Workshop 2: Data types & Conditionals

4 November 2020

Last time you learned:

- why computer programming is important
- that a computer programming language is used to communicate human instructions to computers
- Python is an *interpreted* computer programming language and that throughout this course we'll be using the *Anaconda* installation of Python
- how to find Spyder on a computer
- how to write a basic program using Spyder.
- basic data types (`int`, `float`, `bool`, `str`)

By the end of the session you will:

- know more about data types, including `strings`, `lists` and `tuples`.
- know what mutable/immutable data is
- understand what a conditional is (if statement)
- write Python programs that use conditionals
- use matplotlib's plot function to display data

If you have any questions during this session please do not hesitate to ask one of the demonstrators. You should ensure that you have completed all exercises before the next Workshop.

What to do when you see 'Write a program…'

1. Bring up the Spyder editor
2. At the top of the program, using the docstring (" " " " " "), add your name, and describe what the program does.
3. Sketch how you want your program to work on a piece of paper.
4. Type the commands into the Spyder editor.
5. Run the module using the tabs at the top of the Spyder window.

Template

The template for your Python programs should looks like this:

```
"""
Description of the contents of this script
@author: Your Name Here
"""
Main code goes here
# This line is a comment (note the hashtag, #) and does nothing
Real code goes here
```

Data Types continued: lists, strings, and tuples

In the last workshop we discussed different data types (`int`, `float`, `bool`, `str`). In this workshop we'll discuss more about data types including more about `strings`, `lists`, `tuples`.

What are Lists?

`Lists` are arrangements of values that are identified with an index. For example, in the list **[-2., 'cat', 42]**, The first value is identified with the index 0, the second value, 'cat', is identified with the index 1, and the last value, 42, is identified with the index 2. So the following code,

```
A = [-2.72, -3.14, 42.]
print(A[0])
print(A[1])
print(A[2])
```

will produce the output:

```
-2.72
-3.14
42.0
```

Why is a list useful? Example, create a plot using pyplot
Plotting with pyplot

- Pyplot is a collection of functions that allows a user to interface with computer graphics.
- We will be writing simple programs to use plotting. At the top of your program type

```
"""
An example plotting program.
@author: Your name
"""
import matplotlib.pyplot as plt    # this loads the pyplot package and calls it 'plt'

#  Create two lists, plot them.
x = [1, 2, 3, 4]
y = [1, 7, 3, 5]
plt.plot(x, y)
plt.show()
```

Examine the use of the two lists in the plot function. Figure (1) shows the connection between the elements in the list and the graph pyplot produces. Change the 4 in the first list to 22 and rerun your program. What happens? Change the 7 in the second list to 14 and see what happens. The function `plot` takes 2 lists, the first is a list of values that corresponds to the horizontal axes, and the second list corresponds to the vertical axes. Type "Python pyplot syntax" into a search engine to see a list of all the possible options for the `plot` function.

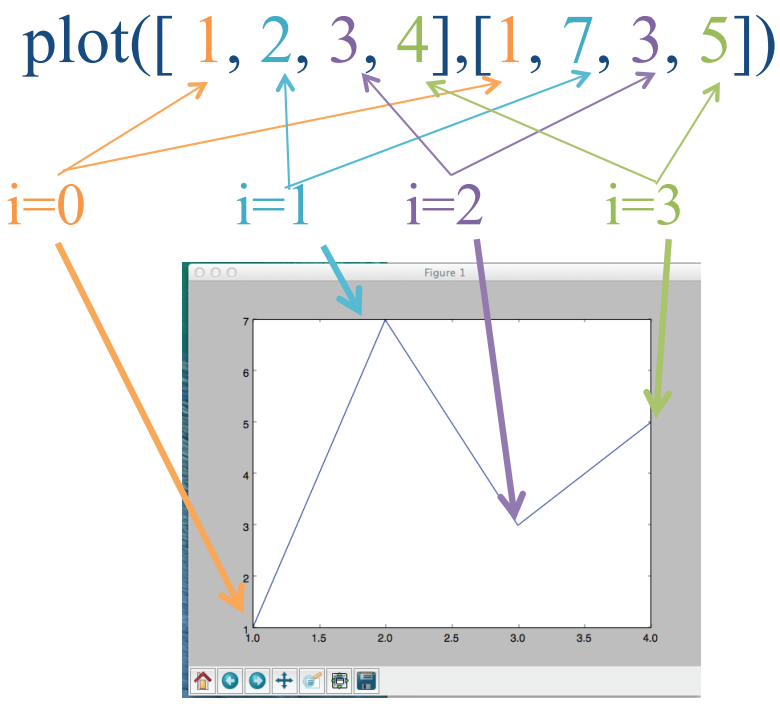

Figure 1: This figure shows how the function `plot` uses two lists to draw a plot. The first list defines the numbers on the horizontal axis, the second list defines the vertical or 'y' values of the function. Consider the first entry in each list, coloured orange. On the horizontal axis the first x value is 1 and the first y value is 1. Find it on the graph. The second point on the plot corresponds to the second entry of the two lists. The second entry of the first list is 2, the second entry of the second list is 7. Find the point (2, 7) on the plot. Do the same for the 3rd and 4th entry.

What are strings?

Like lists, strings (and tuples) are ordered sequences of elements. Strings can be represented by single or double quotes. For example:

```
string1 = 'Minerva'
string2 = "Bridget"

# You can do operations on strings
string3 = string1 + string2
string4 = 4 * string1 + 3 * string2
string5 = string1 + string2

print(string5)

string4 = 4 * string1 + 3 * string2

print(string4)

# You can find out how long a string is using len()

print('length of string1 =', len(string1))
print('length of string4 =', len(string4))

# You can extract individual characters from a string. Notice the use of []
# rather than ()

print(string4[0])
print(string4[5])

# You can do 'slicing' on strings

string6 = string4[0:4]
print(string6)
num_chars_string1 = len(string1)
string7 = string5[0:num_chars_string1]

print(string7)
```

You can perform operations on strings. You can do a web search on "Common string operations Python" and find many different operations such as `string.digits`, `string.uppercase`, `string.letters`, `string.index`. These operations can be performed by using:

`string.<operation>`

where operation is any of the above. For example,

```
string1 = 'Minerva'
```

```
string2 = 'erv'
print(string1.index(string2))
```

---

will produce the output 3.

What are tuples?

Tuples, like strings and lists are also ordered sequences of elements. The difference between tuples and lists are that tuples, once assigned, cannot be changed, unlike lists. Tuples use round brackets and lists use square brackets.

In Python, objects whose data can be changed are called **mutable** and objects whose data cannot be changed are called **immutable**. Tuples are immutable, while lists are mutable.

```
tuple1 = (1, 'Minerva', 'cat', 56)
tuple2 = ('Bridget', 'David', 24.57)
tuple3 = (46, )
print(tuple1)
print(tuple2)
print(tuple3)
print(tuple1[1])
print(tuple1[3])

# Like strings tuples can be concatenated, sliced and indexed
print(tuple1 + tuple2)
print(len(tuple1))
print(len(tuple1 + tuple2))

# Multiplying makes copies
print(5 * tuple1)
print(tuple1[0])
print(tuple3[0])
print(tuple1[1:3])
print((tuple1 + tuple2)[3:7])
```

Conditional Statements (if statements)

Conditionals are blocks of code that make cause a code to *branch*. A conditional statement has three parts:

1. A test that evaluates to either true or false
2. A block of code that is executed if the test evaluates to true

3. Another (*optional*) block of code that is executed if the test evaluates to false

In the lecture we talked about conditionals (also called *if statement*). Here is some information about them. The structure of an if statement looks like this:

```
if <conditional statement>:
    Do Something
elif <conditional statement>:
    Do some other thing
else:
    Do something different still
pass    #  This statement does nothing but can
           # be a useful visual marker for the end of the conditional
Rest of the code continues here
```

The overall structure is '*if, elif, else*'. The elif and else statements are optional and not always needed. *Remember, you must indent the contents of the if statements by 4 spaces!*
We also discussed the following relational operators used in conditional statements:

== > < != >= <=

We also discussed logicals like *and*, *or*, and *not*. For example,

if a != b or b == c:

We discussed a program that calculated the relationship between two integers (i.e., equal, greater than or less than). The code can be found below:

```
if x == y:
    print('x equals y')
else:
    if x > y:
        print('x is greater than y')
    else:
        print('x less than y')
```

1. Copy the code above into your Spyder editor and save the program.
   - Remember to add suitable comments.
2. Run the new program from the Spyder window

3. You will have got a syntax error since x and y have not been defined.

4. Update the code to initialise x to 5 and y to 10 (see below).

```
x = 5
y = 10
if x == y:
    print('x equals y')
else:
    if x > y:
        print('x is greater than y')
    else:
        print('x less than y')
```

5. Test your program to ensure each branch within your code works.

6. Introduce a syntax error by removing one of the ':'

   - Do you understand the error message?

Exercises

These exercises should be completed before next Tuesday, Nov 13^{th}. For each program you write you should make sure you test each aspect (e.g., each branch when using conditional statements) and ensure that you have used suitable comments.

1. Write a new program that prints the larger of two integers (or if they are equal)

2. Write a new program that prints the largest of four integers (or if they all equal one another).

3. Write a new program that prints the first and last element of each list:

```
list1 = [1, 2, 3, 4, 5, 6, 7]
list2 = [16, 2, 43, 22., .11116, 2./3., 72.]
list3 = ['a', 'b', '63']
list4 = ['abcde', '54']
```

4. Write a new program with Spyder and call it alphabet.py (or something like that) and copy in the following text:

```
string1 = 'abcdefghijklmnopqrstuvwzyz'
string3 = 'hij'
```

(a) Modify alphabet.py to check if the 3rd element in each string is a 'j'. If it is, print out the entire string.[1]

(b) Modify alphabet.py to find the index in string1 where the string 'hij' of string 3 begins.[2]

(c) Modify alphabet.py to create a new string that represents the alphabet minus "hij". Though strings are immutable you can form new ones using indexing. Your program should look something like the program below (**N.B.**: The code below contains some intentional errors!)

(d) Compute another new string that does not include "hij" or "opq".

[1] **Hint**: You will find conditionals useful.

[2] **Hint**: use the function "index". If you don't know the syntax look it up online!

```
"""
This program experiments with strings. However, it has some errors in it!!
@author: Beth Wingate, modified by G. Vallis
Date: 9 November, 2018
"""

string1 = 'abcdefghijklmnopqrstuvwzyz'
string3 = 'hij'

# Experiment with strings
# print out each letter of string3

print('first letter of string3 ', string3[1])
print('second letter of string3 ', string3[2])
print('third letter of string3 ', string3[3])

# At which index in string1 does the sequence 'hij' (defined in string3) begin?
hij_index = string1.index(string2)
print(hijindex)

# Form a new string that doesn't contain 'hij'.
newstring = string1[0:hij_index] + string1[hij_index + 3:len(string1)]
print(newstring)
```

5. The roots of a quadratic $ax^2 + bx + c = 0$ are be given by the equation at right[3]

[3]

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Write a program roots.py that calculates the roots of a quadratic using this formula.[4]

[4] **Hint**: You will have to use a conditional statement to deal with what happens if there are no real roots or the roots are equal.

6. Now that you know more about lists, revisit plots. Modify your plotting program like this:

```
"""
An example plotting program.
Create two lists, plot them.
@author: Beth Wingate
"""
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [1, 7, 3, 5, 12, 4.5, 2, 10]
plt.plot(x, y)
plt.show()
```

(a) Modify your plotting program so that it will make a plot only of X[2:6] and Y[2:6].

(b) Modify your plotting program so that it will make a plot only of X[2:6] and Y[3:7].

(c) Modify your plotting program so that it will make a plot only of X[2:6] and Y[3:8]. Why doesn't this work?[5]

[5] **Hint:** check how many values of x there are, then compute how many values of y.