Introduction to Python Programming – Session 2

V1.0

e-mail: training@glasgow.ac.uk

web: gla.ac.uk/services/it/training

Course content created by Blair Thompson

Contents

Introduction

This course runs in three, three hour sessions. It is designed to be an introduction to simple programming in Python for non-programmers. It is not a complete Python programming course. It is intended as course which will enable you to write simple programs to manipulate and analyse data.

Objectives

On successful completion of this course participants will be able to:

- Understand what a computer program is.
- Use the IDLE Shell and Editor windows
- Write a simple print script
- Save your program as a Python program
- Run a Python script from the command prompt
- Include comments in Python scripts
- Assign values to variables
- Perform basic calculations
- Use If statements
- Use For and While Loops in Python.
- Use the Completion tool to speed up your coding
- Manipulate text using Python.
- Open and Save text based files within a Python script
- Define functions and use them in your scripts
- Import modules and use their contained definitions

Session 2

At the end of this session you should be able to

- Create loops in Python to control your scripts
- Manipulate strings
- Write your own scripts more efficiently
- Create and manipulate lists
- Write short scripts without the aid of a safety net

1 Loops (Control Flow Statements)

In all our tasks, so far our code has completed a single action and then stopped running. But when writing code, we may wish to run code a number of times, or perhaps continue to run until we get a desired outcome. We call this **iteration**. For and While loops allow us to do this.

a. While loops

A while loop will run the code contained for as long as a condition is true.

The general Python syntax for a simple **while** statement is

while condition:

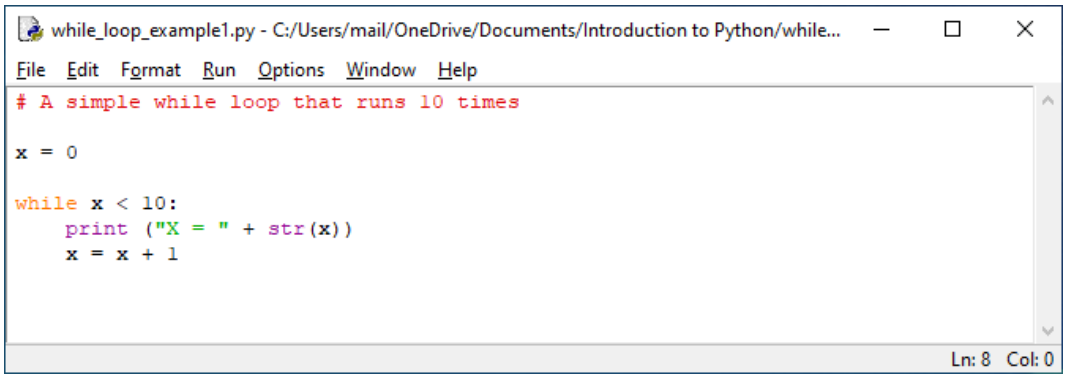
    do this

The condition can only evaluate as true or false and uses the same format and operators that we saw in the if statements before. The while loop will run over and over for as long as the condition is true. If the answer to the condition never changes to false your loop will run indefinitely.

Task: A simple while loop

1 Click **File** and then **New File**

2 Enter the following code:



3 Save the file as **while_loop_example1.py**

4 Run the script

5 Close the script.

With this example, we are performing an action until x < 10, but we can also use while loops to wait for other events to occur. The following tests can be applied to a variable within your while loop.

Test operations

== equal

!= not equal

< less than

<= less than or equal to

> greater than

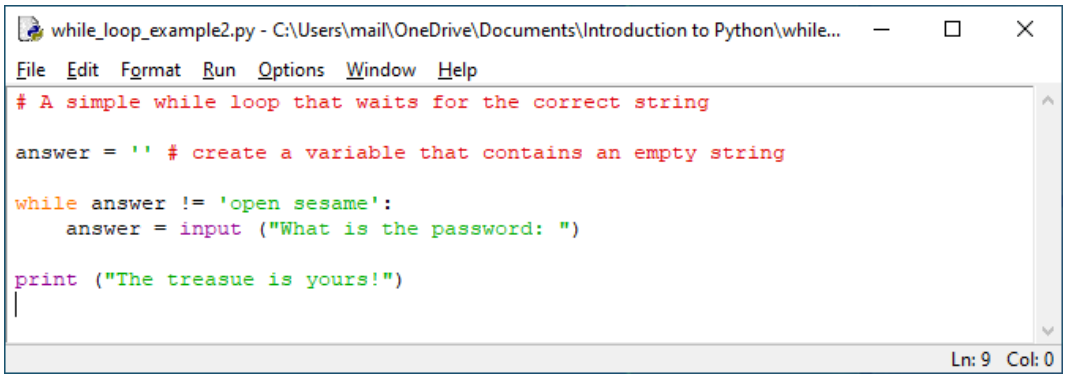
>= greater than or equal to

Odd as it may seem, these test operations can be performed on either numbers or strings.

Task: A while loop that waits for the correct string

1 Click File and then New File

2 Enter the following code:



3 Save the file as **while_loop_example2.py**

4 Run the script

In this example, we used the != operator to compare the answer to the password. != means does not equal, and can be a useful comparison to make.

Additional task: Add a message for those who don't enter the string correctly

To the script you have just created add some additional code that will tell the user politely when they enter an incorrect password.

Be careful as to how you indent the code!

Close the script when you are finished.

b. For loops

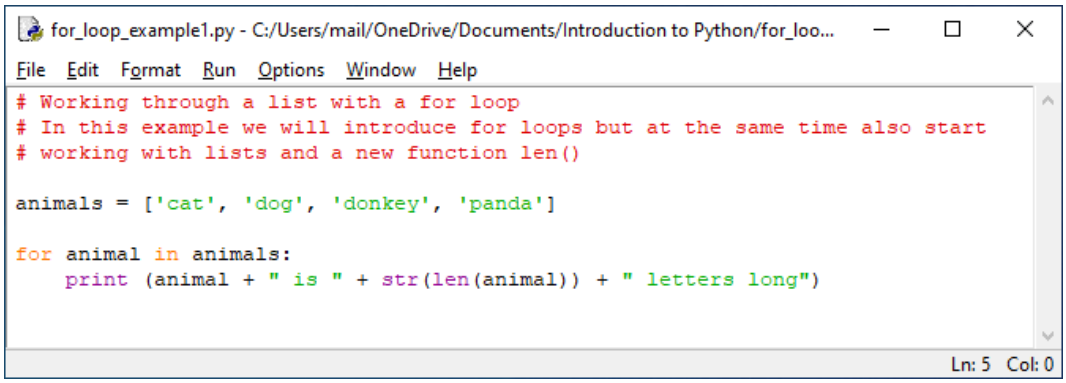
Programs can complete a series of operations repeatedly for a set number of iterations. Python's for statement iterates over the items of any list or string, in the order that they appear in the sequence.

In python, we can use **for** loops to work our way through a list of items or through a number of iterations.

Task: Working through a list with a for loop

1 Click **File** and then **New File**

2 Enter the following code:



3 Save the file as **for_loop_example1.py**

4 Run the script

When you code "**for animal in animals:"** you are creating a new variable animal. This **iteration variable** contains a single item from your list which changes each time the loop is complete.

Note: In this case, the **animals** variable represents a **list**, you can tell it's a list because the items are contained within square brackets.

You could also have used ( ) brackets here too because in this script we are not going to change any of the values in the list. Values stored in ordinary brackets are called a **tuple**. Later in this course we will look at lists in more detail.

We also are introduced to the **len()** function in this example, mostly just so that we could have our for loop do something "useful". len() is one of several functions that performs actions on strings. len() returns the length of a string.

c. A for loop using the range function

We can also combine a for statement with other functions from Pythons built-in functions. The range function returns a list depending on the arguments that it is given. E.g.

```
range (4) == [0, 1, 2, 3] # this is True!
```

This **for** loop has a set starting point, a set end point and a set increment.

The format is:

```
for i in range([start], stop[, step])
    # do something
```

- `start`: Starting number of the sequence.
- `stop`: Generate numbers up to, **but not including** this number.
- `step`: Difference between each number in the sequence.

The items in the square brackets are optional.

Example For Loop

```
For i in range(5, 10, 1):
    print "Looping " + str(i) + " times!"
```

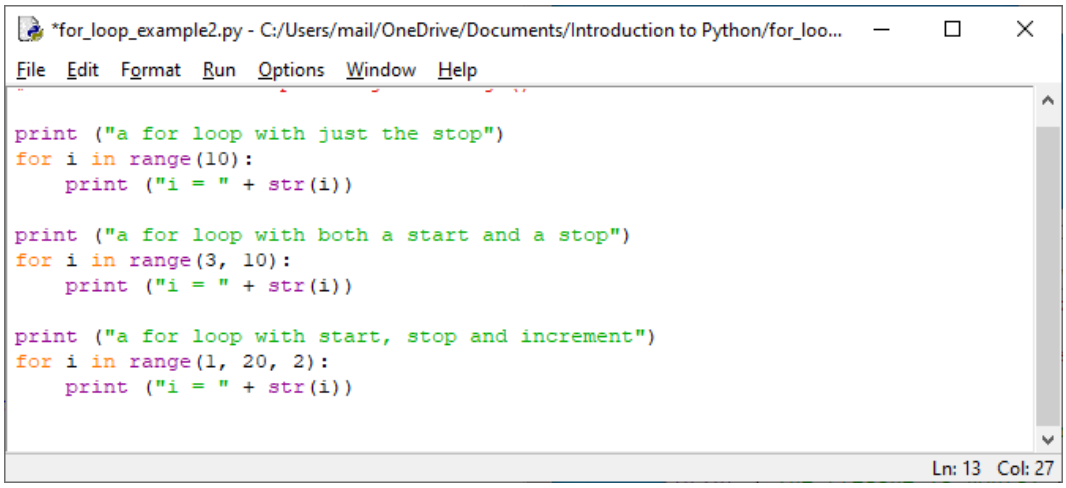
- Starts at variable ‘i=5’
- Runs until variable ‘i=10’ (to be precise no 9 is the last iteration)
- Variable ‘i’ increments by 1 at each iteration.

Task: A For Loop Using the Range Function

Write a script that demonstrates several different uses of the range() function including printing all the even numbers from 2 to 20

1 Click **File** and then **New File**

2 Enter the following code:



3 Save the file as **for_loop_example2.py**

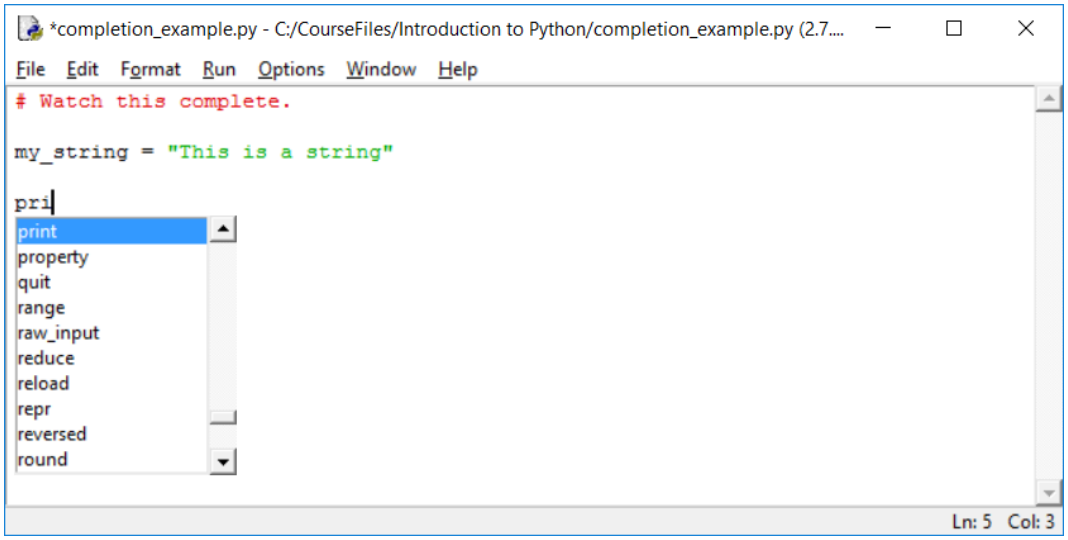
4 Run the script

5 Examine carefully the result of your script.

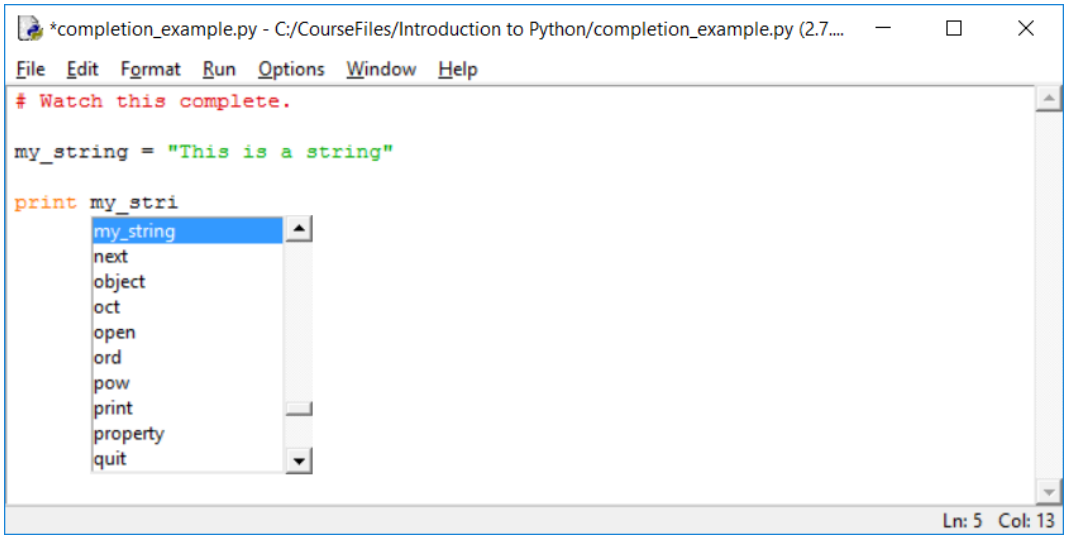
Note: In the for loops using range examples we have created we have used the small letter i (short for index) as our **iteration variable**. We could have used any variable name here and the code will still work. It is convention though to use the letter i in this type of for loop (and not just in python, in most programming languages) and following convention makes it easier for others to read and understand our code.

2 The Completions tool

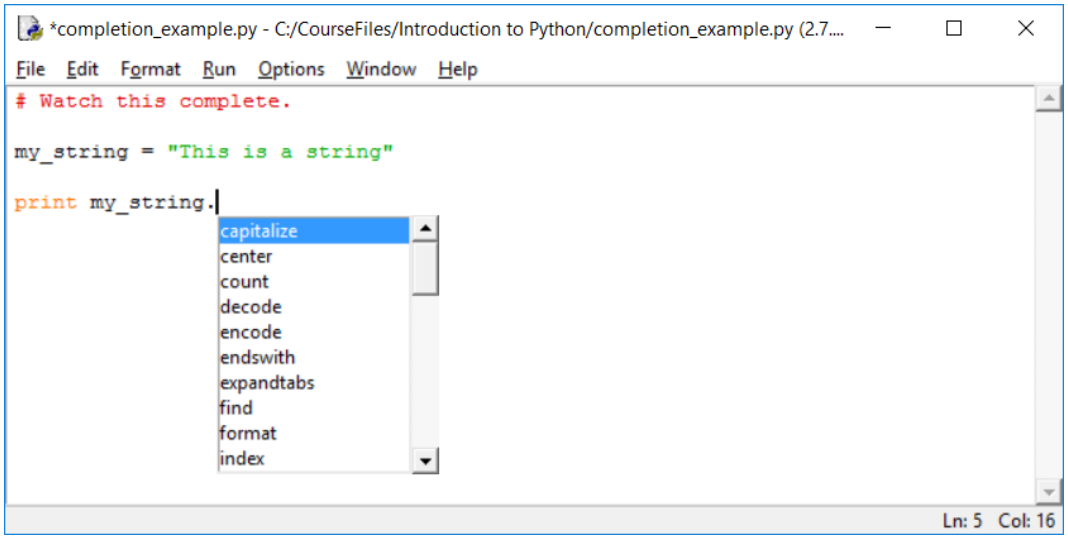
Within Python the completions tool can be very useful to us when we are writing code. When you are inputting code if you hit CTRL + SPACE on the keyboard while you are halfway through typing the name of a variable, list function etc. the editor will produce a drop-down list with suggested completions.



The IDLE editor can be a little inconsistent here, sometimes it will not perform autocompletions correctly until the script has been run once.



Methods can be applied to objects (such as strings as we will see in the next section), the completion tool can be very handy in displaying the methods that are available for an object that you are working with in your code!



You can use the up and down arrows on our keyboard (or the mouse) to select the item you wish to use.

Use the TAB key one you have selected the autocomplete item that you wish to use.

Over the next few exercises try and use the completion tool to speed up your writing of scripts!

3 Strings

- Strings are like any other variables
- Consider strings to be lists of characters.
- We can check for sets of characters, replace characters, reorder characters, etc.
- We can also print them and read them in.
- Remember Python scripts will fail if you try to treat strings as numbers (well unless we take steps to convert them first!)

a. Strings: Indexing

There are many ways to retrieve characters from strings, one of the simpler being by indexing or slicing them.

We can retrieve a character from a string like this.

```
my_string = "University of Glasgow"

print (my_string[0])      # prints the letter U

print (my_string[11])     # prints the letter o
```

The number inside the square brackets is the index of the letter that you wish to return. Remember that in Python we start counting at 0, i.e.

my_string	U	n	i	v	e	r	s	i	t	y		o	f		G	l	a	s	g	o	w
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

As well counting our index from left to right, we can also index characters in our string from right to left like this:

```
print (my_string[-1])      # prints the letter 'w'

print (my_string[-7])      # prints the letter 'G'
```

b. Slicing Python strings

As well as indexing a single character from our string we can also take a slice out from it like this:

```
print (my_string[11:13]) # prints 'of'

print (my_string[:10])     # prints 'University'

print (my_string[-7:])     # prints 'Glasgow'
```

Notice that the index before the colon is the starting position in the string, the index after the end position, but that the character in the end position is not returned.

We can also include a third parameter known as the stride. The stride tells Python how many characters to skip when retrieving characters. If you don't tell Python this it assumes that you want to progress by one each time.

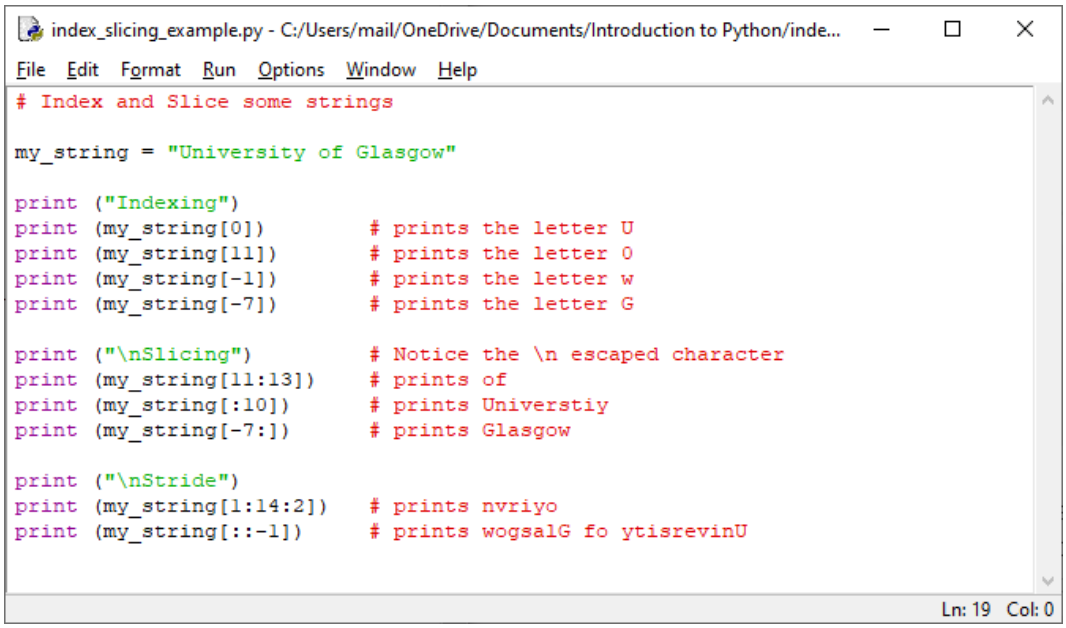
```
print my_string [1:14:2] # prints 'nvriyo '

print my_string [::-1]     # prints 'wogsalG fo ytisrevinU'
```

Yes, you can use negative values for the stride!

Task: Index and Slice a string

1 Click **File** and then **New File**

2 Enter the following code:



3 Save the file as index_slicing_example.py

4 Run the script and observe the results

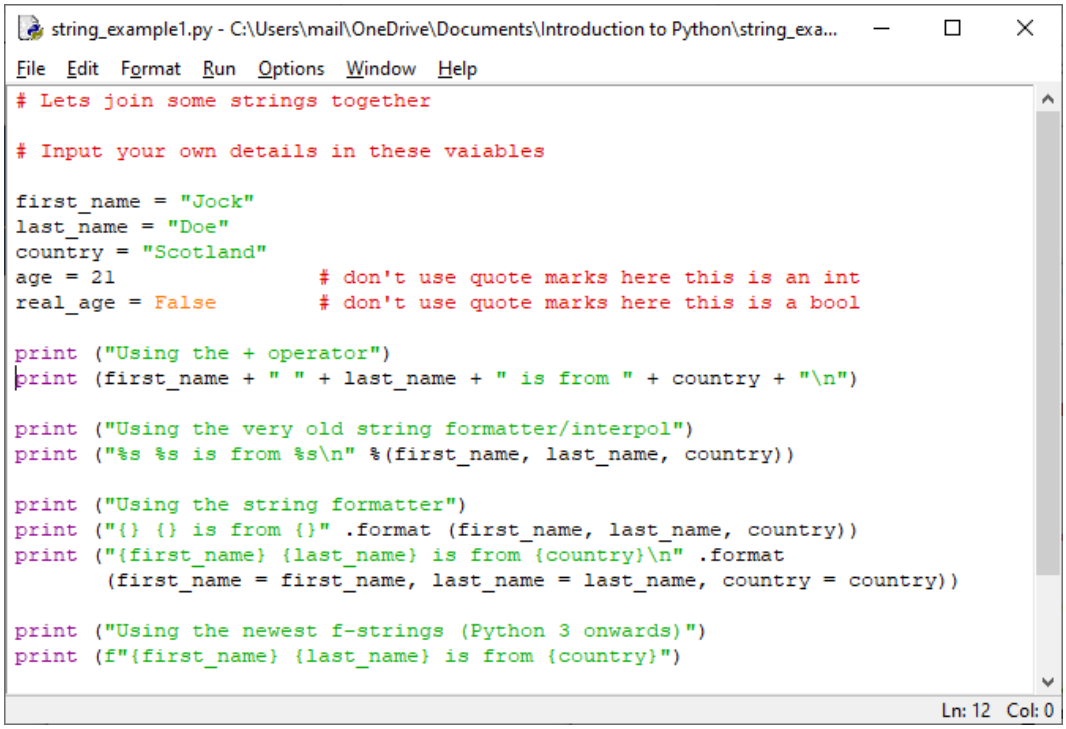
c. Concatenate/merge strings:

There are many ways to concatenate strings within python. So far, we have been using the + sign to join them together. But this is not especially efficient. Let's try some other methods to do the same.

Task: Concatenate strings

1 Click **File** and then **New File**

2 Enter the following code:

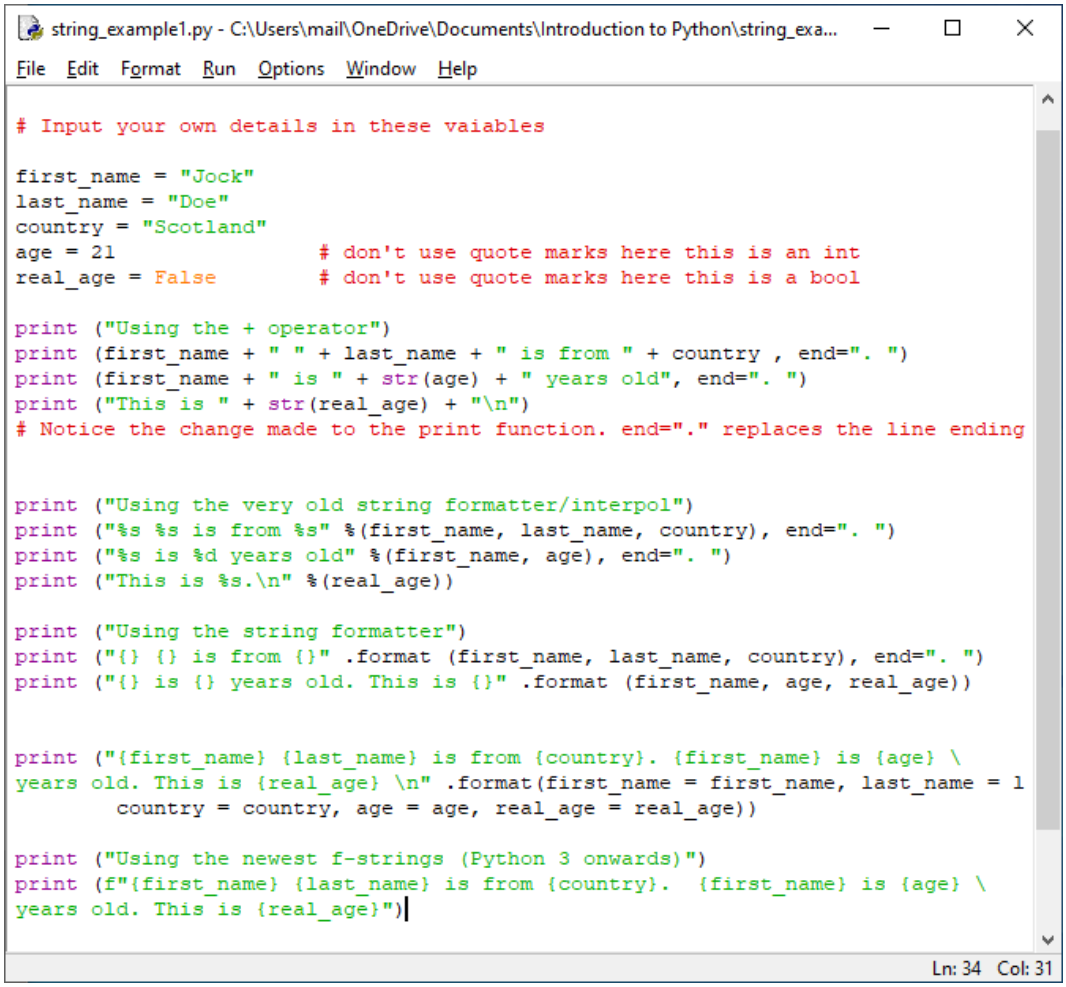


3 Save the file as **string_example1.py**

4 Run the script

5 Examine carefully the result of your script. The concatenated strings that we have printed should all be exactly the same. In this case the code is roughly of equal length, so which should we use?

6 Edit the code to read:



7 Use File – Save As to save the script as **string_example2.py**

8 Run the script

As you can see from the code that we have just executed, using the + operator to concatenate strings works to a degree. While all the variables are strings, it is not too cumbersome.

However, when we start working with more than one data type, we must be very careful that we convert all data into strings (using the str command) before we can add them together. In our example this led to the code stretching across three lines. It also makes it more likely that you might make a mistake and try to concatenate two incompatible data types.

You will see lots of examples of the % formatter being used if you are researching python on the internet or in books, although this formatter works on Python 2 and the newer Python 3, the {} is easier to use and more powerful.

String formatters offer a more powerful tool for concatenating your strings especially the latest str.format() method (the method using {} brackets) As well as concatenating the integer and Boolean (true/false) variables, the str.format() method also automatically converted the variables to the string data type. Hence the code was more compact and less prone to error!

However, the newest f-strings are an even greater improvement even over the string formatter in this example. Being able to inject variables directly into a string like this produces shorter, easier to read code.

Unless you need your code to be backwards compatible to older versions of Python, It will be easiest to use f-strings.

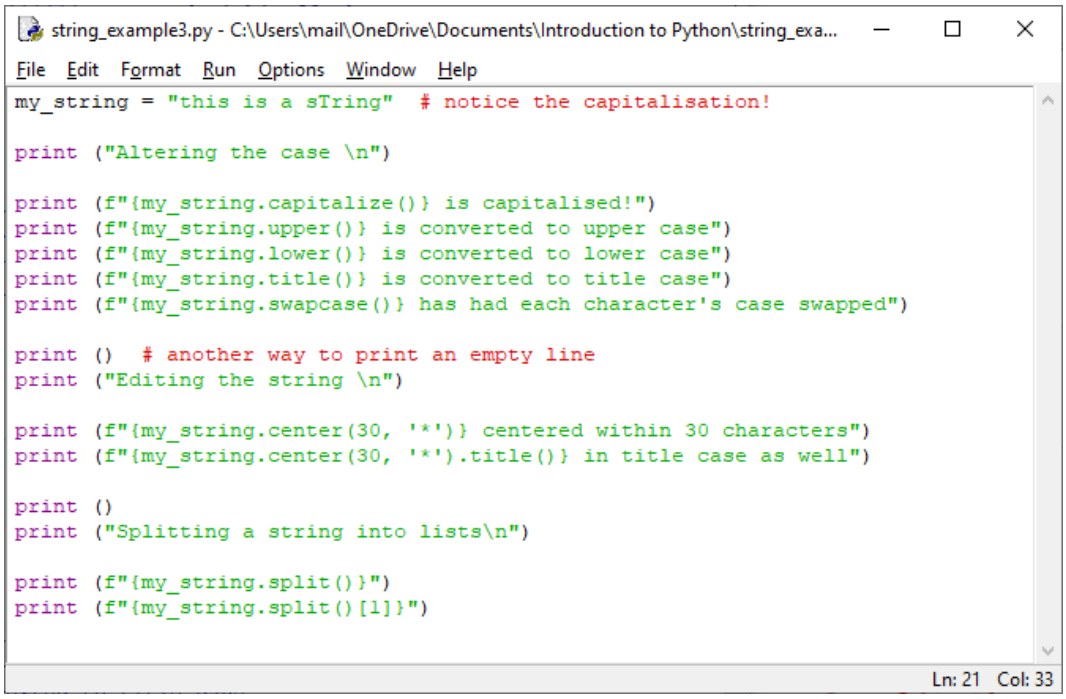
d. Strings: Other methods that manipulate strings

We can use other methods to manipulate out text strings. There are lots and lots of things we can do… The next few tasks looks at a small sample of them.

Task: Further Manipulation of Strings

1 Click **File** and then **New File**

2 Enter the following code:



3 Save the file as **string_example3.py**

4 Run the script

5 Examine the output to see what effect the code has had on our string

Task: Finding strings within strings

1 Using our previous script add the following

```
print ()
print ("Finding text in a string")

print (f"{my_string.count('s')} s's in the string")
print (f"It's {my_string.endswith('ing')} the string ends with ing")
print (f"The characters str are found at position {my_string.find('str')}")
```

Ln: 28 Col: 69

2 Save the file

3 Run the script

4 Examine the output to see the effect of this additional code.

Additional Task: Using Loops and Strings Together

1 Using the previous script add the following

```
# Lets try and make some code that makes use of strings

print ()
print (" Try this ".center(30, '*'))

big_string = "This is a larger string, we can use it to find two strings"
print (big_string.split())

pos = 1
for word in big_string.split():
    print (f"Word : {str(pos).ljust(3)} {word}")
    pos = pos + 1
```

Ln: 41 Col: 17

On the second last line this script uses .ljust(3) method on the string created by str(pos). Take a moment to work out what this line of code is doing (hint look at the .center example earlier in the script).

e. Other String Operations

- my_string = my_string .rstrip() # Removes line termination character.
- my_string = my_string [:-1] # re-assigns the string with the last character removed

Additional Task: Strings

- Write a script that checks to see if a given string is a palindrome.
- Write a script that takes in first name and surname separately and outputs a hello message with the first name and surname on the same line.
- Write a script that looks for “cat”, “dog” & “bird” in the following string: “At school we learned addition, multiplication, subtraction and division.”

4 Lists

Unlike other programming languages, python does not use arrays. Instead it uses an arguably more flexible tool called lists. We saw an example of a list when we were learning about **for** statements.

Lists offer many advantages within our scripts, including:

- We can group items together.
- With lists we can access individual items with an index.
- Grouping like items is easier than working with separate variables. for example, we can then use iteration within our scripts
- Lists are a fundamental part of programming.

a. Creating a list

In Python, we create a list in a similar manner to the way we assign any other variable.

The difference is that we use square brackets to enclose the values and commas to separate them. i.e.:

my_animals = ['donkey', 'elephant', 'tiger', 'giraffe'] # this list contains text strings

my_numbers = [2, 6, 3, 8, 1, 16] # This list contains integers

my_floats = [2.0, 6.0, 3.1, 8.5, 1.0, 16.2] # this list contains floats

my_answers = [True, True, False, True, False] # this list contains bools

mixed_list = ['donkey', 6, 3.1, True] # This list contains all of the above!

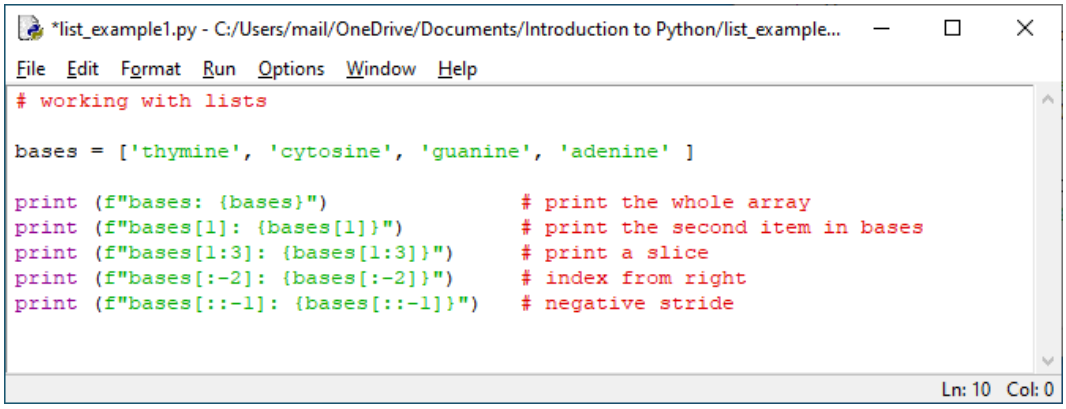
b. Retrieving items from a list

We can retrieve items from a list using **indexes and slicing** in a very similar fashion to the way we retrieved characters from strings.

Task: Create a list and print its members

1 Click **File** and then **New File**

2 Enter the following code:



3 Save the file as **list_example1.py**

4 Run the script

c. Iterating List Elements

Iterating is the process of working our way through a list and performing an action on each of the members of the list. There are several ways that we can iterate through a list in Python.

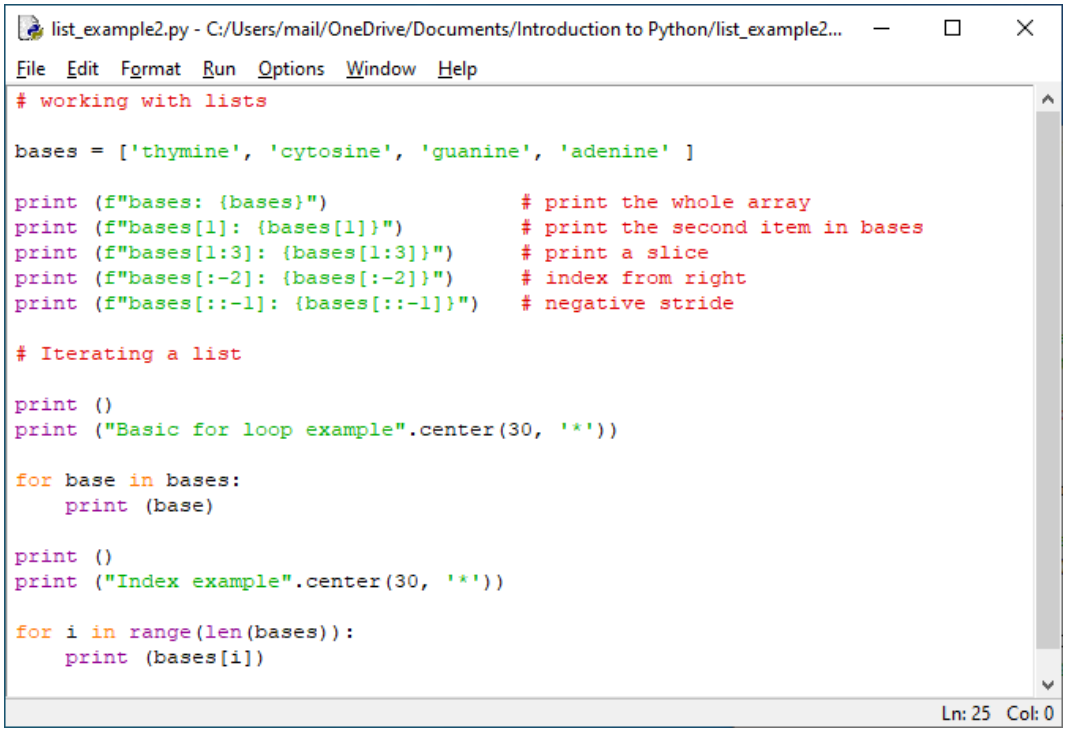
Let's have a look at some ways we can do this:

Task: Iterate a list and print its members

1 Click File and then Save As

2 Save the file as list_example2.py

3 Edit your script to look like this:



4 Run the Script

5 Close the editor

Additional Task Split a String into a List

- Write a program using the split function to put the following into a list

  Bismark, Disraeli, King George, Kaiser Wilhelm

- Print the list values
- Print the list values backwards

Hint: .split()

Programming Challenges: Lists

Following are some programming challenges for you to try. Unlike the earlier exercises we are not giving you much help with these, a lot can be achieved using tools you have already seen. If you have any problems, consider using internet resources for inspiration. This form of problem solving is something that most people will do a lot of as they learn to code!

Write them a little bit by a little bit, running them frequently to see if they are working. Pay attention to any error messages that you receive, often these error messages will lead you to the correct answers.

Try the following:

- Write a script to print the words “Sorry”, “Dave”, “I”, “can’t”, “do” & “that.” from a list into a properly formatted message (i.e. a complete string)

- Write a script to sort a list into ascending order. Contents can be numerical or string.

  *(Hint: There is a method named .**sorted** that can be used on lists, although this is not the only way to achieve this **sort** can be used as well. See if you can find out the difference)*

- Write a script to change the inclusive Vat, on a list of prices, from 17.5% to 20%.

  To find out the price exclusive of 17.5% VAT divide by 1.175,

  To add 20% VAT multiply by 1.2

  Try using loops to simplify your code in this example

Appendix 1 Python Built in Functions

Function	Description
abs()	Return the absolute value of a number.
all()	Return `True` if all elements of the iterable are true (or if the iterable is empty).
any()	Return `True` if any element of the iterable is true. If the iterable is empty, return `False.`
ascii()	Return a string containing a printable representation of an object, but escape the non-ASCII characters.
bin()	Convert an integer number to a binary string.
bool()	Convert a value to a Boolean.
bytearray()	Return a new array of bytes.
bytes()	Return a new "bytes" object.
callable()	Return `True` if the object argument appears callable, `False` if not.
chr()	Return the string representing a character.
classmethod()	Return a class method for the function.
compile()	Compile the source into a code or AST object.
complex()	Create a complex number or convert a string or number to a complex number.
delattr()	Deletes the named attribute of an object.

Function	Description
dict()	Create a new dictionary.
dir()	Return the list of names in the current local scope.
divmod()	Return a pair of numbers consisting of quotient and remainder when using integer division.
enumerate()	Return an enumerate object.
eval()	The argument is parsed and evaluated as a Python expression.
exec()	Dynamic execution of Python code.
filter()	Construct an iterator from elements of iterable for which function returns true.
float()	Convert a string or a number to floating point.
format()	Convert a value to a "formatted" representation.
frozenset()	Return a new `frozenset` object.
getattr()	Return the value of the named attribute of an object.
globals()	Return a dictionary representing the current global symbol table.
hasattr()	Return `True` if the name is one of the object's attributes.
hash()	Return the hash value of the object.
help()	Invoke the built-in help system.
hex()	Convert an integer number to a hexadecimal string.

Function	Description
id()	Return the "identity" of an object.
input()	Reads a line from input, converts it to a string (stripping a trailing newline), and returns that.
int()	Convert a number or string to an integer.
isinstance()	Return `True` if the object argument is an instance.
issubclass()	Return `True` if class is a subclass.
iter()	Return an iterator object.
len()	Return the length (the number of items) of an object.
list()	Return a list.
locals()	Update and return a dictionary representing the current local symbol table.
map()	Return an iterator that applies function to every item of iterable, yielding the results.
max()	Return the largest item in an iterable.
memoryview()	Return a "memory view" object created from the given argument.
min()	Return the smallest item in an iterable.
next()	Retrieve the next item from the iterator.
object()	Return a new featureless object.

Function	Description
oct()	Convert an integer number to an octal string.
open()	Open file and return a corresponding file object.
ord()	Return an integer representing the Unicode.
pow()	Return power raised to a number.
print()	Print objects to the stream.
property()	Return a property attribute.
range()	Return an iterable sequence.
repr()	Return a string containing a printable representation of an object.
reversed()	Return a reverse iterator.
round()	Return the rounded floating point value.
set()	Return a new set object.
setattr()	Assigns the value to the attribute.
slice()	Return a slice object.
sorted()	Return a new sorted list.
staticmethod()	Return a static method for function.
str()	Return a str version of object.

Function	Description
sum()	Sums the items of an iterable from left to right and returns the total.
super()	Return a proxy object that delegates method calls to a parent or sibling class.
tuple()	Return a tuple
type()	Return the type of an object.
vars()	Return the `__dict__` attribute for a module, class, instance, or any other object.
zip()	Make an iterator that aggregates elements from each of the iterables.
__import__()	This function is invoked by the `import` statement.

Appendix 2 DOS Commands

Command	Function	Example
cd	Change Directory	cd.. go back a directory cd c:\perl\myscripts\
Mkdir	Make a new directory in the current folder	“Mkdir newfolder”
Copyfile	copies file1.pl to file2.pl	“Copy file1.pl file2.pl”
Del	deletes file1.pl – be careful!	Del file1.pl
Doskey	starts remembering commands.	
Exit	: closes command prompt	

Useful Shortcut keys

Using keyboard shortcuts can help you become more efficient when creating documents in Microsoft applications. Most keyboard shortcuts require you to use two or more keys at the same time. To use a keyboard shortcut first press and hold down the modifier key or keys (i.e. SHIFT, CTRL, ALT) and then press the corresponding standard key on your keyboard.

Function	Shortcut
Save and run your script (in IDLE)	F5
Open	CTRL+O
Save	CTRL+S
Close	ALT + F4
Cut	CTRL+X
Copy	CTRL+C
Paste	CTRL+V
Select all	CTRL+A
Indent line	CTRL+I or Tab
Cancel	Esc
Undo	CTRL+Z
Re-do	CTRL+SHIFT+Z
Find	CTRL+F
Replace	CTRL+H
Show Completions	CTRL+SPACE