**PYTHON Programming Exercises 1**

Welcome to Python Programming. In these exercises we are going to learn as much as possible about Python Programming from simple examples. The more code that we play around with and use, then the easier Python becomes to learn.

**Task 1: Online Learning**

There are a number of online resources for Learning Python. One such resource is CodeAcademy. Open a web browser and go to their website: [**www.codecademy.com/catalog/language/python**](http://www.codecademy.com/catalog/language/python)

If you haven’t used the website before, then you will need to register with them (just a username, e-mail address and a password). Register and Get Started. Spend 15 minutes or so familiarising yourself with the platform and work through the first few tasks. Once you understand the basics, leave it for now (and come back to it to study in your own time as Private Study time).

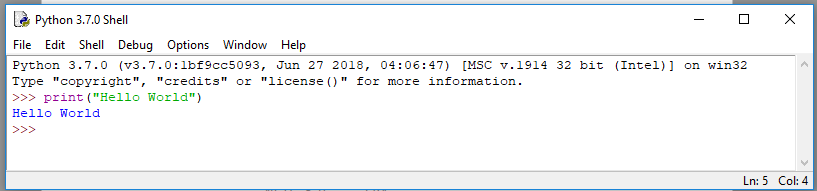
**Task 2: Your First Python Code**

Open up the Python 3.9 (or 3.X) IDLE on your PC. Refer to the lecture slides if you are not sure, (or e-mail me: [david.kidner@southwales.ac.uk](mailto:david.kidner@southwales.ac.uk)). The IDLE Shell window will help us learn about the different elements of our code by using different colours to represent them.

At the “>>>” prompt, type:

print(“Hello World”)

And press ENTER.



Congratulations, you’ve successfully completed your first Python command. Let’s try some more. Type the following commands:

print(“5\*4”)

print(5\*4)

The quotation marks define explicitly what you want printed to the screen. To print the calculation, we need to remove the quotation marks. As Python interprets each expression, we don’t always have to use the Print command for these (as Python will interpret and display the result). Type the following (lazy) commands or full commands:

5\*4 Print(5\*4)

5\*4+3 Print(5\*4+3)

5\*(4+3) Print(5\*(4+3))

The parentheses specify the precedence order of the expression. If in doubt, always use parentheses. Remember: Be Logical! Let’s try some more examples. Type the following commands which use variables:

a=5

b=4

a

b

a+b

a-b

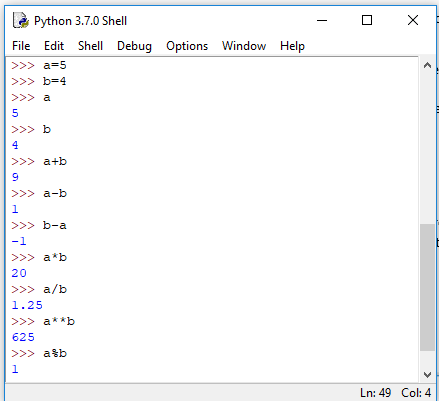
b-a

a\*b

a/b

a\*\*b

a%b



a%b = 1, as 1 is the remainder after dividing 5 by 4. The Modulus is a useful little function which is often used in programming when we are repeating or looping through code, as the remainder can be used to determine a set action. We will come back to this later. And spend some more time working with variables.

In the meantime, let’s try some more mathematical operations:

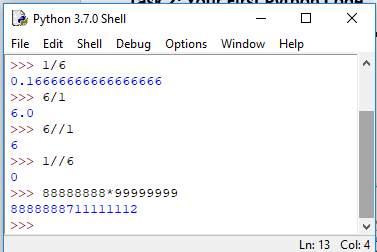
1/6

6/1

6//1

1//6

88888888\*99999999



Division in Python will produce a decimal number. In Python, these are called floats or floating point arithmetic. However, if you need an integer as opposed to a decimal answer, then we can use the double slash as in the examples above. However, the integer part of 1/6 (as 1//6) is zero. The last expression shows us that Python can handle quite large numbers relatively easily. Let’s try a few more:

0.1+0.1

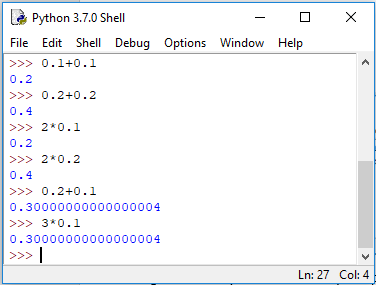
0.2+0.2

2\*0.1

2\*0.2

0.2+0.1

3\*0.1

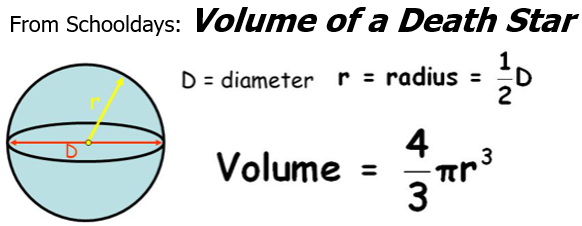


Hmmmm? While Python looks good at dealing with big numbers, it appears **Pants** at dealing with very small numbers? In fact, this is quite a common feature of all programming languages in that we can sometimes get an arbitrary number of decimal places. Python tries to represent numbers as precisely as possible, which is sometimes difficult given how computers have to represent numbers internally. It’s nothing to worry about for now.

**Task 3: Volume of a Death Star**



Python can be used for some extremely important analytical work. For example, we may have an overwhelming desire to calculate the Volume of the Death Star! Assuming that it is a perfect sphere (nearly) and has a Diameter of 160 Kilometres (Source: Wikipedia), how do we go about calculating its volume in Python?





In your Python Shell Window, type the following:

>>> d=160

>>> r=d/2

>>> pi=22/7

>>> volume=4/3\*pi\*r\*r\*r

>>> volume

Make a note of the Volume. Also, in School we might have learnt that a better approximation of pi is 3.14159 and r3 can be simplified as r\*\*3. Now type these 3 lines of code:

>>> pi=3.14159

>>> volume=4/3\*pi\*r\*\*3

>>> volume

Make a note of the Volume. Hopefully, our value of pi is a bit more accurate, so gives us a better volume result. Can we improve upon this? If we can’t remember pi to any more decimal places, then we could import the Python Maths library (like our in-built calculator functions) and use its value of pi. Type the following:

>>> import math

>>> pi=math.pi

>>> volume=4/3\*pi\*r\*\*3

>>> volume

Make a note of the Volume. This should be an accurate enough answer. Note that we do not need to re-enter the values of d and r. These will stay fixed until we define new values or close the Shell Window.

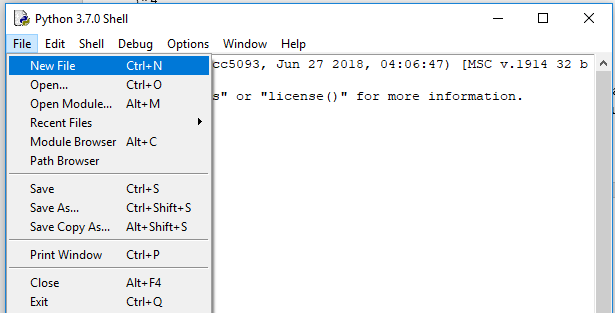
***Challenge:*** *The Death Star is not a perfect sphere. There is a hollow partial sphere in the surface of the northern hemisphere of the Death Star. If the hollow is equivalent to the volume of one third of a sphere of diameter 40 kilometres, then what is the actual volume of the Death Star? (Calculate the Volume of the hollow and subtract this from your original answer).*

**Task 4: Saving and Executing Your Code**

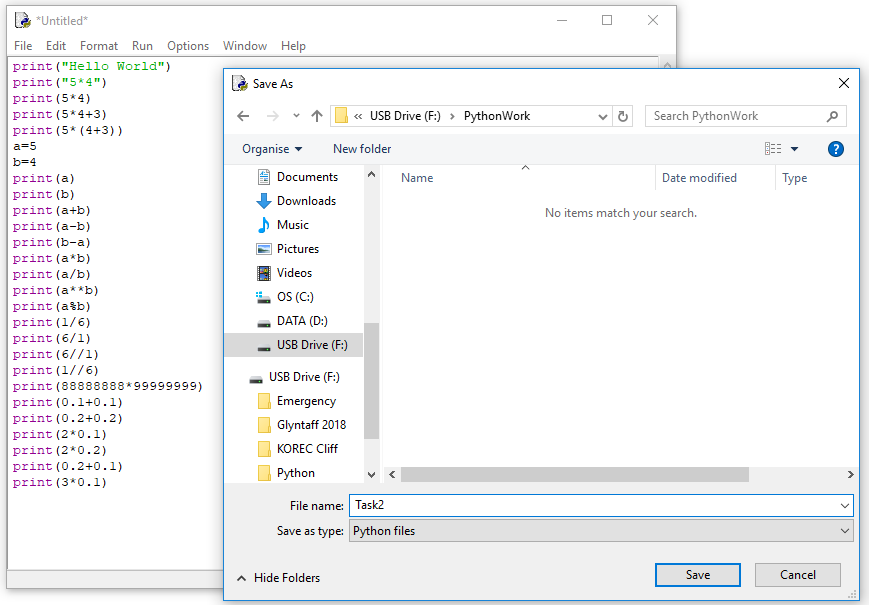
The calculations in Task 2 are “throwaway” answers that are just helping us to familiarise ourselves with Python. However, in Task 3 (Volume of a Death Star), we are performing important calculations that should be documented and saved. Some of the tasks might not be easily repeated again, so it is good practice to save our work.

**Top Tip: Buy a USB Flash Stick dedicated for your Python work. Create a Python folder & SAVE all your programs in this directory. Regularly backup your folder (say in the cloud), or e-mail it to yourself every few days (as well as on your hard drive if working at home; and on your USW OneDrive account). It’s better to be safe than sorry.**

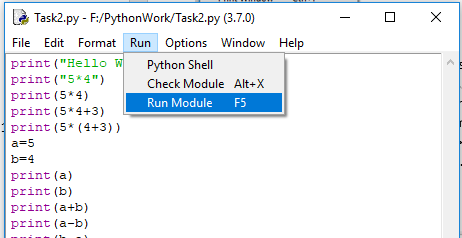
If the Python IDLE Shell isn’t open, open it up and click on **File > New File**.



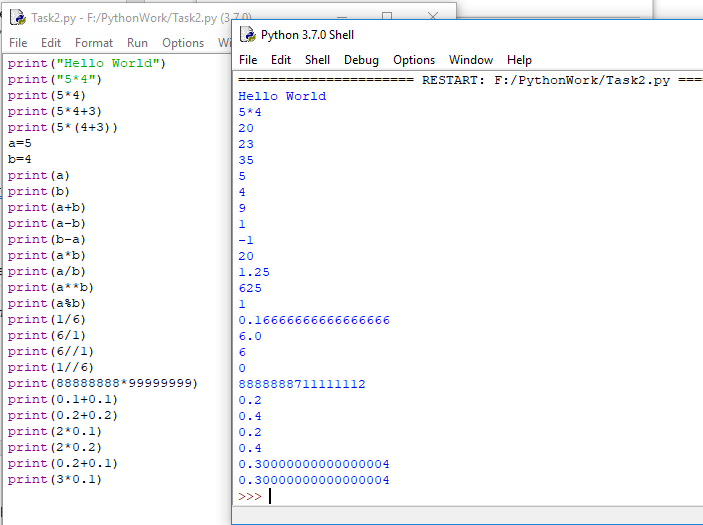
This will open a new window with Untitled as its name. This is the Python IDLE Editor and within it you can enter the code needed to create your future programs. Essentially, the code will appear the same as in the Shell window (without the >>> prompt) and we can enter multiple lines of code before they are interpreted and executed. To test it out, type in **SOME** (e.g. 5 or 6) of the commands from Task 1 above (**REMEMBER:** You will now have to use the **print** command to display these). Click **File > Save As** and navigate to your workspace (e.g. memory stick directory) and give the file a suitable name (e.g. **Task2**). This will save the file as **Task2.py** (where *.py* is the file extension for Python code).



You can execute (run) this program by selecting **Run > Run Module** or hitting **F5**.

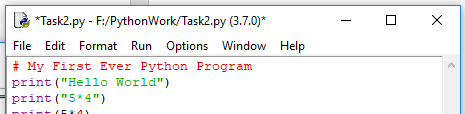


This will send the output to the Shell Window as below:

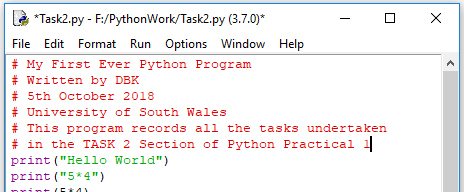


It is good practice to use a method of keeping our code readable by commenting on certain sections. For example, what the program does; what the variables are being used for; or to explain the workings of an algorithm. We might know today what the code does, but will we tomorrow, next week, next year? Or will somebody else?

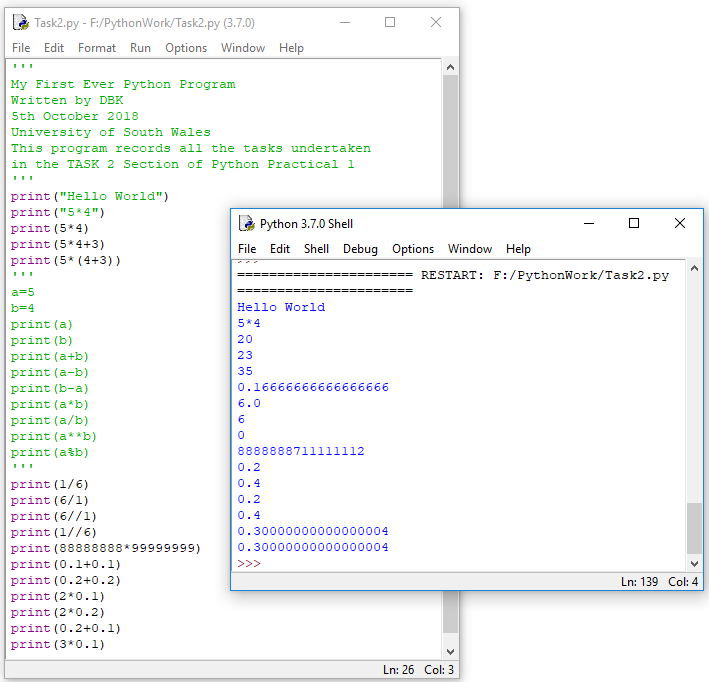
Comments are descriptive statements preceded by the # symbol. Python will not interpret these lines of code. Edit your Task2.py program to include the following heading:

 Then Save and Run it.

We could include a lot more information in our comments:

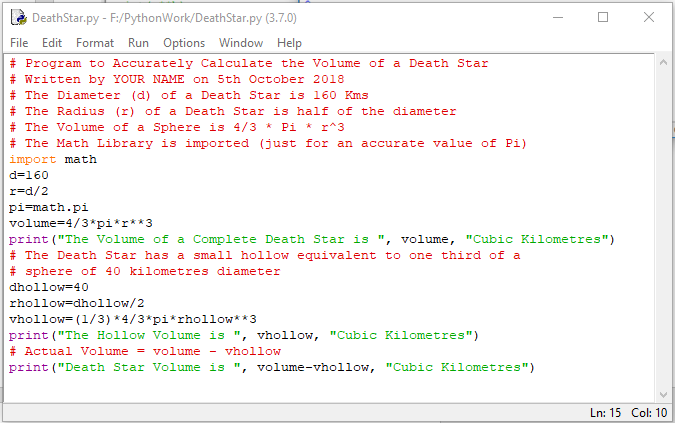
 Then Save and Run it.

Alternatively, we can use three single quotes before and after the section of comments. This is also useful if we want to block out a section of code:

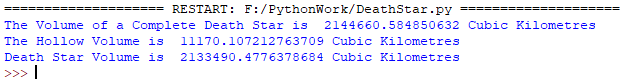


**Task 5: Save a Commented Python Program for the Volume of a Death Star**

Open a New File and enter the code from Task 3. Add some suitable comments …



Save as DeathStar.py in your workspace (USB / Flash) and Run …



**Task 6: Working With Text Variables and Strings**

We’ve already seen in the earlier tasks how we can assign numbers to variable names (e.g. a=5, d=160, pi=math.pi). Now we will explore the use of variables for text or strings of text characters.

Open up a Python Shell Window (the one with the >>> Prompt). Simply type (**with your own First and Last names**):

>>>name=David

(Oooops – as it is a string then we should enclose it in quotes)

>>>name=”David”

>>>name

>>>print(name)

>>>surname=”Kidner”

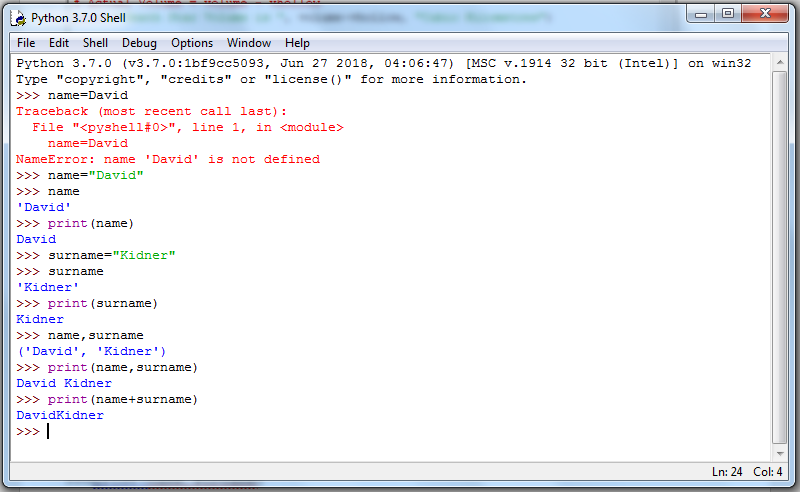
>>>surname

>>>print(surname)

>>>name,surname

>>>print(name,surname)

>>>print(name+surname)



OK, what does this tell us? Firstly, a text string variable must be defined in “” quotes. The IDLE shell is useful at reminding us of that by highlighting the text string in Green. Secondly, our lazy approach of just typing the variable name is not the same as printing the variable name. Typing the variable name returns the text string variable in single ‘’ quote marks to tell us that it is a string. Also, we can separate variable output by using the comma between variable names.

Let’s see what else we can do with text string variables. (Again USE YOUR OWN NAMES …)

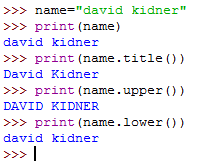
>>>name=”david kidner”

>>>print(name)

>>>print(name.title())

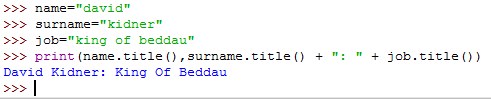
>>>print(name.upper())

>>>print(name.lower())

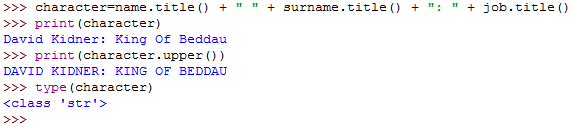


In this example, the lowercase string "david kidner" is stored in the variable name. The method **title()** appears after the variable in the **print()** statement. A ***method***is an action that Python can perform on a piece of data. The dot (.) after name in **name.title()** tells Python to make the **title()** method act on the variable name. Every method is followed by a set of parentheses, because methods often need additional information to do their work. That information is provided inside the parentheses. The **title()** function doesn’t need any additional information, so its parentheses are empty. **title()** displays each word in titlecase, where each word begins with a capital letter. This is useful because you’ll often want to think of a name as a piece of information. For example, you might want your program to recognise the input values David, DAVID, and david as the same name, and display all of them as David. Several other useful methods are available for dealing with case as well. In this example, we can change a string to all uppercase or all lowercase letters.

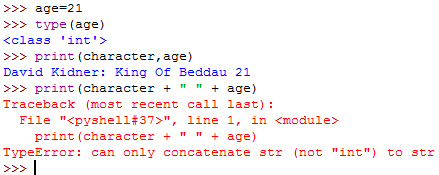
We’ve already seen that we can concatenate variables using the plus symbol (e.g. **print name+surname** above), although it didn’t work too well as it forgot to put a space in between the names! Let’s explore this a bit further. (Again, use your own details in the next few lines of code).



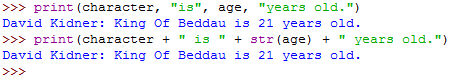
This is awfully fiddly and cumbersome, so take your time to make sure you get it right and understand what it is that Python is doing. If we want to use this new string a number of times, then perhaps it is better representing it as a new variable, rather than having to type this code in all the time? Give this a go …



Here, we have concatenated the three lower case character string variables (name, surname and job) into one new variable (**character**), which is in title case. We have then checked the type of variable that character is, by using the **type()** command. Python tells us that this is a STRING or ‘str’ type.



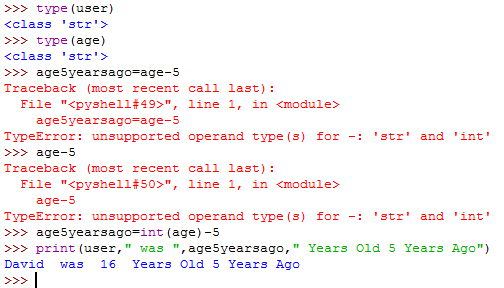
One thing that we should be aware of is that we can’t combine different variable types in the same command (e.g. a string and an integer in the above example). What we need to do is either not use the “+” function, or turn one into the other or vice versa. This is a process known as TypeCasting. We will consider both approaches below:



The former is quick and easy, whereas the latter is essential if we want to create new variables from variables of different types. Another example of TypeCasting is when we ask for input from the user. (We will consider Input in more detail next time, but for now, let’s just consider some simple examples of keyboard entry).



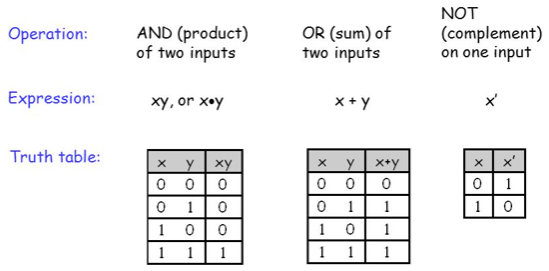
What variable types do you think **user** and **age** are? String and Integer? Check them out for yourself …



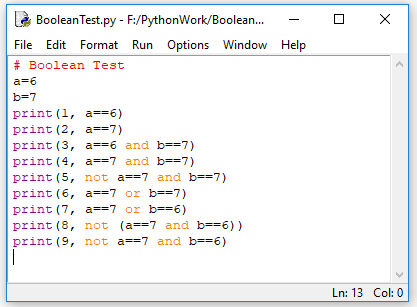
**user** and **age** are both strings, so we can’t do simple calculations with **age** until we convert it to an integer. (In the last **print** example, we don’

**Task 7: Boolean Expressions**

In computer programming we will often use logical expressions or statements that are either TRUE or FALSE. We can use these to compare data and test to see if it’s equal to, less than or greater than. The logical operators most commonly used in Python programming are AND, OR, NOT. The operands (arguments) have Boolean values that are either TRUE or FALSE (represented as 1 and 0 respectively in the tables below, where x, y are the operands).

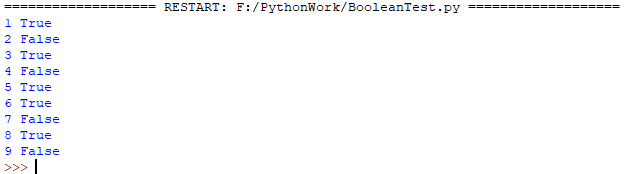


This takes a bit of getting used to, so let’s experiment with these in a Python program. Open a new file and enter the following code and save as BooleanTest.py :



The “==” Equality Operator compares two operands and will return TRUE if both are equal in value, or FALSE otherwise.

Run the Program …

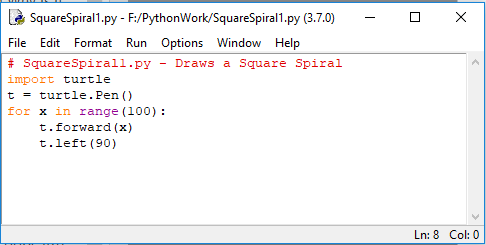


Take a bit of time to study the results and see if you can understand the outcomes. We’ll look at Comparison Tests and Boolean Expressions later.

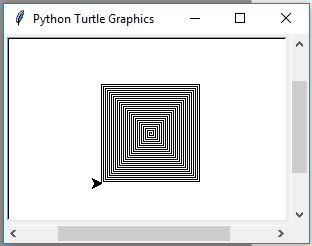
**Task 8: Programming Graphics in Python**

OK, if Task 7 hasn’t spaced you out, then this Task definitely will!! In this task, we're going to develop some very short, simple programs to create beautifully complex visuals. To do this, we’ll use Turtle Graphics (originally developed in the 1960s). In Turtle Graphics, you write instructions that tell a virtual, or imaginary, turtle to move around the screen. The turtle carries a pen, and you can instruct the turtle to use its pen to draw lines wherever it goes. By writing code to move the turtle around in cool patterns, you can make it draw amazing pictures. It's also a good way to learn a little bit more of some Python coding structures and the logic in the code. We will be loading (importing) the turtle graphics library (in the same way as we used the Math Library in Task 3.

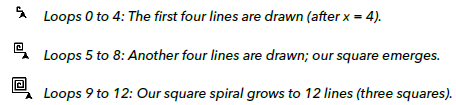
Open a new file from the Python IDLE and type in the following code:



Make sure you indent the last two lines of code (which are in a loop). Save the program as ***SquareSpiral1.py*** in your workspace and then run it. You should see a rather hypnotic square spiral …

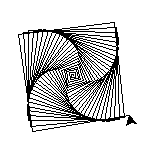


The second line imports the ability to draw turtle graphics. The third line of our program, t = turtle.Pen(), tells the computer that we’ll use the shortcut letter t to stand for the turtle’s pen. This will allow us to draw with the turtle’s pen as the turtle moves around the screen just by typing t.forward() instead of writing out turtle. Pen().forward(). The fourth line is the most complex. Here we’re creating a loop, which repeats a set of instructions a number of times (it loops through those lines of code over and over again). This particular loop sets up a range, or list, of 100 numbers from 0 to 99. (Computers almost always start counting at 0, not 1 like we do.) The loop then steps the variable x through each of the numbers in that range. So x starts as 0, and then it becomes 1, then 2, and so on as it counts all the way up to 99, for a total of 100 steps. The next two lines are indented, which means that they are *in the loop* and go with the line above, so they’ll be repeated each time x gets a new number in the range from 0 to 99, or 100 times. In this instance, the pen will move forward 0, then 1, then 2, then 3 pixels (growing each time) and turning left 90 degrees as follows:

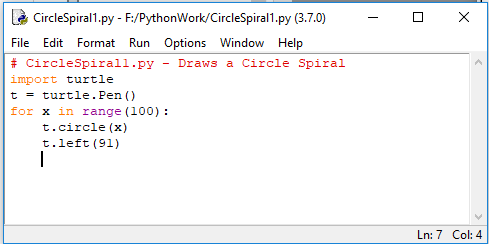


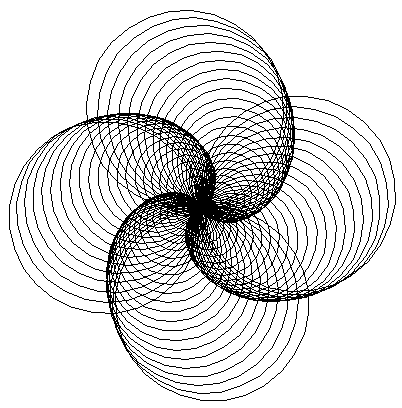
So each four loops through the code creates a square (i.e. four turns of 90 degrees). Change just the last line of the program to **t.left(91)** and save it as **SquareSpiral2.py** and run the program.

Experiment with other angles, e.g. 46, 75, 121, etc. to see what you end up. Save each program as SquareSpiral3, 4, 5.py etc.



Turtle Graphics can draw other shapes – not just straight lines. Edit your program as follows and save as ***CircleSpiral1.py***.

 and run the code …

… once again experiment with other angles.

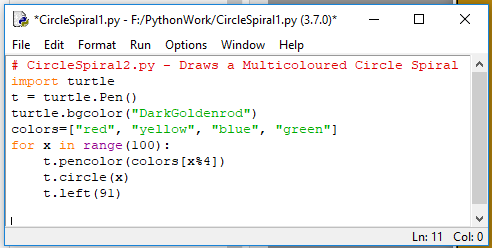
The circle spirals are much larger than the squares, as the value of x in this code is the radius of the circle (and not the length of the side of the square).

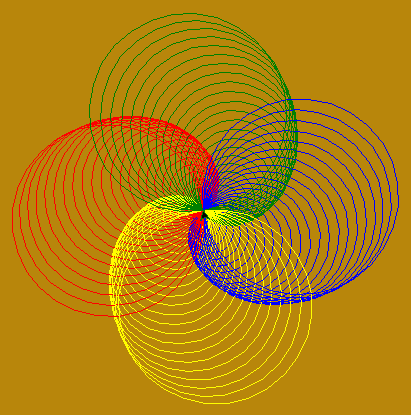
To add a bit of colour to the spirals, add the following line of code t.pencolor("red") before the for loop and run it again.

There are a wide range of colours supported in Turtle Graphics (see: <http://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm>) for a full list. Pick a pretty obscure colour from this list (e.g. “lemon chiffon”) and run it again.

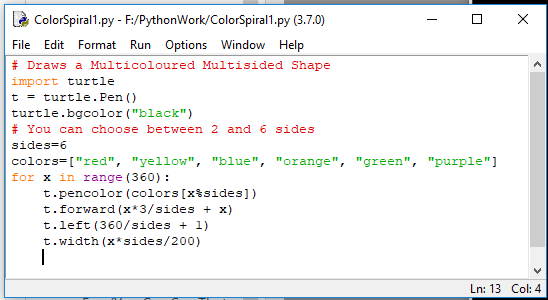
It would be nice to be able to draw each set of radiating spirals in four distinct colours (e.g. red, yellow, blue, green). There are a variety of ways of doing this using various programming constructs.

In the code below, we are going to define a background colour; and then a list of 4 colours for the circles. At each pass through the loop, we will use the modulus function to calculate the remainder of x when divided by 4 (either 0, 1, 2, or 3) which will then determine which pen colour in the list will be used for each spiral. Quite Clever!

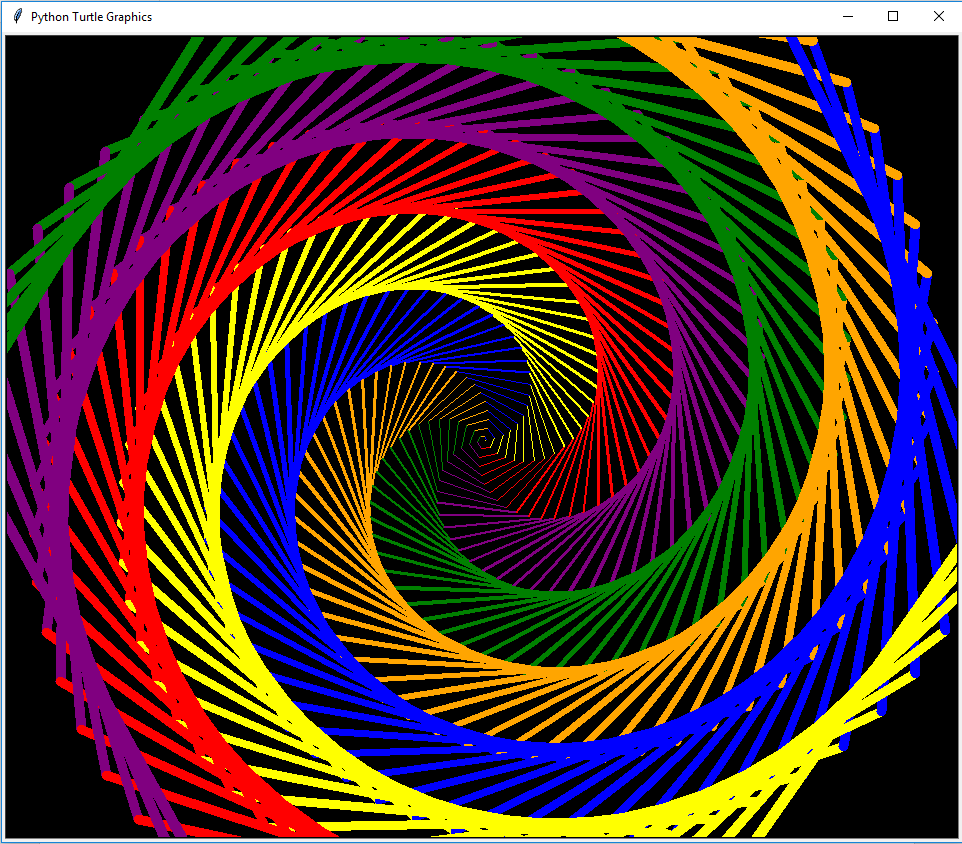




Finally, let’s draw a multi-sided shape with different colours. In the first instance, let’s try it with 6 sides.

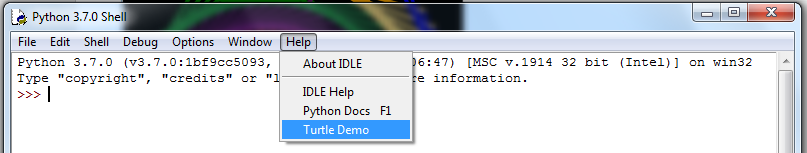


The program can be re-run with sides=5, down to sides=2. If you increase the number of sides to more than 6, then remember to add some more colours to the list.

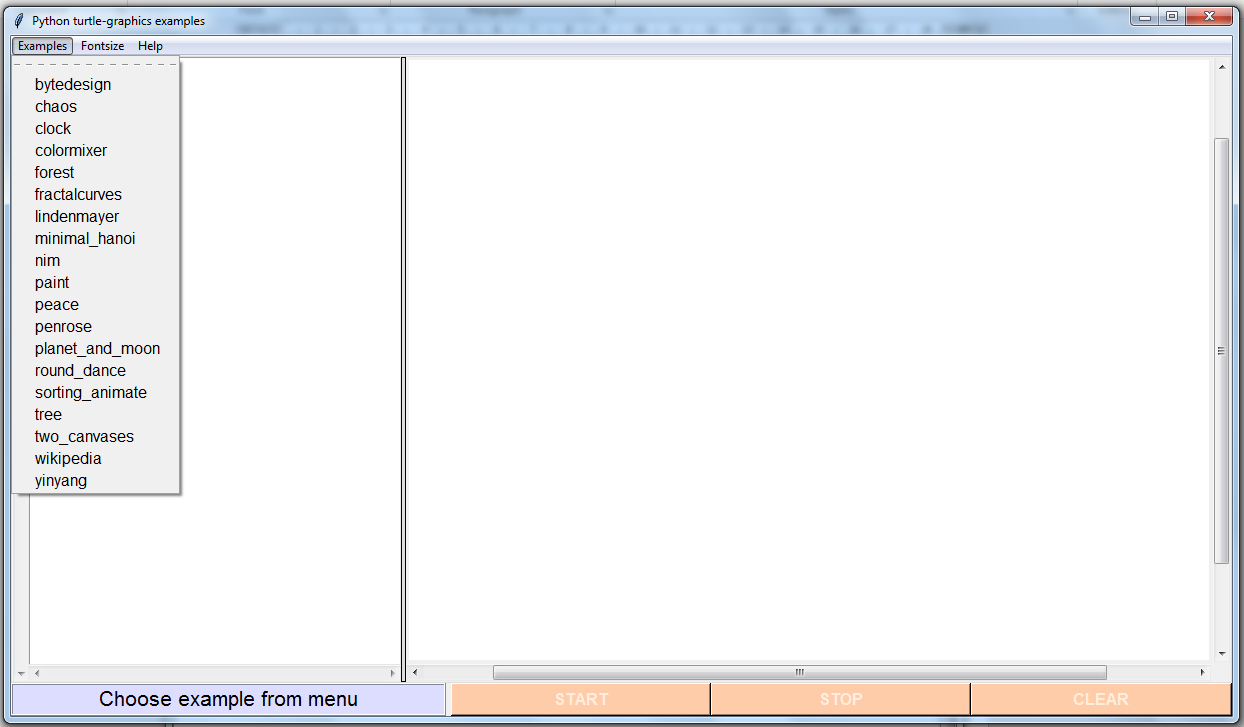


**Task 9: Turtle Graphics Demos**

Finally, you don’t have to do too much this time, but sit back and have a look at the range of possibilities that Turtle Graphics can do. In the Python Shell window, click **Help > Turtle Demo**



This will open up a new window which will allow us to load in some Python scripts which utilise Turtle Graphics. The examples on offer are bundled with different version of Python, so the offering in Python 3.7 might be different to what you see in the list of examples:



Click on each to load the code and hit the START button on the bottom of the window panel. Have a play around with each of these and take a look also at the PYTHON code in the left hand panel of the window. At this stage, the code will still seem a bit daunting, but for the most part many of these demo programs utilise existing Python libraries, so very little effort is required to develop these apps. For example, the Clock demo program utilises the **datetime** library as well as the **turtle** library.

