MATH6005 Introduction to Python Lecture 1

January 20, 2019

1 Python Basics

- Python Syntax
- Variables and data types
- Storing lots of data in memory: Lists and Tuples
- Whitespace
- Functions
- Comments in code

1.1 What does python syntax look like?

```
In [21]: salary = 100000
         tax_rate = 0.2
         salary_after_tax = salary * (1-tax_rate)
         print(salary_after_tax)
```

```
80000.0
```

1.1.1 What parts of Python did we use in that code?

```
In [1]: print('Hello world')
```

```
Hello world
```

1.2 Variables and data types in python

- Variables hold *data*.
- For example, this might be a number (e.g. an integer) or a text string.
- Your computer program uses the data in its operations.

Let's create a really simple variable call `simple_sum` as the sum of two integers.

```
In [2]: simple_sum = 1 + 1
        print(simple_sum)
```

```
2
```

Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Quotient of `a` and `b`, removing fractional parts
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`

Example: the variable `z` is product of variable `x` raised to the power of `y`

```
In [3]: x = 10
        y = 2
        z = x ** y
        print(z)
```

```
100
```

Example: * the variable `foo` is the negation of variable `bar`

```
In [4]: bar = 10
        foo = -bar
        print(foo)
```

```
-10
```

1.2.1 Variables Names

- Variables **names** can only contain *letters*, *numbers*, and *underscores* (_).
- Underscores are used instead of spaces.
- For example, use `student_name` instead of `student name`.

- If you include a space then you will get an `SyntaxError`!

```
In [5]: lecturer name = 'tom'
```

```
  File "<ipython-input-5-bb25698b12f8>", line 1
    lecturer name = 'tom'
                ^
SyntaxError: invalid syntax
```

- Each variable has a **data type**.

- Python is dynamically typed.
- This means that Python does all of the work for you!

```
In [6]: foo = 1000
        bar = 'hello everyone'
        print(type(foo))
        print(type(bar))
```

```
<class 'int'>
<class 'str'>
```

```
In [7]: foo = True
        bar = False
        spam = 3.142
        eggs = 10000000

        print(type(foo))
        print(type(bar))
        print(type(spam))
        print(type(eggs))
```

```
<class 'bool'>
<class 'bool'>
<class 'float'>
<class 'int'>
```

1.3 Introduction to Lists and Tuples

- A Python `List` is a simple and flexible way to store variables and any type of data

```
In [8]: foo = [1, 2, 3, 4, 5]
        print(foo)
```

```
[1, 2, 3, 4, 5]
```

- The elements stored within a `List` have a numbered index
- Indexes start from **zero** (don't forget)

```
In [10]: foo = [1, 2, 3, 4, 5]
         print(foo[0])
         print(foo[1])

         foo[4] = 999
         print(foo)
```

```
1
2
[1, 2, 3, 4, 999]
```

- A `List` is very flexible and can hold different types of variable

```
In [11]: bar = ['spam', 5, 82.96, True]
         bar[1] = 'eggs'
         print(bar)
```

```
['spam', 'eggs', 82.96, True]
```

- A `List` has a length

```
In [12]: length_of_list = len(bar)
         print(length_of_list)
```

```
4
```

1.3.1 Inserting and removing items

New list items: * can be **appended** to the end of a list a `List` * or **inserted** at a specified index
Existing list items: * Can be removed from a specified **index** * or by **value**

```
In [17]: foo = []
         foo.append('spam')
         foo.append('eggs')
         print(foo)

         #foo.insert(1, 'bar')
         #print(foo)

         #del foo[2] # Remove a specific index
         #print(foo)

         #foo.remove('spam') #Remove a specific value
         #print(foo)
```

```
['spam', 'eggs']
```

- A `Tuple` is similar to a `List` with one key difference
- A `List` is mutable where as a `Tuple` is **immutable**

```
In [18]: foo = [1, 2, 3, 4, 5]
         bar = (1, 2, 3, 4, 5)

         print(foo)
         print(bar)
```

```
[1, 2, 3, 4, 5]
(1, 2, 3, 4, 5)
```

```
In [19]: foo[1] = 999 #list
         bar[1] = 999 #tuple


         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-19-0887368b13a8> in <module>()
           1 foo[1] = 999 #list
     ----> 2 bar[1] = 999 #tuple


         TypeError: 'tuple' object does not support item assignment
```

1.4 Functions

- We have already encountered a function: `print()`
- `print()` is one of Python's built in functions
- Python has lots of them!
- If you are not sure how they work you can use the `help()` function!

```
In [20]: help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

1.4.1 Importing functions from Python modules

- Functions are stored within **modules**
- Use the `import` statement to access the modules you need
- Let's generate a random integer between 1 and 100.
- We need to a function from the `random` module

```
In [38]: import random as rnd

         u = rnd.randint(1, 100)
         print('I generated a random integer {0}'.format(u))
```

```
I generated a random integer 4
```

- We can also import specific functions from modules

```
In [40]: from random import randint, gauss

         u1 = randint(1, 100)
         u2 = gauss(0, 1)
         print('I sampled from a random int {0} and a normally distributed value {1}'.format(u1,
```

```
I sampled from a random int 29 and a normally distributed value -0.6915069309944331
```

1.5 Custom Functions

- You can also code your own bespoke functions
- A function is a reusable block of code that has a **single responsibility**
- That means you function should do one thing only

1.5.1 Motivation

- You have been asked to convert a dataset of degrees celsius figures to fahrenheit

```
In [41]: deg_celsius = 20
         fahrenheit = 9.0/5.0 * deg_celsius + 32
         print(fahrenheit)

         deg_celsius = 10
         fahrenheit = 9.0/5.0 * deg_celsius + 32
         print(fahrenheit)
```

```
68.0
50.0
```

- A reusable function would come in very handy here!

```
In [42]: def convert_celsius_to_fahrenheit(deg_celsius):
             deg_fahrenheit = 9.0/5.0 * deg_celsius + 32
             print('{0} degrees celsius is equivalent to {1} degrees fahrenheit'.format(deg_cels
```

```
In [43]: convert_celsius_to_fahrenheit(20)
         convert_celsius_to_fahrenheit(10)
```

```
20 degrees celsius is equivalent to 68.0 degrees fahrenheit
10 degrees celsius is equivalent to 50.0 degrees fahrenheit
```

- An alterantive way to write the same function

- Instead of using `print()` we can `return` the result
- And store the result in a new variable `result_fahrenheit`

```
In [44]: def convert_celsius_to_fahrenheit(deg_celsius):
             deg_fahrenheit = 9.0/5.0 * deg_celsius + 32
             return deg_fahrenheit

In [45]: result_fahrenheit = convert_celsius_to_fahrenheit(22)
         print(result_fahrenheit)
```

```
71.6
```

- Watch out for whitespace rules!
- if you use : then you must indent (use tab or 4 spaces) on the next line

```
In [46]: def convert_celsius_to_fahrenheit(deg_celsius):
         deg_fahrenheit = 9.0/5.0 * deg_celsius + 32
         return deg_fahrenheit
```

```
          File "<ipython-input-46-33c89eb1548a>", line 2
        deg_fahrenheit = 9.0/5.0 * deg_celsius + 32
                     ^
    IndentationError: expected an indented block
```

- If a func returns a value you can pass it to another func

```
In [47]: def add(num1, num2):
             return num1 + num2

         def square(num):
             return num ** 2

In [48]: result = square(add(1, 1))
         print(result)
```

```
4
```

1.6 Comments in code

```
In [49]: def convert_celsius_to_fahrenheit(deg_celsius):
             '''
             Converts d egrees celcius to degrees fahrenheit
             Returns a float representing the temperature in degrees fahrenheit.

             Keyword arguments:
```

```
            deg_celsius -- a float temperature in degress celsius e.g. 18.5
            '''
            deg_fahrenheit = 9.0/5.0 * deg_celsius + 32
            return deg_fahrenheit
```

```
In [50]: help(convert_celsius_to_fahrenheit)
```

```
Help on function convert_celsius_to_fahrenheit in module __main__:

convert_celsius_to_fahrenheit(deg_celsius)
    Converts d egrees celcius to degrees fahrenheit
    Returns a float representing the temperature in degrees fahrenheit.

    Keyword arguments:
    deg_celsius -- a float temperature in degress celsius e.g. 18.5
```

- Using # is another way to add comments to code
- Useful to clarify "complex" code and aid your memory.
- Won't be picked up by `help()`

```
In [51]: from math import pi

         def area_of_circle(radius):
             #  pi x squared(radius)
             area = pi * radius ** 2
             return area
```

```
In [52]: help(area_of_circle)
```

```
Help on function area_of_circle in module __main__:

area_of_circle(radius)
```

1.7 In Python Functions can return multiple values

```
In [70]: def list_info(data):
             '''
             Returns Sum, Length and Mean of list @data
             Keyword arguments
             data -- a list containing numeric data
             '''
             list_sum = sum(data)
             list_length = len(data)
             list_mean = list_sum / list_length
```

```
        return list_sum, list_length, list_mean

    data = [1, 2, 3, 4, 5]

    results = list_info(data)
    print("The variable 'results': {0} has type {1}".format(results, type(results)))

    data_sum, data_length, data_mean = list_info(data)
    print('Split into seperate variables sum {0}, length {1}, mean {2}'.format(data_sum, da
```

```
The variable 'results': (15, 5, 3.0) has type <class 'tuple'>
Split into seperate variables sum 15, length 5, mean 3.0
```

- Please have a go at the lab material.
- You will need to understand the basics to do well on the course