Lecture 1: Introduction to Python

Why Python?

For learning to program Python is one of the most attractive choices out of the huge array of possible programming languages. Often very few lines of code are required to achieve would would otherwise require many lines of code in other languages such as C, C++, Java, C#,...

It may be simple, but Python is also supported by many *libraries* that make it very powerful. For example, the numpy library which we will use extensively in this course allows us to write code that performs sophisticated visualisation, linear algebra and provides useful functions for statistics and other mathematical operations.

If we wanted our code to run as fast as possible, then Python may not be the best choice. This is why software where fast runtimes are of the utmost importance (think banking) languages such as C,C++ and Fortran would be more appropriate.

Overview

In this lecture we are going to cover basic Python syntax and at the end of this lecture you will have the skills to write a very simple Python program. We will cover the following topics:

- Some basic Python notebook commands
- Printing output to the screen
- Variables, including integers, floats, and strings
- How to add comments to your code
- Operators that work on numbers (e.g. add, subtract,...)
- Getting inputs from the user

A few Python notebook basics

- Press Shift + Enter to run a cell
- Any variables defined in a cell will carry through to cells below
- If everything breaks in your notebook, try clicking on Kernel -> Restart and Run all
- Try and limit the use of input statements

The *print* statement

Whatever follows the print statement is output to the screen. We do not actually send anything to a printer! For example

```
In [1]:
```

```
print('This is my first line of Python code')
```

```
This is my first line of Python code
```

```
In [2]:
```

```
print(13)
```

```
13
```

Variables

Variables are the way in which we store values in a program. As the name implies, their value can change throughout the program. We *assign* a value to a variable using the = sign. Let's create a variable called `myvariable` and assign it a value of 2.

```
In [3]:
```

```
myvariable = 2
print(myvariable)

myvariable = 4
print(myvariable)
```

```
2
4
```

Let's now work with several variables

```
In [4]:
```

```
girl_count = 34
boy_count = 10

total = girl_count + boy_count
print(total)
```

```
44
```

The variables we have considered so far all stored numbers, but variables can also store text. In computer programming language, these are called *strings*

String variables (or simply 'strings')

```
In [5]:
```

```
mytext = 'I love Monday morning lectures'
print(mytext)
```

```
I love Monday morning lectures
```

In [6]:

```
string1 = 'I have something important to say '
string2 = ' but I\'m not exactly sure what.'
total_string = string1 + string2
print(total_string)
```

```
I have something important to say  but I'm not exactly sure what.
```

We had to use what is known as an *escape* sequence `\'` to show the apostrophe correctly

Formatting print statements with variables

Sometimes we want to format our print statement by including variables within it. So say for example we have a variable called `n` that represents the number of lollipops we have and we want to print out a sensible message with this number. We can do it like this:

In [7]:

```
print('There are {} students in the class'.format(total))
```

```
There are 44 students in the class
```

Another way to do this but which is now considered outdated is:

In [8]:

```
print('There are %s students in the class' % total)
```

```
There are 44 students in the class
```

We can also insert multiple variables into our print statement:

In [9]:

```
b = 12
w = 15
print('Breadth of beam is {}mm and width is {}mm'.format(b,w))
```

```
Breadth of beam is 12mm and width is 15mm
```

Notice that we need to use brackets when we have more than one variable.

Comments

To help others understand your code, it is a good idea to use comments. But don't overdo it. We signify a comment using the # symbol. Anything after the # symbol is not considered Python code and will be ignored.

In [10]:

```
n_b = 5       # number of beams
n_c = 12      # number of columns
total = n_b + n_c

print(total)
```

```
17
```

Integers

Integers represent whole numbers. As such, they can be either positive or negative.

In [11]:

```
x = -234
y = 234
z = x + y

print(z)
```

```
0
```

Integers are used to represent objects which must be whole numbers. E.g. number of beams, number of nodes, etc.

Any number which has a decimal point will not be considered an integer. We will talk about this later.

Addition, subtraction, division, multiplication and remainder operators

After we define various variables we often want to perform *operations* on these variable using various *operators*. These include the usual addition, subtraction, division and multiplication operators.

Addition and subtraction

Addition and subtraction work straightforwardly:

In [12]:

```
a = 1
b = 2
c = 3
d = a + b - c

print(d)
```

```
0
```

Multiplication

Multiplication also behaves as expected:

```
In [13]:
```

```
s_per_row = 12     # number of students per row
n_row = 30         # number of rows
total_s = s_per_row * n_row

print('Total number of students: {}'.format(total_s))
```

```
Total number of students: 360
```

Division

Division works as expected:

```
In [14]:
```

```
n = 10
d = 2
result = n / d

print('10/2 = {}'.format(result))
```

```
10/2 = 5.0
```

If you want an integer returned which is rounded down, use a double slash:

```
In [15]:
```

```
n = 10
d = 3
result = n // d

print('10/3 = {}'.format(result))
```

```
10/3 = 3
```

Modulo operator

The modulo operator is given by **%**. For example, let's say we want to get the remainder of `11/5`:

```
In [16]:
```

```
n = 11
d = 5
remainder = n % d

print('The remainder of 11/5 = {}'.format(remainder))
```

```
The remainder of 11/5 = 1
```

Note that the `%` symbol is used in two different ways here: the first performs the remainder operator while the second is used to format our printed string.

Floating point numbers

These represent *real* numbers. That is, they are able to represent decimal places. Addition, subtraction, multiplication and division all work as expected.

In [17]:

```
pi = 3.14 # approximate value of pi
radius = 2.123
area = pi * radius * radius

print('The area of our circle is {}'.format(area))
```

```
The area of our circle is 14.152385060000004
```

The power operator

What if we want to raise a number to a power. For example, what is 3^3 ? We do it as follows:

In [18]:

```
answer = 3**3
print(answer)
```

```
27
```

This will also work with floating point numbers

In [19]:

```
radius = 2.0
area = pi * radius**2
print(area)
```

```
12.56
```

Assignment operators

These are not stricly necessary, but sometimes these can make your life a little easier and make you code more succinct.

Common assignment operators include: +=, -=, *= and /=. For example:

In [20]:

```
cement = 12
cement += 5

print('We have {} kg of cement'.format(cement))

cement -= 7

print('We now have {} kg of cement'.format(cement))
```

```
We have 17 kg of cement
We now have 10 kg of cement
```

And similarly,

In [21]:

```
beam_n = 3 # number of beams
beam_n *= 4
print('We have {} beams'.format(beam_n))
```

```
We have 12 beams
```

Input

Sometimes we wish to obtain values from the user as inputs. We can do this as follows:

In [22]:

```
s = input('Please enter text here ---> ')
print(s)
```

```
Please enter text here ---> Some random text
Some random text
```

We need to be a bit careful however when we are working with numbers. For example, we might want to obtain an integer as input and attempt to do this as follows:

In [23]:

```
myinteger = input('Please enter your integer here --> ')
d = myinteger * 2
print(d)
```

```
Please enter your integer here --> 4
44
```

Integers as input

What you must understand is that the raw_input() function turns all input into strings. To convert the input into an integer, the solution is simple: we use a function called int() that converts the string to an integer.

In [24]:

```
myinteger = input('Let\'s now do it correctly --> ')
mycorrectint = int(myinteger)
d = mycorrectint * 2
print(d)
```

```
Let's now do it correctly --> 4
8
```

Floats as input

We can also get floating point numbers in a very similar way using the float() function:

In [25]:

```
data = input('Enter your floating point number --> ')
myfloat = float(data)
x = myfloat * 2

print(x)
```

```
Enter your floating point number --> 4.234
8.468
```

If you want to make your code a bit more concise this can also be written as:

In [26]:

```
myfloat = float(input('Enter your floating point number --> '))
x = myfloat * 2

print(x)
```

```
Enter your floating point number --> 4.234
8.468
```

Example

We want to create a program that computes the maximum bending moment, shear force and bending stress in a simply supported beam . The user can specify the length, and imposed u.d.l. .

Knowns: L (length), w (u.d.l.)

Unknowns: M_{max} (max. bending moment), Q_{max} (max. shear force)

We remember the following formulae: $M_{max} = \frac{wL^2}{8}$, $Q_{max} = \frac{wL}{2}$

```
In [27]:
```

```
L = float(input('Please enter length (m) --> '))
w = float(input('Please enter u.d.l. (kNm) --> '))

m_max = w * L**2 / 8.0
q_max = w * L / 2.0

print('Our beam has a max. bending moment of {} kNm and shear force of {} kN'.fo
rmat(m_max, q_max))
```

```
Please enter length (m) --> 10.2
Please enter u.d.l. (kNm) --> 2.4
Our beam has a max. bending moment of 31.211999999999996 kNm and she
ar force of 12.239999999999998 kN
```

Last edited by RNS 29/1/18