Tools - Boolean, if, while, list, for

# Introduction

Here we make an introduction to a few basic Python programming tools which are continually used. We give the most basic view to enable initial programming to begin. More detail will be provided during the course. We hope that this introduction will aid in doing the early homework problems. If there is any confusion, please ask me for help.

# Summary

* Boolean value – true or false value – constants: True, False
* Boolean operator – test condition - **>**, **<**, **==**, **!=**, **>=**, **<=** 🡪 evaluates to True or False
* **If – select statement to execute based on Boolean condition**
* **while – loop statement(s) base on condition**
* **list – ordered list of values which can be traversed**
* **for – loop statements(s) with values in list**

# Boolean Value

A Boolean value is similar to a numeric value but, instead of a range of number values the Boolean value can only be either true, equal to the Boolean constant True or false, equal to the Boolean constant False. Boolean values are often the result of Boolean operators and are usually used in if or while statements.

# Boolean operator

Boolean operators are similar to and act similarly to arithmetic operators, in that they most often take two operands, e.g., 2 > 1, and evaluate to a result. However, where the arithmetic operator generally gives a numeric result, the Boolean operators give a Boolean value as the result. These operators are often seen in algebra.

Boolean Operator List (partial)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Operator** | | **Meaning** | | **Example** | | **Result** | |
| < | | Less than | | 2<3 | | True | |
| > | | Greater than | | 2>1 | | True | |
| == | | Equal to | | 2==3 | | False | |
| != | | Not equal to | | 2!=3 | | True | |
| <= | | Less or equal to | | 2<=2 | | True | |
| >= | | Greater or equal to | | 2>=2 | | True | |

Examples in IDLE:

>>> 2<3

True

>>> 2>1

True

>>> 2!=2

False

>>>

# if – Conditional execution

The if keyword is used to check a Boolean condition and execute the following statement(s) if that condition is True. The condition may be created by a Boolean constant, e.g., True, a Boolean variable, or a Boolean expression, e.g. (2>1).

The format of the if is:

if *condition* **:**

*statement-1*

*statement-2*

…

**NOTE: The statement(s) to be execute must all be indented to the same amount below the if clause.**

IDLE if examples:

>>> if True:

print("Were True")

Were True

>>> tval = False

>>> if tval:

print("tval:", tval)

>>> tval = True

>>> if tval:

print("tval:", tval)

tval: True

>>> if 2>1:

print("2>1", 2>1)

2>1 True

>>>

# while – conditional looping

The keyword while is used to loop execution while a Boolean condition is true. Like the if statement, this condition may be a Boolean constant, e.g., True, Boolean variable, e.g. tval, or a Boolean expression, e.g., n < 5.

The format of the while is:

while *condition* **:**

*statement-1*

*statement-2*

…

**NOTE: The statement(s) to be execute must all be indented to the same amount below the while clause.**

IDLE while examples:

>>> n = 1

>>> while n < 5:

print("n:", n)

n = n + 1

n: 1

n: 2

n: 3

n: 4

>>>

Please try creating and running an example of while use in a program file, e.g., **my\_while.py**

A start could be the following:

#my\_while.py 15sep2021, crs a sample

n = 1:

while n < 5:

print("n:", n)

n = n + 1

Try some variations. Use other file names to preserve the variations.

For the adventuresome, how about a loop within a loop?

# List – an ordered group of values which can be traversed

To provide help in dealing with groups of values, Python provides a list variable type. The simplest method for constructing a list of values is to place a comma-separated list of the values within square brackets. List Example:

number\_list = [1,2,3,4,5]

IDLE list example:

>>> number\_list = [1,2,3,4,5]

>>> number\_list

[1, 2, 3, 4, 5]

>>>

The built-in function **range(***start\_number, end\_number***)** can be used to create a list of numbers which can be used anywhere a list of numbers is desired (starting with start\_number, end\_number-1).

IDLE range example:

>>> number\_list2 = range(1,5+1)

>>> number\_list2

range(1, 6)

>>> list(number\_list2) # python3 remembers range

[1, 2, 3, 4, 5]

>>>

# for – loop through list of values

lists are very useful and part of that usefulness comes from the fact that the python for statement facilities easy traversing the values within a list. With the **range** function, described above, one is easily able to use the for statement to loop through number ranges.

The format of the for is:

for *iteration\_variable* ***in*** *list*:

*statement-1* # iteration\_variable sequentially takes on the list's element values

*statement-2*

…

Where:

*iteration\_variable* – variable name provided by the programmer to assume successive

Values from the list

list - list to process with the for

statement-1, statement-2, … - indented statement to be executed each loop iteration

getting successive list values via the variable named *iteration\_variable*

if**NOTE: The statement(s) to be execute must all be indented to the same amount below the for clause.**

IDLE for example:

>>> list1 = ["a", "b", "c", "d"]

>>> for letter in list1:

print("letter:", letter)

letter: a

letter: b

letter: c

letter: d

>>>

IDLE for with range example:

>>> for n in range(2,5+1):

print("n:", n)

n: 2

n: 3

n: 4

n: 5

>>>

Please try some examples of your own design. Try a couple of program files e.g., my\_for\_range, starting with the following:

#my\_for\_range.py 15Sep2021 crs, Training example

# Demonstrate for with range

For n in range(1, 5+1):

print("n:", n)