# Python Foundations

Now that you have the latest version of Python installed, you can begin programming. These are your first steps into the wider world of Python and we're here to help you write your first piece of code, save it and run it in the Python IDLE Shell.

We cover variables, numbers and expressions, user input, conditions and loops and the types of errors you will undoubtedly come across in your time with Python. Let's start and see how to get coding.

# Starting Python for the First Time

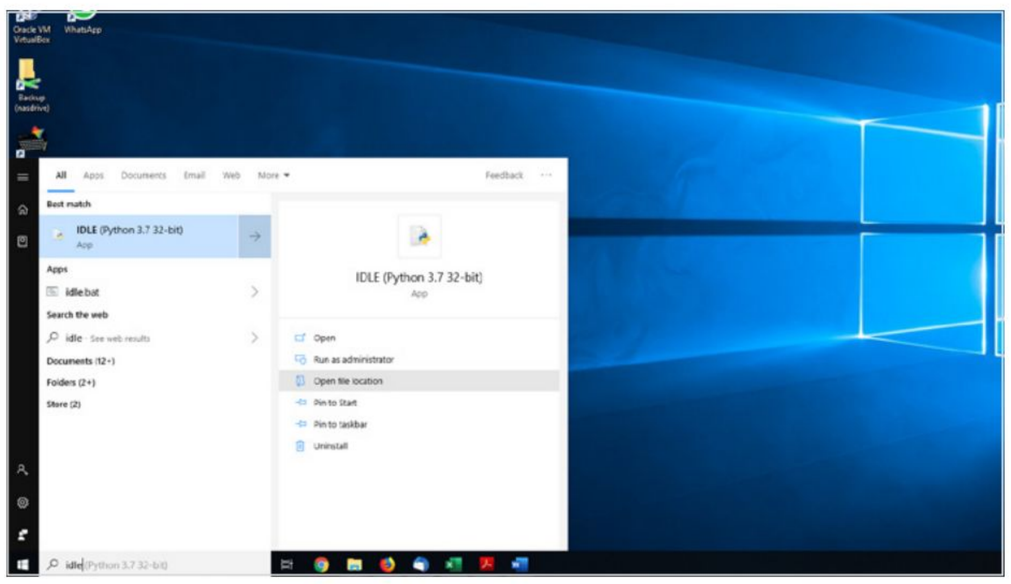
We're using Python 3 under Windows 10 for these following examples. Don't worry if your version of Python is 3.4.2, or something lesser than the current version, as long as you're using Python 3, the code will work.

## STARTING PYTHON

**As when learning anything new, you need to start slow. You can pick up the pace as your experience grows, but for now, let's just get something appearing on the screen. Don't worry, you'll soon be coding like a pro!**

### STEP 1

Click on the Windows Start button, and start typing 'idle'. The result will be the currently installed version of Python, **IDLE (Python 3.7 32-bit)**, for example. You can Pin it to the Start for convenience, otherwise simply click the icon to launch the Python Shell.




### STEP 2

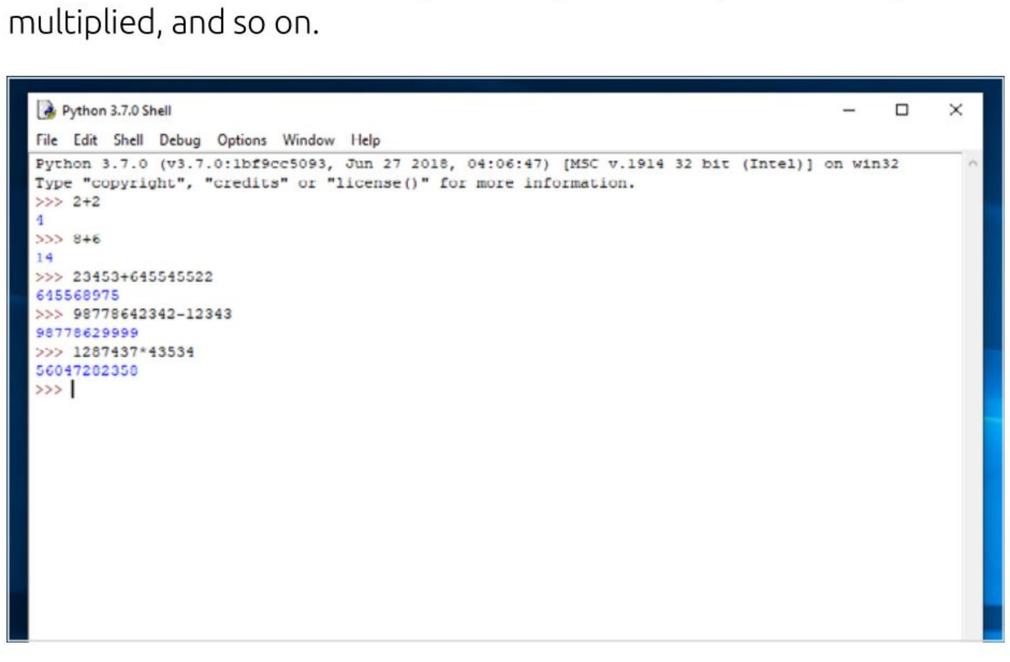
The Shell is where you can enter code and see the responses and output of code you've programmed into Python. This is a kind of sandbox, if you will, where you're able to try out some simple code and processes.


### STEP 3

For example, in the Shell enter: `2+2`
After pressing **Enter**, the next line will display the answer: 4. Basically, Python has taken the 'code' and produced the relevant output.


### STEP 4

The Python Shell acts very much like a calculator, since code is basically a series of mathematical interactions with the system. Integers, which are the infinite sequence of whole numbers, can easily be added, subtracted, multiplied, and so on.



**STEP 5** While that's very interesting, it's not particularly exciting. Instead, try this:

```
print("Hello everyone!")
```

Just enter it into the IDLE as you've done in the previous steps.


**STEP 6** This is a little more like it, since you've just produced your first bit of code. The Print command is fairly self-explanatory, it prints things. Python 3 requires the parentheses as well as quotes in order to output content to the screen, in this case the 'Hello everyone!' bit.

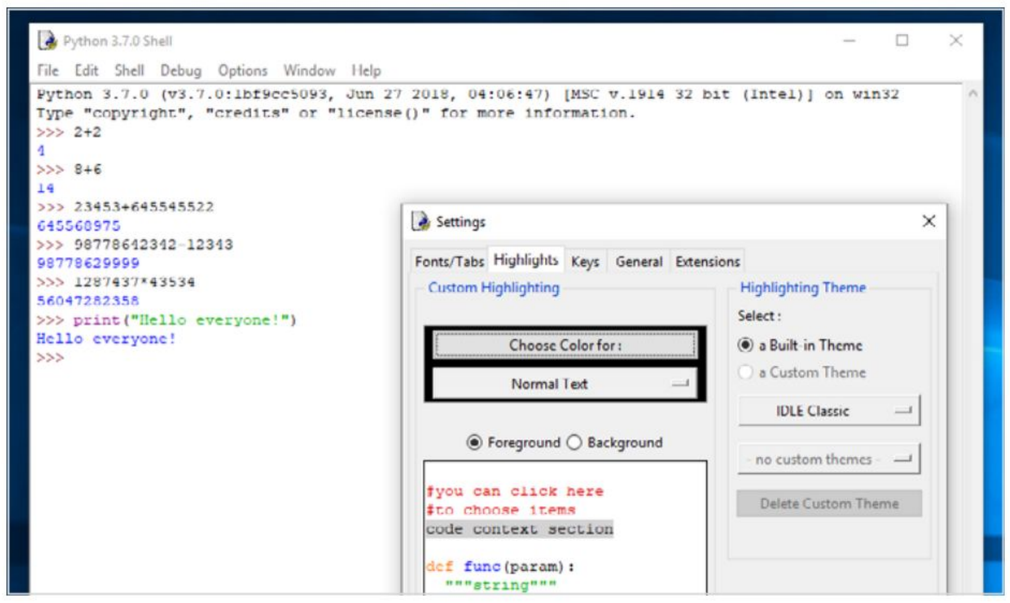

**STEP 7** You'll have noticed the colour coding within the Python IDLE. The colours represent different elements of Python code. They are:

Black – Data and Variables
Green – Strings
Purple – Functions
Orange – Commands
Blue – User Functions
Dark Red – Comments
Light Red – Error Messages

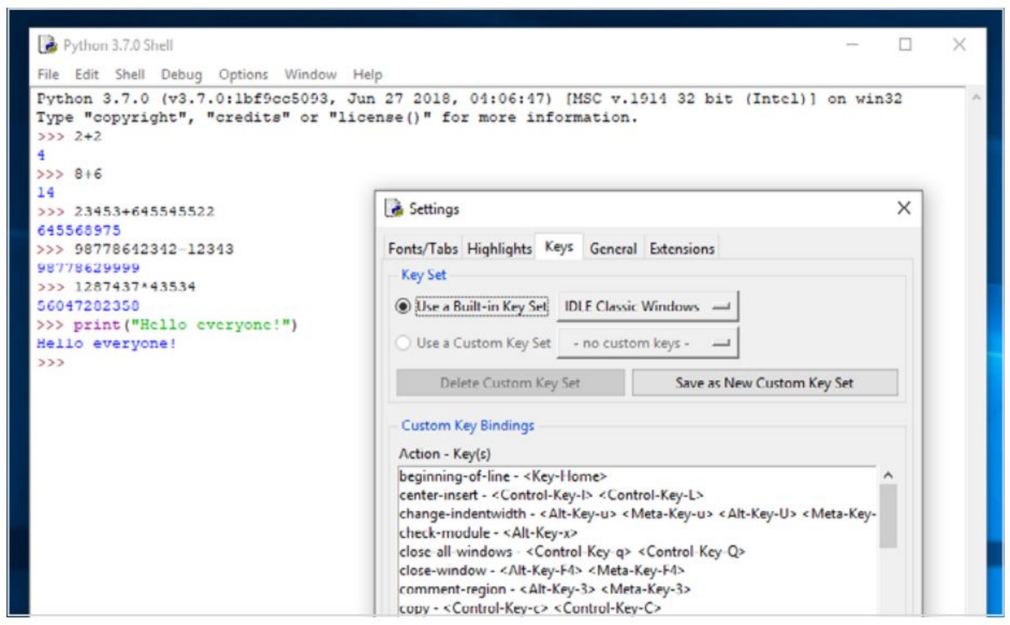
**IDLE Colour Coding**

| Colour | Use for | Examples |
|---|---|---|
| Black | Data & variables | `23.6 area` |
| Green | Strings | `"Hello World"` |
| Purple | Functions | `len() print()` |
| Orange | Commands | `if for else` |
| Blue | User functions | `get_area()` |
| Dark red | Comments | `#Remember VAT` |
| Light red | Error messages | `SyntaxError:` |

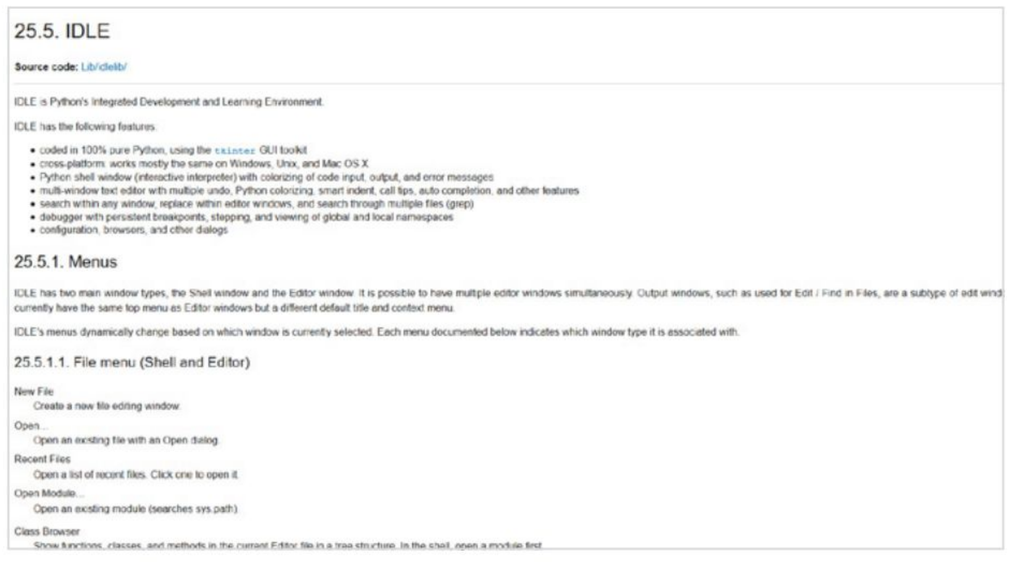
**STEP 8** The Python IDLE is a configurable environment. If you don't like the way the colours are represented, then you can always change them via **Options > Configure IDLE**, and clicking on the **Highlighting** tab. However, we don't recommend that as you won't be seeing the same as our screenshots.




**STEP 9** As with most programs available, regardless of the operating system, there are numerous shortcut keys. We don't have room for them all here, but within the Options > Configure IDLE and under the **Keys** tab, you'll see a list of the current bindings.




**STEP 10** The Python IDLE is a power interface, and one that's actually been written in Python using one of the available GUI toolkits. If you want to know the many ins and outs for the Shell, we recommend you take a few moments to view **https://docs.python.org/3/library/idle.html**, which details many of the IDLE's features.


25.5. IDLE

**Source code:** Lib/idlelib/

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

- coded in 100% pure Python, using the `tkinter` GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- search within any window, replace within editor windows, and search through multiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- configuration, browsers, and other dialogs

25.5.1. Menus

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. Output windows, such as used for Edit / Find in Files, are a subtype of edit windows currently have the same top menu as Editor windows but a different default title and context menu.

IDLE's menus dynamically change based on which window is currently selected. Each menu documented below indicates which window type it is associated with.

25.5.1.1. File menu (Shell and Editor)

New File
Create a new file editing window.

Open...
Open an existing file with an Open dialog.

Recent Files
Open a list of recent files. Click one to open it.

Open Module...
Open an existing module (searches sys.path).

Class Browser
Show functions, classes, and methods in the current Editor file in a tree structure. In the shell, open a module first.

# Your First Code

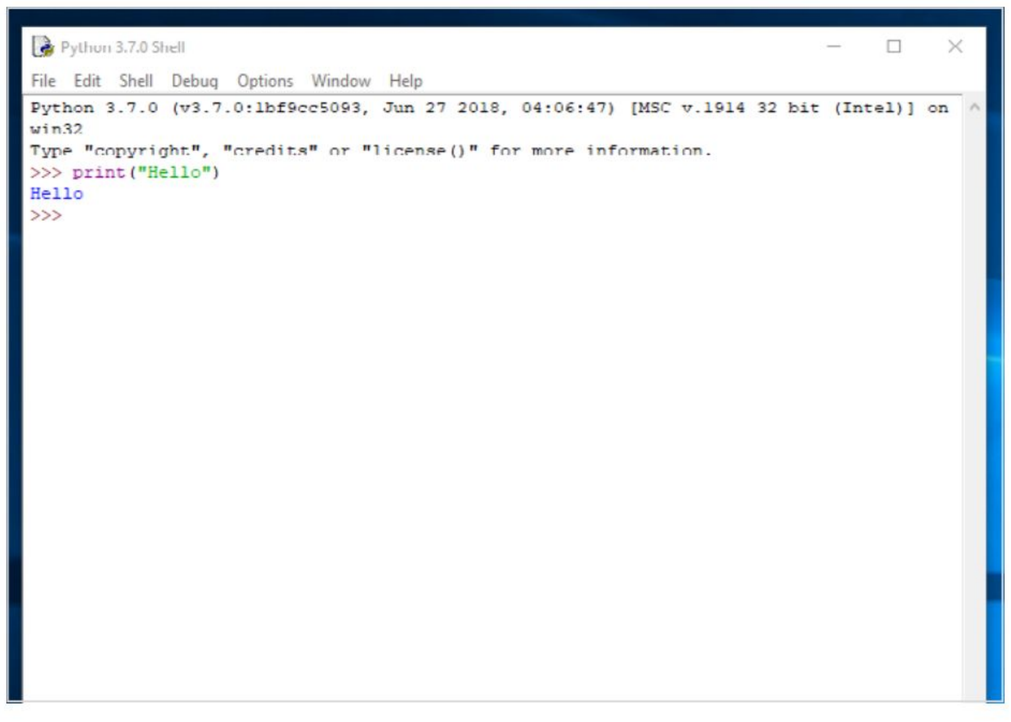
Essentially, you've already written your first piece of code with the print("Hello everyone!") function from the previous tutorial. However, let's expand that and look at entering your code and playing around with some other Python examples.

## PLAYING WITH PYTHON

**As with most languages, computer or human, it's all about remembering and applying the right words to the right situation. You're not born knowing these words, so you need to learn them.**

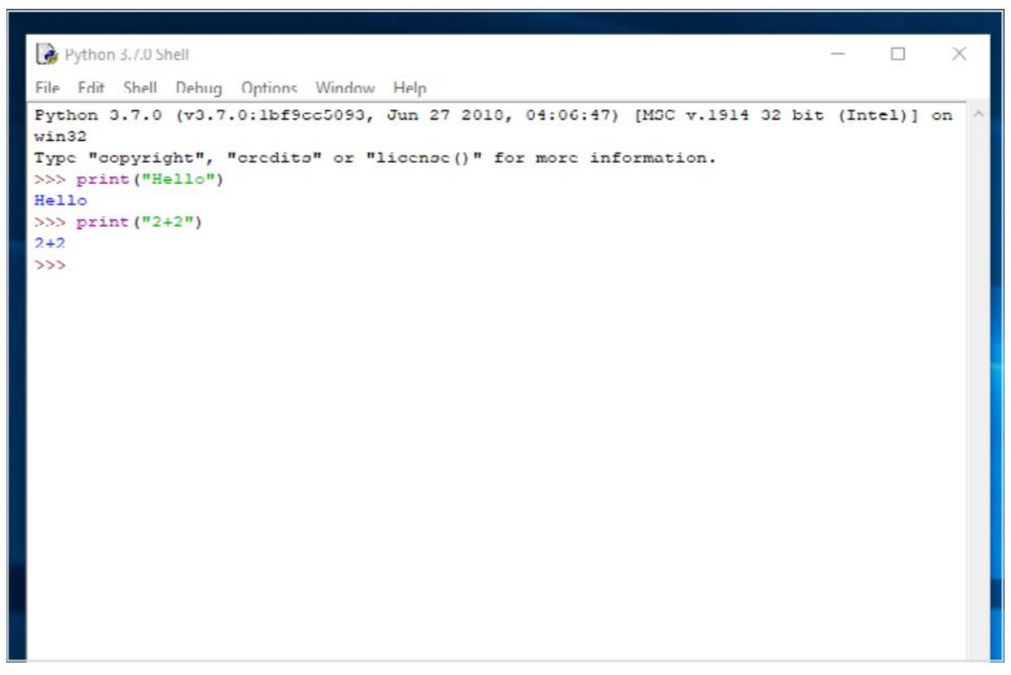
**STEP 1** If you've closed Python 3 IDLE, re-open it as you did in the previous page. In the Shell, enter the familiar following:

```
print("Hello")
```



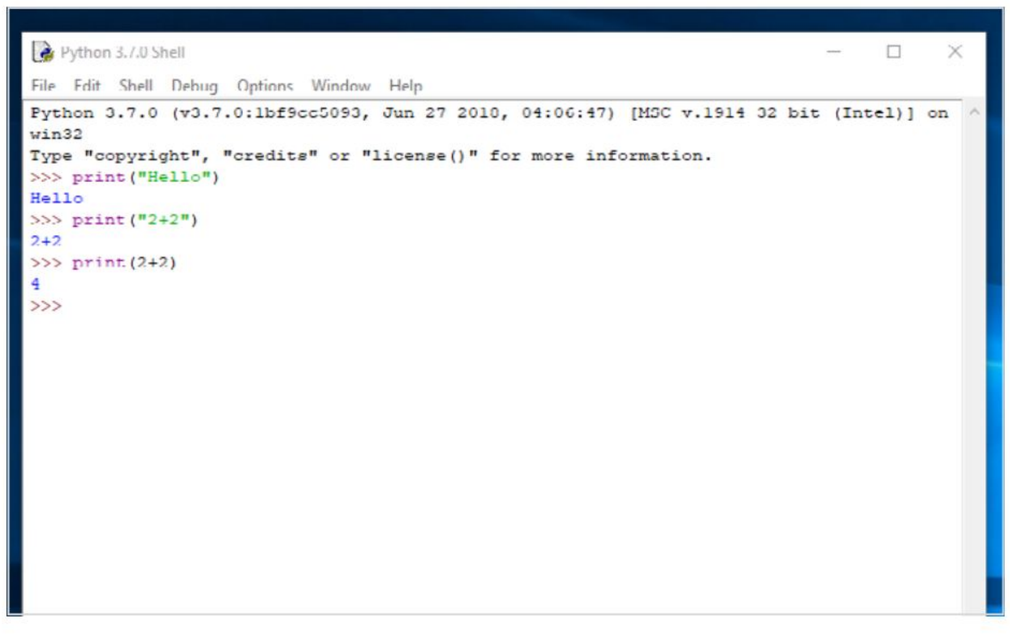

**STEP 2** As predicted, the word Hello appears in the Shell as blue text indicating output from a string. It's fairly straightforward, and doesn't require too much explanation. Now try:

```
print("2+2")
```




**STEP 3** You'll notice that instead of the number 4, the output is the 2+2 you asked to be printed to the screen. The quotation marks are defining what's being outputted to the IDLE Shell, to print the total of 2+2 you'll need to remove the quotes:

```
print(2+2)
```



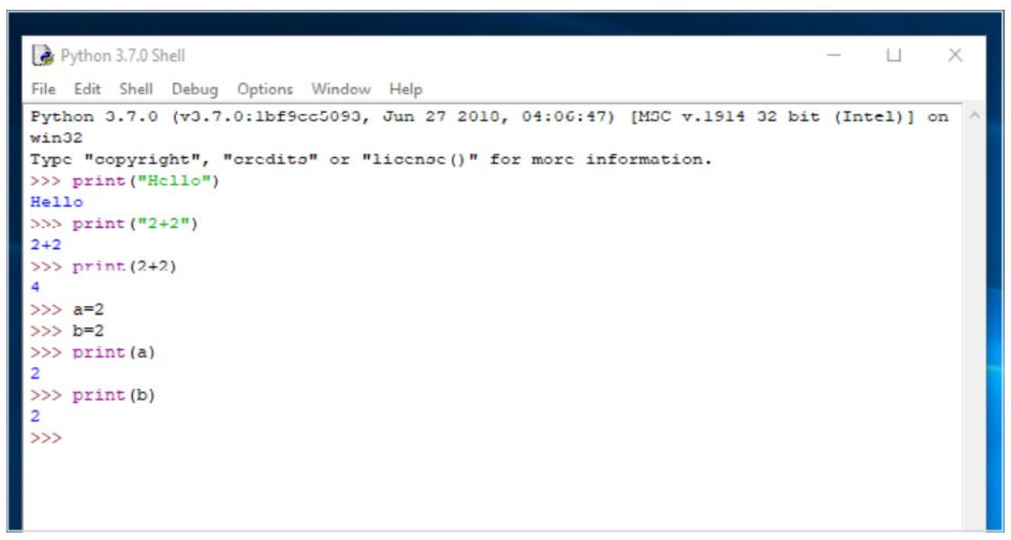

**STEP 4** You can continue as such, printing 2+2, 464+2343 and so on to the Shell. An easier way is to use a variable, which is something we will cover in more depth later. For now, enter:

```
a=2
b=2
```

## STEP 5

What you have done here is assign the letters a and b two values: 2 and 2. These are now variables, which can be called upon by Python to output, add, subtract, divide and so on, for as long as their numbers stay the same. Try this:

```
print(a)
print(b)
```

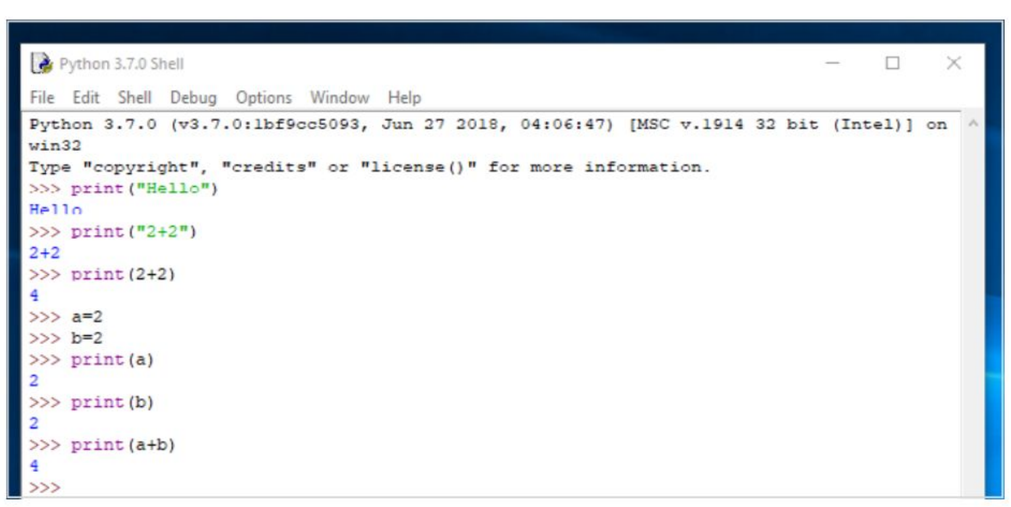



## STEP 6

The output of the last step displays the current values of a and b individually, as essentially you've asked them to be printed separately. If you want to add them up, you can use the following:

```
print(a+b)
```

This code takes the value of  both a and b, adds them together, and outputs the result.




## STEP 7

You can play around with different kinds of variables together with the Print function. For example, we could assign variables for someone's name:

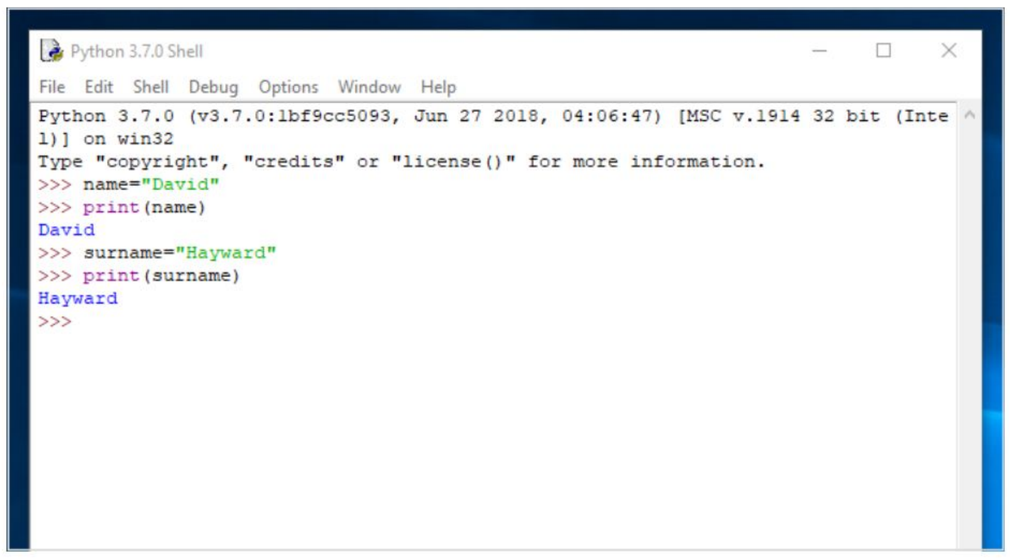
```
name="David"
print(name)
```


## STEP 8

Now let's add a surname:

```
surname="Hayward"
print(surname)
```

We now have two variables containing both a first name and a surname, and we can print them independently.

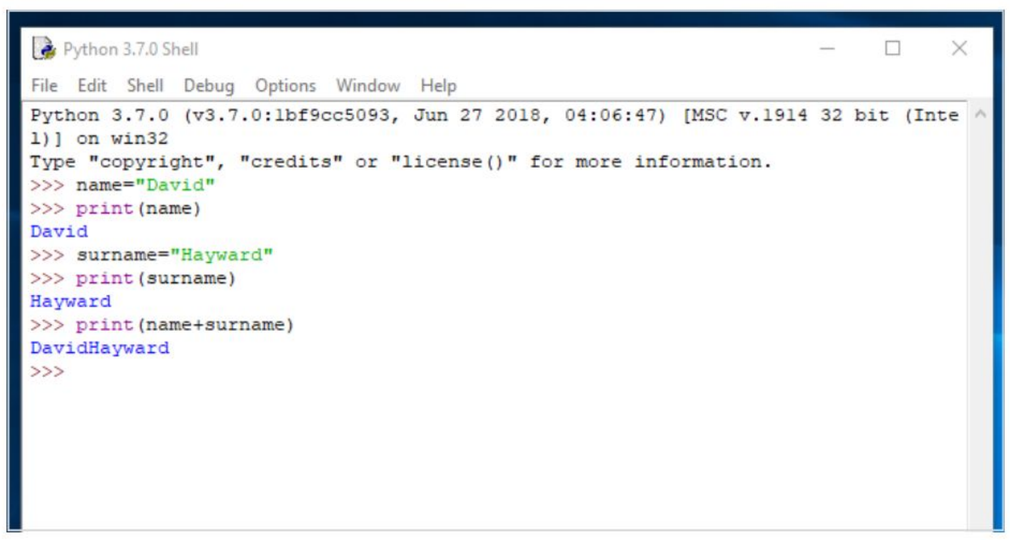



## STEP 9

If we were to apply the same routine as before, using the + symbol, the name wouldn't appear correctly in the output in the Shell. Try it:

```
print(name+surname)
```

We need a space between the two, defining them as two separate values and not something you mathematically play around with.




## STEP 10

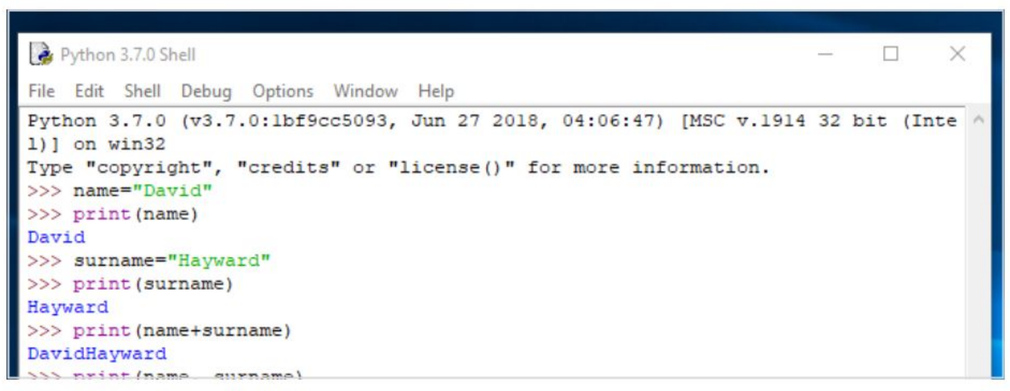
In Python 3 we can separate the two variables with a space by using a comma:

```
print(name, surname)
```

Alternatively, you can add the space yourself:

```
print(name+" "+surname)
```

As you can see, the use of the comma is much neater. Congratulations, you've just taken your first steps into the wide world of Python.



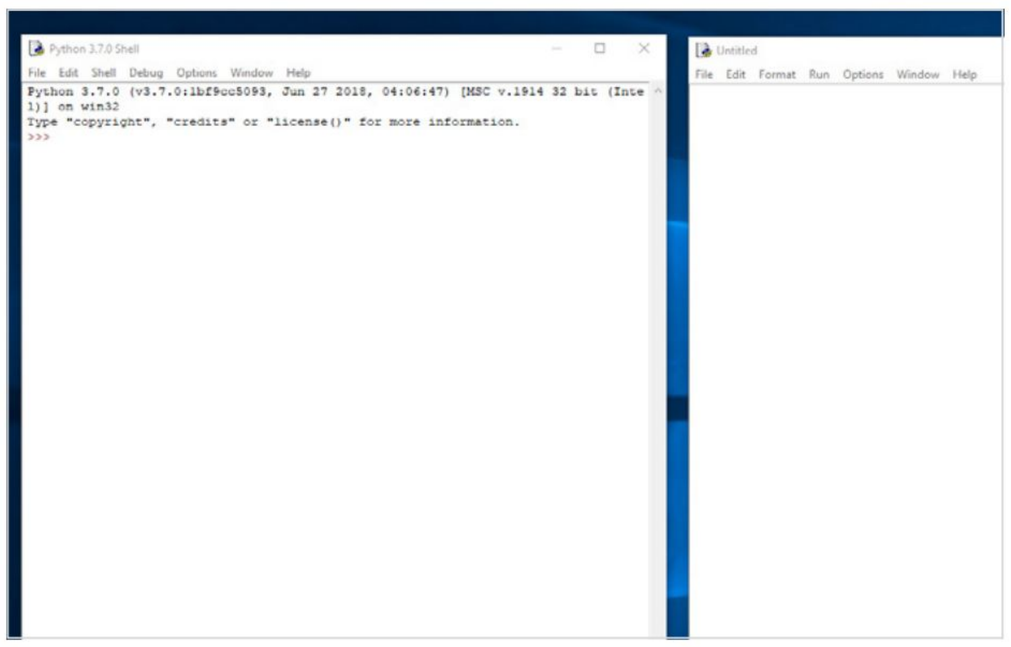
# Saving and Executing Your Code

While working in the IDLE Shell is perfectly fine for snippets of code, it's not designed for entering longer program listings. In this section, we'll introduce you to the IDLE Editor, where most of our code will be entered from now on.

## EDITING CODE

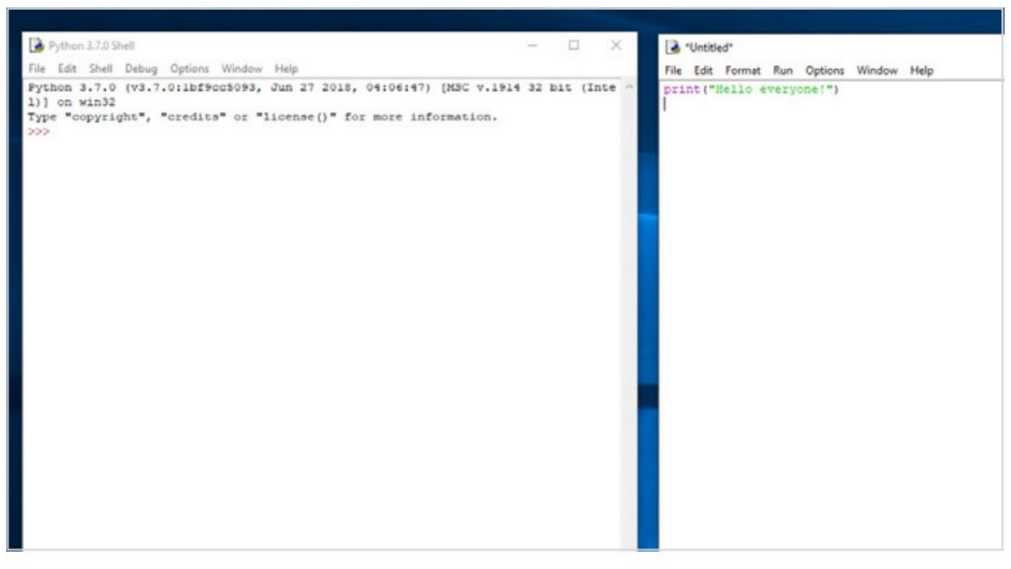
**You will eventually reach a point where you have to move on from inputting single lines of code into the Shell. Instead, the IDLE Editor will allow you to save and execute your Python code.**

**STEP 1** First, open the Python IDLE Shell. When it's up, 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 you need to create your future programs.

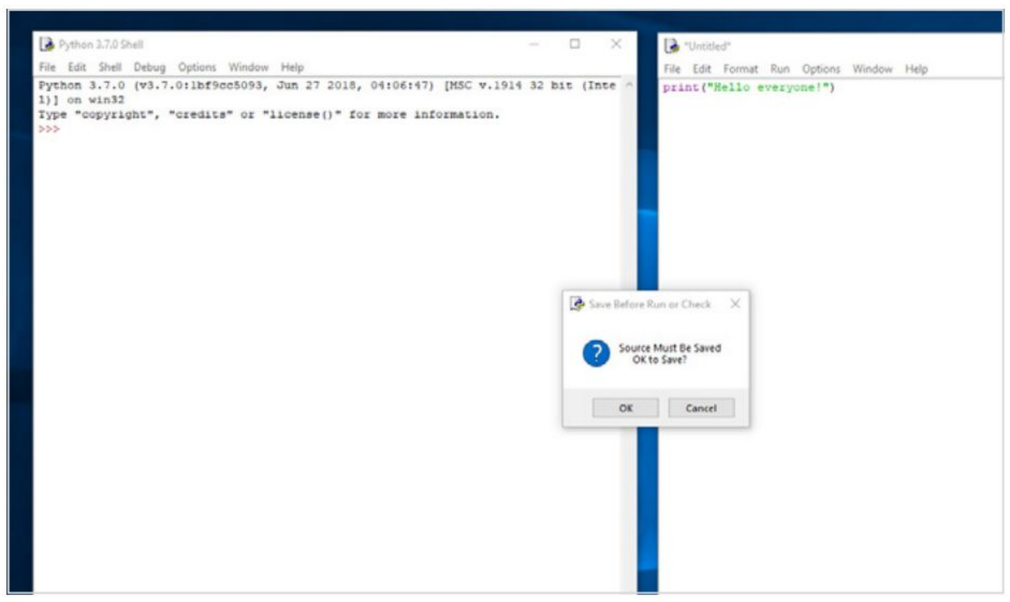



**STEP 2** The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

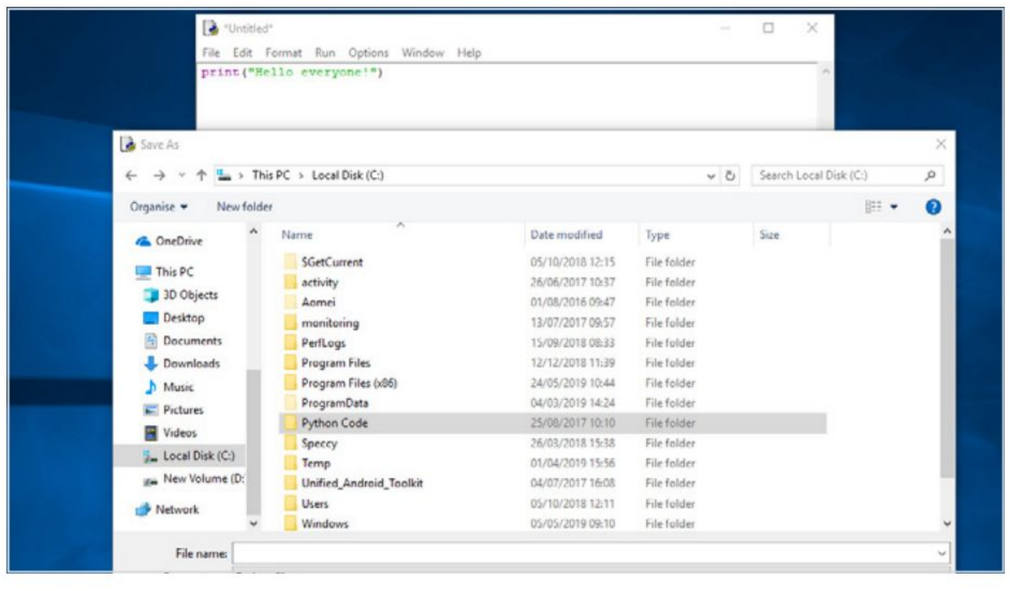
```
print("Hello everyone!")
```



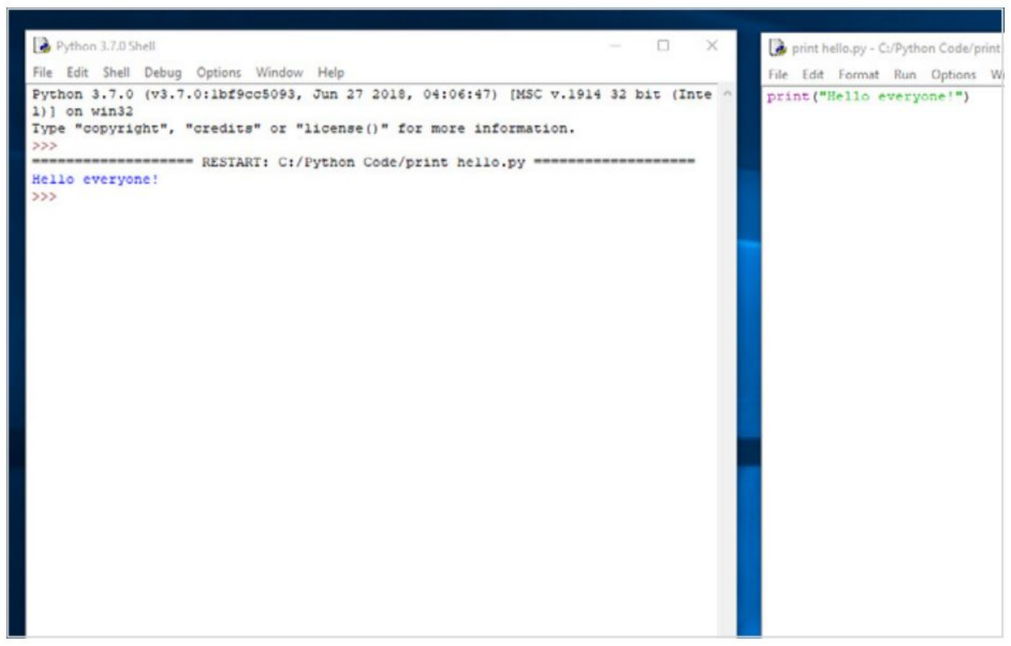

**STEP 3** As you can see the same colour coding is in place in the IDLE Editor as it is in the Shell, enabling you to better understand what's going on with your code. To execute the code, however, you need to first save it. Press **F5** and you'll have a Save…Check box open.




**STEP 4** Click on the **OK** button in the Save box, and select a destination where you'll save all your Python code. The destination can be a dedicated folder called Python, or you can just dump it wherever you like. Remember to keep a tidy file system, though, it'll help you out in the future.

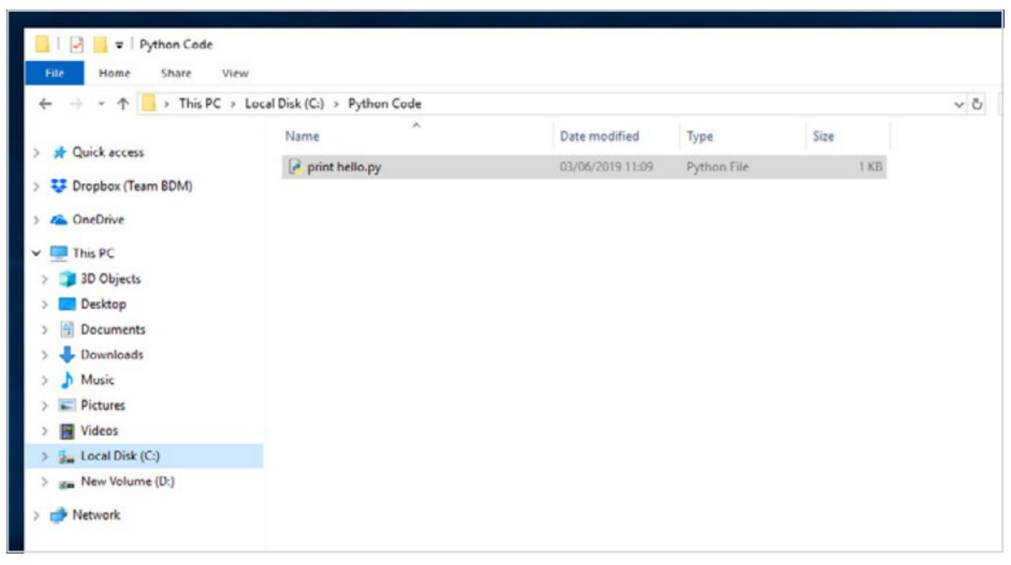


**STEP 5** Enter a name for your code, 'print hello' for example, and click on the Save button. As soon as the Python code is saved, it's executed and the output will be detailed in the IDLE Shell; In this case, the words 'Hello everyone!'.




**STEP 6** This is how the vast majority of your Python code will be conducted. Enter it into the Editor, hit F5, save the code, and look at the output in the Shell. Sometimes things will differ, depending on whether you've requested a separate window, but essentially that's the process and, unless otherwise stated, this is the method we will use.

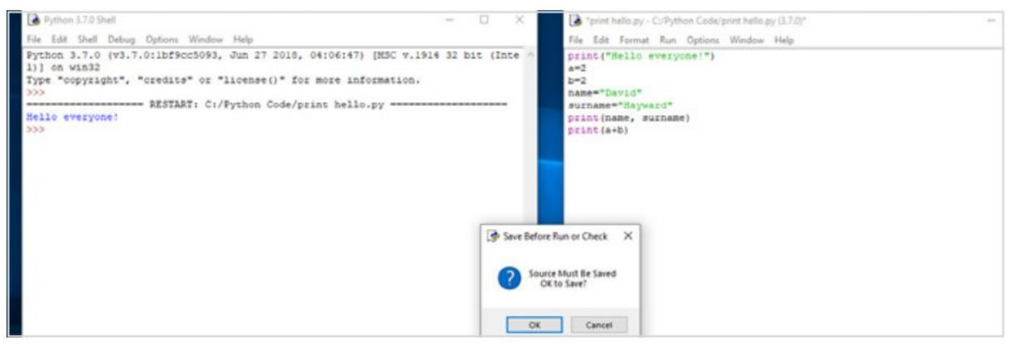

**STEP 7** If you open the file location of the saved Python code, you'll notice that it ends in a .py extension. This is the default Python filename, any code you create will be whatever.py, and any code downloaded from the many Internet Python resource sites will be .py. Just ensure that the code is written for Python 3.



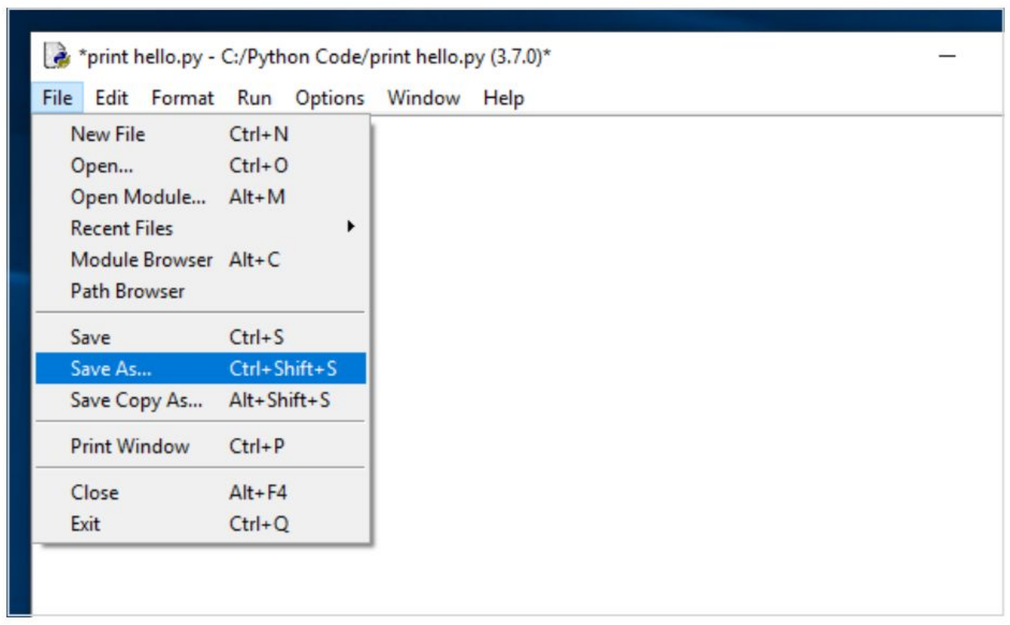

**STEP 8** Let's extend the code and enter a few examples from the previous tutorial:

```
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print (a+b)
```

If you press **F5** now, you'll be asked to save the file again, as it's been modified from before.



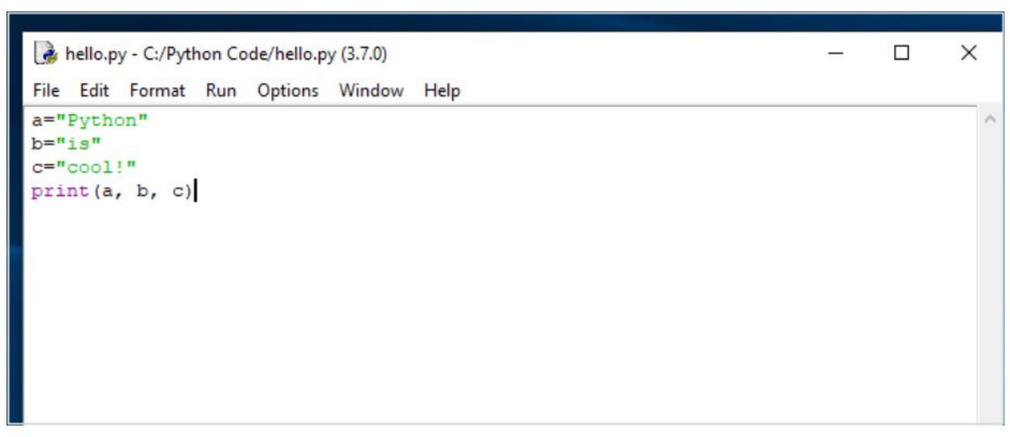

**STEP 9** If you click the **OK** button the file will be overwritten with the new code entries, and executed; with the output in the Shell. It's not a problem with just these few lines, but if you were to edit a larger file overwriting can become an issue. Instead, use **File > Save As** from within the Editor to create a backup.




**STEP 10** Now create a new file. Close the Editor, and open a new instance (File > New File from the Shell). Enter the following, and save it as **hello.py**:

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

We will use this code in the next tutorial.



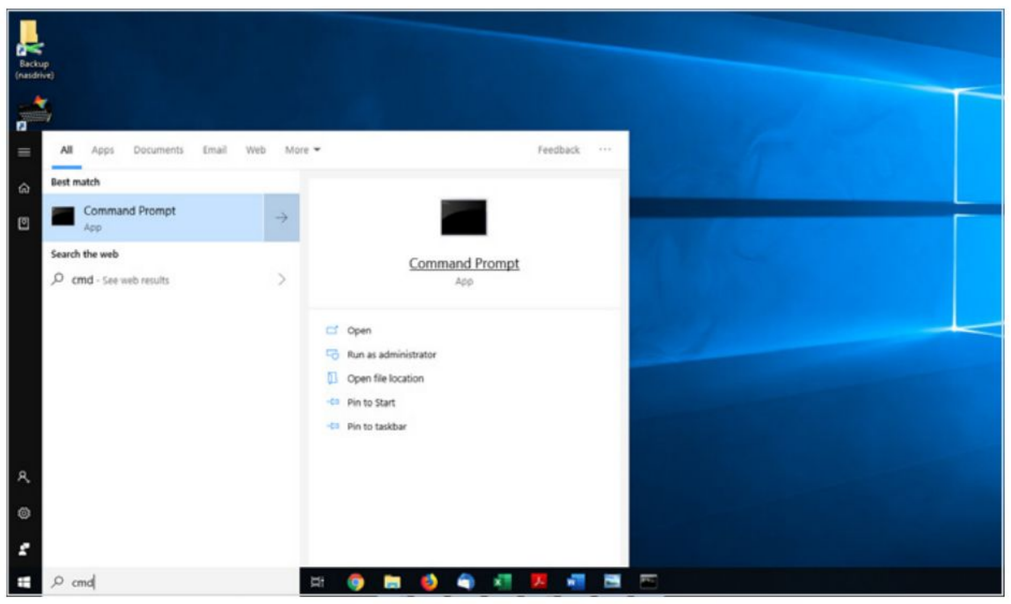
# Executing Code from the Command Line

While we're going to be working from the GUI IDLE, it's worth taking a moment to look at Python's command line handling. Sometimes, depending on the code you write, executing via the command line is a better solution over the IDLE.

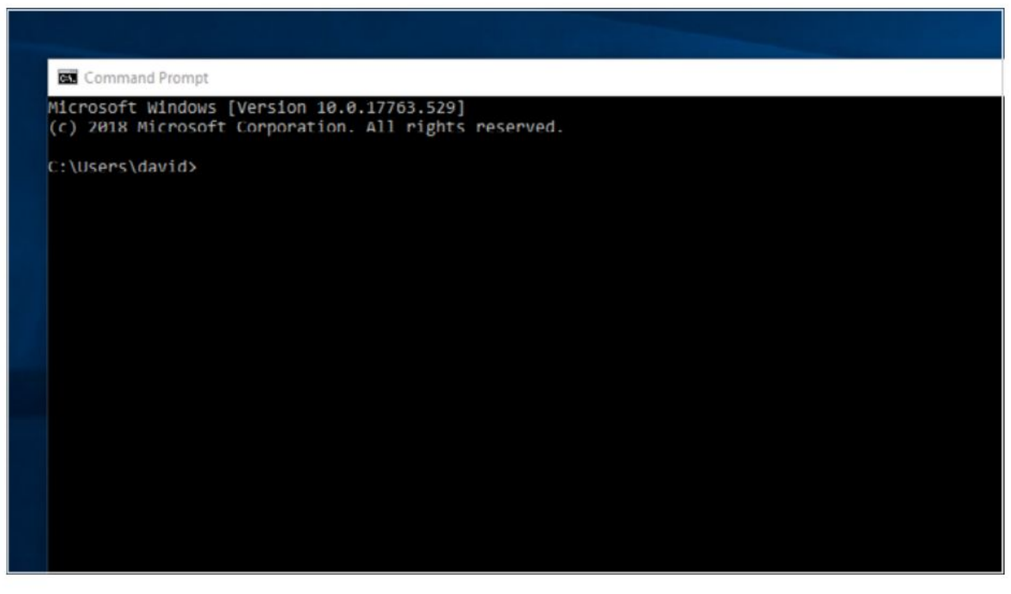
## COMMAND THE CODE

**Using the code we created in the previous tutorial, the one we named hello.py, let's see how we can run code that was made in the GUI at the command line level.**

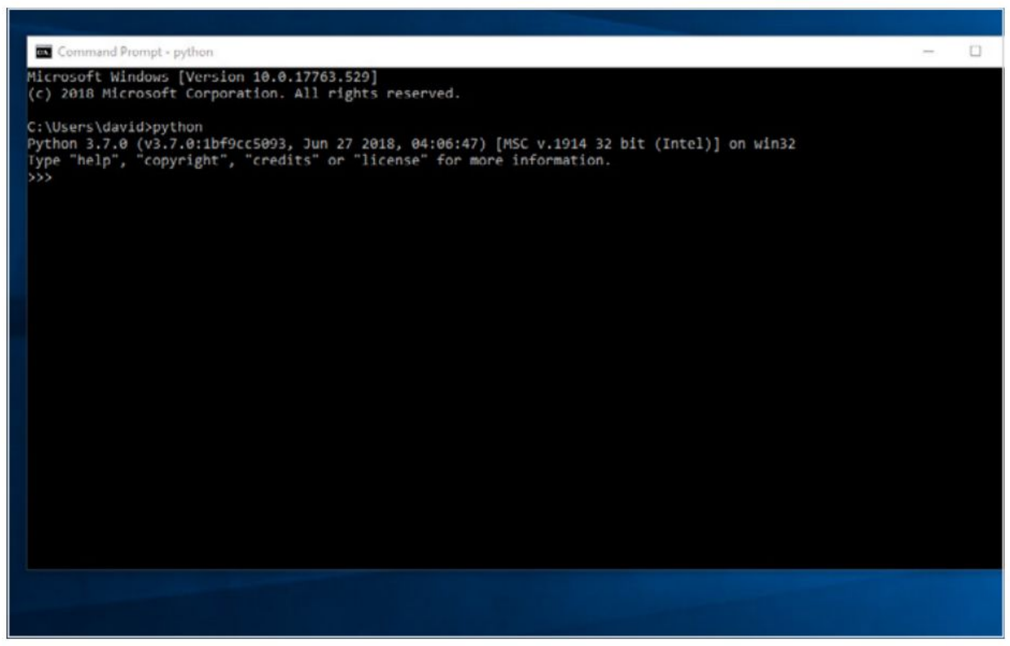
**STEP 1** When you first installed Python, the installation routine automatically included all the necessary components to allow the execution of code outside of the GUI IDLE; in other words, the command line. To begin with, click on the Windows Start Button, and type: `cmd`.




**STEP 2** As you did when launching the Python IDLE, click on the returned result from the search, the Command Prompt App. This will launch a new window, with a black background and white text. This is the command line, also called a Terminal in macOS, Linux, and Raspberry Pi operating systems.



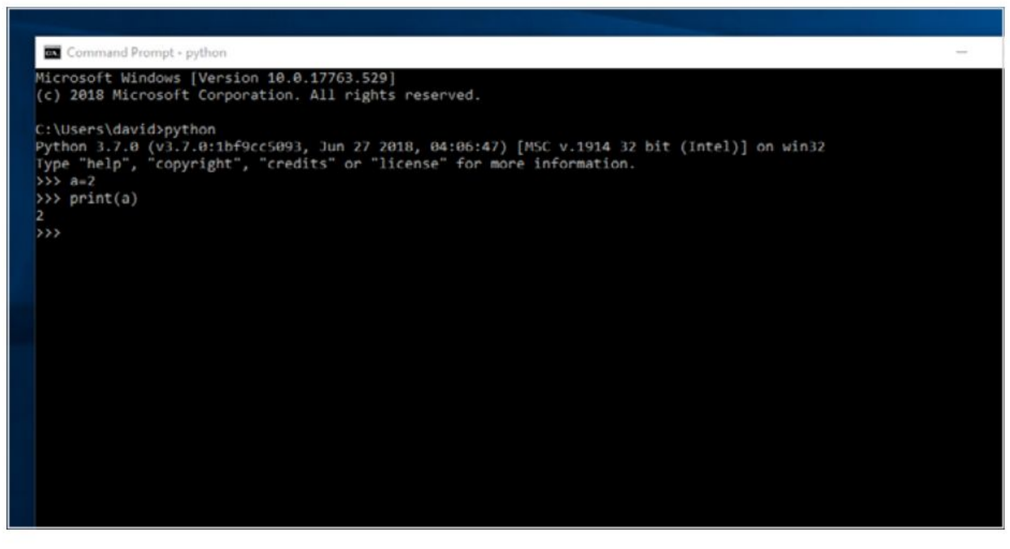

**STEP 3** Now you're at the command line, we can start Python using the command `python` and pressing the Enter key. This will put you into the command line version of the Shell, with the familiar, three right-facing arrows as the cursor (>>>).



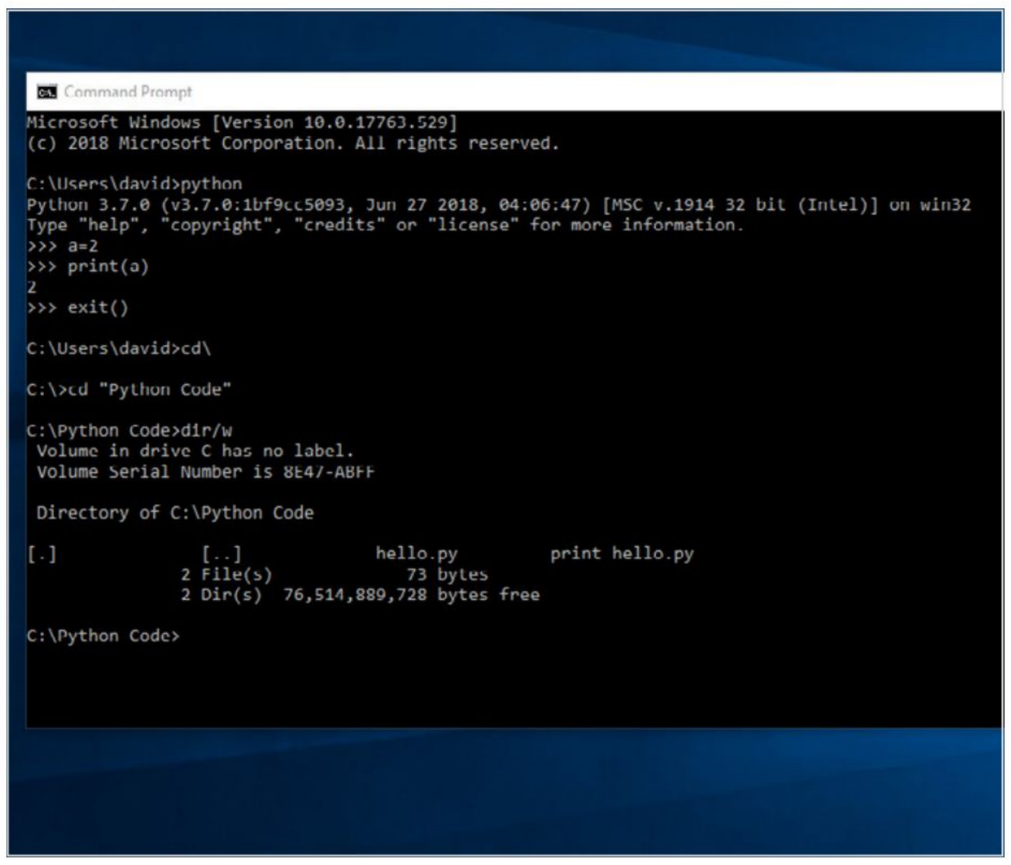

**STEP 4** From here you're able to enter the code you've looked at previously, such as:

```
a=2
print(a)
```

As you can see, it works exactly the same.



**STEP 5** Now enter `exit()` to leave the command line Python session, and return back to the command prompt. Enter the folder where you saved the code from the previous tutorial, and list the available files within; you should see the **hello.py** file.

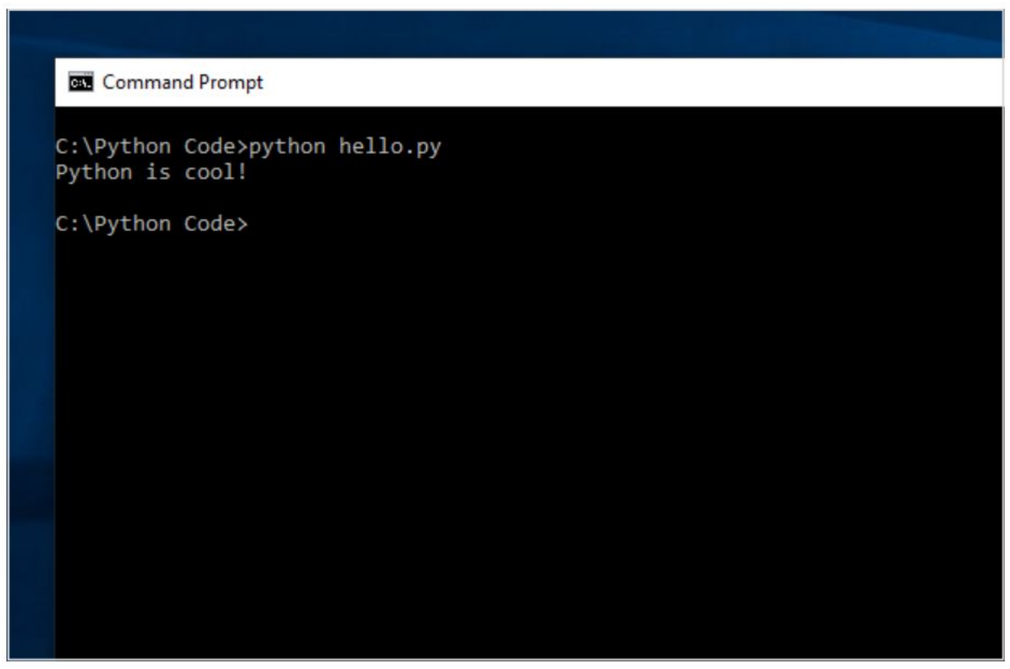



**STEP 6** From within the same folder as the code you're going to run, enter the following into the command line:

```
python hello.py
```

This will execute the code we created, which to remind you is:

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```




## DIFFERENT VERSIONS OF PYTHON

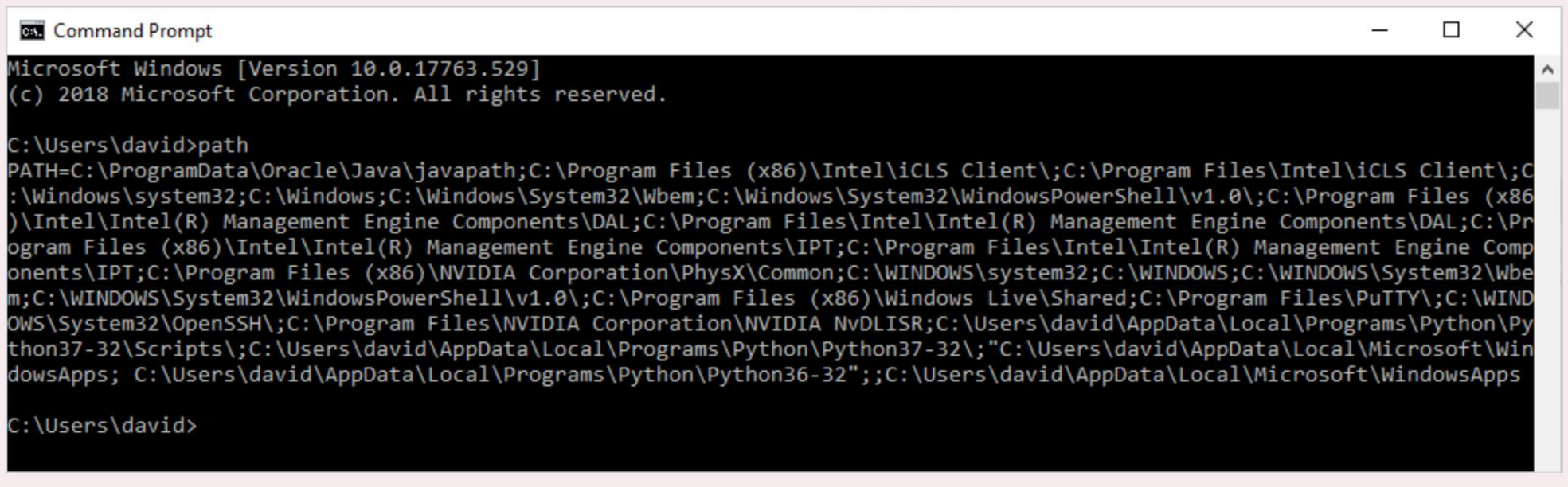
If you've previously used Python 3 on a Mac or Linux, and subsequently the Raspberry Pi, you may be a little confused as to why the Windows version of Python uses the command line: **python**, instead of **python3**.

The reason behind this is that UNIX-like systems, such as macOS and Linux, already have Python libraries pre-installed. These older libraries are present because some of the macOS and Linux system utilities rely on Python 2, and therefore installing a newer version of Python, and thus altering the executable name, could have dire consequences to the system.

As a result, developers decided that the best approach for macOS and Linux systems would be to leave the command line **'python'** as exclusive Python 2 use, and newer versions of user-installed Python would be **'python3'**.

This isn't an issue with Windows, as it doesn't use any Python libraries other than the ones installed by the user themselves when actually installing Python. When a Windows user installs Python, the installation wizard will auto-include the command line instance to the core Windows PATH variable, which you can view by entering: `path` into the command line. This points to the python.exe file required to execute Python code from the command line.

We don't recommend you install both Python 2 and Python 3 within Windows 10; naturally, you can if you want, but realistically, although Python 2 still has a foothold in the coding world, Python 3 is the newest version. However, if you do, then you will need to rename one of the Python versions names; as they will be installed in different folders and both use python.exe as the command line executable. It's a little long-winded, so unless there's a dire need to have both versions of Python installed, it's best to stick to Python 3.



# Numbers and Expressions

We've seen some basic mathematical expressions with Python, simple addition and the like. Now let's expand on that, and see just how powerful Python is as a calculator. You can work within the IDLE Shell, or in the Editor, whichever you like.

## IT'S ALL MATHS, MAN

**You can get some really impressive results from the mathematical powers of Python, as maths is the driving force behind the code with most, if not all, programming languages.**

**STEP 1** Open up the GUI version of Python 3, as mentioned you can use either the Shell or the Editor. For the time being, we're going to use the Shell. If you've opted to use a third-party text editor, note that you need to get to the IDLE Shell for this part of the tutorial.

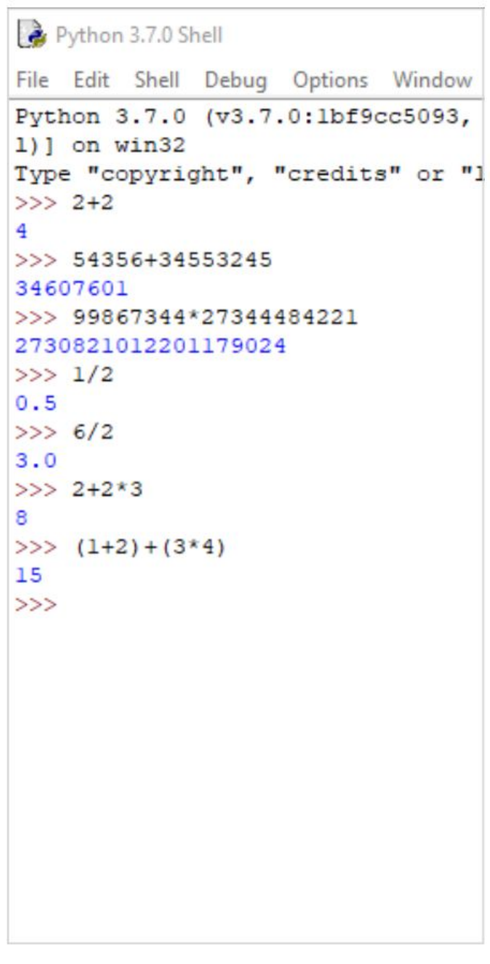

**STEP 2** In the Shell enter the following:

```
2+2
54356+34553245
99867344*27344484221
```

As you can see, Python can handle some quite large numbers.


**STEP 3** You can use all the customary Mathematical operations: divide, multiply, brackets and so on. Practise with a few, for example:

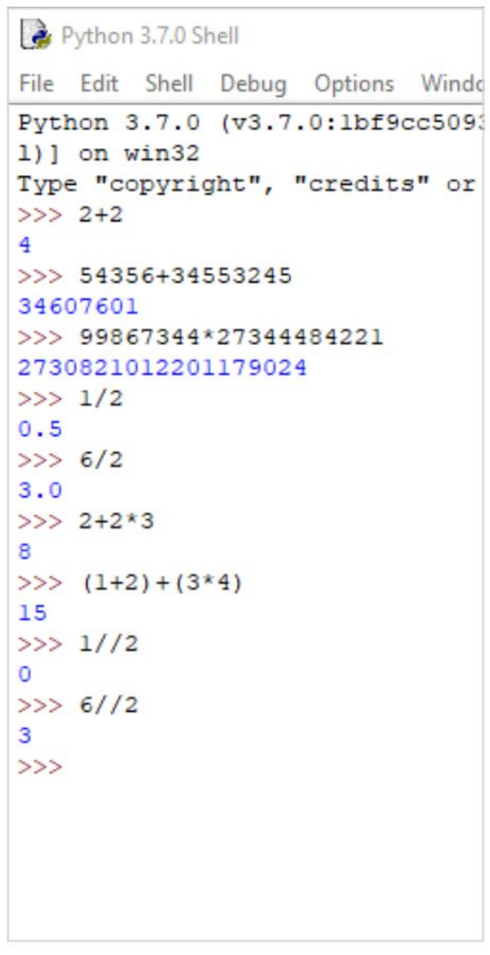
```
1/2
6/2
2+2*3
(1+2)+(3*4)
```




**STEP 4** As you've no doubt noticed, division produces a decimal number. In Python, these are called floats, or floating point arithmetic. If however, you need an integer as opposed to a decimal answer, then you can use a double slash:

```
1//2
6//2
```

and so on.



## STEP 5

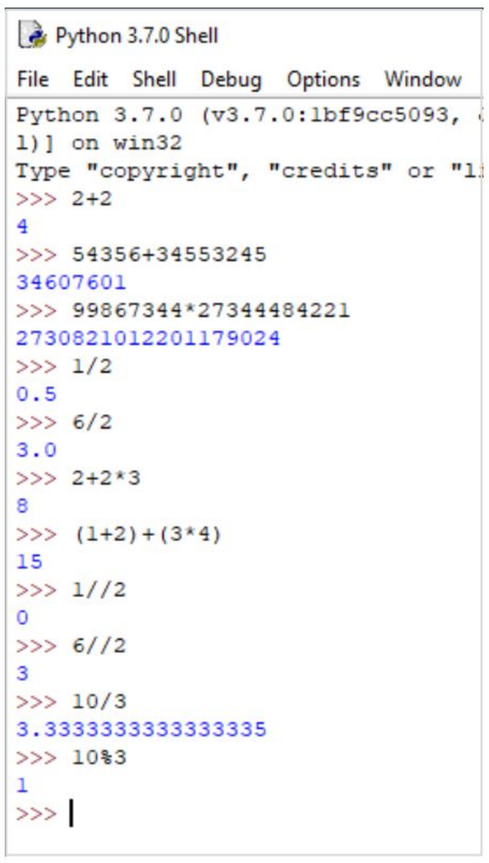
You can also use an operation to see the remainder left over from division. For example:

```
10/3
```

will display 3.333333333, which is, of course, 3.3-recurring. If you now enter:

```
10%3
```

This will display 1, which is the remainder left over from dividing 10 by 3.

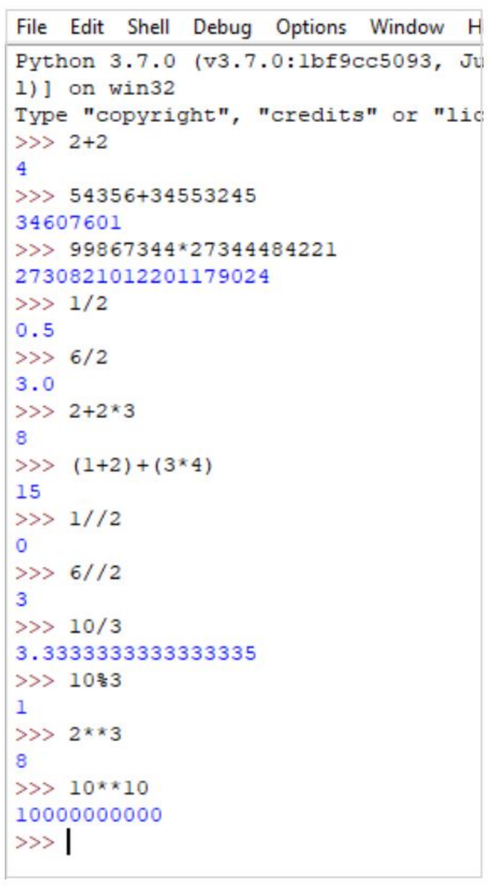



## STEP 6

Next up we have the power operator, or exponentiation if you want to be technical. To work out the power of something you can use a double multiplication symbol, or double-star on the keyboard:

```
2**3
10**10
```

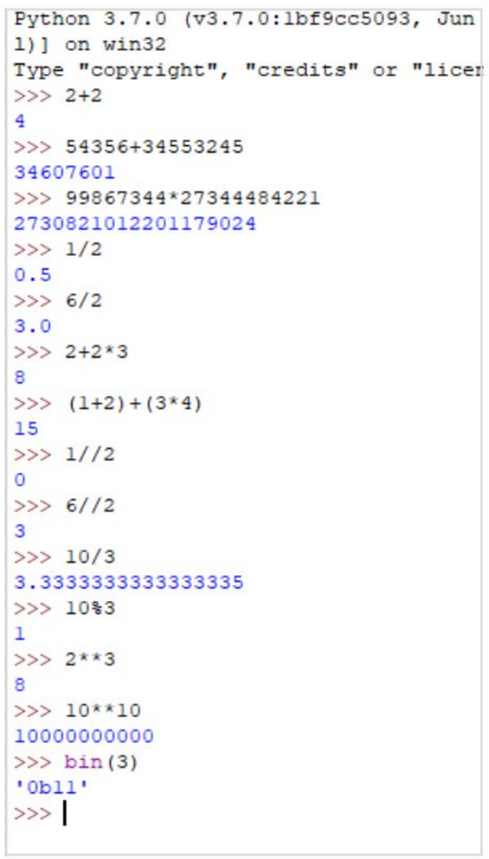
Essentially, it's 2x2x2, but we're sure you already know the basics behind maths operators. This is how you would work it out in Python.




## STEP 7

Numbers and expressions don't stop there. Python has numerous built-in functions to work out sets of numbers, absolute values, complex numbers, and a host of Mathematical expressions and Pythogarian tongue-twisters. For example, to convert a number to binary, use:

```
bin(3)
```

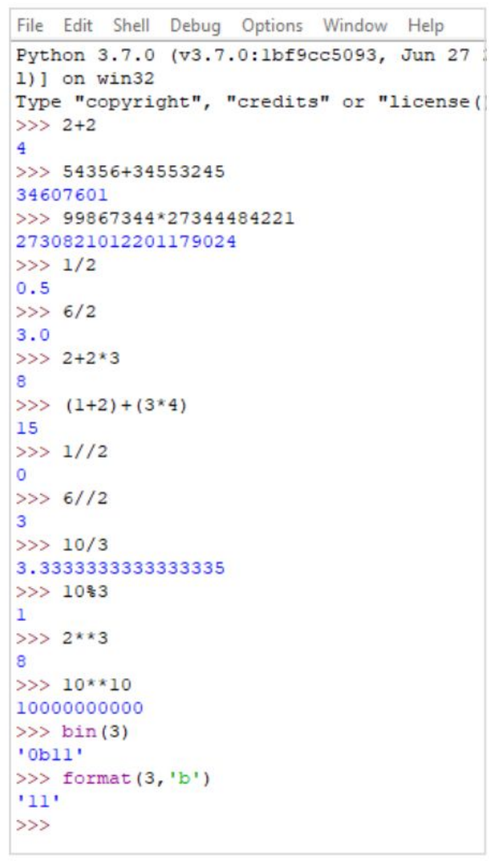



## STEP 8

This will be displayed as '0b11', converting the integer into binary, and adding the prefix 0b to the front. If you want to remove the 0b prefix, then you can use:

```
format(3, 'b')
```

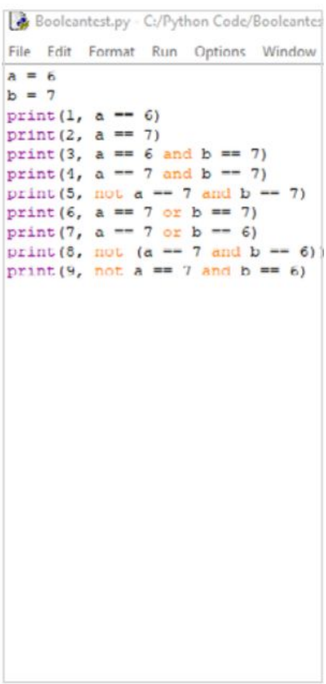
The Format command converts a value, the number 3, to a formatted representation as controlled by the format specification, the 'b' part.




## STEP 9

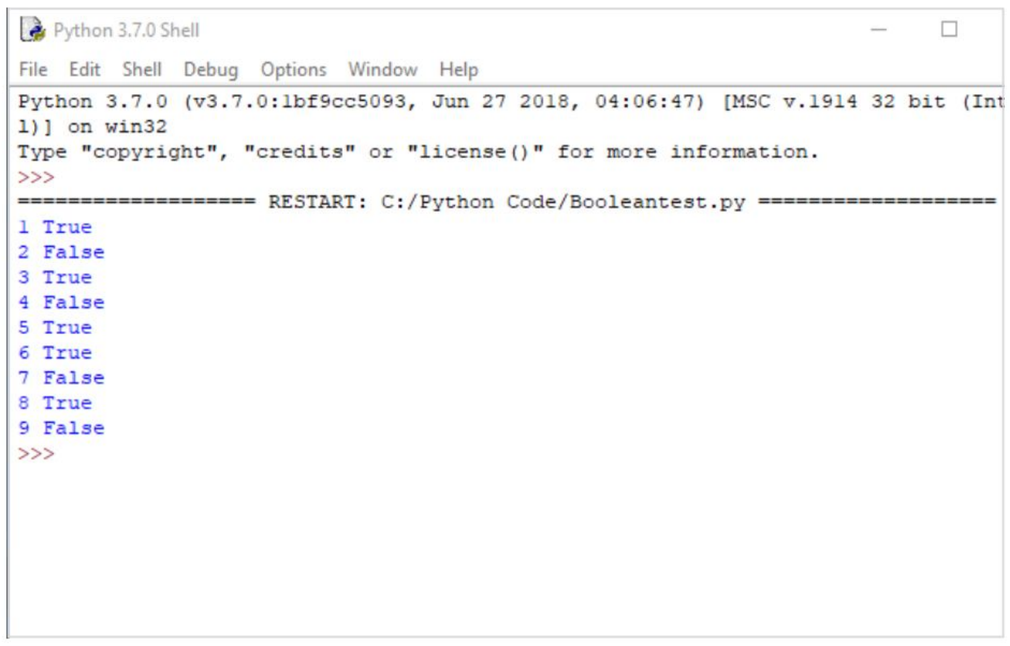
A Boolean Expression is a logical statement that will either be true or false. We can use these to compare data, and test to see if it's equal to, less than, or greater than. Try this in a New File:

```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```




## STEP 10

Execute the code from Step 9, and you'll see a series of True or False statements depending on the result of the two defining values: 6 and 7. It's an extension of what we've looked at, and an important part of programming.



# Using Comments

When writing your code, the flow, what each variable does, how the overall program will operate and so on, is all inside your head. Another programmer could follow the code line by line, but when the code starts to hit thousands of lines, things get a little difficult to read.

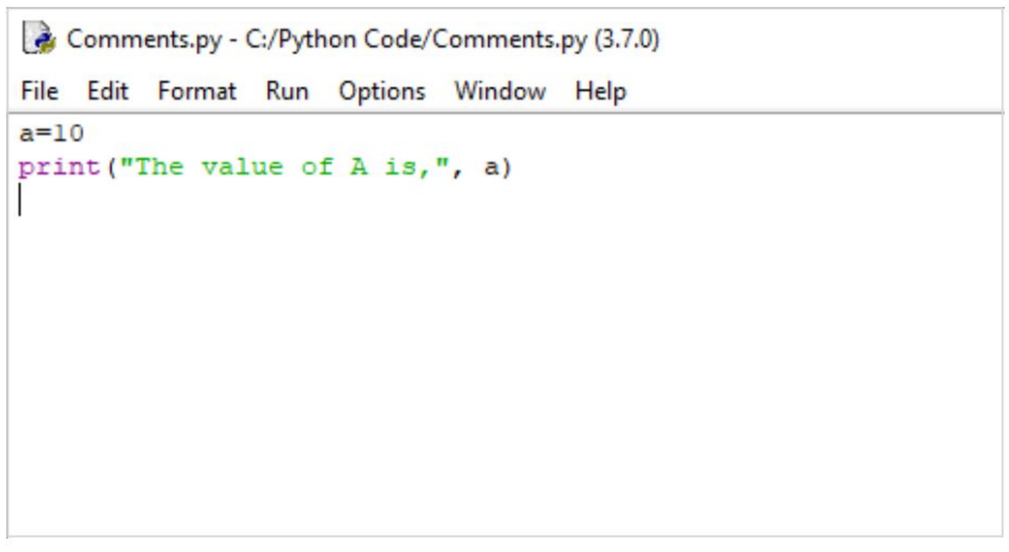
## #COMMENTS!

**A method used by most programmers for keeping their code readable, is by commenting on certain sections. For example, if a variable is used, the programmer comments on what it's supposed to do. It's just good practise.**

**STEP 1** We'll start by creating a new instance of the IDLE Editor (**File > New File**), and then create a simple variable and print command:

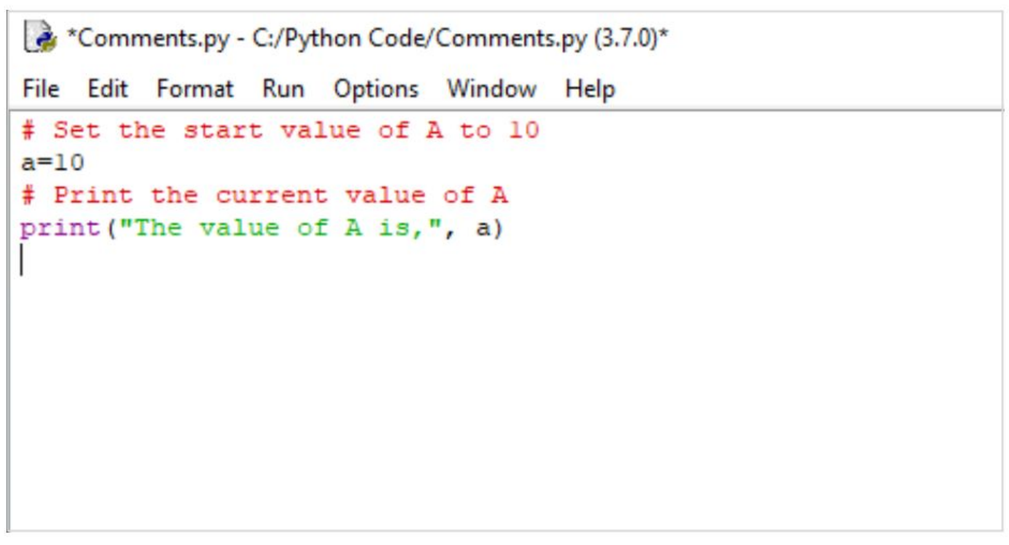
```
a=10
print("The value of A is,", a)
```

Save the file, and execute the code.




**STEP 2** Running the code will return the line: **The value of A is, 10** into the IDLE Shell window – which is what we expected. Now let's add some of the types of comments you'd normally see within code:

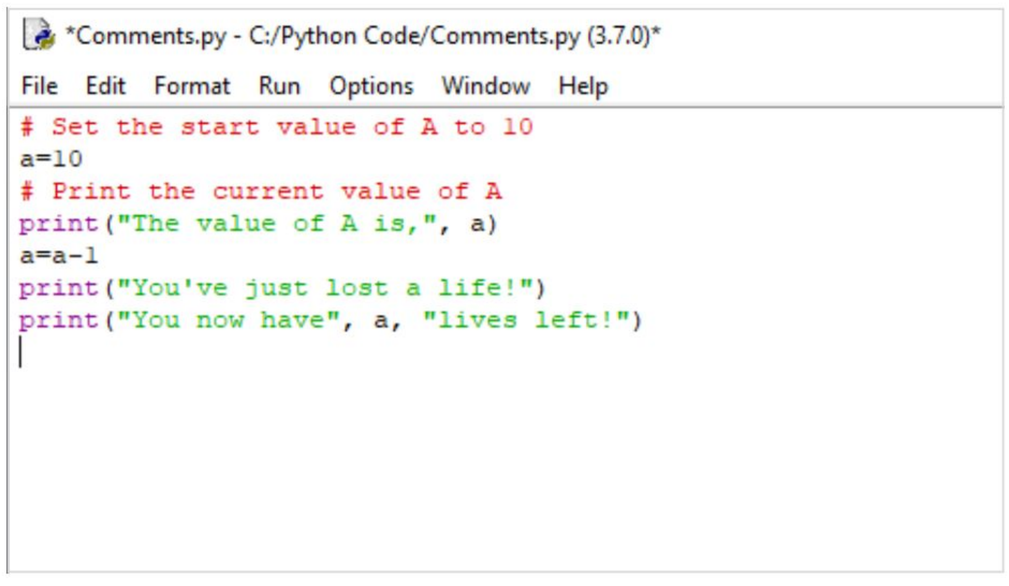
```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
```



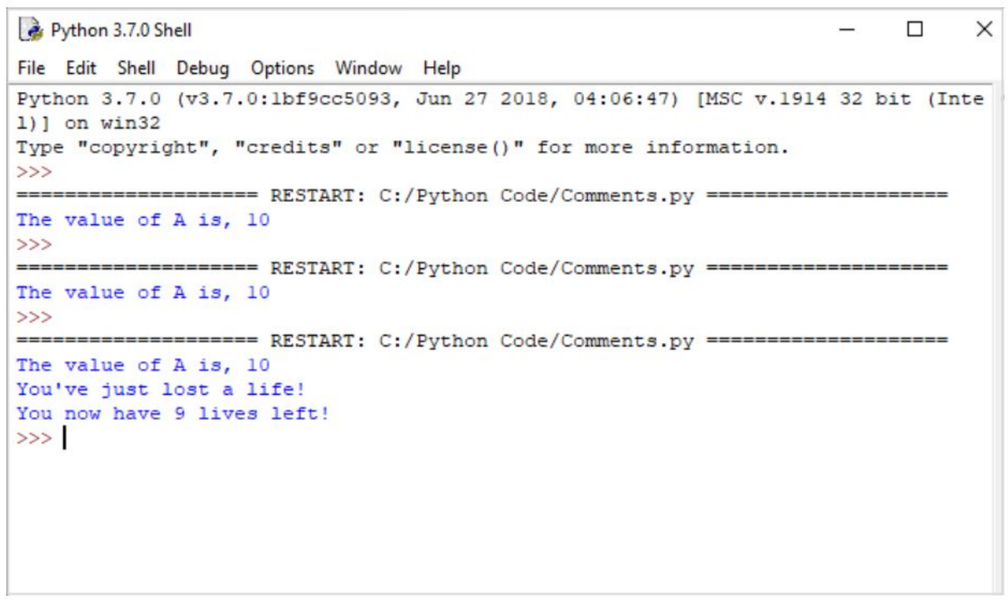

**STEP 3** Re-save the code and execute it. You'll see that the output in the IDLE Shell is still the same as before, despite the extra lines being added. Simply put, the hash symbol (#) denotes a line of text the programmer can insert, to inform them and others of what's going on, without the user being aware.


**STEP 4** Let's assume that the variable A we've created is the number of lives in a game. Every time the player dies, the value decreases by 1. The programmer could insert a routine along the lines of:

```
a=a-1
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

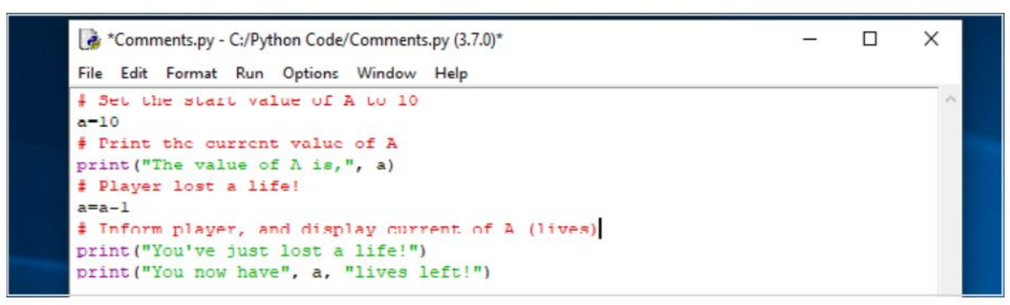


**STEP 5** While we know that the variable A denotes number of lives and the player has just lost one, a casual viewer, or someone checking the code, may not know. Imagine for a moment that the code is twenty thousand lines long, instead of just our seven. You can see how handy comments are.



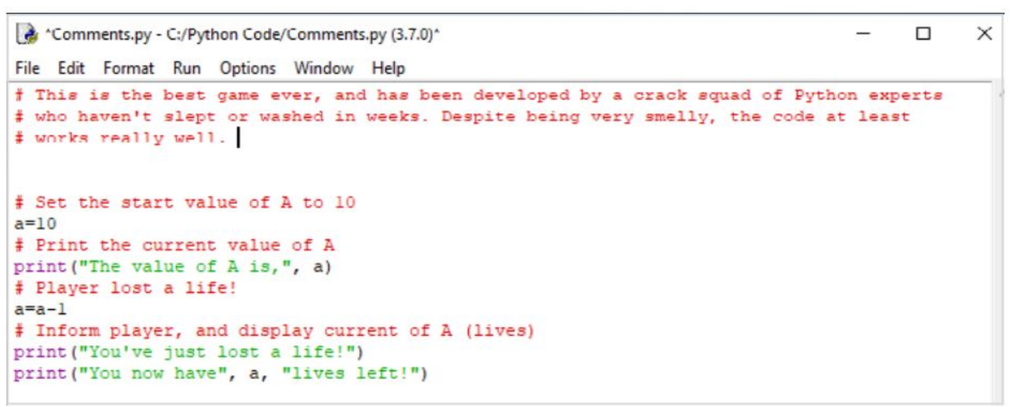

**STEP 6** Essentially, the new code together with comments could look like:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```



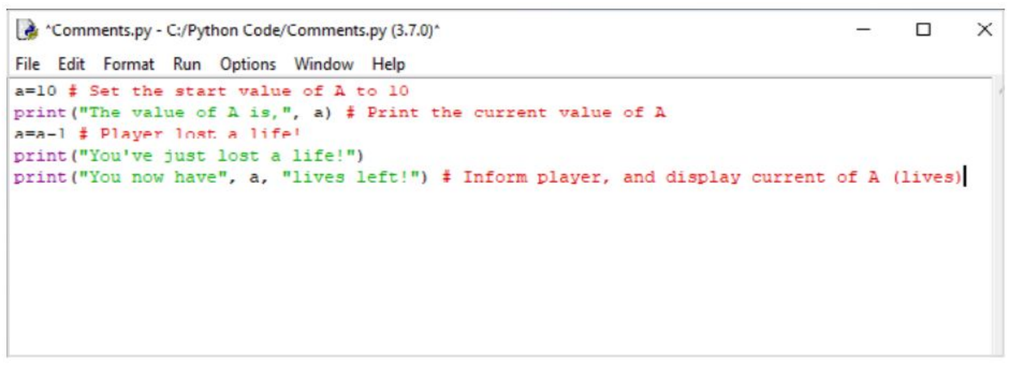

**STEP 7** You can use comments in different ways. For example, Block Comments are a large section of text that details what's going on in the code, such as telling the code reader which variables you're planning on using:

```
# This is the best game ever, and has been
developed by a crack squad of Python experts
# who haven't slept or washed in weeks. Despite
being very smelly, the code at least
# works really well.
```



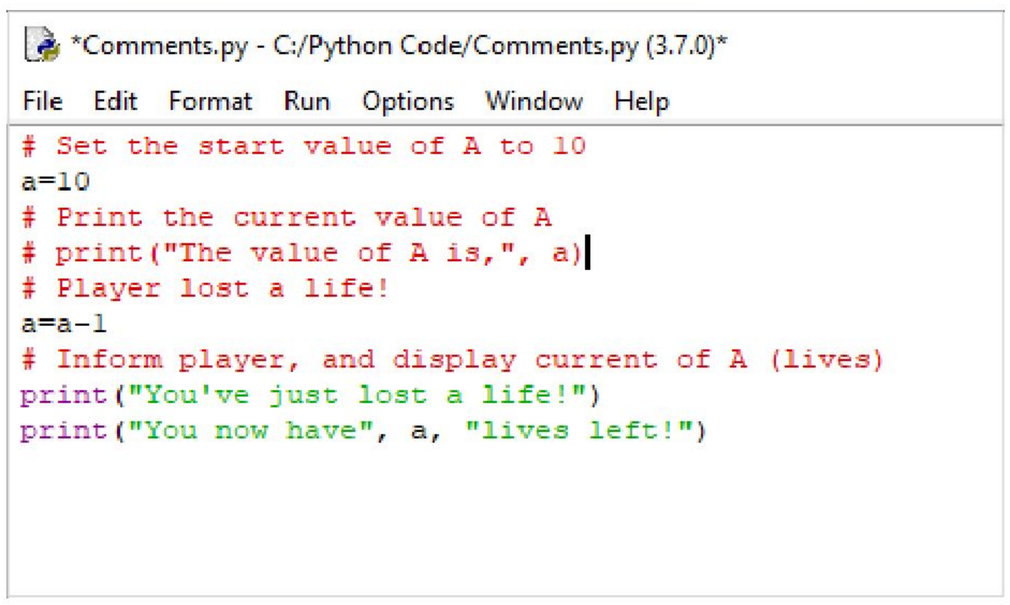

**STEP 8** Inline Comments are comments that follow a section of code. Take our examples from above, instead of inserting the code on a separate line, we could use:

```
a=10 # Set the start value of A to 10
print("The value of A is,", a) # Print the current
value of A
a=a-1 # Player lost a life!
print("You've just lost a life!")
print("You now have", a, "lives left!") # Inform
player, and display current value of A (lives)
```




**STEP 9** The comment, the hash symbol, can also be used to comment out sections of code you don't want to be executed in your program. For instance, if you wanted to remove the first print statement, you would use:

```
# print("The value of A is,", a)
```




**STEP 10** You also use three single quotes to comment out a Block Comment, or multi-line section of comments. For them to work, place them before and after the areas you want to comment:

```
'''
This is the best game ever, and has been developed
by a crack squad of Python experts
who haven't slept or washed in weeks. Despite
being very smelly, the code at least
works really well.   '''
```