Lecture 2: Control flow, loops and functions

Overview

Last week we outlined some of the basic Python syntax, but this week we are going to look at ways of making decisions, performing loops and creating functions. All of these features are commonplace in computer code.

I will show you the basics of these features and at the end of the lecture I will refer you to other references which illustrate other functions and libraries that you might find useful.

Booleans

Before we start with control flow, let's introduce a new variable type called the *boolean*. This is used to signifiy if something is *true* or *false*

In [4]:
```
hungry = True
thirsty = False
print('Am I hungry? {}. Am I thirsty? {}.'.format(hungry, thirsty))
```

```
Am I hungry? True. Am I thirsty? False.
```

The *if/else* statement

The if/else statement is used to make decisions based on the value of a variable. The general format of the if/else statement is as follows:

In [5]:
```
hungry = False

if hungry:
    print('I am very hungry')
else:
    print('I am fine. No food for me please.')
```

```
I am fine. No food for me please.
```

The if statement expects an expression which is either True or False. This can be formed in many ways.

For instance, what if we want to check if a value is greater than, or less than another value? This is simple:

In [6]:
```
x = 2

if x > 0:
    print('x is a positive number')

y = -4

if y < 0:
    print('y is a negative number.')
```

```
x is a positive number
y is a negative number.
```

We can also make use of the operators >= (greater than or equal) and <= (less than or equal).

In [4]:
```
frequency = 12

if frequency <= 12: # we check if the frequency is less than or equal to 12
    print('Vibration is under control.')
else:
    print('Vibration is excessive. Action required!')
```

```
Vibration is under control.
```

In [5]:
```
max_students = 156
student_count = 155

if student_count >= max_students:
    print('Too many students. Get rid of some.')
else:
    print('Ok student number. Proceed with lecture.')
```

```
Ok student number. Proceed with lecture.
```

!= and == operators

We commonly want to check if something is not equal to another value. This is achieved with !=

In [6]:
```
days_in_feb = 29 # number of days in february

if days_in_feb != 29:
    print('We are not in a leap year')
else:
    print('We are in a leap year.')
```

```
We are in a leap year.
```

And similarly, we check if something is equal by using the == operator. Take note, *this is not the same as the = (assignment) operator*

In [7]:
```
rankine_room = 635

if rankine_room == 634:
    print('You\'ve reached Dr. Simpson\'s room.')
else:
    print('You\'ve reached some random professor\'s room. Go away!')
```

```
You've reached some random professor's room. Go away!
```

and, or

There are many cases where we want to check two conditions (or more) before we execute a statement. An example might be if we want to print some code only when a water level is too high *AND* some flood gates are closed. E.g.

In [8]:
```
water_level = 5.0 # current water level reading of 5m
flood_gates_closed = False # our flood gates are closed

if water_level >= 5.0 and flood_gates_closed:
    print('Action required to reduce river level.')
else:
    print('No action required.')
```

```
No action required.
```

But what if we have a situation when we want to execute some code when only one of the expressions is true? We use *or*. E.g.

In [9]:
```
percentage = 40
attendance = True

if percentage < 40 or attendance == False:
    print('Unfortunately you have failed.')
else:
    print('You\'ve passed!')
```

```
You've passed!
```

if/else/elif

We can expand the previous if/else statement to include other conditions. For example, consider a scenario where we want to check if a value lies in one of several ranges. These could be $x < 0$, $x = 0$ and $x > 0$

In [10]:
```
x = 0.0
if x < 0.0:
    print('x is less than zero')
elif x > 0.0: # you can read 'elif' as 'else if'
    print('x is greater than zero')
else:
    print('x must be zero')
```

```
x must be zero
```

We can include several elif statements if necessary:

In [11]:
```
body_mass_i = 19.0 # body mass index

if body_mass_i < 18.5:
    print('Underweight')
elif body_mass_i >= 18.5 and body_mass_i < 25.0:
    print('Normal weight')
elif body_mass_i >= 25.0 and body_mass_i < 30.0:
    print('Overweight')
else:
    print('Obese')
```

```
Normal weight
```

Loops

One of the most powerful aspects of computer programs is their ability to perform the same task multiple times. The language feature which allows us to do this is the *loop*

The *for* loop

The for loop allows us to iterate over a range of values. It is often used with the standard *range* function like so:

In [12]:
```
for i in range(0,10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Note how the counter begins at zero and ends with the integer 9. In total 10 integers are printed.

We are free to choose the lower and upper values of the range function. But the values must be *integers*

In [13]:
```
for x in range(-4, 4):
    print(x)
```

```
-4
-3
-2
-1
0
1
2
3
```

Jumping ahead a little, we can loop over a *list* (lists will be defined more fully in lecture 3)

In [14]:
```
my_list = [1,3,5,7,9]
fav_foods = ['ice cream', 'butter', 'pies']

for x in my_list:
    print(x)

for f in fav_foods:
    print(f)
```

```
1
3
5
7
9
ice cream
butter
pies
```

If we wanted to compute, for example, the square of all numbers between 1 and 10, we could write the following:

In [15]:
```
for x in range(1,11): # note that the upper range is 11, not 10
    square = x**2
    print(square)
```

```
1
4
9
16
25
36
49
64
81
100
```

While loop

The while loop is slightly different from the for loop in the sense that it will continue to loop until some condition is satisfied. A basic while loop might look like the following:

In [16]:
```
x = 0
while x < 10:
    print(x)
    x += 1 # if we didn't include this line the code would run for an infinitel
y long time!
```

```
0
1
2
3
4
5
6
7
8
9
```

A common way to use the while loop is to force the loop to run and then do a check within the loop to work out when to stop. We can exit the loop using the *break* statement. For example:

In [7]:
```
number_students = 103

while True:
    print('There are %s' % number_students)
    number_students -= 10 # let's remove 10 students from the class on every lo
op

    if number_students < 30: # if we're less than 30 students, stop
        break

print('We\'re left with %s students' % number_students)
```

```
There are 103
There are 93
There are 83
There are 73
There are 63
There are 53
There are 43
There are 33
We're left with 23 students
```

Class Exercise (10mins)

Write some code that loops over integer values from 1 to 123 and counts the number of values which are exactly divisible by 3,5 and 6. (Hint: make use of the if statement and the modulo operator '%')

In [17]:
```
three_count = 0
five_count = 0
six_count = 0

for i in range(1,124):

    if i % 3 == 0:
        three_count += 1

    if i % 5 == 0:
        five_count += 1

    if i % 6 == 0:
        six_count += 1

print(123//5)
print('There are {} integers exactly divisible by 3'.format(three_count))
print('There are {} integers exactly divisible by 5'.format(five_count))
print('There are {} integers exactly divisible by 6'.format(six_count))

# Or simply use the integer division operator to get the same answer!
print(123//3)
print(123//5)
print(123//6)
```

```
24
There are 41 integers exactly divisible by 3
There are 16 integers exactly divisible by 5
There are 0 integers exactly divisible by 5
41
24
20
```

Functions

Functions are arguably one of the most important aspects of computer programming. If you find yourself in a situation where you are writing the same code several times, it is almost always a good idea to put that code into a function.

Functions are also a good way of organising your code. Let's look at a scenario where the use of a function is a good idea. Let's say we have a mathematical function $f(x) = x^2 - 3x + 4$ and we want to compute this for three values $x_1 = 3.5, x_2 = 2.3, x_3 = 3.4$. Let's try this:

In [18]:
```
x1 = 3.5
x2 = 2.3
x3 = 3.4

fx1 = x1**2 - 3.0 * x1 + 4.0
fx2 = x2**2 - 3.0 * x2 + 4.0
fx3 = x3**2 - 3.0 * x3 + 4.0

print(fx1)
print(fx2)
print(fx3)
```

```
5.75
2.3899999999999997
5.359999999999999
```

This is rather clumsy since we are repeating the same code three times. Let's define a function that makes our live easier.

In [19]:
```
def myfunc(x):                      # this is a function which we have defined.
    return x**2 - 3.0 * x + 4.0    # notice the return statement to return the ou
tput

x1 = 3.5
x2 = 2.3
x3 = 3.4

fx1 = myfunc(x1)
fx2 = myfunc(x2)
fx3 = myfunc(x3)

print(fx1)
print(fx2)
print(fx3)
```

```
5.75
2.3899999999999997
5.359999999999999
```

You can interpret a function as something that takes some input, performs some work on the input and then returns an output.

Multiple input arguments

We can include multiple arguments as input to a function. For example, consider a function to calculate the maximum bending moment of a beam:

In [20]:
```
def max_bm(L, w):
    return w * L**2 / 8.0

bm_1 = max_bm(1.0, 1.0)
print('Max. bending moment of beam 1 = {} kNm'.format(bm_1))

bm_2 = max_bm(2.0, 3.0)
print('Max. bending moment of beam 2 = {} kNm'.format(bm_2))
```

```
Max. bending moment of beam 1 = 0.125 kNm
Max. bending moment of beam 2 = 1.5 kNm
```

Example 1

Write a function that computes the factorial function i.e. $5! = 5 \times 4 \times 3 \times 2 \times 1$

For this, it makes sense to use a loop within the function.

In [22]:
```
def factorial(n):
    result = 1
    for x in range(1,n+1): # note that we put n here we would miss out the larg
est integer
        result *= x
    return result

f1 = factorial(3)
f2 = factorial(5)

print(f1, f2)
```

```
6 120
```

Built-in functions

The Python library has a number of built-in functions that may come in handy. Built-in function we have been using so far include the `print()` function and `range()` . Others include:

abs()

This determines the *absolute* value of a function. i.e. Given a value x, it computes $|x|$

In [24]:
```
x = 23
y = -99
print(abs(x), abs(y))
```

```
23 99
```

str(), int(), float()

You've seen these before (Lecture 1). They are used to convert variables explicity into a string, integer and float respectively. E.g.

In [21]:
```
myfloat = 4.2345
myint = int(myfloat)

print(myint)
```
```
4
```

Functions from the math library

You will find that you commonly require such mathematical functions in your code. Common examples include e^x, $\sin x$, $\log x$, $\sqrt{x}$. See here for the full set of available functions in the math library https://docs.python.org/2/library/math.html#module-math (https://docs.python.org/2/library/math.html#module-math)

To use functions from the math library we must first 'import' the library. We can do this as follows:

In [22]:
```
import math # import the math library into our code. You only need to this once
in any notebook

x = math.factorial(5) # and then use it!
y = math.sin(0.23)
z = math.log(2.345)
print(x, y, z)
```
```
120 0.2279775235351884 0.8522854018982428
```

We can also import only the functions we want from the library as follows:

In [23]:
```
from math import cos, tan, sin

tol = 1.0e-9 # tolerance
x = 0.5
mytan = sin(x) / cos(x)

print(mytan)
print(tan(x))
```
```
0.5463024898437905
0.5463024898437905
```

and we can import a library using a shorthand notation to avoid typing (I use this frequently throughout the course)

In [30]:
```
import math as m

print(m.factorial(3))  # factorial function
print(m.factorial(5))
```
```
6
120
```

and many more...

There are many other built-in functions which can be found here https://docs.python.org/2/library/functions.html# (https://docs.python.org/2/library/functions.html#) but for this course we will restrict ourselves to a small set of these.

We will find that as we continue through the course that there are also many convenient functions that are defined by various *libraries*. These include plotting functions, matrix functions, statistic functions,.... We will cover some of these functions in later lectures.

References

[Code Academy] https://www.codecademy.com/learn/python (https://www.codecademy.com/learn/python)

[Python math library] https://docs.python.org/2/library/math.html#module-math (https://docs.python.org/2/library/math.html#module-math)