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

TOC

Get started with Python

Why Python?

Python is a high-level object-oriented programming language that is incredibly efficient. 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 as an easy process. You can also experience a fast edit-test-debug cycle with Python; leverage Python in the fields of web development, machine learning, GUI applications such as games in 2D and 3D, cloud- and mobile-based apps, and data visualization.

Prerequisite

Download and install the PyCharm Community edition.

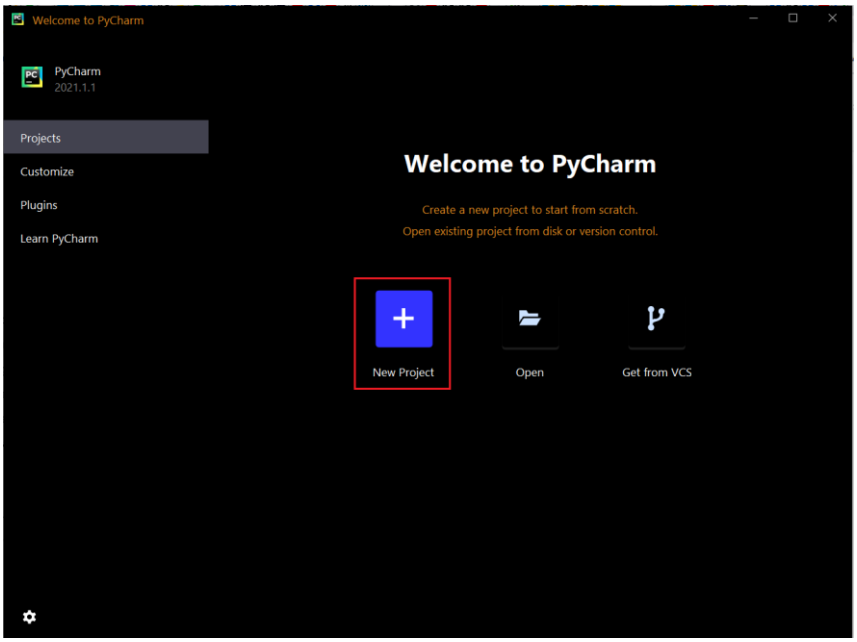
Next step

Write your first Python program

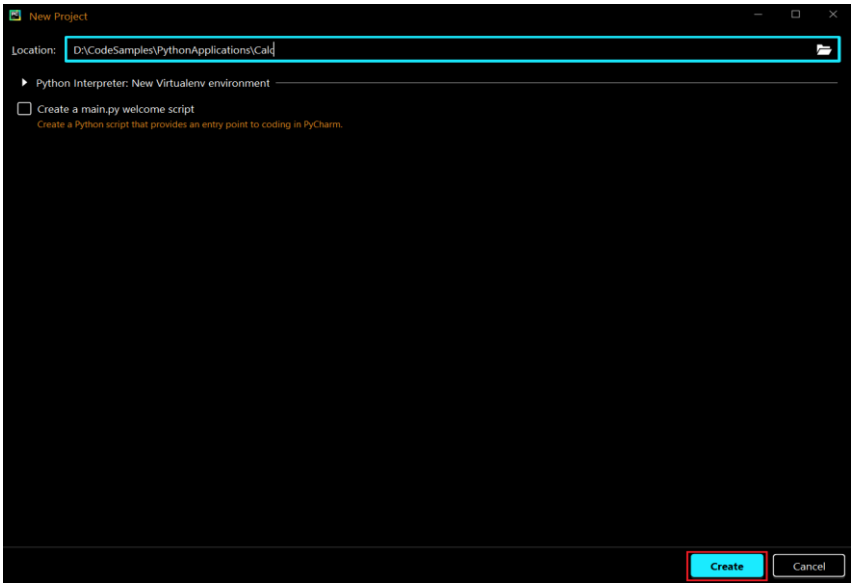
Write your first Python program

Create a new project

1. Open the PyCharm Community Edition developer tool.
2. Click on **New Project**.

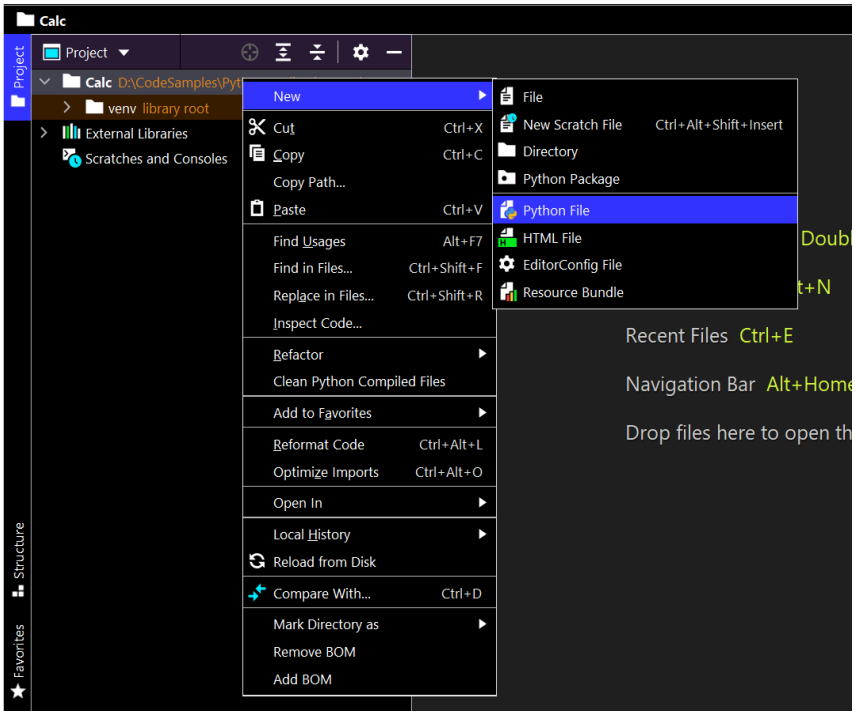


3. Browse the location where you want to save your project. Click on **Create**.
   Note: If the folder does not exist already, it will be created automatically. For example, the folder **Calc** did not exist before clicking **Create**.



4. Right-click on your project name, select **New** > **Python File**. Enter the name of your **New Python File** as **Arithmetic.py**.

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



Next step

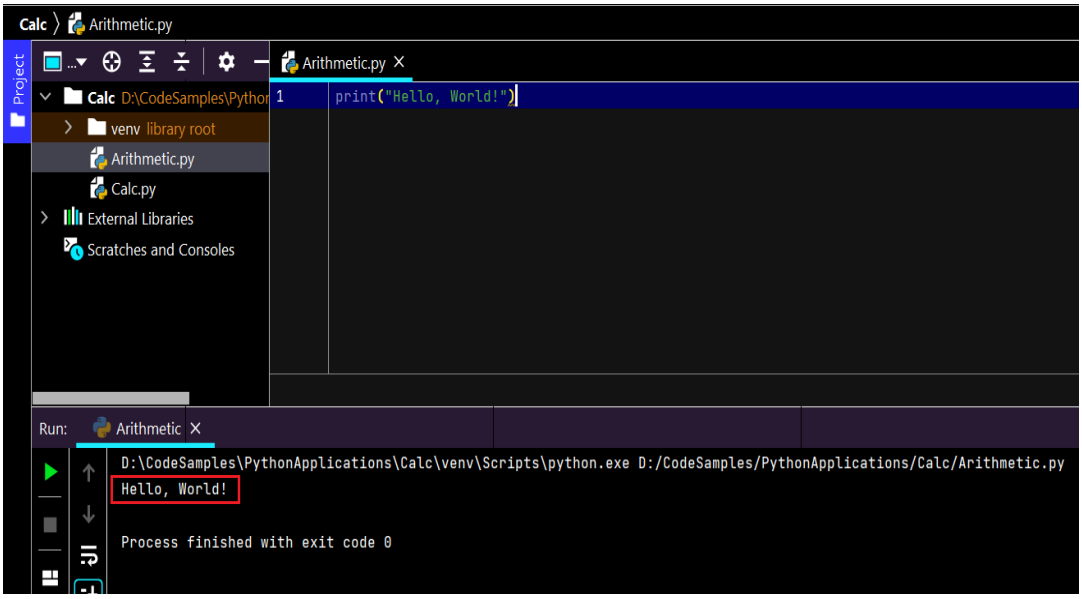
[Try sample commands]

Try sample commands

Once you have created your project, you are now ready to explore the single-line commands, starting from the classic “Hello, World!”. This will provide you a quick insight to the readable syntax offered by Python.

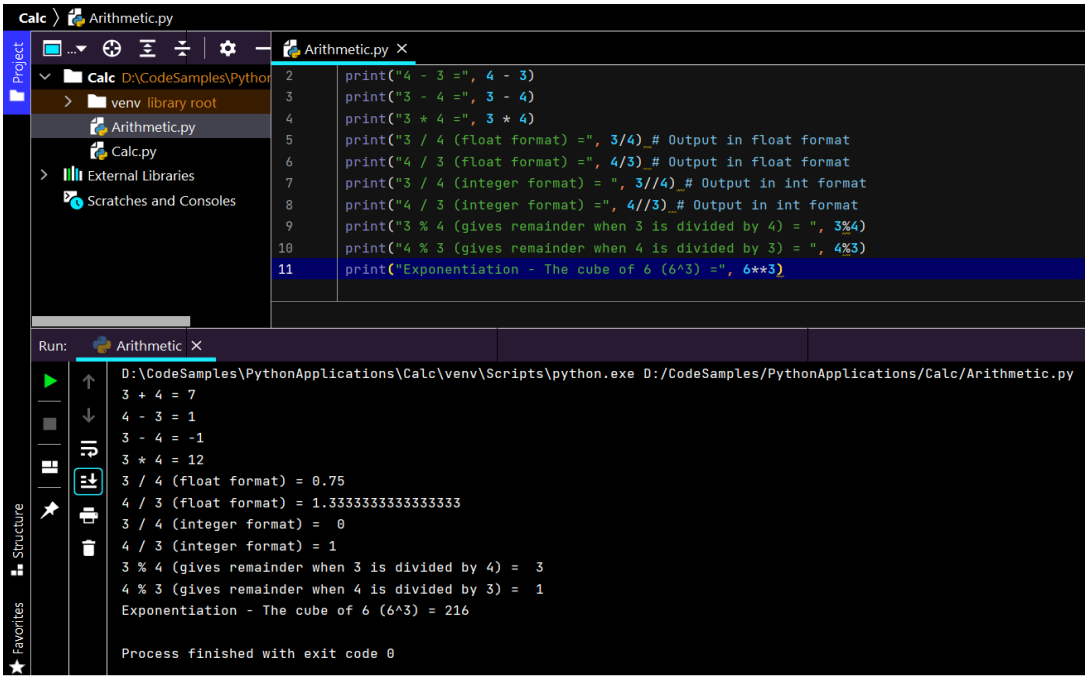
1. In *Calc.py*, write the command to display “Hello World”- *print(“Hello, World!”)*.
2. Run this command by using any of the following two methods:
    a. Right-click on Calc.py and click on **Run Arithmetic.py**.
    b. Use the **Ctrl+Shift+F10** shortcut key.

    You can see the output in the **Run** window at the bottom.



3. Try out the following arithmetic operators in Python. Copy the following commands to the **Arithmetic.py** file:

```
print("4 - 3 =", 4 - 3)
print("3 - 4 =", 3 - 4)
print("3 * 4 =", 3 * 4)
print("3 / 4 (float format) =", 3/4) # Output in float format
print("4 / 3 (float format) =", 4/3) # Output in float format
print("3 / 4 (integer format) = ", 3//4) # Output in int format
print("4 / 3 (integer format) =", 4//3) # Output in int format
print("3 % 4 (gives remainder when 3 is divided by 4) = ", 3%4)
print("4 % 3 (gives remainder when 4 is divided by 3) = ", 4%3)
print("Exponentiation - The cube of 6 (6^3) =", 6**3)
```



Note: The # symbol is used to mark the beginning of a single-line comment.

Quick exercise: The order of precedence of these operators is as below. Make changes to the commands tested in the step above and verify the order yourself.
Highest to lowest: ***, - (negation), [*, /, //, %], [+, -]*.

Next step

[Create a Calculator program]

Create a Calculator program

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

1. Create a new Python file and name it as **Calc.py**.
2. Copy the following code structure in your IDE and run as-is.

```
if __name__ == "__main__":
    print("Calculator:\n1. Add\n2. Subtract\n3. Multiply\n4. Divide\n5.
Remainder\n6. Power\n")

    a = int(input("Enter the first digit:"))  # By default, a is of type String
    b = int(input("Enter the second digit:")) # By default, a is of type String
    choice = int(input("Enter the choice of operation:"))
```

For information on *if __name__ == "__main__":*, see Dunder name. For further information on syntax used here, see Python syntax.

3. Create function definitions to perform the chosen operation. Add the following definition for the *sum(a,b)* function before the *if __name__ == "__main__":* block.
Note: Ensure to use the return statement while working with functions.

```
def sum(a, b):

    '''
    This is called a Docstring, that explains what the function is doing, how
many parameters it takes and what does it
    returns.
    The function takes the value of two integer parameters and returns their
sum.

    It is a good practice to show an example with the expected outcome.

    Example:
    sum(3,4)
    7

    '''
    sum = a + b
    return sum
```

4. Similarly, you can create the function definition for the *subtract(a,b)* function.

```
def subtract(a, b):
    '''
    The function takes the value of two input parameters and returns their
difference by subtracting the smaller number from the larger number.

    Example:
    sum(12,24)
    12
    '''
    # Introducing if-else construct
    if a > b:
        return a - b
    else:
        return b - a

    # Another way to write this: return a - b if a > b else b - a.
```

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

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

```
def product(a, b):
    '''
        The function takes the value of two input parameters and returns their
product.

        Example:

        product(5,7)
        35

    '''

    return (a * b)
```

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

```
def div(a, b):
    return a / b


def power(a, b):
    return a ** b

def modulo(a, b):
    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.

```
if __name__ == "__main__":
    print("Calculator:\n1. Add\n2. Subtract\n3. Multiply\n4. Divide\n5.
Remainder\n6. Power\n")
    a = int(input("Enter the first digit:"))
    # By default, a is of type String. Convert it to type int.
    b = int(input("Enter the second digit:"))
    # By default, b is of type String. Convert it to type int.
    choice = int(input("Enter the option number:"))
    # Converting choice to type int.
    if choice == 1:
        result = sum(a, b)
    elif choice == 2:
        result = subtract(a, b)
    elif choice == 3:
        result = product(a, b)
    elif choice == 4:
        result = div_f(a, b)
    elif choice == 5:
        result = modulo(a, b)
    elif choice == 6:
        result = power(a, b)

    print(result)    # This is outside the if-else-if construct.
```

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

8. You can now execute this program.

Food for thought

Did you observe how Python doesn't compile the code but directly runs it? That's because Python interprets the code at the run time and then shows an error, if any.

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:

```
elif choice == 6:
    result = power(a, b)

# Start here
else:
    print("Invalid option.")
    result = "Invalid"  # See what happens if you do not write this statement.
print(result)    # This is outside the if-else-if construct.
```

Next steps

- Concepts

Concepts

Python syntax

In this topic, we will discuss the syntax used in Write your first Python program.

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 the how you can leverage this command. You can run the help(print) command to know more about the optional keyword arguments.

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

Commands	**Class type**
a = 42 print(type(a))	<class 'int'>
print(type(3.14))	<class 'float'>
print(type(0.5 + 6j))	<class 'complex'>
print(type(True))	<class 'bool'>
print(‘hello’)	<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**
print(type (str(**4**))) print(str(**4**))	<class 'str'> 4
print(type(float(**4**))) print(float(**4**))	<class 'float'> 4.0

print(type(bool(0))) print(bool(0)) print(type(bool(**4**))) print(bool(**4**))	<class 'bool'> False <class 'bool'> True
print(type(int(**4.0**))) print(int(**4.0**))	<class 'int'> 4
print(type(int('4'))) print(int('4'))	<class 'int'> 4
def funcNone(): print(**4**) print(funcNone())	4 None This happened because there was no return statement in the *funcNone* 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 with a similar syntax and caters to the decision-making process.

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

```
One = 'A' # ASCII value 65
Two = 'a' # ASCII value 97
if One < Two:
    print('Two')
    print(Two)
```

Output

```
Two
a
```

Dunder name

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

Python's interpreter behavior

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

Note: Each module is a program in Python.

Your module is the main program

When you a create a Python file, it gets an extension as *.py*. Python has an extensive range of modules and libraries. You can import any existing module in your piece of code. Test with the following code sample:

1. Create a new Python project and name it as *DunderName.py*.
2. Create a new Python file and name it as *foo.py*.

```python
# foo.py.

print("Before import math")

import math

print("Before functionBlue")

def functionBlue():
    print("Inside function Blue")

print("Before functionRed")

def functionRed():
    print("Inside function Red {0}, {1}".format(math.sqrt(100), 3))

print("Before __name__")

if __name__ == '__main__':
    functionBlue()
    functionRed()
print("After __name__")
```

3. Run this program and observe the output.

```
Before import math
Before functionBlue
Before functionRed
Before __name__
Inside function Blue
Inside function Red 10.0, 3
After __name__
```

New module imports your module

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

1. Continue with the code sample by creating a new file called *bar.py*. Import the *foo* module.

```python
# bar.py

import foo
```

2. Python interpreter searches for the *foo* module and assigns the *__name__* variable to *foo.*

```python
__name__ = "foo"
```

3. Since *__name__ = "foo"*, the body of the *if* statement gets skipped.

```
if __name__ == '__main__':
    functionBlue()
    functionRed()
```

4. Run this program as-is and compare the output observed in the previous section.

```
Before import math
Before functionBlue
Before functionRed
Before __name__
After __name__
```

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