SIX WEEKS INDUSTRIAL TRAINING REPORT

**PYTHON PROGRAMMING**

Submitted to: Mr. CHAKSHU GOEL

A project report submitted in partial fulfillment of the requirements for the award of

summer training at

Allsoft Solutions pvt. Ltd. Mohali (From 5th of June- 20th of July 2023)



**Under the Guidance of: Mr. Mayank Raghuwanshi Submitted by:**

**Name: Tamana Verma University Roll No: 211558 ECE Department**

# DECLARATION

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING SHAHEED BHAGAT SINGH STATE UNIVERSITY

FEROZEPUR

I Tamana Verma hereby declare that I have undertaken six weeks industrial training at **Allsoft Solutions Pvt. Limited MOHALI** During a period of 5th of June 2023 to 20th of July 2023 in partial fulfilment of requirements for the degree of B-Tech (Electronics and Communication Engineering) at Shaheed Bhagat Singh State University, Ferozepur. The work which is being presented in the training report is an authentic record of training work.

**Tamana Verma**

**211558**

# ACKNOWLEDGEMENT

Through this acknowledgment, I express my sincere gratitude to all those people who are associated with this project and are helping me with it to make it a worthwhile experience. First and foremost I would like to thank almighty for giving me courage to make this project .At the outset, I would like to propose a word of thanks for the people who gave me unending support and help in numerous ways.

Firstly, I express our thanks to Dr. Vishal Sharma (HOD ECE DEPTT) who give this opportunity to learn the subject in a partial approach who guided me and gave Me valuable suggestions regarding the project report. Secondly, I would like to thanks Mr. Chakshu Goel (project guide) and FACULTY MEMBERS OF ALLSOFTS SOLUTIONS who are giving me their support in completing the project. The atmosphere provided is full of gaining more and more knowledge and to keep enthusiastic nature. The teacher provide a lot of help in resolving my doubts and making this project successful in shorter time.

Lastly, I would also like to thank my parents and friends who help me a lot in finishing the project in limited time. I am making this project not only for marks but also to increase my knowledge. Thanks again to all who are helping me in completion of this project.

**COMPANY PROFILE**



Allsoft Solutions and Services Pvt Ltd, IBM Business Partner was started in 2000 having headquarters in the USA, with the establishment of three branches in India. Allsoft hasbeen committed to providing the highest quality and needs-based services to its clients, both locally and internationally. We offer IT consulting, Industrial Training Program in various technologies, Web Development and Cloud services as well. We believe in “Making of the era with professionals” in this highly competitive workplace. Our policy towards introducing new, on demand and emerging technology makes us competitive in the market as our goals are to provide the best possible value and to lead in our markets through service and innovation. Being one of the IBM trusted and certified business partner, we offer training in IBM technologies to provide students an exposure to the IBM tools also.

Allsoft offer’s:

Several Training Programs, choose with your interest

Network, Big Data, Cyber Security, AI, Ethical Hacking, IOT, Java, Web Developer, Android, Full Stack, ML, Cloud, Python.

100% live projects to provide a full hands-on presentation. Flexible batches available, weekdays & weekends.

Highly qualified IBM trainers with many years of industrial background VOICE:+7773070136

E-Mail: [info@allsoftsolutions.in](mailto:info@allsoftsolutions.in)

Website: https://[www.allsoftsolutions.in/IBM/MAIN](http://www.allsoftsolutions.in/IBM/MAIN)

# Introduction

Work is a major part of our course. It is a period in which we are introduced to the industrial environment or in other words we can say that industrial training is providedfor the familiarization with the industrial environment, with the advancement in computer applications.

# TRAINING

The objective of training work is to raise the level of performance in one or more of its aspects and this may be achieved by providing new knowledge and information relevant to a job. By teaching new trend, by imbuing an individual with new attitude, motives and other personalities characteristic. Often these technology are utilized with segments of a work force regardless of the existing performance level to operate efficiency.

Practical training is an important part of theoretical studies. It covers all that remains uncovered in the classroom i.e., without it our studies remain ineffective and incomplete. Also, it explores a student to an invaluable treasure of experience and offers an exposure to real management in a business organization. Also, it is a well-known fact that practical training plays a very important role in future building of an individual. Only gaining theoretical knowledge is just not sufficient for sure success in life, practical training is a must and I have been given an opportunity to gain practical experience at ALLSOFT SOLUTIONS, MOHALI. I avail this instance in a very satisfactory manner and think it will very beneficial for me in building my future.

INDEX

|  |  |  |
| --- | --- | --- |
| S no. | Table of content | Pages |
| 1. | Introduction to Python Language |  |
| 2. | Introduction to Tuples, Lists and Dictionaries |  |
| 3. | Loops in Python |  |
| 4. | Statements in Python |  |
| 5. | Python Operators |  |
| 6. | Function |  |
| 7. | Python Cheat Sheet |  |
| 8. | GUI Programming |  |
| 9. | Introduction to the project |  |
| 10. | Project Code |  |

1. **Introduction to Python Language**

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more

# efficiently.

* 1. **LANGUAGE FEATURES**

**(a)Interpreted**

1.There are no separate compilation and execution steps like C and C++. 2.Directly *run* the program from the source code.

1. Internally, Python converts the source code into an intermediate form called bytecodes which is then translated into native language of specific computer to run it.
2. No need to worry about linking and loading with libraries etc. **(b)Platform Independent**
3. Platform Independent Python programs can be developed and executed on multiple operating system platforms.
4. Python can be used on Linux, Windows, Macintosh, Solaris and many more. 3.Free and Open Source**;** Redistributable

**(c)High-level Language**

In Python, no need to take care about low-level details such as managing the memory used by the program.

1.Simple

1. Closer to English language Easy to Learn.
2. More emphasis on the solution to the problem rather than the syntax. 2.Embeddable

Python can be used within C/C++ program to give scripting capabilities for the program’s users.

3.Robust

Exceptional handling features

Memory management techniques in built **(d)Rich Library Support**

Python Standard Library is very vast.

Known as the “batteries included” philosophy of Python; It can help do various things Readable. involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, email, XML, HTML, WAV files, cryptography, GUI and many more.

* 1. **Features of Python**

Easy to learn.

Free.

Cross Platform.

Open Source.

Memory Management.

Large Standard Library, Exception Handling.

# What can you do with Python?

Web Development Machine learning Data Analysis Scripting

Game development Desktop Applications Selenium

**2 Introduction to Tuples, List, Dictionaries**

Tuples, List, &Dictionaries Variables are great at what they do – storing a piece of information, which doesn’t change over time?

Say, for example, the names of the months of the year.

Or may be a long list of information, that does change over time?

For these three problem, Python uses three different solution-Tuples, List and dictionaries:

* 1. **Lists:**

1.Collection of Items: A list in Python is a collection of items. These items can be of any data type, such as numbers, strings, or even other lists.

1. Mutable and Ordered: Lists are mutable, meaning you can change their contents by adding, removing, or modifying elements. They are also ordered, meaning the items in a list have a specific order, and this order is maintained.
2. Square Brackets: Lists are defined using square brackets []. To create a list, you enclose the items inside square brackets, separated by commas. For example: my\_list = [1, 2, 3, 'apple'].
3. Indexing and Slicing: Each item in a list has a unique index (position), starting from 0. You can access individual elements using their index, and you can also extract a subset of elements using slicing. For example: my\_list[0] would give the first element.
4. Common Operations: Lists support various operations like adding elements (append()), removing elements (remove()), finding the length (len()), and more. They provide a versatile way to store and manipulate collections of data in Python

# List Operations

**List Creation Code:**

my\_string\_list = ["apple", "banana", "orange"]

print(my\_string\_list)

Accessing List Elements:

fruits = ["apple", "banana", "orange", "grape", "kiwi"]

print("The first fruit is:", fruits[0])

Modifying List Elements:

numbers = [1, 2, 3, 4, 5]

numbers[0] = 10

print("Modified list after changing the first element:", numbers)

**List Slicing:**

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

subset = numbers[2:7] # Extract elements from index 2 (inclusive) to index 7 (exclusive)

print("Subset of numbers:", subset)

**List Length:**

my\_list = [1, 2, 3, 4, 5]

length = len(my\_list)

print("Length of the list:", length)

**Adding Elements to a List:**

my\_list = [1, 2, 3]

my\_list.append(4)

print(my\_list)

**Checking if an Element Exists in a List:**

my\_list = [1, 2, 3, 4, 5]

# Check if an element exists in the list using the 'in' keyword

if 3 in my\_list:

print("3 exists in the list.")

else:

print("3 does not exist in the list.")

# Check if an element does not exist in the list using the 'not in' keyword

if 6 not in my\_list:

print("6 does not exist in the list.")

else:

print("6 exists in the list."

# Tuples

Ordered Collection: Tuples are ordered collections of elements. This means that the order in which you define elements in a tuple is maintained.

Immutable: Once you create a tuple, you cannot change its elements. This property is known as immutability. You can't add, remove, or modify elements in a tuple after it's created.

Mixed Data Types: Tuples can contain elements of different data types, such as integers, floats, strings, etc. You can have a mix of data types within the same tuple.

Enclosed in Parentheses: Tuples are defined by enclosing the elements in parentheses (). For example: my\_tuple = (1, 2, 'hello').

Used for Unchanging Data: Due to their immutability, tuples are often used to represent unchanging data, such as coordinates, configuration settings, or fixed sets of values in a program.

# Tuple Operations:

**Tuple Creation:**

my\_tuple = (1, 2, 3, 4, 5)

**Accessing Tuple Elements:**

my\_tuple = (1, 2, 3, 4, 5)

print("The first element is:", my\_tuple[0])

print("The third element is:", my\_tuple[2])

print("The last element is:", my\_tuple[-1])

print("Slice of elements from index 1 to 3:", my\_tuple[1:4])

print("All elements except the first one:", my\_tuple[1:])

**Modifying Tuple Elements (Immutable):**

my\_tuple = (1, 2, 3, 4, 5)

# Attempt to modify an element (this will raise an error)

try:

my\_tuple[0] = 10

except TypeError as e:

print("Error:", e)

**Modifying Tuple Elements (Immutable):**

my\_tuple = (1, 2, 3, 4, 5)

# Attempt to modify an element (this will raise an error)

try:

my\_tuple[0] = 10

except TypeError as e:

print("Error:", e)

**Tuple Length:**

my\_tuple = (1, 2, 3, 4, 5)

# Find the length of the tuple

length = len(my\_tuple)

# Print the length

print("Length of the tuple:", length)

**Tuple Concatenation:**

tuple1 = (1, 2, 3)

tuple2 = (4, 5, 6)

# Concatenate the tuples

concatenated\_tuple = tuple1 + tuple2

# Print the concatenated tuple

print("Concatenated tuple:", concatenated\_tuple)

* 1. **Dictionary**

1. **Key-Value Pairs:**

A dictionary is a collection of key-value pairs. Think of it like a real-world dictionary where words (keys) have definitions (values).

1. **Syntax:**

Created using curly braces {}.

Example: my\_dict = {'name': 'John', 'age': 25}.

1. **Accessing Values:**

Retrieve values by specifying the key in square brackets. **Example:** name = my\_dict['name'] gives 'John'.

1. **Mutable and Dynamic:**

Dictionaries are mutable, meaning you can change their values. Easily add or remove key-value pairs.

**Example**: my\_dict['city'] = 'New York' adds a new key-value pair.

# Dictionary Operations

**Dictionary Creation:**

my\_dict = {"name": "John", "age": 30, "city": "New York"}

**Accessing Dictionary Elements:**

my\_dict = {"name": "John", "age": 30, "city": "New York"}

# Accessing elements by key

print("Name:", my\_dict["name"]) # Output: John

print("Age:", my\_dict["age"]) # Output: 30

print("City:", my\_dict["city"])

**Modifying Dictionary Elements:**

my\_dict = {"name": "John", "age": 30, "city": "New York"}

my\_dict["age"] = 35

print("Modified age:", my\_dict["age"]) # Output: 35

my\_dict["gender"] = "Male"

print("Added gender:", my\_dict["gender"]) # Output: Male

**Adding Elements to a Dictionary:**

my\_dict = {"name": "John", "age": 30}

my\_dict["city"] = "New York"

print("Updated dictionary:", my\_dict)

**Removing Elements from a Dictionary:**

my\_dict = {"name": "John", "age": 30, "city": "New York"}

# Remove a specific key-value pair

del my\_dict["age"]

print("Dictionary after removing 'age':", my\_dict)

**Checking if a Key Exists in a Dictionary:**

my\_dict = {"name": "John", "age": 30, "city": "New York"}

# Check if a key exists in the dictionary using the 'in' keyword

if "age" in my\_dict:

print("Age exists in the dictionary.")

else:

print("Age does not exist in the dictionary.")

# Check if a key does not exist in the dictionary using the 'not in' keyword

if "gender" not in my\_dict:

print("Gender does not exist in the dictionary.")

else:

print("Gender exists in the dictionary."

# Loops in Python

In Python, loops are used to repeatedly execute a block of code until a certain condition is met.

In Python, both for and while loops are used for repetitive execution of code, but they differ in their syntax and the way they control the flow of execution. Here's a breakdown of the differences between for and while loops in Python:

# For loop

A for loop is used to iterate over a sequence (such as a list, tuple, string, or range) or any other iterable object. The loop variable takes each value from the sequence, and the block of code inside the loop is executed for each iteration.

**Code**

numbers = [1, 2, 3, 4, 5]

# Iterate over each element in the list and print it

for number in numbers:

print(number)

**Output**

1

2

3

4

5

# While loop

A while loop is used to repeatedly execute a block of code as long as a given condition is true. The loop continues until the condition becomes false.

**Code**

count = 0

# Define the condition

while count < 5:

print("Count:", count)

count += 1

**Output**

Count: 0

Count: 1

Count: 2

Count: 3

Count: 4

# Nested loop

Nested loops involve using one loop inside another loop. The outer loop controls the iteration of the inner loop. For each iteration of the outer loop, the inner loop runs completely.

**Code**

fruits = ["apple", "banana", "cherry"]

# Define a list of colors

colors = ["red", "yellow", "pink"]

# Iterate over each fruit

for fruit in fruits:

# Iterate over each color

for color in colors:

print(fruit, color)

**Output**

apple red

apple yellow

apple pink

banana red

banana yellow

banana pink

cherry red

cherry yellow

cherry pink

**4 Statements in Python**

# Break Statement

The `break` statement is used to exit or terminate a loop prematurely.

**Code**

for num in range(5):

print(num)

if num == 2:

break

**Output**

0

1

2

# Continue Statement:

The `continue` statement allows you to skip certain iterations and move to the next iteration of the loop.

**Code**

for num in range(5):

if num == 2:

continue

print(num)

**Output**

0

1

3

# Pass Statement

The `pass` statement is a placeholder statement that does nothing. It is used when you need a statement syntactically, but you don't want it to do anything.

**Code**

def my\_function():

pass

# Define a class with no methods

class MyClass:

pass

# Use pass in a loop

for i in range(5):

if i == 3:

pass

else:

print(i)

# Python Operators

Operators are symbols or special characters that perform specific operations on values or variables in Python.

Arithmetic operators(\*, +, -, /, //, %, \*\*) Assignment operators(=, +=, -=) Comparison operators()

Logical operators() Membership operators() Identity operators ()

# Arithmetic operators

Arithmetic operators are used for performing mathematical calculations.

Arithmetic operators in Python are fundamental tools for performing mathematical calculations within programs. Python supports various arithmetic operators, including addition (+), subtraction (-), multiplication (\*), division (/), integer division (//), exponentiation (\*\*), and modulus (%). These operators allow for the manipulation of numeric data types such as integers and floating-point numbers. Addition and subtraction operators perform the respective mathematical operations on operands, while the multiplication operator computes the product of two numbers. Division can be performed with either the regular division operator, which returns a floating-point result, or the integer division operator, which returns the integer quotient. Exponentiation raises a number to a specified power, and the modulus operator returns the remainder of the division of two numbers. Together, these arithmetic operators provide the foundation for mathematical computations in Python, enabling developers to build powerful and flexible algorithms for a wide range of applications.

**Example**

# Addition

result\_addition = 10 + 5

print("Addition:", result\_addition) # Output: 15

# Subtraction

result\_subtraction = 10 - 5

print("Subtraction:", result\_subtraction) # Output: 5

# Multiplication

result\_multiplication = 10 \* 5

print("Multiplication:", result\_multiplication) # Output: 50

# Division

result\_division = 10 / 5

print("Division:", result\_division) # Output: 2.0

# Integer Division

result\_integer\_division = 10 // 5

print("Integer Division:", result\_integer\_division) # Output: 2

# Exponentiation

result\_exponentiation = 10 \*\* 2

print("Exponentiation:", result\_exponentiation) # Output: 100

# Modulus

result\_modulus = 10 % 3

print("Modulus:", result\_modulus) # Output: 1

# Comparison Operators:

Comparison operators are used to compare values and determine the relationship between them.

Comparison operators in Python are essential for evaluating conditions and making decisions within programs. These operators allow for the comparison of values and return a Boolean result (True or False) based on the comparison. Python supports various comparison operators, including equality (==), inequality (!=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=). These operators are used to compare two operands and determine the relationship between them. For example, the equality operator (==) checks if two values are equal, while the inequality operator (!=) checks if they are not equal. Similarly, the greater than (>) and less than (<) operators compare values to determine if one is greater than or less than the other. The greater than or equal to (>=) and less than or equal to (<=) operators evaluate whether one value is greater than or equal to or less than or equal to the other, respectively. Comparison operators are fundamental in conditional statements, loops, and other control structures, enabling programmers to create dynamic and responsive applications that react to different conditions based on the comparison results.

**Example**

# Equality

print(5 == 5) # Output: True

print(5 == 10) # Output: False

# Inequality

print(5 != 10) # Output: True

print(5 != 5) # Output: False

# Less Than

print(5 < 10) # Output: True

print(10 < 5) # Output: False

# Greater Than or Equal To

print(10 >= 5) # Output: True

print(5 >= 5) # Output: True

print(5 >= 10) # Output: False

# Less Than or Equal To

print(5 <= 10) # Output: True

print(5 <= 5) # Output: True

print(10 <= 5) # Output: False

# Logical Operators

Logical operators are used to combine and manipulate logical values (True or False).

Logical operators in Python are used to combine multiple conditions and evaluate them as a single expression.

**Python supports three logical operators:** and, or, and not. These operators are commonly used in conditional statements, loops, and Boolean expressions to control the flow of the program based on logical conditions.

**and:** The and operator returns True if both conditions it connects are True, otherwise it returns False.

**or:** The or operator returns True if at least one of the conditions it connects is True, otherwise it returns False.

**not:** The not operator is a unary operator that returns the opposite Boolean value of the operand. If the operand is True, not returns False, and if the operand is False, not returns True.

**Example**

# Example using and operator

x = 5

print(x > 0 and x < 10) # Output: True

# Example using or operator

y = 15

print(y < 0 or y > 10) # Output: True

# Example using not operator

is\_valid = False

print(not is\_valid) # Output: True

# Assignment Operators

Assignment operators are used to assign values to variables.

Assignment operators in Python are used to assign values to variables. In addition to simple assignment (=), Python supports several compound assignment operators, which combine arithmetic or bitwise operations with assignment. These compound assignment operators provide a concise way to modify the value of a variable based on its current value.

**Example**

x = 10

x += 5 # Equivalent to x = x + 5

x -= 3 # Equivalent to x = x – 3

x \*= 2 # Equivalent to x = x \* 2

x /= 4 # Equivalent to x = x / 4

x //= 3 # Equivalent to x = x // 3

# 5.6 Identity Operators and Membership Operators

Identity operators are used to compare the identity of objects.

is- True if the operands are identical.

V is not - True if the operands are not identical.

Membership operators are used to test if a value is a member of a sequence or not.

In -True if value is found in the sequence.

not in - True if value is not found in the sequence.

**Code**

x=5

y=5

is\_operator = x is y

print(is\_operator) # Output: True

is\_not\_operator = x is not y

print(is\_not\_operator) # Output:False

**Membership operator code:**

Fruits = [“apple”,”banana”,”orange”]

in\_operator = “banana” in fruits

print(in\_operator) # Output:True

not\_in\_operator= “grape” in fruits

print(not\_int\_operator) # Output:False

# Functions

Definition: A function in Python is like a mini-program within a program. It's a block of code designed to perform a specific task. You give a name to the function, and you can call that function whenever you need to execute its code.

Input and Output: Functions can take input values called parameters or arguments. These inputs are used by the function to perform some actions. Functions can also return a result or output after the actions are completed.

Reuse of Code: Functions help in organizing your code and make it reusable. Instead of writing the same code multiple times, you can define a function and call it whenever you need that particular functionality. This makes your code shorter and easier to manage.

Scope: Variables created inside a function are local to that function, meaning they only exist within that function. Variables created outside any function are global and can be accessed from anywhere in the code. This helps prevent unintended changes to variables.

Syntax: Defining a function involves using the def keyword, followed by the function name, parameters in parentheses, and a colon. The code inside the function is indented. To call a function, you use its name followed by parentheses containing the necessary arguments.

**Code**

Def greet(name):

Print(f”Hello,{name}!”}

**Output**

Hello, Alice

# PYTHON CHEAT SHEET

Python is a beautiful language. It's easy to learn and fun, and its syntax is simple yet elegant. Python is a popular choice for beginners, yet still powerful enough to back some of the world’s most popular products and applications from companies like NASA, Google, Mozilla, Cisco, Microsoft, and Instagram, among others. Whatever the goal, Python’s design makes the programming experience feel almost as natural as writing in English.

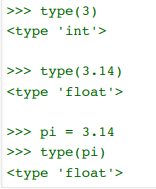
Check out Real Python to learn more about Python and web development**.**

1.Primitives NUMBERS

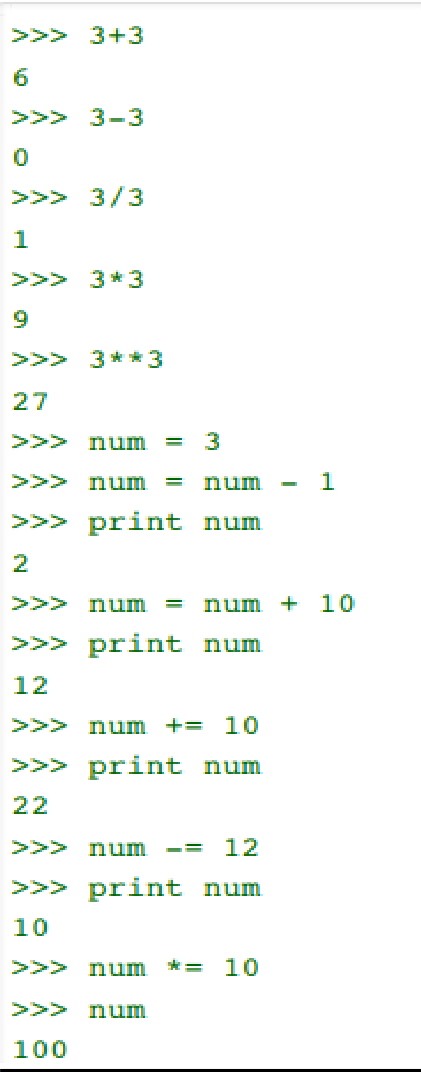
Python has integers and floats.

Integers are simply whole numbers, like 314, 500, and 716. Floats, meanwhile, arefractional numbers like 3.14, 2.867, 76.88887.

You can use the type method to check the value of an object



# You can use the basic mathematical operators:



**What happens when you divede 9 by 4 ?**



What is the actual answer? 2 remainder 1 right? Or: 2.25

In Python 2.7x, when you divide a whole number by a whole number and the answer is a fractional number, Python returns a whole number without the remainder. In other words, this type of division rounds the fraction down to the nearest whole number (commonly known as flooring the results)

# GUI Programming

Graphical User Interface (GUI) is a form of user interface which allows users to interact with computers through visual indicators using items such as icons, menus, windows, etc.

# What is Tkinter?

Tkinter is the inbuilt python module that is used to create GUI applications.

It is one of the most commonly used modules for creating GUI applications in Python as it is simple and easy to work with.

It gives an object-oriented interface to the Tk GUI toolkit.

Some other Python Libraries available for creating our own GUI applications are; Kivy, Python Qt, wxPython

Among all Tkinter is most widely used

Creating windows and dialog boxes: Tkinter can be used to create windows and dialog boxes that allow users to interact with your program. These can be used to display information, gather input, or present options to the user.

Building a GUI for a desktop application: Tkinter can be used to create the interface for a desktop application, including buttons, menus, and other interactive elements.

Adding a GUI to a command-line program: Tkinter can be used to add a GUI to a command-line program, making it easier for users to interact with the program and input arguments.

Creating custom widgets: Tkinter includes a variety of built-in widgets, such as buttons, labels, and text boxes, but it also allows you to create your own custom widgets.

Prototyping a GUI: Tkinter can be used to quickly prototype a GUI, allowing you to test and iterate on different design ideas before committing to a final implementation.

# What are Widgets?

Widgets in Tkinter are the elements of GUI application which provides various controls (such as Labels, Buttons, ComboBoxes, CheckBoxes, MenuBars, RadioButtons and many more) to users to interact with the application.

**Label:** Labels are used to display text or images. They are non-interactive and provide information to the user.

**Button:** Buttons are interactive widgets that perform an action when clicked. You can bind functions to button clicks.

**Entry:** Entry widgets are used to accept single-line text input from the user. They are often used for forms or to get user input.

**Text:** Text widgets are used to display multi-line text. They are useful for displaying large amounts of text or allowing the user to input multi-line text.

**Frame:** Frames are containers that hold other widgets. They are used to organize the layout of your GUI.

**Checkbutton:** Checkbuttons are interactive widgets that allow the user to toggle a binary state (checked or unchecked).

**Radiobutton:** Radiobuttons are used when you have a set of options and the user can only select one option at a time.

**Scrollbar:** Scrollbars are used with other widgets like Text or Listbox to allow scrolling through content that doesn't fit within the widget's visible area.

**Listbox:** Listboxes display a list of items from which the user can select one or more items.

**Menu:** Menus provide a set of options to the user, typically displayed at the top of the window. They can contain items like File, Edit, etc., with submenus and commands.

1. **INTRODUCTION TO THE PROJECT**

The main objective of the project “SIMPLE CALCULATOR**”** Is to perform the arithmetic operation.

# Software and Hardware Requirements

This project in python requires you to have the knowledge of python programming andTkinter for GUI.

**Python:-**Python is an interpreted, object-oriented, high-level Programming language with dynamic semantics.

**Tkinter:-**Tkinter is a Python binding to the TK GUI tool kit.

Tkinter is included with standard Linux,, Microsoft Windows and MacOs installs of python. # 4GB RAM(Minimum)# Python version 3.6

Anaconda Python 3 for installing Python and the required Moduless.

# Tkinter to make the GUI of the project that helps users to interact easily with project.

**#** You can use any OS-macOS, Windows and Linux-basedOS.

**Example**

**#!/usr/bin/pythonimport Tkinter top = Tkinter.Tk()** # Code to add widgets will go here... top.mainloop() **This would create a following window:**



**Tkinter Widgets** Tkinter provides various controls, such as buttons, labels and text boxesused in a GUI application. These controls are commonly called widget

In Tkinter, widgets have various attributes that can be used to customize their appearance and behavior. Here are some standard attributes commonly used across different Tkinter widgets:

**bg (background):** Sets the background color of the widget.

**fg (foreground):** Sets the foreground color (text color) of the widget.

**font:** Sets the font style, size, and weight of the text.

**width and height:** Sets the width and height of the widget in characters or pixels.

**relief:** Sets the border decoration of the widget. Common values include "flat", "raised", "sunken", "ridge", and "groove".

**state:** Sets the state of the widget, which can be "normal", "active", or "disabled".

**command:** Specifies a function to call when the widget is interacted with, such as when a button is clicked.

**text or textvariable:** Sets the text content of the widget. For some widgets, you can also use a text variable that dynamically updates the text.

**variable:** Associates a Tkinter variable with the widget, such as StringVar, IntVar, DoubleVar, etc.

**anchor:** Sets the alignment of the text or image within the widget. Common values include "n", "ne", "e", "se", "s", "sw", "w", "nw", and "center".

**padx and pady:** Sets the padding (in pixels) on the left/right and top/bottom of the widget, respectively.

**justify:** Sets the alignment of multi-line text. Common values include "left", "center", and "right".

**wrap:** Specifies how text should wrap within the widget. Values can be "none" (no wrapping), "char" (wrap at character boundaries), or "word" (wrap at word boundaries).

**image:** Sets an image to be displayed on the widget (e.g., for buttons).

1. **PROJECT CODE**







# FINAL RESULT

I have developed a straightforward calculator application using Tkinter, a powerful GUI toolkit for Python. Tkinter enables the creation of desktop applications with an intuitive graphical user interface. Leveraging this technology, my calculator provides users with a user-friendly interface for performing basic arithmetic operations. The Tkinter library facilitated the implementation of interactive buttons, an entry field for input, and a dynamic display area to showcase calculation results. The application seamlessly integrates the functionality of a calculator into a visually appealing and responsive desktop environment. Tkinter's versatility and ease of use played a pivotal role in bringing this desktop calculator to life, offering a practical solution for users in need of a quick and accessible tool for basic mathematical calculations**.**

**Output**

