Conditional Logic and Looping in Python

Leigh J. Little SUNY Brockport November 15, 2021

Contents

1 Introduction

This document will discuss how to perform conditional testing and looping operations in Python.

2 Python 2 versus Python 3

Like most languages, Python is occasionally updated to incorporate new features and capabilities. In 2008, Python underwent a major update from Python 2 to Python 3. One important consequence of this update is that some Python 2 commands and functions had changes to their syntax. In this course we will be using Python 3.

Though most new programs are written in Python 3, there is still a large amount of existing code that was written in Python 2. If you ever have to modify a Python program, you should first try to determine in which version of Python the program was written. This is especially important if you are looking on the internet for samples of Python code.

3 Conditional Logic

Conditional logic in Python behaves the same way as in other languages with respect to program flow, but the syntax is different. The relational test operators in Python are given in Table 1. If you need to assign a logical variable, you can do it using the syntax

```
In[]: a = True
In[]: b = False
```

3.1 `if-then` Statements

There are 3 types of `if-then` structures in Python.

Operator	Test
==	Equal to
!=	Not equal to
<=	Less than or equal to
<	Less than
>	Greater than
>=	Greater than or equal to
and	Logical And
or	Logical Or
not	Logical Not

Table 1: Relational Operators in Python

3.1.1 The Simple `if-then` Statement

The syntax for the simple `if-then` statement is given by

```
if conditional statement:
   ...
   Section to execute if statement is true
   ...
First executable statement outside if-then structure
```

There are two key differences between the Python and MATLAB syntax:

a) Note the colon (`:`) at the end of the `if` line

b) There is no statement that ends the `if` block. This is implied by the level of indenting (hence, the many previous warnings to indent the bodies of your `if-then` blocks). This is characteristic of many structures in Python.

c) It doesn't matter how many spaces you chose for your indenting.

d) All statements that are part of the same structure must be indented at the same level. For example,

```
if a == b:
   c = a - b
   d = a + b
print('End of if block')
```

is acceptable, but

```
if a == b:
   c = a - b
      d = a + b
print('End of if block')
```

is not. This is because the two lines in the `if` body are part of the same `if` block and must have the same indenting.

3.1.2 The `if-then-else` Statement

This structure has the syntax

```
if conditional statement:
   ...
   Section to execute if statement is true
   ...
else:
```

```
   ...
   Section to execute if statement is false
   ...
First executable statement outside if-then structure
```

3.1.3 The `if-then-elseif` Statement

This structure has the syntax

```
if conditional statement 1:
   ...
   Section to execute if statement 1 is true
   ...
elif conditional statement 2:
   ...
   Section to execute if statement 2 is true
   ...
elif conditional statement 3:
   ...
   Section to execute if statement 3 is true
   ...
else:
   ...
   Section to execute if none of the above statements are true
   ...
First executable statement outside if-then structure
```

Like with MATLAB, the `else` is optional. You can have as many `elif` clauses as you need. The program will execute the body of the first block that is true and ignore any remaining clauses. Also, any type of `if-then` structure can be nested within the body of any other type of `if-then` structure.

3.1.4 Example 1

The code below is an example of nesting `if-then` structures. The bodies of the inner `if` statement need a further level of indenting.

```
a = 2
b = 1
c = 5
if a > b:
   print('a is greater than b')
   if b > c:
      print('a is greater than b, b is greater than c')
   else:
      print('a is greater than b, c is greater than b')
print('End of program')
```

Note that the `print` statement is indented at the level of the outer `if` block, so this statement ends both blocks. This is equivalent to the following MATLAB code:

```
a = 2;
b = 1;
c = 5;
if a > b
   disp('a is greater than b')
   if b > c
      disp('a is greater than b, b is greater than c')
   else
      disp('a is greater than b, c is greater than b')
   end
```

```
end
disp('End of program')
```

4 Looping Structures

Looping structures in Python are also possible. For now we will only examine `for` and `while` loops.

4.1 The `for` Loop

The syntax for a standard counting `for` loop is given by

```
# start is the number you want to start with
# end is the number you want to end at
for i in range(start,end+1):
    ...
    Body of loop
    ...
First statement outside loop
```

Some notes on the `for` loop

a) `i` is called the index variable. The body of the loop should not modify the value of `i`.

b) As with the `if-then` structure, there is no statement that ends the loop; this is implied by the indenting.

c) The `range` statement behaves much like an array index in MATLAB however, the syntax is unusual. For example, the statement `range(1,11)` generates the numbers 1 through 10. There is a reason for this, but it's easier to hold off on an explanation for now.

You can increment by values other than 1

```
for i in range(start,end+1,2):     # Increment by 2
    ...
    Body of loop
    ...
First statement outside loop
```

and count backwards

```
for i in range(start,end-1,-1):    # Increment by -1
    ...
    Body of loop
    ...
First statement outside loop
```

As with MATLAB, it is possible for the `range` function to return a set with no values in it (for example, `range(5,1)`) so loops can be empty (*i.e.,* never execute).

4.1.1 Example 2

A simple loop that counts to 10 is given in the code below

```
for i in range(1,11):
    print(i)
print('End of program')
```

4.2 `while` Loops

Python also has a `while` loop structure. The syntax for this is

```
while condition:
    ...
    Block to execute while condition is true
    ...
First statement outside loop
```

As with MATLAB `while` loops, you control when the loop exits by choosing the condition that must be met. Be sure the condition eventually becomes false, otherwise the loop will be infinite.

4.2.1 Example 3

A simple `while` loop that counts to 10 is given in the code below:

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
print('End of program')
```

4.3 The `break` Statement

The `break` statement is used to immediately exit a loop when needed.

4.3.1 Example 4

An example of the `break` statement is given in the code below

```
i = 1
while i <= 10:
    print(i)
    if i == 6:
         break
    i = i + 1
print('End of program')
```

As in MATLAB, you can nest loops within other loops. More generally, you can nest any `if-then` or looping structure within any other `if-then` or looping structure.