

Below is a continuous sample with multiple pages, portraying a hierarchical structure.

## TOC

[Get started with Python](#)

[Why Python](#)

[Setup your dev environment](#)

[Create a new workspace](#)

[Say Hello World](#)

[Evaluate arithmetic expressions](#)

[Write a Calculator program](#)

[Reference guide](#)

[Python syntax](#)

[Dunder name](#)

# Get started with Python

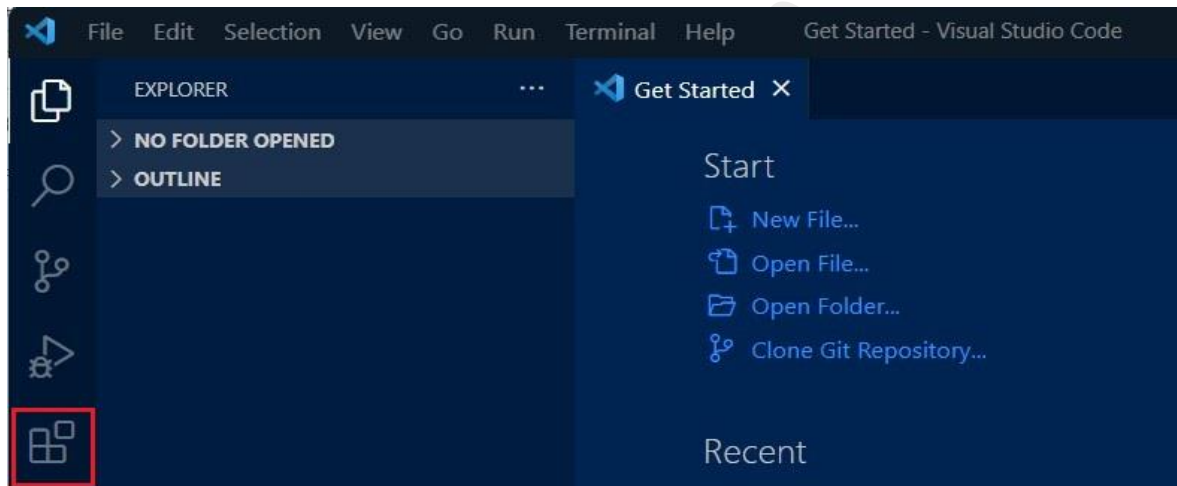
## Why Python

Python is a high-level object-oriented programming language that is incredibly efficient. As compared to the other existing languages, Python's easy-to-read and quick-to-code syntax make it stand out. It supports an ever-increasing list of modules and packages that enable bundling multiple applications together. You can also experience a fast edit-test-debug cycle with Python; leverage Python in the fields of web development, machine learning, task automation, GUI applications such as games in 2D and 3D, cloud- and mobile-based apps, and data visualization.

This quickstart shows how to create a project in Visual Studio Code and enables you to perform basic operations in Python.

## Setup your dev environment

1. Based on your system specifications and project needs, [download](#) and install the appropriate Python version.
2. [Download](#) and install Visual Studio Code (VS Code).
3. Open VS Code and click the **Extensions** icon.



4. Enter *Python* in the **Extensions Marketplace** search box. **Install** the Python extension published and verified by Microsoft.



5. Follow these steps to select a suitable Python interpreter:
  - a. From the VS Code menu, select **View** > **Command Palette**.
  - b. Type *Python: Select Interpreter* command and select it to see the available interpreters.
  - c. Select the recommended option for your use. Once selected, it should appear at the bottom-left-hand side of the VS Code window.

Next step

[Create a new workspace](#)

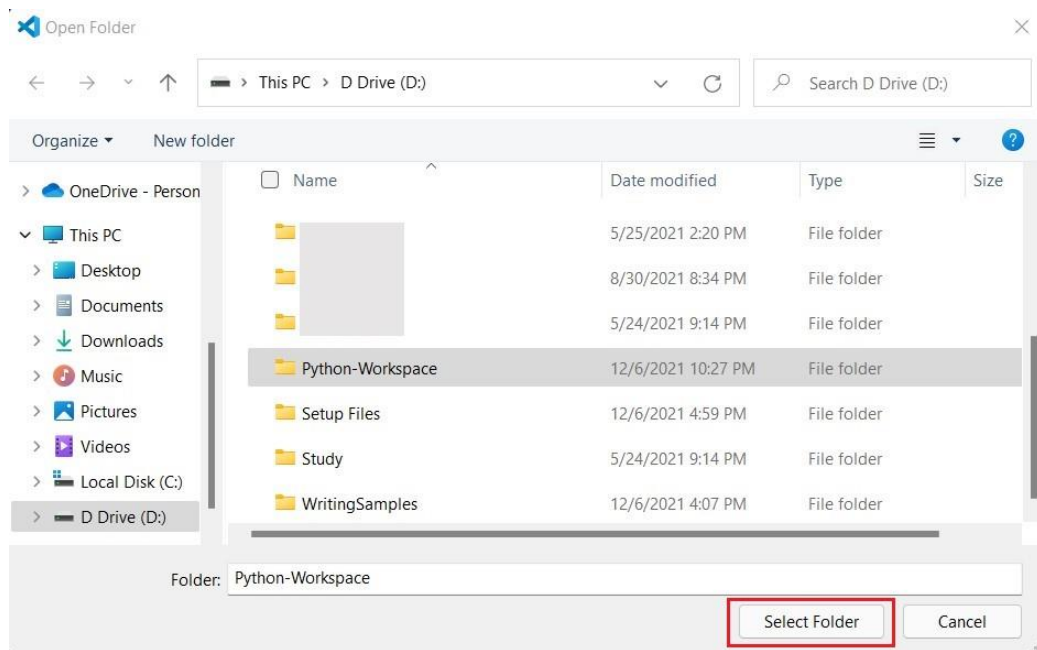
Namandeep's writing sample - Do not copy

## Create a new workspace

A workspace is defined as one folder or a combination of multiple folders. This depends on your development workflow.

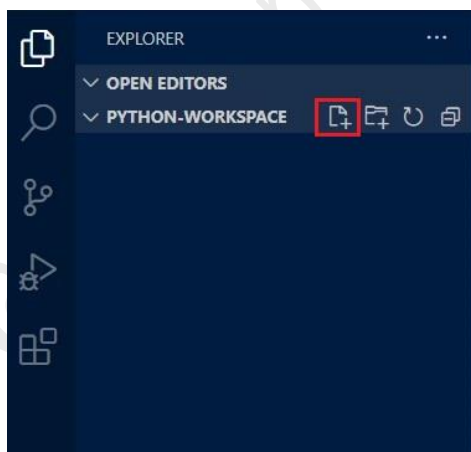
1. From the VS Code menu, select **File > Open folder**. Browse the location where you want to save the source code. You can either create a new workspace (folder) or select an existing workspace.
2. Once you select the desired workspace, click **Select folder**. In this quickstart, we name our workspace as *Python-Workspace*.

**Note:** If required, you can create a hierarchy of folders inside the selected workspace.



3. Your selected workspace will open up in VS Code's document explorer. Select your workspace folder and then select the **New File** option. Enter the name as *Hello.py*.

**Note:** The extension of a Python file is *.py*.

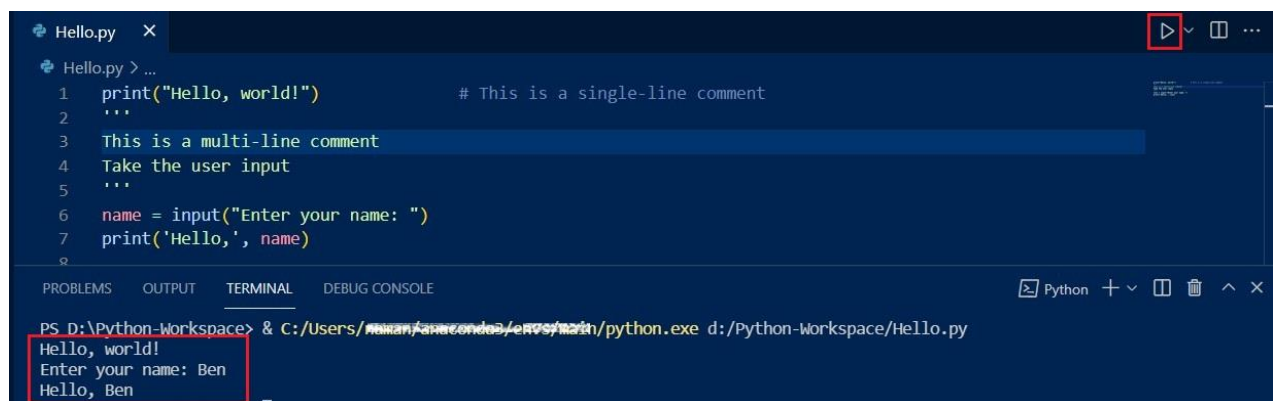


## Say Hello World

You are now ready to explore the syntax and semantics of the Python language.

1. In *Hello.py*, write the command to display "Hello, world!".

2. *Input* function enables you to accept an input from the user. By default, the user input is of string data type.



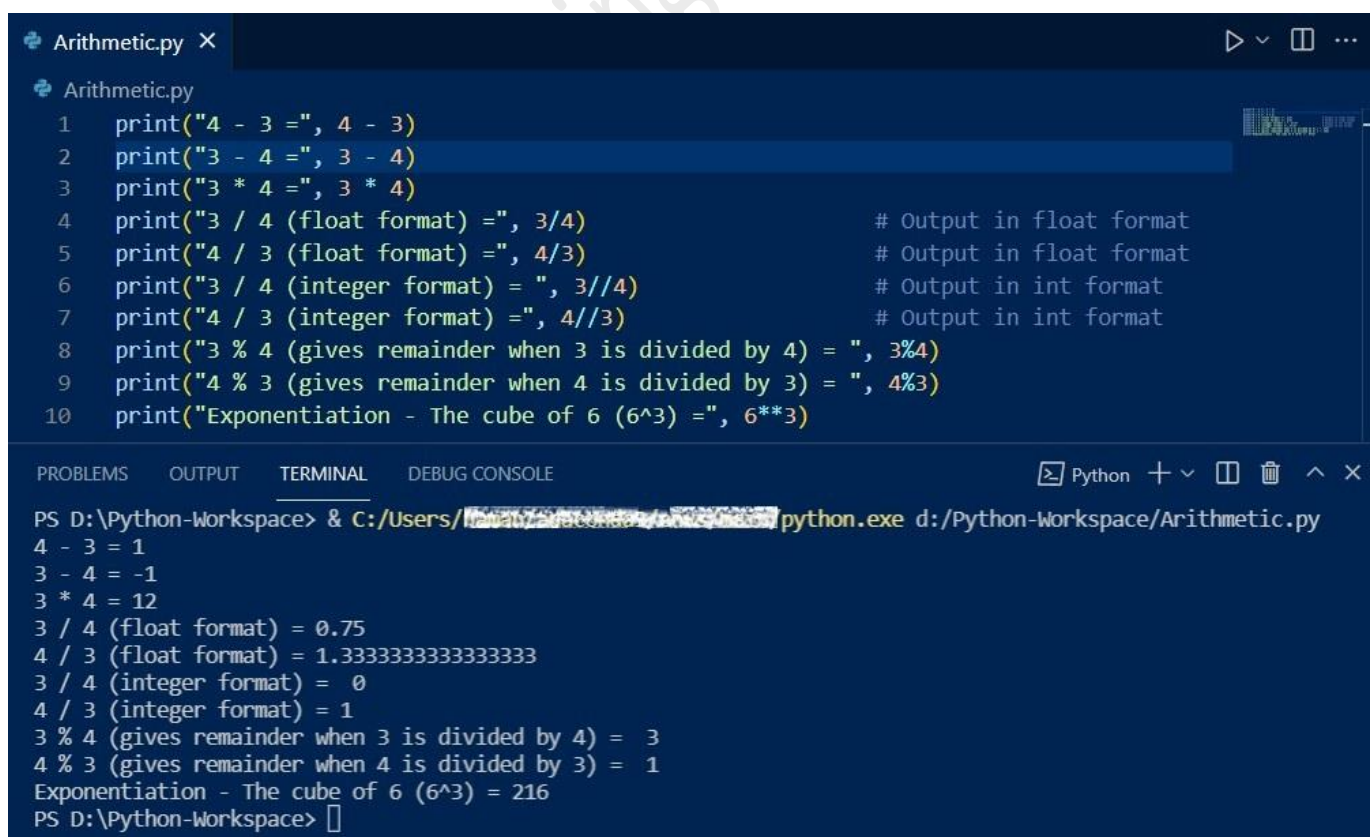
```
PS D:\Python-Workspace> & C:/Users/roman/Anaconda/Scripts/python.exe d:/Python-Workspace/Hello.py
Hello, world!
Enter your name: Ben
Hello, Ben
```

3. You can execute the file using one of the following options:
  - a. Select the **Run Python File** (play) button top-right-hand side of the editor window.
  - b. Right-click anywhere in the editor window and select **Run Python File in Terminal** option.
  - c. Press **Ctrl + F9** shortcut key. This option is specific to the Windows OS.

**Note:** If you have experience writing code in other language(s), you may notice that Python doesn't need a compiler. It uses an interpreter that compiles as well as executes the code. Therefore, you can view the errors directly at the run time.

### Evaluate arithmetic expressions

In the same workspace, create a new file and name it as *Arithmetic.py*. Add the following *print* statements in your file and observe the output.



```
PS D:\Python-Workspace> & C:/Users/roman/Anaconda/Scripts/python.exe d:/Python-Workspace/Arithmetic.py
4 - 3 = 1
3 - 4 = -1
3 * 4 = 12
3 / 4 (float format) = 0.75
4 / 3 (float format) = 1.3333333333333333
3 / 4 (integer format) = 0
4 / 3 (integer format) = 1
3 % 4 (gives remainder when 3 is divided by 4) = 3
4 % 3 (gives remainder when 4 is divided by 3) = 1
Exponentiation - The cube of 6 (6^3) = 216
PS D:\Python-Workspace>
```

**Quick exercise:** The precedence of these operators is as below. Make changes to the above expressions and verify the order yourself.

**Highest to lowest:** `**`, `-` (negation), `*`, `/`, `//`, `%`, `+`, `-`.

Next step

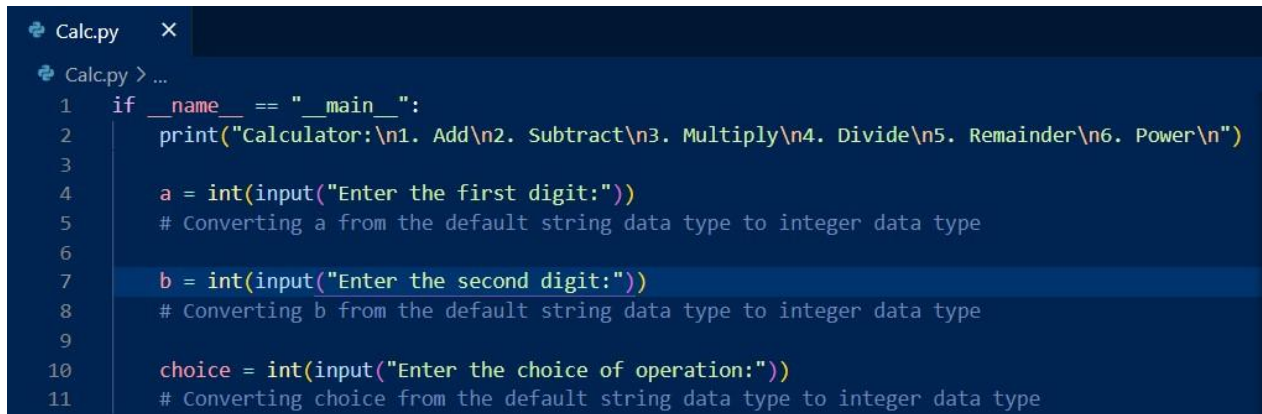
[Create a Calculator program](#)

Namandeep's writing sample - Do not copy

## Create a Calculator program

Once you get familiarized with the introductory commands, get started with the following instructions to create your own calculator program:

1. In the same workspace, create a new Python file and name it as *Calc.py*.
2. Write the following code structure in your editor window.

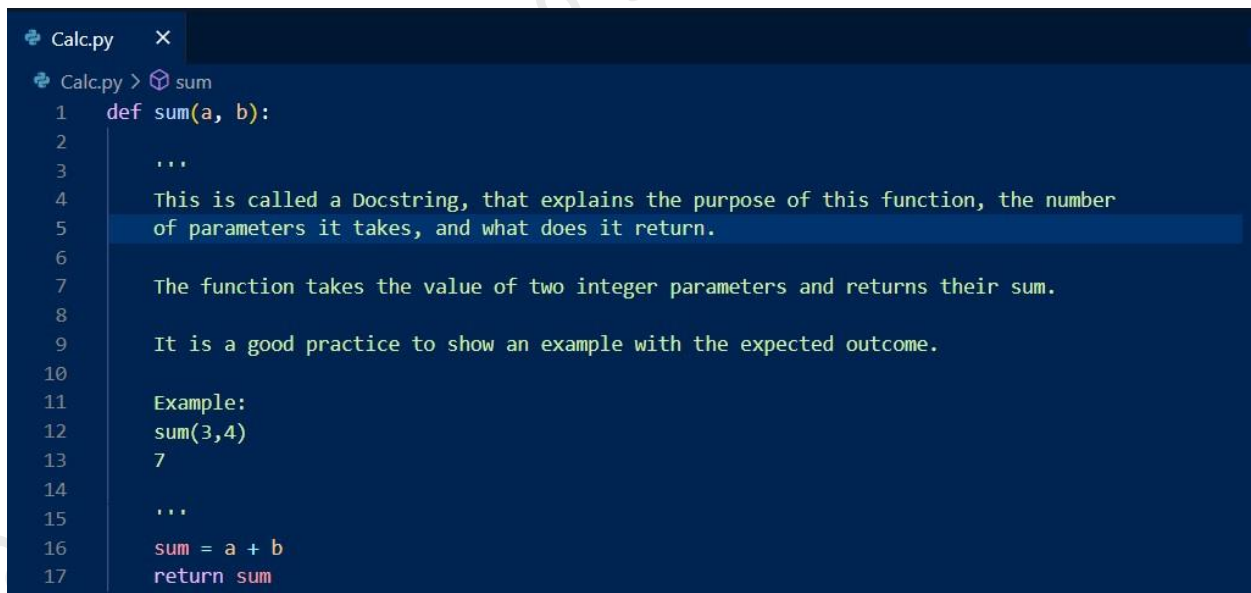


```
Calc.py
Calc.py > ...
1  if __name__ == "__main__":
2      print("Calculator:\n1. Add\n2. Subtract\n3. Multiply\n4. Divide\n5. Remainder\n6. Power\n")
3
4      a = int(input("Enter the first digit:"))
5      # Converting a from the default string data type to integer data type
6
7      b = int(input("Enter the second digit:"))
8      # Converting b from the default string data type to integer data type
9
10     choice = int(input("Enter the choice of operation:"))
11     # Converting choice from the default string data type to integer data type
```

For information on `if __name__ == "__main__":` and the syntax used here, see [Reference guide](#).

3. Create (user-defined) function definitions to perform the chosen operation. Before the `if __name__ == "__main__":` block, add the following definition for the `sum(a,b)` function.

**Note:** Ensure to use the return statement while working with functions.



```
Calc.py
Calc.py > sum
1  def sum(a, b):
2
3      ...
4      This is called a Docstring, that explains the purpose of this function, the number
5      of parameters it takes, and what does it return.
6
7      The function takes the value of two integer parameters and returns their sum.
8
9      It is a good practice to show an example with the expected outcome.
10
11     Example:
12     sum(3,4)
13     7
14
15     ...
16     sum = a + b
17     return sum
```

4. Similarly, create the function definition for the `subtract(a,b)` function.

**Note:** You can also combine the if-else construct with the return statement. Try it out by using the code in the last comment.

```

19 def subtract(a, b):
20     '''
21     The function takes the value of two input parameters and returns their difference by
22     subtracting the smaller number from the larger number.
23
24     Example:
25     sum(12,24)
26     12
27     '''
28     # Introducing if-else construct
29     if a > b:
30         return a - b
31     else:
32         return b - a
33
34     # Another way to write this: return a - b if a > b else b - a.

```

5. For the *product(a,b)* function, we have simply added a *return(a\*b)* statement. Write the following function definition in your editor window. You can use a similar statement for other functions as well.

```

36 def product(a, b):
37     '''
38     The function takes the value of two input parameters and returns their product.
39
40     Example:
41
42     product(5,7)
43     35
44
45     '''
46
47     return (a * b)
48

```

6. For rest of the functions, add a suitable *docstring* along with an example.

```

49 def div(a, b):
50     return a / b
51
52
53 def power(a, b):
54     return a ** b
55
56 def modulo(a, b):
57     return a % b

```

7. You can now call these functions inside the *if \_\_name\_\_ == "\_\_main\_\_":* block. Based on the choice of operation, we have used the *if-else-if* construct to call the required function.

Update the *if \_\_name\_\_ == "\_\_main\_\_":* block.

**Note:** Observe how we have used a variable *result* to catch the value returned from the function calls.



```

59 if __name__ == "__main__":
60     print("Calculator:\n1. Add\n2. Subtract\n3. Multiply\n4. Divide\n5. Remainder\n6. Power\n")
61
62     a = int(input("Enter the first digit:"))
63     # Converting a from the default string data type to integer data type
64
65     b = int(input("Enter the second digit:"))
66     # Converting b from the default string data type to integer data type
67
68     choice = int(input("Enter the choice of operation:"))
69     # Converting choice from the default string data type to integer data type
70
71     if choice == 1:
72         result = sum(a, b)
73
74     elif choice == 2:
75         result = subtract(a, b)
76
77     elif choice == 3:
78         result = product(a, b)
79
80     elif choice == 4:
81         result = div(a, b)
82
83     elif choice == 5:
84         result = modulo(a, b)
85
86     elif choice == 6:
87         result = power(a, b)
88
89     print(result) # This is outside the if-else-if construct.

```

8. Now you can execute this program.

Food for thought

What would happen if you enter an option that does not exist in the suggestions? Try it out!

Once you have successfully executed the program, add the following piece of code to the *if-else-if* construct:

```

86     elif choice == 6:
87         result = power(a, b)
88
89     # Start here
90     else:
91         print("Invalid option.")
92         result = "Invalid" # See what happens if you do not write this statement.
93
94     print(result) # This is outside the if-else-if construct.

```

Next steps

[Reference guide](#)

## Reference guide

### Python syntax

In this topic, we will discuss the syntax used in *Hello.py* and *Arithmetic.py*.

**Note:** As we proceed further, should you want to explore the type of arguments that the built-in functions offer, you can run the `help(functionName)` command in your IDE. An example would be `help(print)`.

- **print** - You can use the in-built print function to display any message or output on the screen. The table below describes you can leverage this command. You can run the `help(print)` command to know more about the optional keyword arguments.

Command	Description
<code>print("Hello, World!")</code>	Prints a string. You can use the escape characters to make the string appear as desired. For example, <code>\n</code> is the newline character.
<code>print("Hello"+"World!")</code>	For strings, the unary plus operator acts as a concatenation operator. This prints <i>HelloWorld!</i> without a space.
<code>print('Hello','World!')</code>	Prints <i>Hello World!</i> . Observe that the single-quotes work just as well.
<code>a= 10</code> <code>print(a)</code>	Prints value of a variable. For this example, the output is <i>10</i> .
<code>print('M', end=', ')</code> <code>print('A', end=', ')</code> <code>print('R', 'S', sep=', ')</code>	Prints in the following order: <i>M,</i> <i>A,</i> <i>R, S.</i> <i>end</i> is an optional argument that appends the string after the last value. Default is a newline. <i>sep</i> is a separator string inserted between the values. Default is a space.

- **Built-in types** - You can identify the built-in type of a variable or any value by using the `type(objName)` command in your editor window.

Commands	Class type
<code>a = 42</code> <code>print(type(a))</code>	<class 'int'>
<code>print(type(3.14))</code>	<class 'float'>
<code>print(type(0.5 + 6j))</code>	<class 'complex'>
<code>print(type(True))</code>	<class 'bool'>
<code>print('hello')</code>	<class 'str'>

- **input** function - You can prompt the user to provide an input using the `input` function. By default, the input is read as a string. When working with types other than string, ensure that you convert them to the desired type.

print type	Output
<code>print( type (str(4) ) )</code>	<class 'str'>
<code>print(str(4))</code>	4

print( type( float(4) ) )	<class 'float'>
print(float(4))	4.0
print( type( bool(0) ) )	<class 'bool'>
print(bool(0))	False
print( type( bool(4) ) )	<class 'bool'>
print(bool(4))	True
print( type( int(4.0) ) )	<class 'int'>
print(int(4.0))	4
print( type( int('4') ) )	<class 'int'>
print(int('4'))	4
def funcNone(): print(4)	4
print(funcNone())	None
	This happened because there was no return statement in the <i>funcNone</i> function.

**Quick exercise:** Test more examples with different values and built-in types. Observe what happens if you run the `print(int('4.4'))` command. Do you know what could fix this? Run the `print(int(float('4.4')))` command to fix the error.

- *If-else-if* construct - The general syntax of an *if-else-if* block is as shown in [Create a Calculator program](#). An *if-else* block is widely used across multiple programming languages and caters to the decision-making process.

**Quick exercise:** Test out some examples by starting with an *if* block, modify your code to explore an *if-else* block, *if-elif* block, and *if-elif-else* block. Here's how you can get yourself started:

```

if-else.py > ...
1  One = 'A' # ASCII value 65
2  Two = 'a' # ASCII value 97
3  if One < Two:
4      print('Two')
5      print(Two)
6

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```

PS D:\Python-Workspace> & C:/Users/naman/anaconda3/Scripts/python.exe d:/Python-Workspace/if-else.py
Two
a
PS D:\Python-Workspace> 

```

## Dunder name

In this article, we will explore the purpose and behaviour of `if __name__ == "__main__":` block and Python's module import mechanism. The use of the word "Dunder" is to signify the double underscores.

## How does Python interpreter behaves

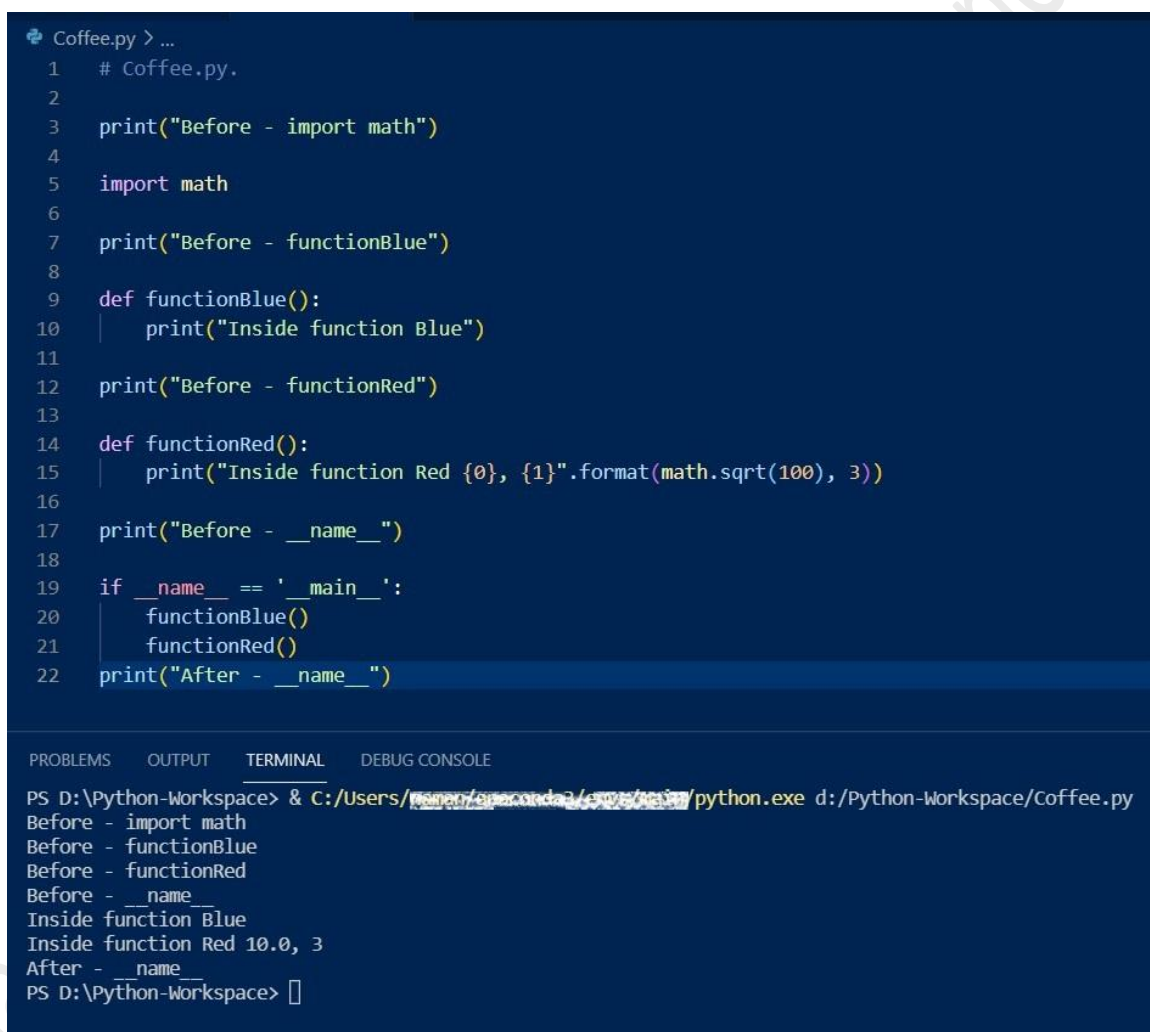
The Python interpreter reads a file and assigns a hard-coded string (`__main__`) to a special variable called `__name__`. While running a program, the interpreter inserts this string at the top of your module.

**Note:** Each module is a program in Python.

## Your module is the main program

Python has an extensive range of modules and libraries. You can import any existing module in your piece of code.

1. Create a new Python file and name it as *Coffee.py*.
2. Add the following code to *Coffee.py*. Run this file and observe the output.



```
1 # Coffee.py.
2
3 print("Before - import math")
4
5 import math
6
7 print("Before - functionBlue")
8
9 def functionBlue():
10 |     print("Inside function Blue")
11
12 print("Before - functionRed")
13
14 def functionRed():
15 |     print("Inside function Red {0}, {1}".format(math.sqrt(100), 3))
16
17 print("Before - __name__")
18
19 if __name__ == '__main__':
20 |     functionBlue()
21 |     functionRed()
22 print("After - __name__")
```

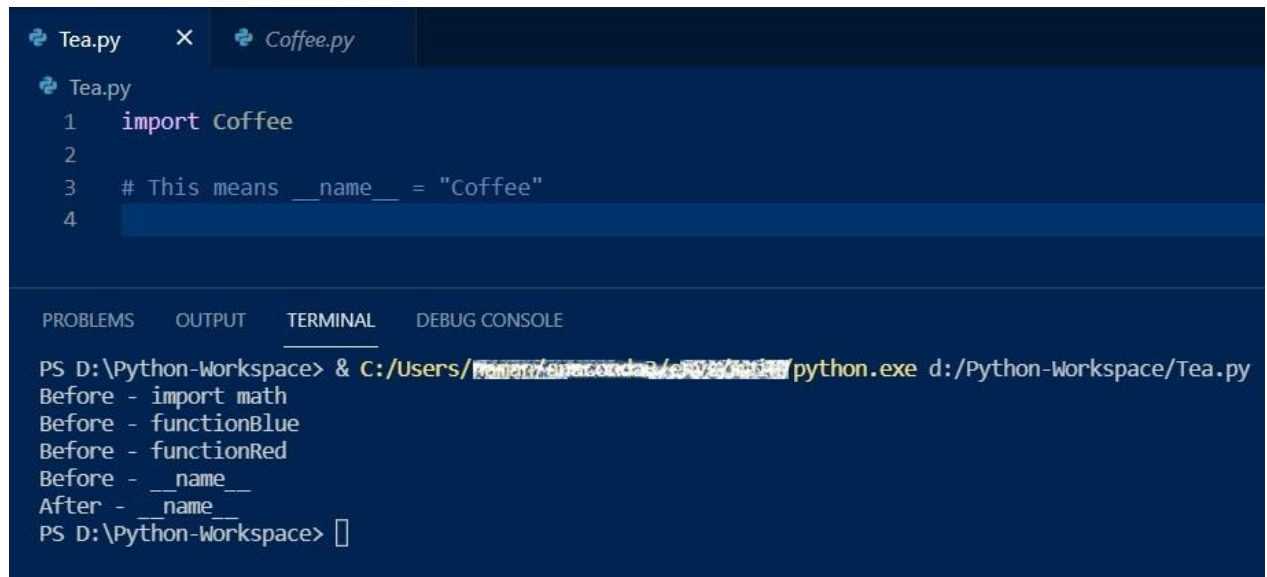
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS D:\Python-Workspace> & C:/Users/rohan/OneDrive/Desktop/Python/python.exe d:/Python-Workspace/Coffee.py
Before - import math
Before - functionBlue
Before - functionRed
Before - __name__
Inside function Blue
Inside function Red 10.0, 3
After - __name__
PS D:\Python-Workspace>
```

## New module imports your module

What happens when a new module imports your module? How does the interpreter's behavior changes?

1. Create a new file and name it as *Tea.py*. Import the *Coffee* module.
2. When you import the *Coffee* module, the interpreter will search for this module and assign the special variable `__name__` to *Coffee*.
3. Run this program as-is and compare the output observed in the [previous section](#).



```
Tea.py  X  Coffee.py

Tea.py
1  import Coffee
2
3  # This means __name__ = "Coffee"
4

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Python-Workspace> & C:/Users/Namandeep/AppData/Local/Programs/Python/Python38-32/python.exe d:/Python-Workspace/Tea.py
Before - import math
Before - functionBlue
Before - functionRed
Before - __name__
After -  name
PS D:\Python-Workspace> 
```

**Quick exercise:** Add an `if __name__ == "__main__":` block and print statements in the `Tea.py` file. Now run it and observe the output.