## ABSTRACT

Weather forecasting is the prediction of  the state of the atmosphere for a given location using the application of science and technology. This includes temperature, rain, cloudiness, wind speed, and humidity. Weather warnings are a special kind of short-range forecast carried out for the protection of human life. Weather warnings are issued by the governments throughout the world for all kinds of threatening weather events including  tropical storms and tropical cyclones depending upon the location. The forecast may be short-range or Long-range. It is a very interesting and challenging task. This report  provides  a basic understanding of the purpose and scope of weather forecasts, the basic principles and the general models developed for forecasting.

# CHAPTER 1 INTRODUCTION TO WEATHER FORECAST

## 1.1Intorduction

Weather simply refers to the condition of air on the earth at a given place and time. It is a continuous, data-intensive, multidimensional, dynamic and chaotic process. These properties make weather forecasting is a formidable challenge. Forecasting is the process of estimation in unknown situations from the historical data. Weather forecasting is one of the most scientifically and technologically challenging problems around the world in the last century. To make an accurate prediction is indeed, one of the major challenges that meteorologists are facing all over the world. Since ancient times, weather prediction has been one of the most interesting and fascinating domains. Scientists have tried to forecast meteorological characteristics using a number of methods, some of these methods being more accurate than others.

Knowledge of meteorology forms the basis of scientific weather forecasting, which revolves around predicting the state of the atmosphere for a given location. Weather forecasting as practiced by humans is an example of having to make judgments in the presence of uncertainty. Weather forecasts are often made by collecting quantitative data about the current state of the atmosphere and using scientific understanding of atmospheric processes to project how the atmosphere will evolve in future. Over the last few years the necessity of increasing knowledge about the cognitive process in weather forecasting has been recognized. For human practitioners, forecasting the weather becomes a task for which the details can be uniquely personal, although most human forecasters use approaches based on the science of meteorology in common to deal with the challenges of the task.

Weather forecasting entails predicting how the present state of the atmosphere will change. Present weather conditions are obtained by ground observations, observations from ships, observation from aircraft, radio sounds, doppler radar and satellites. This information is sent to meteorological centers where the data are collected, analyzed and made into a variety of charts, maps and graphs. Modern high-speed computers transfer the many thousands of observations onto surface and upper-air maps.

Weather forecasts provide critical information about future weather. There are various techniques involved in weather forecasting, from relatively simple observation of the sky to highly complex computerized mathematical models. Weather prediction could be one day/one week or a few months ahead. The accuracy of weather forecasts however, falls significantly beyond a week. Weather forecasting remains a complex business, due to its chaotic and unpredictable nature. It remains a process that is neither wholly science nor wholly art. It is known that persons with little or no formal training can develop considerable forecasting skill. For example, farmers often are quite capable of making their own short term forecasts of those meteorological factors that directly influence their livelihood, and a similar statement can be made about pilots, fishermen, mountain climbers, etc. Weather phenomena, usually of a complex nature, have a direct impact on the safety and/or economic stability of such persons. Accurate weather forecast models are important to third world countries, where the entire agriculture depends upon weather. It is thus a major concern to identify any trends for weather parameters to deviate from its periodicity, which would disrupt the economy of the country. This fear has been aggravated due to threat by the global warming and green house effect. The impact of extreme weather phenomena on society is growing more and more costly, causing infrastructure damage, injury and the loss of life.

As practiced by the professionally trained meteorologist, weather forecasting today is a highly developed skill that is grounded in scientific principle and method and that makes use of advanced technological tools. The notable improvement in forecast accuracy that has been achieved since 1950 is a direct outgrowth of technological developments, basic and applied research, and the application of new knowledge and methods by weather forecasters. High-speed computers, meteorological satellites, and weather radars are tools that have played major roles in improving weather forecasts.

Several other factors have contributed significantly to this increase in forecasting accuracy. One is the development of statistical methods for enhancing the scope and accuracy of model predictions. Another is the improved observational capability afforded by meteorological satellites. A third primary reason for the increase in accuracy is the continued improvement of the initial conditions prepared for the forecast models. Statistical methods allow a wider variety of meteorological elements to be predicted than do the models alone, and they tailor the geographically less precise model forecasts to specific locations. Satellites now provide the capability for nearly continuous viewing and remote sensing of the atmosphere on a global scale. The improvement in initial conditions is the result of an increased number of observations and better use of the observations in computational techniques.

# CHAPTER 2 HARDWARE AND SOFTWARE SPECIFICATION

## 2.1 Hardware Requirement

| Hardware | Specification |
|---|---|
| RAM | 8 GB |
| Processor | Ryzen 5 |

## 2.2 Software Requirement

| Software | Specification |
|---|---|
| Operating System | Window 11 |
| IDE | Python 3 |

## 2.3 Technology

Python language use for creating weather forecasting project.

Tkinter library used for creating weather forecasting GUI.

Open weather map api use for taking current weather condition.

# CHAPTER 3 INTRODUCTION TO PYTHON

## 3.1What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

## 3.2What can Python do?

Python can be used on a server to create web applications and also python can be used alongside software to create workflows..It can connect to database systems and also can read and modify files.
Also it is used to handle big data and perform complex mathematics and used for rapid prototyping, or for production-ready software development.

## 3.3Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

**Good to know**

The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, PyCharm, NetBeans or Eclipse which are particularly useful when managing larger collections of Python files.

## 3.4Python Syntax compared to other programming languages

Python was designed for readability, and has some similarities to the English language with influence from mathematics. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses .Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

**Example**

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

## 3.5Features in Python

There are many features in Python, some of which are discussed below as follows:

**1. Free and Open Source**

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

**2. Easy to code**

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

**3. Easy to Read**

As you will see, learning Python is quite simple. As was already established, Python's syntax is straight forward. The code block is defined by the indentations rather than by semicolons or brackets.

**4. Object-Oriented Language**

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

**5. GUI Programming Support**

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

**6. High-Level Language**

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

**7. Extensible feature**

Python is an Extensible language. We can write some Python code into C or C++ language and also, we can compile that code in C/C++ language.

**8. Easy to Debug**

Excellent information for mistake tracing. You will be able to quickly identify and correct most of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

**9. Python is a Portable language**

Python language is also a portable language. For example, if we have Python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

**10. Python is an integrated language**

Python is also an integrated language because we can easily integrate Python with other languages like C, C++, etc.

# CHAPTER 4 APPLICATION OF PYTHON IN VARIOUS DISCIPLINES

Python is known for its general-purpose nature that makes it applicable in almost every domain of software development. Python makes its presence in every emerging field. It is the fastest-growing programming language and can develop any application.

**1) Web Applications**

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser, etc. Python provides many useful frameworks, and these are given below:

- Django and Pyramid framework(Use for heavy applications)
- Flask and Bottle (Micro-framework)
- Plone and Django CMS (Advance Content management)

**2) Desktop GUI Applications**

The GUI stands for the Graphical User Interface, which provides a smooth interaction to any application. Python provides a **Tk GUI library** to develop a user interface. Some popular GUI libraries are given below.

- Tkinter or Tk
- wxWidgetM
- Kivy (used for writing multitouch applications)

**3) Console-based Application**

Console-based applications run from the command-line or shell. These applications are computer program which are used commands to execute. This kind of application was more popular in the old generation of computers. Python can develop this kind of application very effectively. It is famous for having REPL, which means **the Read-Eval-Print Loop** that makes it the most suitable language for the command-line applications.

**4) Software Development**

Python is useful for the software development process. It works as a support language and can be used to build control and management, testing, etc.

- **SCons** is used to build control.
- **Buildout** and **Apache** Gumps are used for automated continuous compilation and testing.
- **Round** or **Trac** for bug tracking and project management.

**5) Scientific and Numeric**

This is the era of Artificial intelligence where the machine can perform the task the same as the human. Python language is the most suitable language for Artificial intelligence or machine learning. It consists of many scientific and mathematical libraries, which makes easy to solve complex calculations. Implementing machine learning algorithms require complex mathematical calculation. Python has many libraries for scientific and numeric such as NumPy, Pandas, SciPy, Scikit-learn, etc. If you have some basic knowledge of Python, you need to import libraries on the top of the code. Few popular frameworks of machine libraries are given below.

- SciPy
- Scikit-learn
- NumPy
- Pandas
- Matplotlib

# CHAPTER 5 GETTING STARTED

## 5.1 Python Install

Many PCs and Macs will have python already installed. To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

```
C:\Users\Your Name>python --version
```

To check if you have python installed on a Linux or Mac, then on Linux open the command line or on Mac open the Terminal and type:

```
Python --versionpython --version
```

If you find that you do not have Python installed on your computer, then you can download it for free from the following website: https://www.python.org/

**Python QuickStart**

Python is an interpreted programming Language; this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed. The way to run a python file is like this on the command line:

```
C:\Users\Your Name>python helloworld.py
```

Where "helloworld.py" is the name of your python file.

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

helloworld.py

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

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:

```
C:\Users\Your Name>python helloworld.py
```

The output should read:

```
Hello, World!
```

Congratulations, you have written and executed your first Python program.

**The Python Command Line**

To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

Type the following on the Windows, Mac or Linux command line:

```
C:\Users\Your Name>python
```

Or, if the "python" command did not work, you can try "py":

```
C:\Users\Your Name>py
Python 3.6.4 (v3.6.4: d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
```

Which will write "Hello, World!" in the command line:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4: d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
```

Whenever you are done in the python command line, you can simply type the following to quit the python command line interface

```
exit()
```

# CHAPTER 6 BASICS OF PROGRAMMING IN PYTHON

## 6.1Python Variables

**Variables**

Variables are containers for storing data values.

**Creating Variables**

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

**Example**

```
x= 5
y=       "John"
print(x)
print(y)
```

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

**Example**

```
x= 4      #x is of type int
x = "Sally" # x is now of type string
print(x)
```

**Casting**

If you want to specify the data type of a variable, this can be done with casting.

**Example**

```
x= str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3) # z will be 3.0
```

## 6.2Python Data Types

**Built-in Data Types**

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

**Table 6.1 Data Types**

| | |
|---|---|
| Text Type: | Str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | Dict |
| Set Types: | set, frozeset |
| Boolean Type: | Bool |
| Binary Types: | bytes,byte,array,memorview |
| None Type: | None Type |

## 6.3Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

**If**
An "if statement" is written by using the if keyword.

**Example**

If statement:

```
a= 33
b= 200
if b>a:
  print("b is greater than a")
```

**Indentation**
Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

**Example**

If statement, without indentation (will raise an error):

```
a= 33
b= 200
if b>a:
print("b is greater than a") # you will get an error
```

**Elif**
The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

**Example**

```
a= 33
b= 33
if b>a:
  print("b is greater than a")

elif a==b:
  print("a and b are equal")
```

**Else**
The else keyword catches anything which isn't caught by the preceding conditions.

**Example**

```
a= 200
b= 33
```

```
if b>a:
  print("b is greater than a")

elif a==b:
  print("a and b are equal")

else:
  print("a is greater than b")
```

You can also have an else without the elif:

**Example**

```
a= 200
b= 33
if b>a:
  print("b is greater than a")

else:
  print("b is not greater than a")
```

**Nested If**
You can have if statements inside if statements, this is called *nested* if statements.

**Example**

```
x= 41
if x> 10:
  print("Above ten,")

  if x> 20:
    print("and also above 20!")

  else:
    print("but not above 20.")
```

**6.4Python Loops**

Python has two primitive loop commands:

- while loops
- for loops

**The while Loop:**

With the while loop we can execute a set of statements as long as a condition is true.

**Example**

Print i as long as i is less than 6:

```
i= 1
while i< 6:
  print(i)
  i += 1
```

**Note:** remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1

**The break Statement:**

With the break statement we can stop the loop even if the while condition is true:

**Example:** Exit the loop when i is 3:

```
i= 1
while i< 6:
  print(i)
  if i==3:
    break
  i += 1
```

**The continue Statement:**
With the continue statement we can stop the current iteration, and continue with the next

**Example**

Continue to the next iteration if i is 3:

```
i= 0
while i< 6:
  i+= 1
  if i== 3:
    continue
  print(i)
```

**Python For Loops**
A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

**Example**

Print each fruit in a fruit list:

```
fruits=["apple",   "banana",   "cherry"]
for x in fruits:
  print(x)
```

The for loop does not require an indexing variable to set beforehand.

**Looping Through a String**
Even strings are iterable objects, they contain a sequence of characters:

**Example**

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

**The range() Function**
To loop through a set of code a specified number of times, we can use the range() function,
The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

**Example**

Using the range() function:

```
for x in range(4):

  print(x)
```

```
Output:0
        1
        2
        3
```

**Nested Loops:**
A nested loop is a loop inside a loop.
The "inner loop" will be executed one time for each iteration of the "outer loop":

**Example**

Print each adjective for every fruit:

```
adj=["red",          "big",          "tasty"]
fruits=["apple", "banana", "cherry"]

for  x   in   adj:
  for y in fruits:
  print(x, y)
```

Output:

```
red   apple

red banana

red  cherry

big   apple

big banana

big  cherry

tasty apple

tasty    banana

tasty cherry
```

## 6.5Python Functions:

A function is a block of code which only runs when it is called.You can pass data, known as parameters, into a function.A function can return data as a result.

**Creating a Function**

In Python a function is defined using the def keyword:

**Example**

```
def my_function():
  print("Hello from a function")
```

**Calling a Function**

To call a function, use the function name followed by parenthesis:

**Example**

```
def my_function():
  print("Hello from a function")
```

Output:

my_function()

**Arguments**

Information can be passed into functions as arguments. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma. The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

**Example**

```
def my_function(fname):
  print(fname+          "Refsnes")
my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

*Arguments* are often shortened to *args* in Python documentations

# CHAPTER 7 DEVELOPING A GUI WITH TKINTER

## 7.1What is Tkinter?

**Tkinter** is a **standard library** in python used for creating **Graphical User Interface (GUI)** for Desktop Applications. With the help of **Tkinter** developing **desktop applications** is not a tough task.
The **primary GUI toolkit** we will be using is Tk, which is Python's default GUI library. We'll access Tk from its Python interface called **Tkinter** (short for **Tk interface**).

## 7.2Features of Tkinter

Here are important features of Tkinter:
Learn tkinter which consists of more than six hundred classes covering a range of features such as

- Graphical User Interfaces
- SQL Databases
- Web toolkits
- XML processing
- Networking

## 7.3How to install Tkinter?

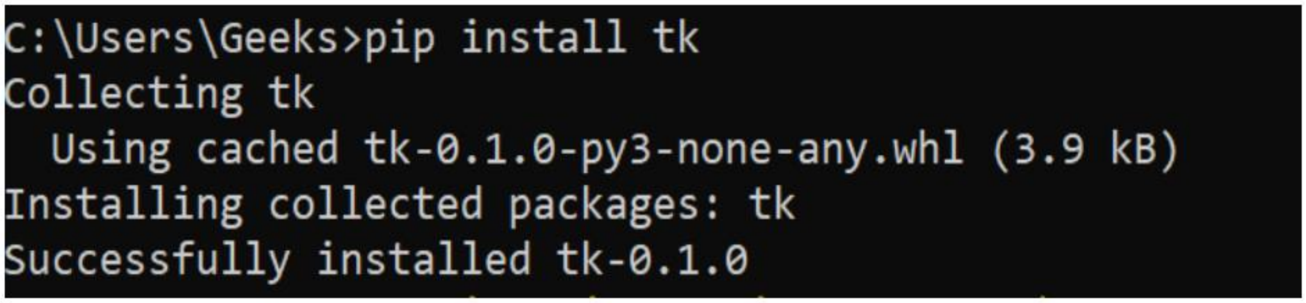
In this tkinter tutorial, we will see the two ways of installing PyQt:
To install tkinter,
**Step 1)** Open the Command Prompt or PowerShell in your Windows machine.

**Step 2)** Type in the following.

pip install tk

**Step 3)** Installation successful.

This step in this tkinter tutorial will download the tkinter package and install it on your system.



```
C:\Users\Geeks>pip install tk
Collecting tk
  Using cached tk-0.1.0-py3-none-any.whl (3.9 kB)
Installing collected packages: tk
Successfully installed tk-0.1.0
```

**Fig.7.1 Tkinter Install**

## 7.4Basic Tkinter Concepts and Programs

Now that you have successfully installed Tkinter in your computer, you are ready to write Python GUI design applications.
Let’s start with a simple app in this Tkinte tutorial which will display an empty window on your screen.
**Program 1**

```
  import Tkinter
  top = Tkinter.Tk()
# Code to add widgets will go here...
  top.mainloop()
```

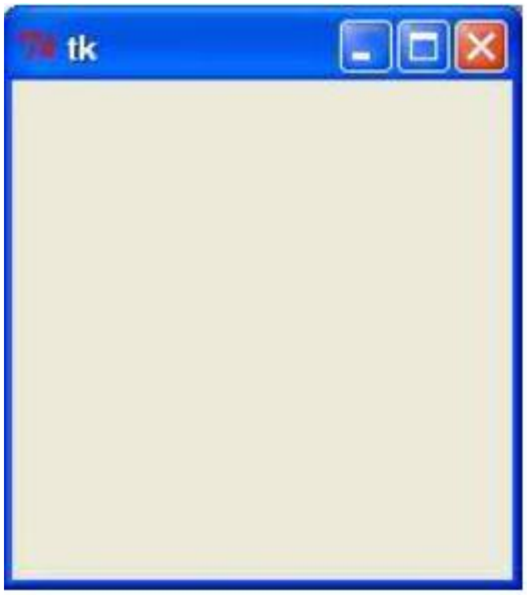


**Fig.7.2 Tkinter GUI**

## 7.5Tkinter Widgets

In general, **Widget** is an element of Graphical User Interface (GUI) that displays/illustrate information or gives a way for the user to interact with the OS. In **Tkinter** , **Widgets** are objects ; instances of classes that represent buttons, frames, and so on.
Each separate widget is a Python object. When creating a widget, you must pass its parent as a parameter to the widget creation function. The only exception is the “root” window, which is the top-level window that will contain everything else and it does not have a parent.

## 7.6 Types of Widgets

Frame:
Outlines the frame for your Tkinter window with a fixed size. Just like the human skeleton, a Tkinter window requires a frame to support it and give it structure.

```
from tkinter import *


root = Tk()
root.geometry("200x150")
frame = Frame(root)
frame.pack()


leftframe = Frame(root)
leftframe.pack(side=LEFT)


rightframe = Frame(root)
rightframe.pack(side=RIGHT)


label = Label(frame, text = "Hello world")
label.pack()


button1 = Button(leftframe, text = "Button1")
button1.pack(padx = 3, pady = 3)
button2 = Button(rightframe, text = "Button2")
button2.pack(padx = 3, pady = 3)
button3 = Button(leftframe, text = "Button3")
button3.pack(padx = 3, pady = 3)


root.title("Test")
root.mainloop()
```

Buttons:

The Python Tkinter **Button** is a standard Tkinter widget. A button is used as a way for the user to interact with the User interface. Once the button is clicked, an action is triggered by the program.

```
from tkinter import *

def set():
    print("Hello World")

root = Tk()
root.geometry("200x150")

frame = Frame(root)
frame.pack()
button = Button(frame, text = "Button1", command = set,
                fg = "red", font = "Verdana 14 underline",
                bd = 2, bg = "light blue", relief = "groove")
button.pack(pady = 5)

root.mainloop()
```

Entry:
A standard Tkinter widget used to take input from the user through the user interface. A simple box is provided where the user can input text.

```
from tkinter import *

root = Tk()
root.geometry("200x150")

frame = Frame(root)
frame.pack()

my_entry = Entry(frame, width = 20)
my_entry.insert(0,'Username')
my_entry.pack(padx = 5, pady = 5)

my_entry2 = Entry(frame, width = 15)
my_entry2.insert(0,'password')
my_entry2.pack(padx = 5, pady = 5)

root.mainloop()
```

Check Button:

A check button is a Tkinter GUI widget that presents to the user a set of predefined options. The user may select more than one option.

```
from tkinter import *

root = Tk()
root.geometry("200x150")

frame = Frame(root)
frame.pack()

Var1 = IntVar()
Var2 = IntVar()

ChkBttn = Checkbutton(frame, width = 15, variable = Var1)
ChkBttn.pack(padx = 5, pady = 5)

ChkBttn2 = Checkbutton(frame, width = 15, variable = Var2)
ChkBttn2.pack(padx = 5, pady = 5)
```

Radio Button:

A radio button is a Tkinter GUI widget that allows the user to choose only one of a predefined set of mutually exclusive options.

```
from tkinter import *


root = Tk()
root.geometry("200x150")
frame = Frame(root)
frame.pack()

Var1 = StringVar()

RBttn = Radiobutton(frame, text = "Option1", variable = Var1,
                    value = 1)
RBttn.pack(padx = 5, pady = 5)

RBttn2 = Radiobutton(frame, text = "Option2", variable = Var1,
                     value = 2)
RBttn2.pack(padx = 5, pady = 5)

root.mainloop()
```

Label:
A Tkinter widget used to display simple lines of text on a GUI. Also very versatile, and can even be used to display images.

```
from tkinter import *

root = Tk()
root.geometry("200x150")

frame = Frame(root)
frame.pack()

var = StringVar()
var.set("Hello World")

label = Label(frame, textvariable = var )
label.pack()

root.mainloop()
```

Menu:
The Tkinter Menu widget is used to create various types of **menus** in the GUI such as top-level menus, which are displayed right under the title bar of the parent window.

This is fairly complex widget, but essential in creating a modern and powerful GUI.

```
from tkinter import *


def save():
    #Code to be written
    pass


def load():
    #Code to be written
    pass


root = Tk()
root.geometry("200x150")
frame = Frame(root)
frame.pack()

mainmenu = Menu(frame)
mainmenu.add_command(label = "Save", command= save)
mainmenu.add_command(label = "Load", command= load)
mainmenu.add_command(label = "Exit", command= root.destroy)

root.config(menu = mainmenu)

root.mainloop()
```

ComboBox:
A special extension of Tkinter, the `ttk` module brings forward this widget. A combobox presents a drop down list of options and displays them one at a time. Has a more modern approach than other similar widgets.

```
from tkinter import *
from tkinter import ttk


root = Tk()
root.geometry("200x150")


frame = Frame(root)
frame.pack()


vlist = ["Option1", "Option2", "Option3",
          "Option4", "Option5"]


Combo = ttk.Combobox(frame, values = vlist)
Combo.set("Pick an Option")
Combo.pack(padx = 5, pady = 5)


root.mainloop()
```

List Box:
Another Tkinter widget that is used to display a list of options for the user to select. Displays all of the options in one go. Furthermore, all options are in text format.

```
from tkinter import *


root = Tk()
root.geometry("200x220")
frame = Frame(root)
frame.pack()

label = Label(root,text = "A list of Grocery items.")
label.pack()

listbox = Listbox(root)

listbox.insert(1,"Bread")
listbox.insert(2, "Milk")
listbox.insert(3, "Meat")
listbox.insert(4, "Cheese")
listbox.insert(5, "Vegetables")

listbox.pack()
root.mainloop()
```

Toplevel:
A widget in Tkinter that allows for the easy spawning of new Tkinter Windows. Toplevel is a better alternative to spawning extra tkinter windows by using `tk()`.

```
from tkinter import *


def NewWindow():
    window = Toplevel()
    window.geometry('150x150')
    newlabel = Label(window, text = "Settings Window")
    newlabel.pack()



root = Tk()
root.geometry('200x200')


myframe = Frame(root)
myframe.pack()


mybutton = Button(myframe, text = "Settings", command = NewWindow)
mybutton.pack(pady = 10)


root.mainloop()
```

Scrollbar:
A useful widget in GUI’s, which allows you to scroll in a Tkinter window or enable scroll for certain widgets. Typically used when you’re limited in space for your Tkinter window, but want more space for the widget (e.g Canvas).

```
from tkinter import *


root = Tk()
root.geometry("200x250")


mylabel = Label(root, text ='Scrollbars', font = "30")
mylabel.pack()


myscroll = Scrollbar(root)
myscroll.pack(side = RIGHT, fill = Y)


mylist = Listbox(root, yscrollcommand = myscroll.set )
for line in range(1, 100):
    mylist.insert(END, "Number " + str(line))
mylist.pack(side = LEFT, fill = BOTH )


myscroll.config(command = mylist.yview)


root.mainloop()
```

Scale:

The **Tkinter Scale** widget is used to implement a graphical slider to the User interface giving the user the option of picking through a range of values. The Maximum and minimum values on the Scale can be set the programmer.

```
from tkinter import *


root = Tk()
root.geometry("200x200")
frame = Frame(root)
frame.pack()


Scala = Scale(frame, from_=0, to=10)
Scala.pack(padx=5, pady=5)


Scala2 = Scale(frame, from_=0, to=10, orient=HORIZONTAL)
Scala2.pack(padx=5, pady=5)


root.mainloop()
```

Canvas:

One of the more advanced Tkinter widgets. As the name suggests, it’s used to draw graphs and plots on. You can even display images on a Canvas. It’s like a drawing board on which you can paint or draw anything.

```
from tkinter import *

root = Tk()

frame=Frame(root,width=300,height=300)
frame.pack(expand = True, fill=BOTH)

canvas = Canvas(frame,bg='white', width = 300,height = 300)

coordinates = 20, 50, 210, 230
arc = canvas.create_arc(coordinates, start=0, extent=250, fill="blue")
arc = canvas.create_arc(coordinates, start=250, extent=50, fill="red")
arc = canvas.create_arc(coordinates, start=300, extent=60, fill="yellow")

canvas.pack(expand = True, fill = BOTH)

root.mainloop()
```

Menu Button:
A combination of both the **button** and **menu** widget, this button widget displays a drop down menu with a list of options once clicked. This widget is unique because it's able to incorporate aspects of both **check buttons** and **radio buttons** into it.

```
from tkinter import *

root = Tk()
root.geometry("200x150")
frame = Frame(root)
frame.pack()

MenuBttn = Menubutton(frame, text = "Favourite food", relief = RAISED)

Var1 = IntVar()
Var2 = IntVar()
Var3 = IntVar()

Menu1 = Menu(MenuBttn, tearoff = 0)

Menu1.add_checkbutton(label = "Pizza", variable = Var1)
Menu1.add_checkbutton(label = "Cheese Burger", variable = Var2)
Menu1.add_checkbutton(label = "Salad", variable = Var3)

MenuBttn["menu"] = Menu1

MenuBttn.pack()
root.mainloop()
```

TABLE 7.1 Types Of Widgets

| SN | Widget | Description |
|---|---|---|
| 1 | Button | The Button is used to add various kinds of buttons to the python application. |
| 2 | Canvas | The canvas widget is used to draw the canvas on the window. |
| 3 | Checkbutton | The Checkbutton is used to display the CheckButton on the window. |
| 4 | Entry | The entry widget is used to display the single-line text field to the user. |
| 5 | Frame | It can be defined as a container to which, another widget can be added and organized. |
| 6 | Label | A label is a text used to display some message or information about the other widgets. |
| 7 | ListBox | The ListBox widget is used to display a list of options to the user. |
| 8 | Menubutton | The Menubutton is used to display the menu items to the user. |
| 9 | Menu | It is used to add menu items to the user. |
| 10 | Message | The Message widget is used to display the message-box to the user. |
| 11 | Radiobutton | In Radiobutton user is provided with various options and the user can select only one option among them. |
| 12 | Scale | It is used to provide the slider to the user. |
| 13 | Scrollbar | It provides the scrollbar to the user so that the user can scroll the window up and down. |
| 14 | Text | It is different from Entry because it provides a multi-line text field to the user so that the user can write the text and edit the text inside it. |
| 14 | Toplevel | It is used to create a separate window container. |
| 15 | Spinbox | It is an entry widget used to select from options of values. |
| 16 | PanedWindow | It is like a container widget that contains horizontal or vertical panes. |
| 17 | LabelFrame | A LabelFrame is a container widget that acts as the container |
| 18 | MessageBox | This module is used to display the message-box in the desktop based applications |

# CHAPTER 8 CODING & IMPLEMENTATION

## 8.1 Code to create weather forecasting gui app.

```
from tkinter import *
import tkinter as tk
from geopy.geocoders import Nominatim
from tkinter import ttk, messagebox
from timezonefinder import TimezoneFinder
from datetime import *
import requests
import pytz
from PIL import Image, ImageTk

root = Tk()
root.title("weather forecast")
root.geometry("890x470+300+200")
root.configure(bg="#57adff")
root.resizable(False, False)

def getWeather():
    city = textfeild.get()

    geolocator = Nominatim(user_agent="geoapiExercises")
    location = geolocator.geocode(city)
    obj = TimezoneFinder()

    result = obj.timezone_at(lng=location.longitude, lat=location.latitude)

    timezone.config(text=result)
    long_lat.config(text=f"{round(location.latitude,4)}°N,{round(location.longitude,4)}°E")

    home = pytz.timezone(result)
    local_time = datetime.now(home)
    current_time = local_time.strftime("%I:%M %p")
    clock.config(text=current_time)

    ##weather
    api = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid=06c921750b9a82d8f5d1294e1586276f"

    # api = f"https://api.open-
meteo.com/v1/forecast?latitude={location.latitude}&longitude={location.longitude}&hourly=temperature_2m"
    json_data = requests.get(api).json()
    print(json_data)
    condition = json_data['weather'][0]['main']
    temp = int(json_data['main']['temp'] - 273.15)
    pressure = json_data['main']['pressure']
    humidity = json_data['main']['humidity']
    wind = json_data['wind']['speed']

    t.config(text=(temp, "°C"))
    h.config(text=(humidity, "%"))
    p.config(text=(pressure, "hPa"))
    w.config(text=(wind, "km/h"))
    c.config(text=condition)
    # Forecast
    forecast_api =
f"https://api.openweathermap.org/data/2.5/forecast?q={city}&appid=06c921750b9a82d8f5d1294e1586276f"
    forecast_data = requests.get(forecast_api).json()

    # Update forecast labels
    forecast = forecast_data['list'][:7]
    for i, day in enumerate(forecast):
        temperature = int(day['main']['temp'] - 273.15)
        description = day['weather'][0]['description']
        forecast_temperatures[i].config(text=f"{temperature}°C")
        forecast_descriptions[i].config(text=description)



    #first call
    firstdayimage = forecast_data['list'][0]['weather'][0]['icon']
    photo1 = ImageTk.PhotoImage(file=f"icon{firstdayimage}@2x.png")
    firstimage.config(image=photo1)

    #second call




    # days
    first = datetime.now()
    day1.config(text=first.strftime("%A"))

    second = first + timedelta(days=1)
    day2.config(text=second.strftime("%A"))

    third = first + timedelta(days=2)
    day3.config(text=third.strftime("%A"))

    fourth = first + timedelta(days=3)
    day4.config(text=fourth.strftime("%A"))

    fifth = first + timedelta(days=4)
    day5.config(text=fifth.strftime("%A"))

    sixth = first + timedelta(days=5)
    day6.config(text=sixth.strftime("%A"))

    seventh = first + timedelta(days=6)
    day7.config(text=seventh.strftime("%A"))

##icons
image_icon = PhotoImage(file="Images/logo.png")
root.iconphoto(False, image_icon)

Round_box = PhotoImage(file="Images/Rounded Rectangle 1.png")
Label(root, image=Round_box, bg="#57adff").place(x=30, y=110)


#labels
label1 = Label(root, text="Temperature", font=("Helvetica", 11), fg="white", bg="#203243")
label1.place(x=50, y=120)
```

```
label1 = Label(root, text="Humidity", font=("Helvetica", 11), fg="white", bg="#203243")
label1.place(x=50, y=140)

label1 = Label(root, text="Pressure", font=("Helvetica", 11), fg="white", bg="#203243")
label1.place(x=50, y=160)

label1 = Label(root, text="Wind Speed", font=("Helvetica", 11), fg="white", bg="#203243")
label1.place(x=50, y=180)

label1 = Label(root, text="Condition", font=("Helvetica", 11), fg="white", bg="#203243")
label1.place(x=50, y=200)

##search box
Search_image = PhotoImage(file="Images/Rounded Rectangle 3.png")
myimage = Label(image=Search_image, bg="#57adff")
myimage.place(x=270, y=120)

weat_image = PhotoImage(file="Images/Layer 7.png")
weatherimage = Label(root, image=weat_image, bg="#203243")
weatherimage.place(x=290, y=127)

textfeild = tk.Entry(root, justify="center", width=15, font=("poppins", 25, "bold"), bg="#203243", border=0,
fg="white")
textfeild.place(x=370, y=130)
textfeild.focus()

Search_icon = PhotoImage(file="Images/Layer 6.png")
myimage_icon = Button(image=Search_icon, borderwidth=0, cursor="hand2", bg="#203243", command=getWeather)
myimage_icon.place(x=645, y=125)

##Bottom Box
frame = Frame(root, width=900, height=180, bg="#212120")
frame.pack(side="bottom")

# bottom boxes
firstbox = PhotoImage(file="Images/Rounded Rectangle 2.png")
secondbox = PhotoImage(file="Images/Rounded Rectangle 2 copy.png")

##Clock
clock = Label(root, font=("Helvetica", 30, "bold"), fg="white", bg="#57adff")
clock.place(x=30, y=20)

##timezone
timezone = Label(root, font=("helvetica", 14, "bold"), fg="white", bg="#57adff")
timezone.place(x=700, y=20)

long_lat = Label(root, font=("Helvetica", 14, "bold"), fg="white", bg="#57adff")
long_lat.place(x=700, y=50)


###THPWC
t = Label(root, font=("Helvetica", 11), fg="white", bg="#203243")
t.place(x=150, y=120)
h = Label(root, font=("Helvetica", 11), fg="white", bg="#203243")
h.place(x=150, y=140)
p = Label(root, font=("Helvetica", 11), fg="white", bg="#203243")
p.place(x=145, y=160)
```

```
w = Label(root, font=("Helvetica", 11), fg="white", bg="#203243")
w.place(x=145, y=180)
c = Label(root, font=("Helvetica", 11), fg="white", bg="#203243")
c.place(x=150, y=200)

# first cell
firstframe = Frame(root, width=230, height=132, bg="#282829")
firstframe.place(x=35, y=315)

day1 = Label(firstframe, font="arial 16", bg="#282829", fg="#fff", anchor='nw')
day1.place(x=5, y=5, relwidth=1, relheight=1)

firstimage=Label(firstframe, bg="#282829", fg="white")
firstimage.place(x=1, y=100)

# second cell
secondframe = Frame(root, width=70, height=115, bg="#282829")
secondframe.place(x=305, y=325)

day2 = Label(secondframe, font="arial 11", bg="#282829", fg="#fff", anchor='nw')
day2.place(x=5, y=5, relwidth=1, relheight=1)

secondimage=Label(secondframe, bg="#282829", fg="white")
secondimage.place(x=7,y=100)

# third cell
thirdframe = Frame(root, width=70, height=115, bg="#282829")
thirdframe.place(x=405, y=325)

day3 = Label(thirdframe, font="arial 11", bg="#282829", fg="#fff", anchor='nw')
day3.place(x=5, y=5, relwidth=1, relheight=1)

thirdimage=Label(thirdframe, bg="#282829", fg="white")
thirdimage.place(x=7,y=100)

# fourth cell
fourthframe = Frame(root, width=70, height=115, bg="#282829")
fourthframe.place(x=505, y=325)

day4 = Label(fourthframe, font="arial 11", bg="#282829", fg="#fff", anchor='nw')
day4.place(x=5, y=5, relwidth=1, relheight=1)

fourthimage= Label(fourthframe, bg="#282829", fg="white")
fourthimage.place(x=7,y=100)

# fifth cell
fifthframe = Frame(root, width=70, height=115, bg="#282829")
fifthframe.place(x=605, y=325)

day5 = Label(fifthframe, font="arial 11", bg="#282829", fg="#fff", anchor='nw')
day5.place(x=5, y=5, relwidth=1, relheight=1)

fifthimage = Label(fifthframe, bg="#282829", fg="white")
fifthimage.place(x=7,y=100)

# sixth cell
sixthframe = Frame(root, width=70, height=115, bg="#282829")
```

```
sixthframe.place(x=705, y=325)

day6 = Label(sixthframe, font="arial 11", bg="#282829", fg="#fff", anchor='nw')
day6.place(x=5, y=5, relwidth=1, relheight=1)

sixthimage=Label(sixthframe, bg="#282829", fg="white")
sixthimage.place(x=7,y=100)

# seventh cell
seventhframe = Frame(root, width=70, height=115, bg="#282829")
seventhframe.place(x=805, y=325)

day7 = Label(seventhframe, font="arial 11", bg="#282829", fg="#fff", anchor='nw')
day7.place(x=5, y=5, relwidth=1, relheight=1)

seventhimage=Label(seventhframe, bg="#282829",fg="white")
seventhimage.place(x=7,y=100)

# Forecast Labels
forecast_temperatures = [day1, day2, day3, day4, day5, day6, day7]
for temp_label in forecast_temperatures:
    temp_label.place(x=5, y=35, relwidth=1, relheight=0.6)

forecast_descriptions = [firstimage, secondimage, thirdimage, fourthimage, fifthimage, sixthimage, seventhimage]

root.mainloop()
```