'number' variable at that time is printed.

If you want to *break* out of a loop if a given condition is met, you can use the *break* statement. The *break* statement **leaves the innermost loop only**.

```
for i in range(0, 10):
    print(f"Number: {f}")
    if i == 7:
        break
# Prints "Number: 0", "Number: 1", ... , "Number: 6", "Number: 7"
```

Though the above example isn't very practical (why not just bound it between 0 and 7?), but I feel that it illustrates the *break* function quite well. The loop stops when 'i is equal to 7'.

Loop type:	**for**	**while**
End early:	can use break statement	can use break statement
Number of iterations:	bounded	unbounded
Counter:	inbuilt	initialized prior