Conditional Logic and Looping in Python

Leigh J. Little SUNY Brockport November 6, 2020

Contents

1 Introduction

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

2 Integer versus Floating Point Variables

Most of the time, Python will not care about variable types, but this is not always the case. An integer is a number without a decimal point (2, 5, -6, *etc.*). A floating point number has a decimal point (4.5, -3.3, π). Because computers cannot store an infinite number of digits, there are limits to the portion of the real number line that can be represented.

2.1 Integers

In most languages, the standard integer variable is represented internally as a 32 bit binary number. The first bit is used for the sign of the number and the remaining 31 bits store the digits. The range of values for an integer is -2,147,483,647 to 2,147,483,648. Attempting to create an integer outside this range will result in an *overflow* error.

2.2 Double Precision Floating Point Numbers

There are many ways to store floating point numbers, but most computer architectures obey the IEEE 754 floating point standard. The most common type of floating point value in modern scientific computing is the 64 bit double precision value. A double precision value has a precision roughly equivalent to 15-16 decmial digits. The range of positive values is approximately $[10^{-308}, 10^{308}]$. A calculation that results in a value greater (less) than 10^{308} (10^{-308}) results in an *overflow* (*underflow*) error.

2.3 Single Precision Floating Point Numbers

Single precision floating point values were the most popular data type back when computer memory cost a small fortune. This is a 32 bit number that has a precision roughly equivalent to 7-8 decimal digits. The range of positive values is approximately $[10^{-38}, 10^{38}]$.

3 The `input` Statement

The input statement is used to get user input from the keyboard. This is where the data types for Python become important. If you type the statement

```
>>> n = input("Input a value for n ")
```

the value of `n` will be a string variable, not a number (even if you enter a number). If you need to obtain numerical input, you first need to decide if the value will be an integer or a double. For example,

```
>>> ii = int(input("Input an integer value for ii "))
>>> aa = float(input("Input a double value for aa "))
```

4 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.

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

If you need to assign a logical variable, you can do it using the syntax

```
>>>  a = True
>>>  b = False
```

4.1 `if-then` Statements

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

4.1.1 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 syntaxes:

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

b) Note that there is no statement that ends the `if` block. This is implied by the indenting (hence, the many previous warnings to indent the bodies of your `if-then` blocks).

4.1.2 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
```

4.1.3 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. Also, any type of `if-then` structure can be nested within the body of any other type of `if-then` structure.

4.1.4 Example 1

The code below is an example of nesting `if-then` structures.

```
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 last 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')
```

5 Looping Structures

Looping structures in Python are also possible.

5.1 The `for` Loop

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

```
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 syntax on the `range` statement 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).

5.1.1 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')
```

5.2 The `while` Loops

Python also has a `while` looop 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.

5.2.1 Example 3

A simple 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')
```

5.3 The `break` Statement

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

5.3.1 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')
```