Control

Relevant for: COMP5711M

Brandon Bennett

School of Computing
University of Leeds

`B.Bennett@leeds.ac.uk`

Last Updated: August 2019

Learning Goals

After this lecture you should:

- Understand the meaning and purpose of a *conditional* command.
- Know how to form a conditional command in Python using the `if` keyword.
- Understand the purpose of loops in computer programs.
- Know how to construct `while` and `for` loops in Python.

Conditionals — `if`

A conditional statement specifies that a certain block of code should be executed if and only if certain condition is true.

Here is a simple example:

```
if ( 1+1 == 2 ):
    print( "Your computer can do sums." )
```

Conditionals — `if`

A conditional statement specifies that a certain block of code should be executed if and only if certain condition is true.

Here is a simple example:

```
if ( 1+1 == 2 ):
    print( "Your computer can do sums." )

if ( 1+1 == 3 ):
    print( "You computer is faulty." )
```

Code Blocks and Indentation

Following a conditional (`if`) statement or other control construct, we usually get an *indented* code block. This specifies 1 or more lines of code to be executed.

```
if ( x == 1 ):
   print( "Choice 1:" )
   print( "Go to park" )
if ( x == 2 ):
   print( "Choice 2:" )
   print( "Go swimming" )
if ( x == 3 ):
   print( "Choice 3:" )
   print( "Visit museum" )
```

Python is unusual in specifying code blocks by indentation.

Using `if` with `else`

It is common in programming that we want the program to do one think if a condition is true and another thing if it is not true.

This type of choice is implemented in most programming languages using the keyword `else`.

Here is an example:

```
x = input("Enter your favourite number: ")
if (x == 7):
   print( "That's my lucky number!")
else:
   print( "What's so good about number", x, "?" )
```

Note that the `else` needs to be at the same indentation level as the corresponding `if`

Negated Conditions

Sometimes you want to execute a command if something is not true.

One way of doing this is by using the negation operator ‘`not`’, which reverses the truth of the test operation.

For example:

```
if not (x == 7):
   print( "That’s NOT my lucky number!")
```

You can read the ‘`not`’ operator as:

It is not the case that ...

`and` **and** `or`

Sometimes we want to employ more complex test conditions that are logical combinations of two or more separate tests.

```
if (x > 10 and x < 20 ):
   print( "The value is in the correct range")

if (x == 13 or x < 0 ):
   print( "The value is unacceptable")
```

Boolean Values and Variables

A test operation such as `7 > 5` has a value of type *Boolean* — in other words, its value is either `True` or `False`.

```
x = 1000
isHigh = x > 100
if isHigh:
    print( "The value is too high!" )

print( isHigh )
```

Loops

Python provides a wide range of looping constructs.

Here are some basic examples:

```
x = 1

while (x < 100):
    print( x )
    x = x + 1
```

The indented block following the `while` statement's condition, will be executed repeatedly as long as the condition remains true.

The `break` Statement

The `break` command allows you to abandon a loop at a point within it.
It is normally used within an `if` statement (within the loop).

Use `True` for a condition that is always true

```
while (True):
    instring = input()
    if ( instring == "b" ):
        print( "Breaking out of loop!" )
        break
    print( "Your input was", len(instring),
           "characters long." )
```

Using `for` and the `range` function

```
for i in range(10):
       print( "Hello" )

for i in range(5,12):
       print( "The value of i is", i )
       print( "----------" )
```

To see exactly what values will be returned by the `range` function you can use the `list` function to get an actual list of numbers:

```
>>>  list( range( 400, 0, -17 ) )
[300, 283, 266, 249, 232, 215, 198, 181, 164, 147,
130, 113, 96, 79, 62, 45, 28, 11]
```

A More Complex Looping Example

The following code illustrates *nested* use of `for` loops.

A multiplication table is printed, where the outer loop controls the generation of each row, and the inner loop prints individual numbers for each column of the row.

```
for i in range(12):
    for j in range(10):
        print( "{:3d}".format(i*j), end = '')
    print()
```

Take note of the string formatting operation using the `.format(str)` method. This can be used to insert information within a string. This has many uses. There are wide variety of different ways this operation can be used.