Chapter 1 - Python Basics

This chapter will be a crash course in Python programming including how to install Python and the PyCharm IDE and verify the installations using the Python version of Hello World. We will create the Hello World program using ChatGPT AI as a "programming assistant". We will then cover basic Python coding such as comments, variables, data types, operators, input & output, selection code, functions, looping and iteration, list data structure, classes, objects, modules, and an introduction to graphical user interface (GUI) programming. I recommend that you type all example programs rather than copying my source code. It is like learning a new language, after awhile, your mind will begin speaking the new language fluently or in this case typing fluently.

Python and PyCharm Installation

Many installation tutorials and videos are available so I won't repeat them here. Watch the YouTube video: How to Install Python 3.11 and PyCharm on Windows. Install Python 3.11.5 rather than 3.11.4 as shown in the video. Install the latest version of PyCharm.

It is important to know about installation of earlier Python versions because 3rd Party Libraries tend to lag the release of new Python versions and you may need to install an earlier version to use a Library.

Ensure that your python directory is added to the Path in the Advanced settings. Open a command window and enter python --version. You should see the following if the installation and path are ok.

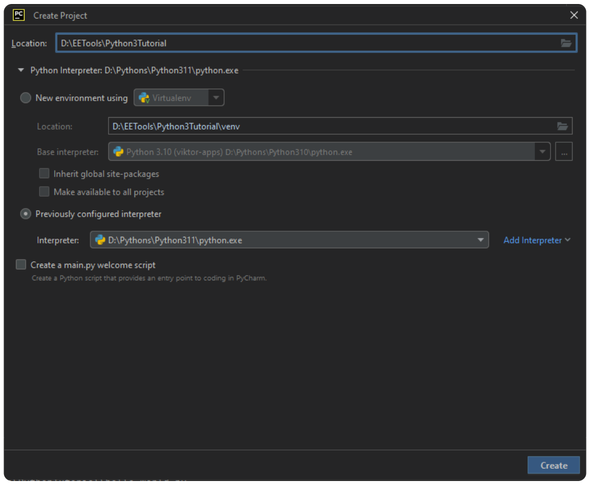
```
D:\EETools\Python3Tutorial>python --version
Python 3.11.5
```

Project Setup

Create a project directory, my directory is D:/EETools. The subdirectories in the location will contain the PyCharm projects for the tutorials. Run PyCharm and create a new project. In the Create Project dialog, set the location to D:\EETools\Python3Tutorial and set the previously configured interpreter to your Python 3.11.5 python.exe file as shown in the video.

One of the best features of PyCharm is that it allows easy configuration of "Virtual Environments" which isolates this project from all other projects allowing you to define the python interpreter and 3rd party libraries to this project.

Create Project Dialog



Location: D:\EETools\Python3Tutorial

Python Interpreter: D:\Pythons\Python311\python.exe

○ New environment using Virtualenv

Location: D:\EETools\Python3Tutorial\venv

Base interpreter: Python 3.10 (viktor-apps) D:\Pythons\Python310\python.exe

☐ Inherit global site-packages

☐ Make available to all projects

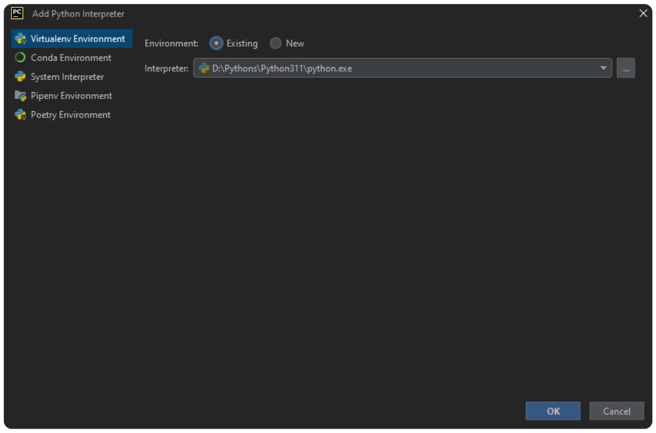
● Previously configured interpreter

Interpreter: D:\Pythons\Python311\python.exe

☐ Create a main.py welcome script

Create a Python script that provides an entry point to coding in PyCharm.

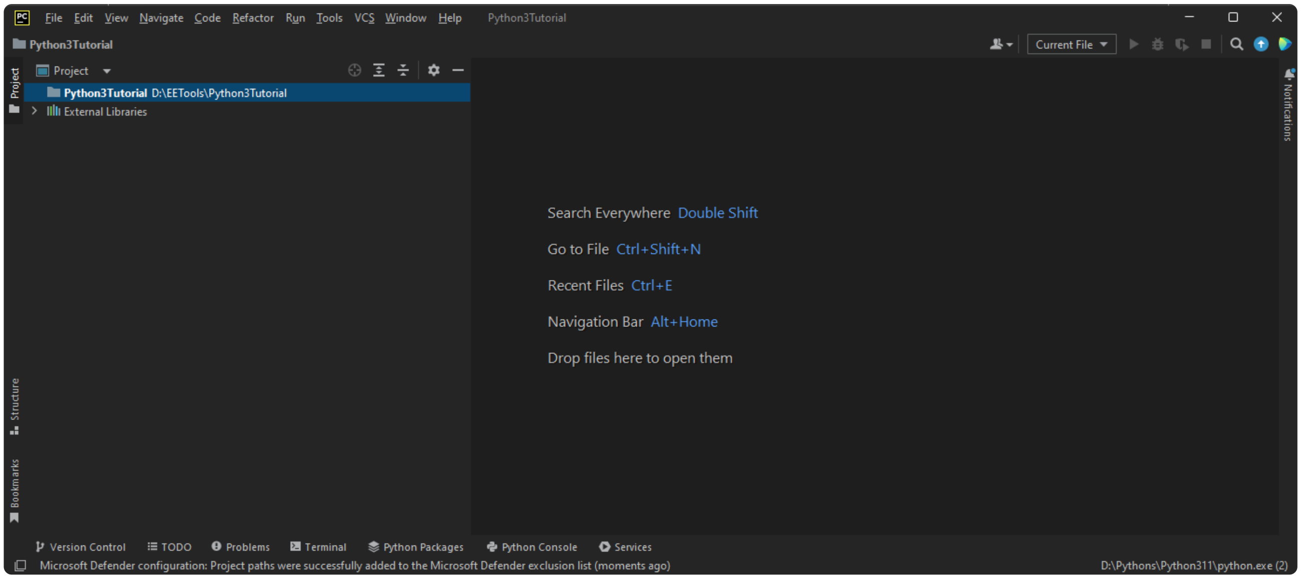
Python Interpreter Dialog



Environment: ● Existing ○ New

Interpreter: D:\Pythons\Python311\python.exe

If all goes well, the PyCharm IDE user interface will open.



Python3Tutorial D:\EETools\Python3Tutorial

External Libraries

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open them

Microsoft Defender configuration: Project paths were successfully added to the Microsoft Defender exclusion list (moments ago)

D:\Pythons\Python311\python.exe (2)

The current Python version is shown in the lower right corner. The Python directory is shown and the “venv” or virtual environment is listed. Source code is shown in the upper right window. The lower window is used to access the Terminal, Python Console, and run console as we will see after the first program is run.

Hello World

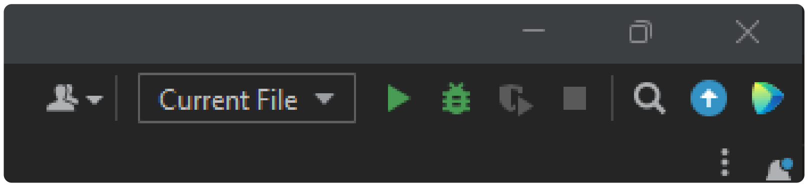
Many python tutorials call the main python program main.py. However, it is not necessary to do that. Create a new python file by selecting File > New > Python File > Name: hello_world. Note that it is not necessary to add the .py extension if you choose Python File. A new python file is created called hello_world.py. You can also right click on the project name: Python Tutorial > New > Python File > Name: hello_world. If you select File rather than Python File you will need to add the .py extension. If you need to change the name of a file, right click on the filename > Refactor > Rename > Name: hello_world.py.

On line 1 in the hello_world.py file enter the following python code:

```
print("Hello World!")
```

Note that PyCharm will verify compliance to PEP8 coding style by issuing warnings when styles are violated. In our simple Hello World program, a blank line is needed after the print line to comply with PEP 8. The style guide can be found at PEP 8 - Style Guide for Python Code. I have found that if I follow the style guide, I am a better programmer.

Check the Select Run/Debug Configuration drop down that it is set to "Current File". Run the program by clicking on the green arrow button.



PyCharm calls the configured interpreter (Python 3.11.5) and produces the output in the console:

```
D:\Pythons\Python311\python.exe D:\EETools\Python3Tutorial\hello_world.py
Hello World!

Process finished with exit code 0
```

If the program runs successfully, it demonstrates that the Python and PyCharm installations are working and we are ready to proceed with the Tutorial.

Let's look at the anatomy of the hello_world.py program.

```
print("Hello World!")
```

print() is a built-in function in Python 3 that sends the specified message to the standard output device which in this case is the PyCharm output console. We will learn how and why to define functions later. The message in this case is a string. A string is a sequence of characters enclosed in single or double quotes. We will learn more about strings in a later section.

This may seem like a trivial example but we will use print() functions in all programs, even those with a graphical user interface to debug the program. This is accomplished by placing print() statements at key locations in the program to display data in order to troubleshoot a problem. We can also use the PyCharm debugger to find problems in the code. We will discuss both debugging methods later.

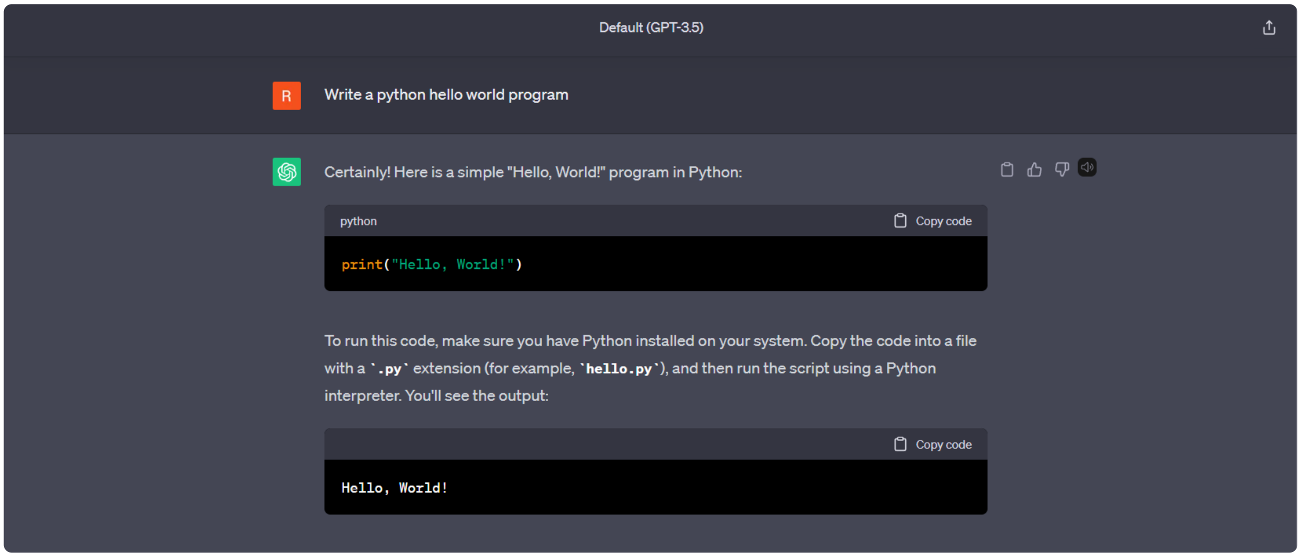
Pat yourself on the back for completing installation and running your first Python program. Try experimenting with the program by changing the string to print your name in the message.

Artificial Intelligence (AI) Programming Assistant

An AI such as ChatGPT can be used as a programming assistant for small coding challenges. Note that a program generated by AI may contain programming errors or "bugs".

Open ChatGPT and start a new chat. Note that currently ChatGPT-3.5 is free to use.

Send the following message:
Write a python hello world program.



Write a python hello world program

Certainly! Here is a simple "Hello, World!" program in Python:

```python
print("Hello, World!")
```

To run this code, make sure you have Python installed on your system. Copy the code into a file with a `.py` extension (for example, `hello.py`), and then run the script using a Python interpreter. You'll see the output:

```
Hello, World!
```

ChatGPT correctly creates our hello_world.py program and even shows the expected output. To verify the code, copy the program into a PyCharm file and run it. AI is very helpful as a programming assistant to give you ideas on how to write code to perform a specific task. The "Regenerate" function will rerun the request and often shows another method to write the code. This can be much faster than using a search engine to find the same methods. Note that ChatGPT can be configured with a voice interface.

Comments

Comments are messages to a programmer reading the source code. Comments are not executed by the python interpreter. Python supports single-line comments, multi-line comments, and inline comments. Comments can be used by the programmer to "comment out" a line or multiple lines to try new code before deleting old code or to see the effect of removing some code from execution.

Create a new program called comments.py.

```python
# Single line comments

"""
Multi-line comment
using
triple quotes
"""

print("Python comments are awesome")  # This is an inline comment
```

```
# print("This line is commented out")
```

Later we will learn about doc strings which are used to document functions and classes.

Variables & Operators

A variable is a human readable name for data that is stored in computer memory. Without variables, the programmer would need to provide the memory address where the data is stored, not very user friendly.

Variables are "assigned" a value that can be of different "data types" such as:

- integers (number without decimal point)
- floats (number with a decimal point)
- string which we have already encountered
- boolean - true or false
- data containers such as list, dictionary, set, tuple
- custom user types
- many other types

Operators perform "operations" on values and variables called "operands". Types of operators in Python include:

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Identity Operators
- Membership Operators

Create a new program called variables_and_operators.py

```
# Assigning various data types to variables

# Integers
```

```
a = 1
b = 2

# Integer math operators
add = a + b
sub = b - a
mult = a * b
div = a / b

# Print string and values for math operations
print("a + b = ", add, " b - a = ", sub, " a * b = ", mult, " a / b = ", div)
# Print on a single line
print("a + b = ", add, "\nb - a = ", sub, "\na * b = ", mult, "\na / b = ",
div)  # Print on multiple lines
```

Console Output

```
a + b =  3  b - a =  1  a * b =  2  a / b =  0.5
a + b =  3
b - a =  1
a * b =  2
a / b =  0.5
```

We will cover other data types and operators as they are needed in the examples. We will also create a scientific calculator later with a graphic user interface (GUI) that uses these operators.

Input and output

User input refers to interaction between the program user (human) and the program (computer). Output refers to computer generated data that is displayed on the computer screen such as the print() function we have already discussed. Although console programs are easy to code and see quick results, most if not all modern programs have a GUI interface. We will discuss GUI programming in detail later.

Create a new program called input_output.py

```
# Get user input
name = input("What is your name?: ")

# Output a message to the computer screen using the provided input
print("Hello ", name)
```

Console Output - Note that I entered my name: Rick

```
What is your name?: Rick
Hello Rick
```

Selection using if...elif...else

Computer programs can make decisions based on certain conditions in the program. The if...elif...else is a python programming statement that allows the program to execute code blocks if conditions are met. All programming languages have a way to define a code block. Many languages use curly brackets to define the block. Python uses indentation to define a code block. The indentation is usually 4 spaces for code readability.

In Python, the simply if statement syntax is:

if condition:
code executed if condition is true

Create a new program called selection.py

```
# Define a variable for the ambient temperature
temperature = 12

# Simple if statement
if temperature < 32:
    print("It is cold outside")

# if..else statement
temperature = 75
if temperature < 32:
    print("It is cold outside")
```

```
else:
    print("It is not too cold today")

# if..elif..else statement
temperature = 90
if temperature < 32:
    print("It is cold outside")
elif 32 <= temperature <= 80:
    print("It is a pleasant day today")
else:
    print("It is a hot day today")
```

Console Output

```
It is cold outside
It is not too cold today
It is a hot day today
```

Loops

Computer programs can execute code very quickly and are ideal for executing the same code in a loop. Python supports multiple types of loops:

- for loop
- while loop

Create a new program called loops.py

```
# for loop
for i in range(0, 5):
    print(i)

# while loop
i = 0
while i < 5:
    print(i)
    i = i + 1
```

Console Output

```
0
1
2
3
4
0
1
2
3
4
```

The for loop will be the most used loop in later project.

Functions

Functions provide a way of "reusing" code. Functions are called from using a function call after a function is defined. They receive data from the program using "arguments" and can return data to the program.

Function syntax:

```
def <function name>(arguments):
        print(arguments)
        return 0
```

Create a new program called function.py

```
# Print your name 6 times
print("Rick")
print("Rick")
print("Rick")
print("Rick")
print("Rick")
print("Rick")
```

```
def print_name(name):
    for i in range(0,6):
        print(name)


print_name("Rick")
```

Console Output:

```
Rick
Rick
Rick
Rick
Rick
Rick
Rick
Rick
Rick
Rick
Rick
Rick
```

`__main__` Function

`__main__` is a special type in Python called a "magic method". I use this method to define what to do if a program or module is run as the main program. It works well for testing purposes and setting up a large program main file.

Create a new program called test_main.py (note that it does not need to be named main.py)

```
# Print your name 6 times
def print_name(name):
    for i in range(0,6):
        print(name)
```

```
# Program main entry point
if __name__ == "__main__":
    print_name("Rick")
```

Console Output

```
Rick
Rick
Rick
Rick
Rick
Rick
```

This allows the program to run as a main file or as a module. We will discuss modules later in the tutorial.

List Data Structure

The list data structure is a collection of data. It is a very powerful feature in Python because the list can consist of any data type and each element can be a different data type. It is like a dynamically sized array but with any data type.

Define an empty list

```
a_list = []
```

Define a list of strings

```
a_list = ["Hello", "Python", "World"]
```

Define a list of data of different types

```
a_list = [1, 53.999, "Stuff"]
```

Lists have many pre-defined functions or methods available, let's write a program to demonstrate some of them.

Write a new program called test_list.py

```
# Define a list
a_list = [10, 25.2, "Hey"]

# Access the list elements
print("a_list[0]: ", a_list[0])
print("a_list[1]: ", a_list[1])
print("a_list[2]: ", a_list[2])

# Iterate over a list
for item in a_list:
    print(item)

# Add a boolean item to the end of the list
a_list.append(True)
print(a_list)

# Print the length of the list
print("List length:", len(a_list))

# Create a list of lists
list_of_lists = [
    [1, 2, 3],
    ["1", "2", "3"],
    [True, False, True]
]
print(list_of_lists)

# Access the second item in the second list
print(list_of_lists[1][1])

# Clear a list
a_list.clear()
print("a_list after clear: ", a_list)
```

Console Output

```
a_list[0]:  10
a_list[1]:  25.2
a_list[2]:  Hey
10
25.2
Hey
[10, 25.2, 'Hey', True]
List length: 4
[[1, 2, 3], ['1', '2', '3'], [True, False, True]]
2
a_list after clear:  []
```

We will use lists extensively in the projects as you will see.

Classes and Objects

Python supports object-oriented programming (OOP) as well as procedural and functional programming. As we will be using OOP extensively in all projects we need to understand classes, objects, inheritance, abstraction, encapsulation, and polymorphism. The process of writing OOP programs includes:

- Define a class
- Instantiate (create) an object from the class
- Use the object like a custom data type in the program

Let's write a program to demonstrate basic OOP called test_oop.py

```python
# Define a class
class Shape:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.loc = (x, y)

        print("Shape Class initialized")

    def draw(self):
        print("Drawing shape at x,y:", self.loc)
```

```
class Rectangle(Shape):
    def __init__(self, x, y):
        super().__init__(x, y)

        print("Rectangle Class initialized")

    def draw(self):
        print("Drawing rectangle at x,y:", self.loc)


# Instantiate the class object called rect
rect = Rectangle(10,10)

# Use the rect object
rect.draw()
```

Console Output

```
Shape Class initialized
Rectangle Class initialized
Drawing rectangle at x,y: (10, 10)
```

We will discuss OOP in more detail as we dive into the projects.

Multithreading

Most modern desktop computers have multiple core processors which is like having more than one computer in a single microprocessor. Python has a `threading` library which allows you to setup a separate "thread" than the main program thread. One way that multithreading can be used is when we need a continuous operation that if implemented in the main thread would greatly impact the performance of the program.

Lets create a program that toggles the state of a variable between True and False at a known rate or frequency.

First we need to import the threading and time libraries.

```
import threading
import time
```

Next we setup a global variable called `state` and set it to False.

```
import threading
import time

state = False
```

We define a function called `background_calculation()` which sets the timer sleep() function to 1 second. When the timer has finished the function toggles the state variable to True or False depending on its current state. Finally, we recursively call the `background_calculation()` function and the process starts over.

```
def background_calculation():
    # set the time
    time.sleep(1)

    # Toggle and print state
    global state
    state = not state
    print(state)
    background_calculation()
```

In the main entry point to the program, we define a thread and set the target to the `background_calculation()` function. Next we start the thread which runs the function in a new thread. The thread runs continuously, in this case.

```
import threading
import time

state = False


def background_calculation():
    # set the time
    time.sleep(1)
    # Toggle and print state
    global state
    state = not state
    print(state)
    background_calculation()


def main():
    thread = threading.Thread(target=background_calculation)
    thread.start()


if __name__ == '__main__':
    main()
```

Console Output

```
True
False
True
False
True
False
True
False
True
False
True
```

Modules

In large, advanced programming projects, the number of lines of code can become large and difficult to mange. At this point in the project development, it is a good idea to break the project into modules that are easier to maintain and manage. Modular programming also enables reusable code that can be shared with other projects, programmers, and teams. The Python module and packaging system can be difficult to master so for this basic tutorial we will create a single module file and use "import" to load the module into the main file.

Create a module file called shape.py in the same directory with the other files

```python
# Define a class
class Shape:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.loc = (x, y)

        print("Shape Class initialized")

    def draw(self):
        print("Drawing shape at x,y:", self.loc)


class Rectangle(Shape):
    def __init__(self, x, y):
        super().__init__(x, y)

        print("Rectangle Class initialized")

    def draw(self):
        print("Drawing rectangle at x,y:", self.loc)
```

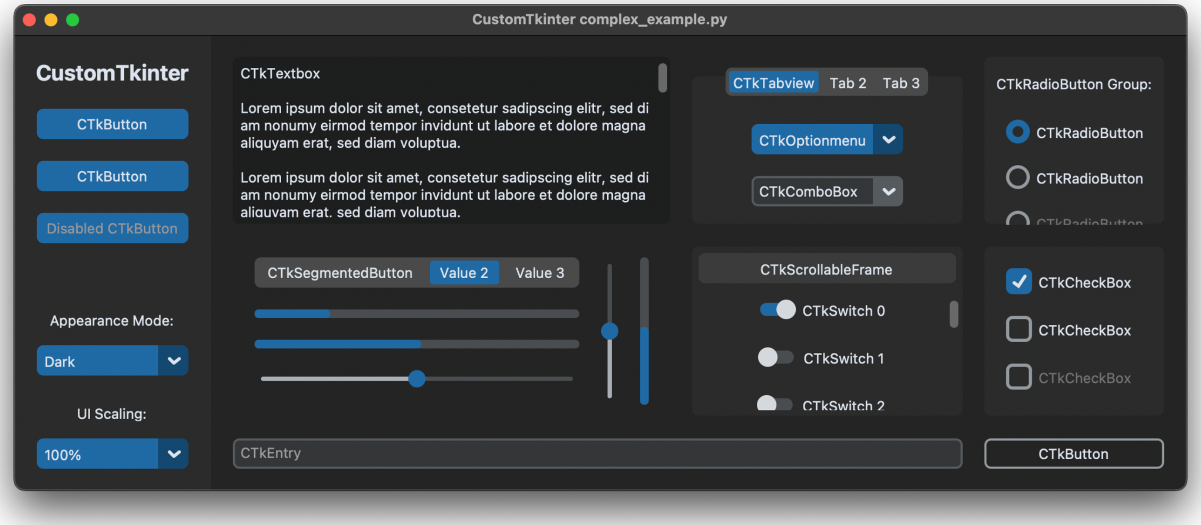
Now, create another file called test_shape.py

```python
from shape import Rectangle


# Instantiate the class object called rect
rect = Rectangle(10,10)

# Use the rect object
rect.draw()
```

Note that the shape classes are defined in a module or library and are "imported" into the main test shape file.

GUI Programming

Console programs such as those created in this tutorial are not very interesting. Most if not all modern programs provide a graphic user interface (GUI) that is user-friendly. One such GUI library is tkinter which ships with the Python installation, however the tkinter graphics are dated. I will use a modern version of tkinter called CustomTkinter created by Tom Schimanski. As you can see from the image below taken from the documentation, it is much nicer.



Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

As we will be using CustomTkinter for all projects, let's write a simple GUI program using OOP that displays a button that when pressed prints a message to the console.

Create a new program called test_ctk.py

```python
import customtkinter as ctk


class App(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.geometry("400x150")
        self.title("Test CustomTkinter")

        self.button = ctk.CTkButton(self, text="my button",
command=self.button_callbck)
        self.button.pack(padx=20, pady=20)

    def button_callbck(self):
        print("button clicked")
```

```
app = App()
app.mainloop()
```

Before we can run the program, we need to install the CustomTkinter library, open the Terminal tab and enter:

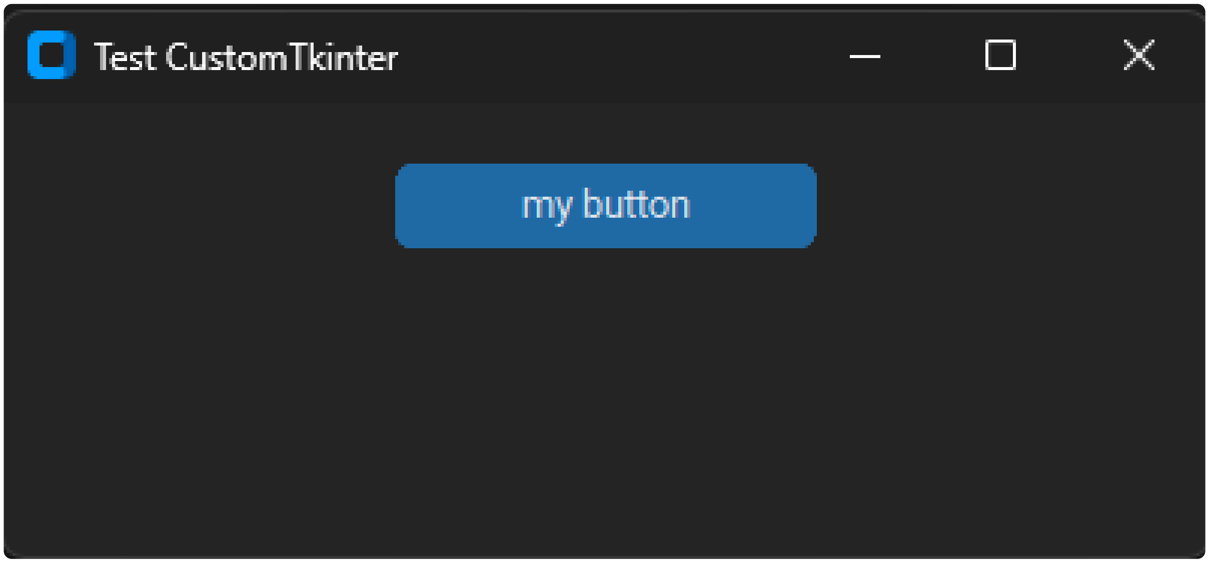
```
py -m pip install customtkinter
```

```
D:\EETools\Python3Tutorial>
D:\EETools\Python3Tutorial>py -m pip install customtkinter
Collecting customtkinter
  Using cached customtkinter-5.2.0-py3-none-any.whl (295 kB)
Collecting darkdetect
  Using cached darkdetect-0.8.0-py3-none-any.whl (9.0 kB)
Installing collected packages: darkdetect, customtkinter
Successfully installed customtkinter-5.2.0 darkdetect-0.8.0

[notice] A new release of pip available: 22.2.2 -> 23.3
[notice] To update, run: D:\Pythons\Python39\python.exe -m pip install --
upgrade pip
```

It is a good idea to update pip if a new release is available

```
D:\EETools\Python3Tutorial>python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in d:\pythons\python311\lib\site-packages
(23.2.1)
Collecting pip
  Obtaining dependency information for pip from
https://files.pythonhosted.org/packages/e0/63/b428aaca15fcd98c39b07ca7149e24bc1
4205ad0f1c80ba2b01835aedde1/pip-23.3-py3-none-any.whl.metadata
  Downloading pip-23.3-py3-none-any.whl.metadata (3.5 kB)
Using cached pip-23.3-py3-none-any.whl (2.1 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.2.1
    Uninstalling pip-23.2.1:
      Successfully uninstalled pip-23.2.1
Successfully installed pip-23.3
```

Now, run the program



Console Output after clicking the button

```
button clicked
```

Summary

This concludes our whirlwind tour of the basic Python tutorial. If you need additional tutorials and resources to learn python, please refer the Appendix on Python References. If you are like me, it is time to start creating useful programs that every Electrical Engineer uses in his daily routine. We will start with the beginner project on creating a scientific calculator in the next chapter.