Functional Programming Paradigm

Introduction

- Functional programming is a programming paradigm in which it is tried to bind each and everything in pure mathematical functions. It is a declarative type of programming style that focuses on what to solve rather than how to solve.
- Functional programming paradigm is based on lambda calculus.
- Instead of statements, functional programming makes use of expressions. Unlike a statement, which is executed to assign variables, evaluation of an expression produces a value.
- Functional programming is a declarative paradigm because it relies on expressions and declarations rather than statements. Unlike procedures that depend on a local or global state, value outputs in FP depend only on the arguments passed to the function.
- Functional programming consists only of PURE functions.
- In functional programming, control flow is expressed by combining function calls, rather than by assigning values to variables.

Example:

```
sorted(p.name.upper() for p in people if len(p.name) > 5)
```

Characteristics of Functional Programming

- Functional programming method focuses on results, not the process
- Emphasis is on what is to be computed
- Data is immutable
- Functional programming Decompose the problem into 'functions
- It is built on the concept of mathematical functions which uses conditional expressions and recursion to do perform the calculation
- Functional programming languages are designed on the concept of mathematical functions that use conditional expressions and recursion to perform computation.
- Functional programming supports higher-order functions and lazy evaluation features.
- Functional programming languages don’t support flow Controls like loop statements and conditional statements like If-Else and Switch Statements. They directly use the functions and functional calls.

Functional Programming vs Procedure Programming

```python
#Functional Programming

num = 1
def function_to_add_one(num):
    num += 1
    return num

print("Num is :",function_to_add_one(num)) #global num =1
print("Num is :",function_to_add_one(num)) #global num =1
print("Num is :",function_to_add_one(num)) #global num =1
```

```
Num is : 2
Num is : 2
Num is : 2
```

```python
#Procedural  Programming
'''The basic rules for global keyword in Python are:
When we create a variable inside a function, it's local by default.
When we define a variable outside of a function, it's global by default. ...
We use global keyword to read and write a global variable inside a function'''

num = 1
def procedure_to_add_one():
    global num
    num += 1
    return num

procedure_to_add_one()
procedure_to_add_one()
procedure_to_add_one()
```

Mathematical Notation of Functional Programming

Notation:

F(x) = y

x , y -> Arguments or parameters

x -> domain and

y->codomain

Types:

1. Injective if f(a) ≠ f(b)
2. Surjective if f(X)= Y
3. Bijective (support both)

Functional Rules:

1. (f+g)(x)=f(x)+g(x)
2. (f-g)(x)=f(x)-g(x)
3. (f*g)(x)=f(x)*g(x)
4. (f/g)(x)=f(x)/g(x)
5. (gof)(x)=g(f(x))
6. f f-1=idy

Mathematical Notation of Functional Programming

```
f(x) = 3x 2 - 2x + 5
def f(x):
        return 3 * x ** 2 - 2 * x + 5
>>> f(3)
26
>>> f(0)
 5
>>> f(-1)
 10
>>> result = f(3) + f(-1)
>>> result  #output 36
```

Function composition is a way of combining functions such that the result of each function is passed as the argument of the next function. Example:

```
def f(x):
...     return 2 * x
>>> def g(x):
...     return x + 5
>>> def h(x):
...     return x ** 2 - 3
>>> f(3)              # 6
>>> g(3)              # 8
>>> h(4)              # 13
>>> f(g(3))           # 16
>>> g(f(3))           # 11
>>> h(f(g(0)))        # 97
```

Mathematical Notation of Functional Programming

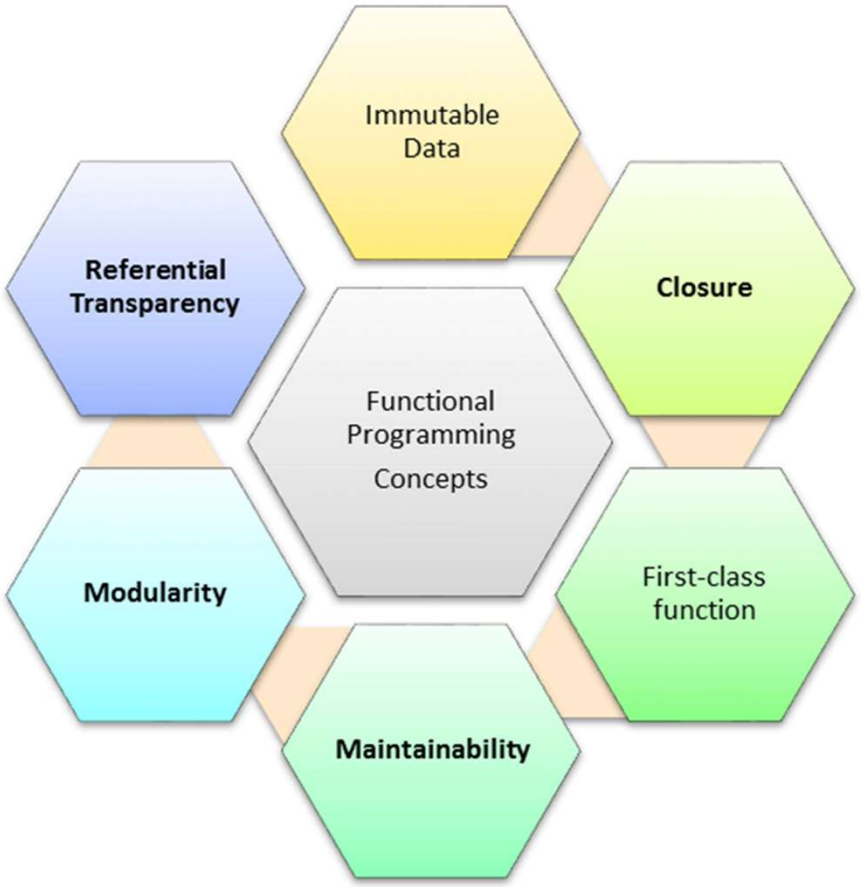
```
def compose2(f, g):
        return lambda x: f(g(x))


>>> def double(x):
...     return x * 2


>>> def inc(x):
...     return x + 1
...
>>> inc_and_double = compose2(double, inc)
>>> inc_and_double(10)
#output 22
```

Concepts of FP

- Pure functions
- Recursion
- Referential transparency
- Functions are First-Class and can be Higher-Order
- Immutability



1. Pure functions

- Pure functions always return the same results when given the same inputs. Consequently, they have no side effects.
- Properties of Pure functions are:
  - First, they always produce the same output for same arguments irrespective of anything else.
  - Secondly, they have no side-effects i.e. they do modify any argument or global variables or output something.
- A simple example would be a function to receive a collection of numbers and expect it to increment each element of this collection.
- We receive the numbers collection, use map with the ***inc*** function to increment each number, and return a new list of incremented numbers.

Example:

```
def inc(x):
        return x+1
list=[8,3,7,5,2,6]
x=map(inc,list)   #print(list)
print(x)
```

```
var z = 15;

function add(x, y) {
    return x + y;
}
```

Note : Function involving Reading files,
using global data, random numbers are impure functions

Note: if a function relies on the global variable or class member's data, then it is not pure. And in such cases, the return value of that function is not entirely dependent on the list of arguments received as input and can also have side effects.

A side effect is a change in the state of an application that is observable outside the called function other than its return value

2. Recursion

- In the functional programming paradigm, there is no for and while loops. Instead, functional programming languages rely on recursion for iteration. Recursion is implemented using recursive functions, which repetitively call themselves until the base case is reached.

Immutability

- In functional programming you cannot modify a variable after it has been initialized. You can create new variables and this helps to maintain state throughout the runtime of a program.

```
var x = 1;
x = x + 1;
```

- In imperative programming, this means “take the current value of x, add 1 and put the result back into x.” In functional programming, however, x = x + 1 is illegal. That’s because there are technically no variables in functional programming.
- Using immutable data structures, you can make single or multi-valued changes by copying the variables and calculating new values,
- Since FP doesn’t depend on shared states, all data in functional code must be immutable, or incapable of changing

Referential transparency

- An expression is said to be referentially transparent if it can be replaced with its corresponding value without changing the program's behaviour. As a result, evaluating a referentially transparent function gives the same value for fixed arguments. If a function consistently yields the same result for the same input, it is referentially transparent.
- Functional programs don’t have any assignment statements. For storing additional values in a program developed using functional programming, new variables must be defined. State of a variable in such a program is constant at any moment in time
  - pure function + immutable data = referential transparency

```
int add(int a, int b)
 {
 return a + b
 }
int mult(int a, int b)
 {
 return a * b;
 }
int x = add(2, mult(3, 4));
```

- Let’s implement a square function:
  - This (pure) function will always have the same output, given the same input.
  - Passing “2” as a parameter of the square function will always returns 4. So now we can replace the (square 2) with 4.

LAMBDA FUNCTION

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

Syntax:

lambda *arguments* : *expression*

```
x = lambda a : a + 10
    print(x(5))
x = lambda a, b : a * b
    print(x(5, 6))
```

Functions are First-Class and can be Higher-Order

- A programming language is said to have First-class functions when functions in that language are treated like any other variable. For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable.
- Higher-order functions are functions that take at least one first-class function as a parameter
- Examples:
  - name_lengths = map(len, ["Bob", "Rob", "Bobby"])
- Higher Order functions are map, reduce, filter
- The **reduce**() method reduces the array to a single value

Functional style of getting a sum of a list:

```
new_lst = [1, 2, 3, 4]
def sum_list(lst):
    if len(lst) == 1:
            return lst[0]
    else:
            return lst[0] + sum_list(lst[1:])
print(sum_list(new_lst))
```

 # or the pure functional way in python using higher order function

```
import functools
new_lst = [1, 2, 3, 4]
print(functools.reduce(lambda x, y: x + y, new_lst))
```

Map

Functions are First-Class and can be Higher-Order

- map() can run the function on each item and insert the return values into the new collection.
- The map() method creates a new array with the results of calling a function for every array element. The map() method calls the provided function once for each element in an array, in order.

map(*function*, *iterables*)

Example:

```
import functools
def myfunc(a, b):
  return a + b

x = map(myfunc, ('apple', 'banana', 'cherry'), ('orange', 'lemon', 'pineapple'))
print(x)
print(list(x))
```

reduce()

- reduce() is another higher order function for performing iterations. It takes functions and collections of items, and then it returns the value of combining the items

Example:

```
sum = reduce(lambda a, x: a + x, [0, 1, 2, 3, 4])
print sum                    // 10
```

Functions are First-Class and can be Higher-Order ..cont

filter()

- filter function expects a true or false value to determine if the element should or should not be included in the result collection. Basically, if the call-back expression is true, the filter function will include the element in the result collection. Otherwise, it will not.
- filter() Parameters
- filter() method takes two parameters:

function - function that tests if elements of an iterable return true or false

If None, the function defaults to Identity function - which returns false if any elements are false

iterable - iterable which is to be filtered, could be sets, lists, tuples, or containers of any iterators

FILTER-EXAMPLE

```
# list of letters
letters = ['a', 'b', 'd', 'e', 'i', 'j', 'o']

# function that filters vowels
def filter_vowels(letter):
    vowels = ['a', 'e', 'i', 'o', 'u']

    if(letter in vowels):
        return True
    else:
```

FILTER-EXAMPLE

```
 return False

filtered_vowels = filter(filter_vowels, letters)

print('The filtered vowels are:')
for vowel in filtered_vowels:
    print(vowel)
```

Functional Programming – Non Strict Evaluation

- In programming language theory, lazy evaluation, or call-by-need[1] is an evaluation strategy which delays the evaluation of an expression until its value is needed (non-strict evaluation) and which also avoids repeated evaluations
- Allows Functions having variables that have not yet been computed

In Python, the logical expression operators and, or, and if-then-else are all non-strict. We sometimes call them short-circuit operators because they don't need to evaluate all arguments to determine the resulting value.

The following command snippet shows the and operator's non-strict feature:

```
>>> 0 and print("right")
0
>>> True and print("right")
Right
```

When we execute the preceding command snippet, the left-hand side of the and operator is equivalent to False; the right-hand side is not evaluated. When the left-hand side is equivalent to True, the right-hand side is evaluated

Functional Programming – lambda calculus

Lambda expressions in Python and other programming languages have their roots in lambda calculus. Lambda calculus can encode any computation. Functional languages get their origin in mathematical logic and lambda calculus

In Python, we use the lambda keyword to declare an anonymous function, which is why we refer to them as "lambda functions". An anonymous function refers to a function declared with no name.

when you want to pass a function as an argument to higher-order functions, that is, functions that take other functions as their arguments

Characteristics of Python lambda functions:

- A lambda function can take any number of arguments, but they contain only a single expression. An expression is a piece of code executed by the lambda function, which may or may not return any value.
- Lambda functions can be used to return function objects.
- Syntactically, lambda functions are restricted to only a single expression.

Syntax:

```
lambda argument(s): expression
```

Example:

```
remainder = lambda num: num % 2
print(remainder(5))
```

```
[(lambda x: x*x)(x) for x in [2,6,9,3,6,4,8]]
```

Functional Programming – Closure

Basically, the method of binding data to a function without actually passing them as parameters is called closure. It is a function object that remembers values in enclosing scopes even if they are not present in memory.

Example:.

```
def counter(start=0, step=1):
    x = [start]
    def _inc():
        x[0] += step
        return x[0]
    return _inc
c1 = counter()
c2 = counter(100, -10)
c1()
//1
c2()
90
```

Functional Programming in Python

- Functional Programming is a popular programming paradigm closely linked to computer science's mathematical foundations. While there is no strict definition of what constitutes a functional language, we consider them to be languages that use functions to transform data.

- Python is not a functional programming language but it does incorporate some of its concepts alongside other programming paradigms. With Python, it's easy to write code in a functional style, which may provide the best solution for the task at hand.

Pure Functions in Python

- If a function uses an object from a higher scope or random numbers, communicates with files and so on, it might be impure

```
def multiply_2_pure(numbers):
    new_numbers = []
    for n in numbers:
        new_numbers.append(n * 2)
    return new_numbers

original_numbers = [1, 3, 5, 10]
changed_numbers = multiply_2_pure(original_numbers)
print(original_numbers) # [1, 3, 5, 10]
print(changed_numbers)  # [2, 6, 10, 20]
```

```
[1, 3, 5, 10]
[2, 6, 10, 20]
```

```
# Example1 : Impure Function
A = 5
def impure_sum(b): # A is out side function ,it has side effect
    return b + A

impure_sum(8)
```

```
13
```

```
# Example2 : Pure Function

def pure_sum(a, b):    #a and b inside function
    return a + b
print(impure_sum(6))
```

```
11
```

Built-in Higher Order Functions

Map

- The map function allows us to apply a function to every element in an iterable object

Filter

- The filter function tests every element in an iterable object with a function that returns either True or False, only keeping those which evaluates to True.

Combining map and filter

- As each function returns an iterator, and they both accept iterable objects, we can use them together for some really expressive data manipulations!

List Comprehensions

- A popular Python feature that appears prominently in Functional Programming Languages is list comprehensions. Like the map and filter functions, list comprehensions allow us to modify data in a concise, expressive way.

Anonymous Function

- In Python, anonymous function is a function that is defined without a name.
- While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.

Characteristics of Python lambda functions:

- A lambda function can take any number of arguments, but they contain only a single expression. An expression is a piece of code executed by the lambda function, which may or may not return any value.
- Lambda functions can be used to return function objects.
- Syntactically, lambda functions are restricted to only a single expression.

Syntax of Lambda Function in python

lambda arguments: expression

Example:

```
double = lambda x: x * 2
print(double(5))
# Output: 10
```

```
product = lambda x, y : x * y
print(product(2, 3))
```

Note: you want to pass a function as an argument to higher-order functions, that is, functions that take other functions as their arguments

map() Function

Example Map with lambda

tup= (5, 7, 22, 97, 54, 62, 77, 23, 73, 61)

newtuple = tuple(map(lambda x: x+3 , tup))

print(newtuple)

//with multiple iterables

list_a = [1, 2, 3]

list_b = [10, 20, 30]

map(lambda x, y: x + y, list_a, list_b)

Example with Map

from math import sqrt

map(sqrt, [1, 4, 9, 16])

[1.0, 2.0, 3.0, 4.0]

map(str.lower, ['A', 'b', 'C'])

['a', 'b', 'c']

#splitting the input and convert to int using map

print(list(map(int, input.split(‘ ‘)))

```
numbers_list = [2, 6, 8, 10, 11, 4, 12, 7, 13, 17, 0, 3, 21]

mapped_list = list(map(lambda num: num % 2, numbers_list))

print(mapped_list)
```

map() Function

- map() function is a type of higher-order. As mentioned earlier, this function takes another function as a parameter along with a sequence of iterables and returns an output after applying the function to each iterable present in the sequence.

Syntax:

map(function, iterables)

Example without Map

```
my_pets = ['alfred', 'tabitha', 'william', 'arla']
uppered_pets = []
for pet in my_pets:
        pet_=pet.upper()
        uppered_pets.append(pet_)
print(uppered_pets)
```

Example with Map

```
my_pets = ['alfred', 'tabitha', 'william', 'arla']
uppered_pets=list(map(str.upper,my_pets)) print(uppered_pets)
//map with multiple list as input
circle_areas = [3.56773, 5.57668, 4.00914, 56.24241, 9.01344, 32.00013]
result = list(map(round, circle_areas, range(1,7)))
print(result)
```

filter() Function

- filter extracts each element in the sequence for which the function returns True.
- filter(), first of all, requires the function to return boolean values (true or false) and then passes each element in the iterable through the function, "filtering" away those that are false

Syntax:

filter(func, iterable)

The following points are to be noted regarding filter():

- Unlike map(), only one iterable is required.
- The func argument is required to return a boolean type. If it doesn't, filter simply returns the iterable passed to it. Also, as only one iterable is required, it's implicit that func must only take one argument.
- filter passes each element in the iterable through func and returns only the ones that evaluate to true. I mean, it's right there in the name -- a "filter".

Example:

```
def isOdd(x): return x % 2 == 1
filter(isOdd, [1, 2, 3, 4])
# output --->  [1, 3]
```

filter() Function

Example:

Python 3

scores = [66, 90, 68, 59, 76, 60, 88, 74, 81, 65]

def is_A_student(score):

return score > 75

over_75 = list(filter(is_A_student, scores))

print(over_75)

```
'''
ages = [5, 12, 17, 18, 24, 32]

def myFunc(x):
  if x < 18:
    return False
  else:
    return True

adults = filter(myFunc, ages)
for x in adults:
  print(x)
```

```
# Python 3
dromes = ("demigod", "rewire", "madam", "freer",
"anutforajaroftuna", "kiosk")

palindromes = list(filter(lambda word: word == word[::-1],
dromes))

print(palindromes)
```

```
#function that filters vowels
def fun(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable in letters):
        return True
    else:
        return False
# sequence
sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']

# using filter function
filtered = filter(fun, sequence)

print('The filtered letters are:')
for s in filtered:
    print(s)
```

reduce() Function

- reduce, combines the elements of the sequence together, using a binary function. In addition to the function and the list, it also takes an initial value that initializes the reduction, and that ends up being the return value if the list is empty.
- The “reduce” function will transform a given list into a single value by applying a given function continuously to all the elements. It basically keeps operating on pairs of elements until there are no more elements left.
- reduce applies a function of two arguments cumulatively to the elements of an iterable, optionally starting with an initial argument

Syntax:

reduce(func, iterable[, initial])

Example:

```
reduce(lambda s,x: s+str(x), [1, 2, 3, 4], '')
#output '1234‘
my_list = [3,8,4,9,5]
reduce(lambda a, b: a * b, my_list)
#output 4320 ( 3*8*4*9*5)
```

```
from functools import reduce
y = filter(lambda x: (x>=3), (1,2,3,4))
print(list(y))


reduce(lambda a,b: a+b,[23,21,45,98])


nums = [92, 27, 63, 43, 88, 8, 38, 91, 47, 74, 18, 16,
      29, 21, 60, 27, 62, 59, 86, 56]
sum = reduce(lambda x, y : x + y, nums) / len(nums)
```

map(), filter() and reduce() Function

Using filter() within map():

```
c = map(lambda x:x+x,filter(lambda x: (x>=3), (1,2,3,4)))
print(list(c))
```

Using map() within filter():

```
c = filter(lambda x: (x>=3),map(lambda x:x+x, (1,2,3,4))) #lambda x: (x>=3)
print(list(c))
```

Using map() and filter() within reduce():

```
d = reduce(lambda x,y: x+y,map(lambda x:x+x,filter(lambda x: (x>=3), (1,2,3,4))))
print(d)
```

map(), filter() and reduce() Function

```
from functools import reduce
# Use map to print the square of each numbers rounded# to two decimal places
my_floats = [4.35, 6.09, 3.25, 9.77, 2.16, 8.88, 4.59]
# Use filter to print only the names that are less than or equal to seven letters
my_names = ["olumide", "akinremi", "josiah", "temidayo", "omoseun"]
# Use reduce to print the product of these numbers
my_numbers = [4, 6, 9, 23, 5]
map_result = list(map(lambda x: round(x ** 2, 3), my_floats))
filter_result = list(filter(lambda name: len(name) <= 7, my_names))
reduce_result = reduce(lambda num1, num2: num1 * num2, my_numbers)
print(map_result)
print(filter_result)
print(reduce_result)
```

Function vs Procedure

S.No	Functional Paradigms	Procedural Paradigm
1	Treats computation as the evaluation of mathematical functions avoiding state and mutable data	Derived from structured programming, based on the concept of modular programming or the *procedure call*
2	Main traits are Lambda alculus, compositionality, formula, recursion, referential transparency	Main traits are Local variables, sequence, selection, iteration, and modularization
3	**Functional** programming focuses on **expressions**	**Procedural** programming focuses on **statements**
4	Often recursive. Always returns the same output for a given input.	The output of a routine does not always have a direct correlation with the input.
5	Order of evaluation is usually undefined.	Everything is done in a specific order.
6	Must be stateless. i.e. No operation can have side effects.	Execution of a routine may have side effects.
7	Good fit for parallel execution, Tends to emphasize a divide and conquer approach.	Tends to emphasize implementing solutions in a linear fashion.

Function vs Object Oriented

S.No	Functional Paradigms	Object Oriented Paradigm
1	FP uses Immutable data.	OOP uses Mutable data.
2	Follows Declarative Programming based Model.	Follows Imperative Programming Model.
3	What it focuses is on: "What you are doing. in the programme."	What it focuses is on "How you are doing your programming."
4	Supports Parallel Programming.	No supports for Parallel Programming.
5	Its functions have no-side effects.	Method can produce many side effects.
6	Flow Control is performed using function calls & function calls with recursion.	Flow control process is conducted using loops and conditional statements.
7	Execution order of statements is not very important.	Execution order of statements is important.
8	Supports both "Abstraction over Data" and "Abstraction over Behavior."	Supports only "Abstraction over Data".

GUI & Event Handling Programming Paradigm

Event Driven Programming Paradigm

- Event-driven programming is a programming paradigm in which the flow of program execution is determined by events - for example a user action such as a mouse click, key press, or a message from the operating system or another program.
- An event-driven application is designed to detect events as they occur, and then deal with them using an appropriate event-handling procedure.
- In a typical modern event-driven program, there is no discernible flow of control. The main routine is an event-loop that waits for an event to occur, and then invokes the appropriate event-handling routine.
- Event callback is a function that is invoked when something significant happens like when click event is performed by user or the result of database query is available.

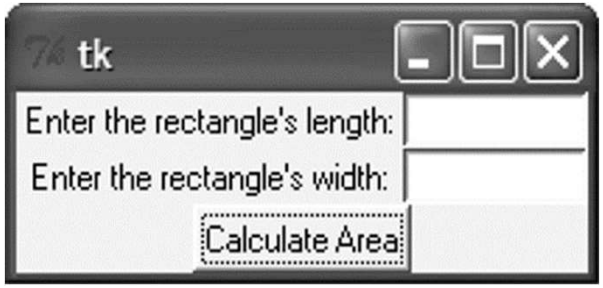
Event Handlers: Event handlers is a type of function or method that run a specific action when a specific event is triggered. For example, it could be a button that when user click it, it will display a message, and it will close the message when user click the button again, this is an event handler.

Trigger Functions: Trigger functions in event-driven programming are a functions that decide what code to run when there are a specific event occurs, which are used to select which event handler to use for the event when there is specific event occurred.

Events: Events include mouse, keyboard and user interface, which events need to be triggered in the program in order to happen, that mean user have to interacts with an object in the program, for example, click a button by a mouse, use keyboard to select a button and etc.

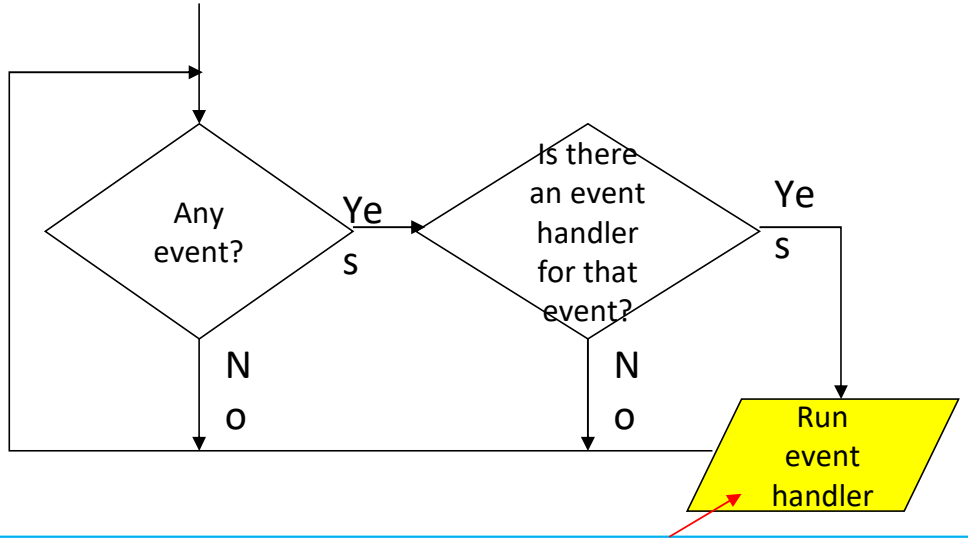
Introduction

- A graphical user interface allows the user to interact with the operating system and other programs using graphical elements such as icons, buttons, and dialog boxes.
- GUIs popularized the use of the mouse.
- GUIs allow the user to point at graphical elements and click the mouse button to activate them.
- GUI Programs Are Event-Driven
- User determines the order in which things happen
- GUI programs respond to the actions of the user, thus they are event driven.
- The tkinter module is a wrapper around tk, which is a wrapper around tcl, which is what is used to create windows and graphical user interfaces.



Introduction

- A major task that a GUI designer needs to do is to determine what will happen when a GUI is invoked
- Every GUI component may generate different kinds of “events” when a user makes access to it using his mouse or keyboard
- E.g. if a user moves his mouse on top of a button, an event of that button will be generated to the Windows system
- E.g. if the user further clicks, then another event of that button will be generated (actually it is the click event)
- For any event generated, the system will first check if there is an event handler, which defines the action for that event
- For a GUI designer, he needs to develop the event handler to determine the action that he wants Windows to take for that event.



GUI Using Python

- Tkinter: Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.
- wxPython: This is an open-source Python interface for wxWindows
- PyQt –This is also a Python interface for a popular cross-platform Qt GUI library.
- JPython: JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine

Tkinter Programming

- Tkinter is the standard GUI library for Python.
- Creating a GUI application using Tkinter

Steps

- Import the Tkinter module.

  Import tKinter as tk

- Create the GUI application main window.

  root = tk.Tk()

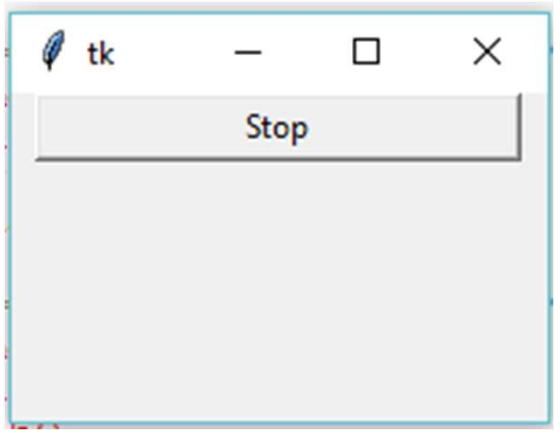
- Add one or more of the above-mentioned widgets to the GUI application.

  button = tk.Button(root, text='Stop', width=25, command=root.destroy)

  button.pack()

- Enter the main event loop to take action against each event triggered by the user.

  root.mainloop()

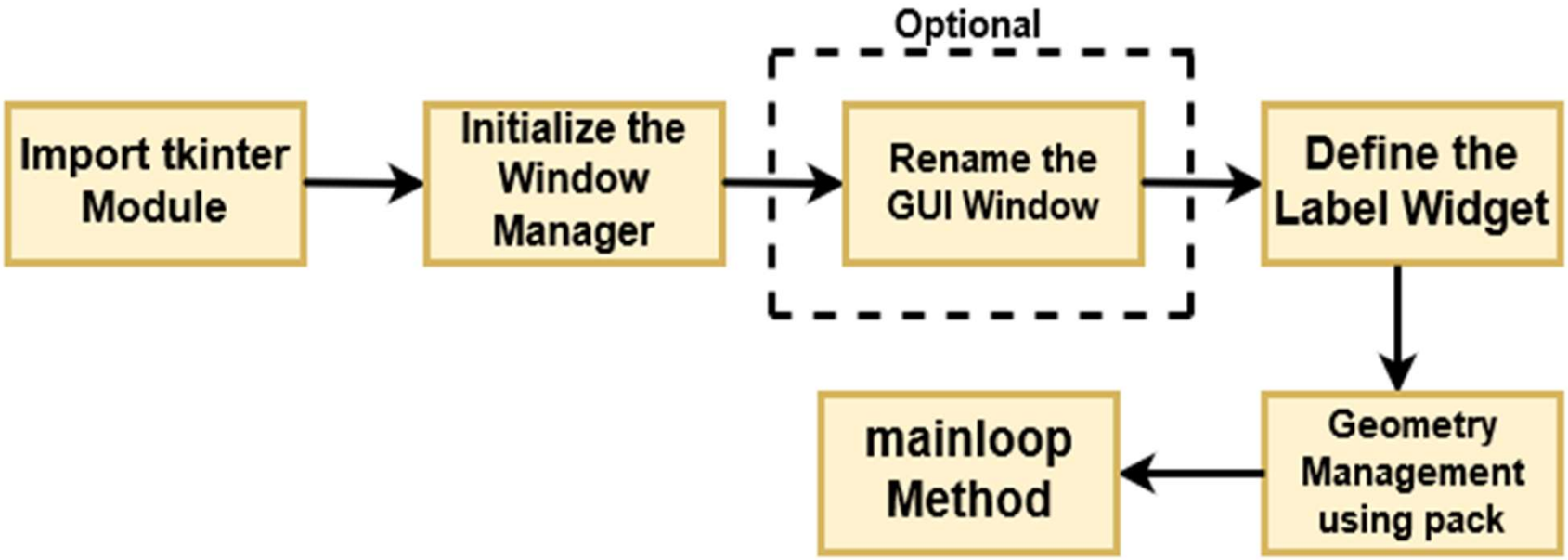


Tkinter widgets

- Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

Widget	Description
Label	Used to contain text or images
Button	Similar to a Label but provides additional functionality for mouse overs, presses, and releases as well as keyboard activity/events
Canvas	Provides ability to draw shapes (lines, ovals, polygons, rectangles); can contain images or bitmaps
Radiobutton	Set of buttons of which only one can be "pressed" (similar to HTML radio input)
Checkbutton	Set of boxes of which any number can be "checked" (similar to HTML checkbox input)
Entry	Single-line text field with which to collect keyboard input (similar to HTML text input)
Frame	Pure container for other widgets
Listbox	Presents user list of choices to pick from
Menu	Actual list of choices "hanging" from a Menubutton that the user can choose from
Menubutton	Provides infrastructure to contain menus (pulldown, cascading, etc.)
Message	Similar to a Label, but displays multi-line text
Scale	Linear "slider" widget providing an exact value at current setting; with defined starting and ending values
Text	Multi-line text field with which to collect (or display) text from user (similar to HTML TextArea)
Scrollbar	Provides scrolling functionality to supporting widgets, i.e., Text, Canvas, Listbox, and Entry
Toplevel	Similar to a Frame, but provides a separate window container

Operation Using Tkinter Widget



Geometry Managers

- The pack() Method − This geometry manager organizes widgets in blocks before placing them in the parent widget.

  widget.pack(pack_options)

  options

  - expand − When set to true, widget expands to fill any space not otherwise used in widget's parent.
  - fill − Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically).
  - side − Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.
- The grid() Method − This geometry manager organizes widgets in a table-like structure in the parent widget.

  widget.grid(grid_options)

  options −

  - Column/row − The column or row to put widget in; default 0 (leftmost column).
  - Columnspan, rowsapn− How many columns or rows to widgetoccupies; default 1.
  - ipadx, ipady − How many pixels to pad widget, horizontally and vertically, inside widget's borders.
  - padx, pady − How many pixels to pad widget, horizontally and vertically, outside v's borders.

Geometry Managers

- The place() Method − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

  widget.place(place_options)

  options −

  - anchor − The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)
  - bordermode − INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.
  - height, width − Height and width in pixels.
  - relheight, relwidth − Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
  - relx, rely − Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
  - x, y − Horizontal and vertical offset in pixels.

Common Widget Properties

Common attributes such as sizes, colors and fonts are specified.

- Dimensions
- Colors
- Fonts
- Anchors
- Relief styles(eg. Flat, Raised)
- Bitmaps
- Cursors

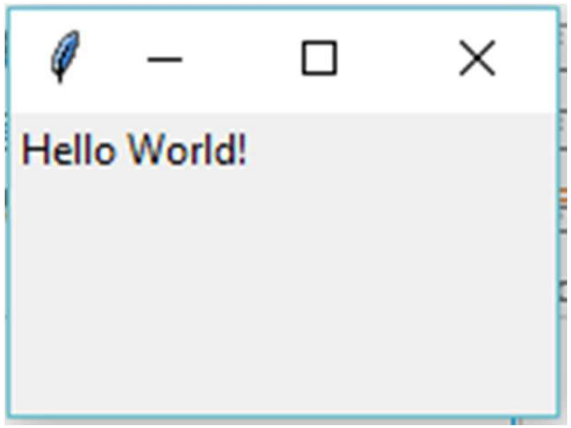
Label Widgets

- A label is a widget that displays text or images, typically that the user will just view but not otherwise interact with. Labels are used for such things as identifying controls or other parts of the user interface, providing textual feedback or results, etc.
- Syntax

    tk.Label(parent,text="message")

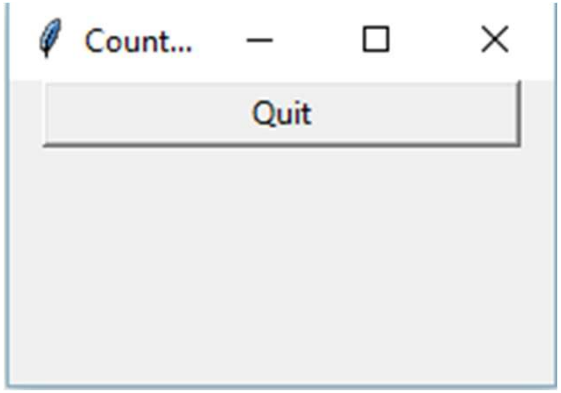
Example:

```
import tkinter as tk
root = tk.Tk()
label = tk.Label(root, text='Hello World!')
label.grid()
root.mainloop()
```



Button Widgets

```
import tkinter as tk
r = tk.Tk()
r.title(‘Example')
button = tk.Button(r, text='Stop', width=25)
button.pack()
r.mainloop()
```



Button Widgets

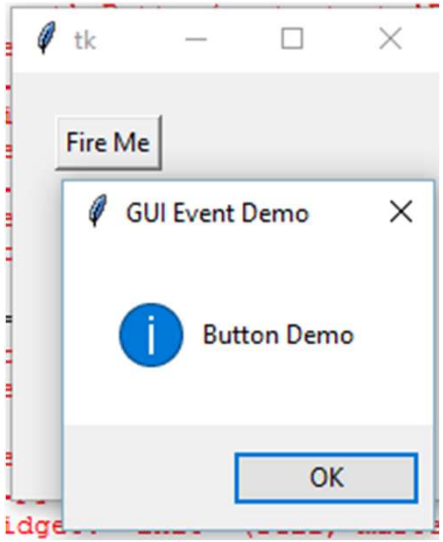
- A button, unlike a frame or label, is very much designed for the user to interact with, and in particular, press to perform some action. Like labels, they can display text or images, but also have a whole range of new options used to control their behavior.

Syntax

button = ttk.Button(parent, text='ClickMe', command=submitForm)

Example:

```
import tkinter as tk
from tkinter import messagebox
def hello():
  msg = messagebox.showinfo( "GUI Event Demo","Button Demo")
root = tk.Tk()
root.geometry("200x200")
b = tk.Button(root, text='Fire Me',command=hello)
b.place(x=50,y=50)
root.mainloop()
```



Button Widgets

- Button:To add a button in your application, this widget is used.

Syntax :

w=Button(master, text="caption" option=value)

- master is the parameter used to represent the parent window.
- activebackground: to set the background color when button is under the cursor.
- activeforeground: to set the foreground color when button is under the cursor.
- bg: to set he normal background color.
- command: to call a function.
- font: to set the font on the button label.
- image: to set the image on the button.
- width: to set the width of the button.
- height: to set the height of the button.

Entry Widgets

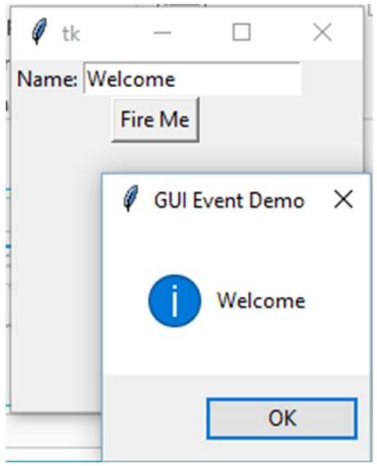
- An entry presents the user with a single line text field that they can use to type in a string value. These can be just about anything: their name, a city, a password, social security number, and so on.

Syntax

name = ttk.Entry(parent, textvariable=username)

Example:

```
def hello():
  msg = messagebox.showinfo( "GUI Event Demo",t.get())
root = tk.Tk()
root.geometry("200x200")
l1=tk.Label(root,text="Name:")
l1.grid(row=0)
t=tk.Entry(root)
t.grid(row=0,column=1)
b = tk.Button(root, text='Fire Me',command=hello)
b.grid(row=1,columnspan=2);
root.mainloop()
```



Canvas

- The Canvas is a rectangular area intended for drawing pictures or other complex layouts. You can place graphics, text, widgets or frames on a Canvas.
- It is used to draw pictures and other complex layout like graphics, text and widgets.

Syntax:

w = Canvas(master, option=value)

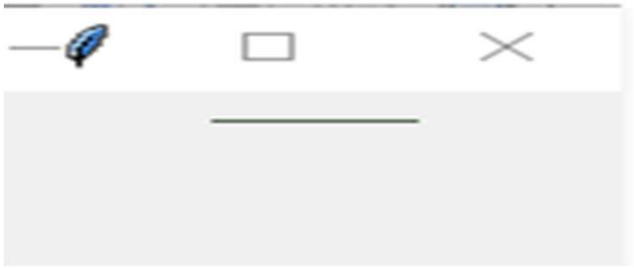
- master is the parameter used to represent the parent window.
- bd: to set the border width in pixels.
- bg: to set the normal background color.
- cursor: to set the cursor used in the canvas.
- highlightcolor: to set the color shown in the focus highlight.
- width: to set the width of the widget.
- height: to set the height of the widget.

To Create line: create_line(coords, options)

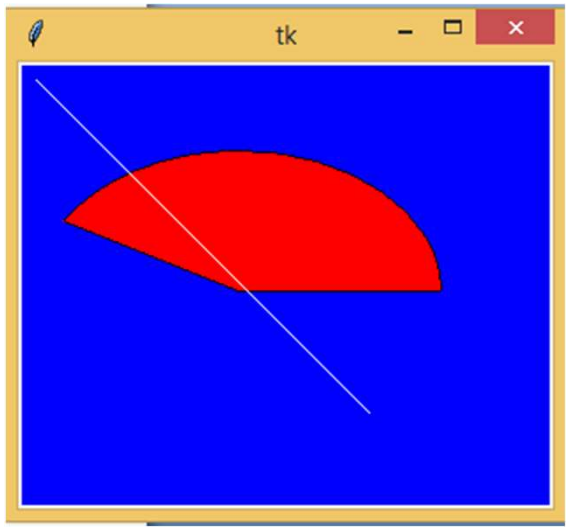
To create_arc(*x0*, *y0*, *x1*, *y1*, *option*, ...)

Canvas

```
from tkinter import *
master = Tk()
w = Canvas(master, width=40, height=60)
w.pack()
canvas_height=20
canvas_width=200
y = int(canvas_height / 2)
w.create_line(0, y, canvas_width, y )
mainloop()
```



```
from tkinter import *
from tkinter import messagebox
top = Tk()
C = Canvas(top, bg = "blue", height = 250, width = 300)
coord = 10, 50, 240, 210
arc = C.create_arc(coord, start = 0, extent = 150, fill = "red")
line = C.create_line(10,10,200,200,fill = 'white')
C.pack()
top.mainloop()
```



Checkbutton

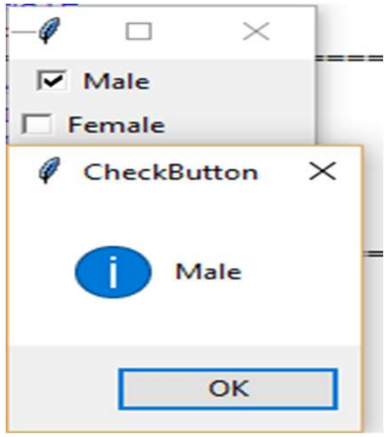
- A checkbutton is like a regular button, except that not only can the user press it, which will invoke a command callback, but it also holds a binary value of some kind (i.e. a toggle). Checkbuttons are used all the time when a user is asked to choose between, e.g. two different values for an option.

Syntax

w = CheckButton(master, option=value)

Example:

```
from tkinter import *
root= Tk()
root.title('Checkbutton Demo')
v1=IntVar()
v2=IntVar()
cb1=Checkbutton(root,text='Male', variable=v1,onvalue=1, offvalue=0, command=test)
cb1.grid(row=0)
cb2=Checkbutton(root,text='Female', variable=v2,onvalue=1, offvalue=0, command=test)
cb2.grid(row=1)
root.mainloop()
```




```
def test():
    if(v1.get()==1 ):
        v2.set(0)
        print("Male")
    if(v2.get()==1):
        v1.set(0)
        print("Female")
```

radiobutton

- A radiobutton lets you choose between one of a number of mutually exclusive choices; unlike a checkbutton, it is not limited to just two choices. Radiobuttons are always used together in a set and are a good option when the number of choices is fairly small
- **Syntax**

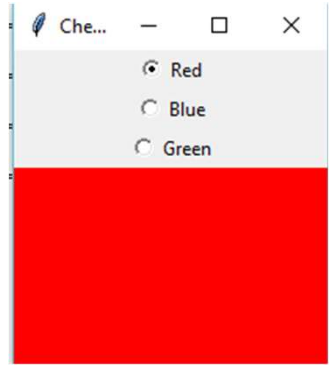
w = RadioButton(master, option=value)

Example:

from tkinter import*

```
root= Tk()
root.geometry("200x200")
radio=IntVar()
rb1=Radiobutton(root,text='Red', variable=radio,width=25,value=1, command=choice)
rb1.grid(row=0)
rb2=Radiobutton(root,text='Blue', variable=radio,width=25,value=2, command=choice)
rb2.grid(row=1)
rb3=Radiobutton(root,text='Green', variable=radio,width=25,value=3, command=choice)
rb3.grid(row=3)
```

```
def choice():
    if(radio.get()==1):
        root.configure(background='red')
    elif(radio.get()==2):
        root.configure(background='blue')
    elif(radio.get()==3):
        root.configure(background='green')
```



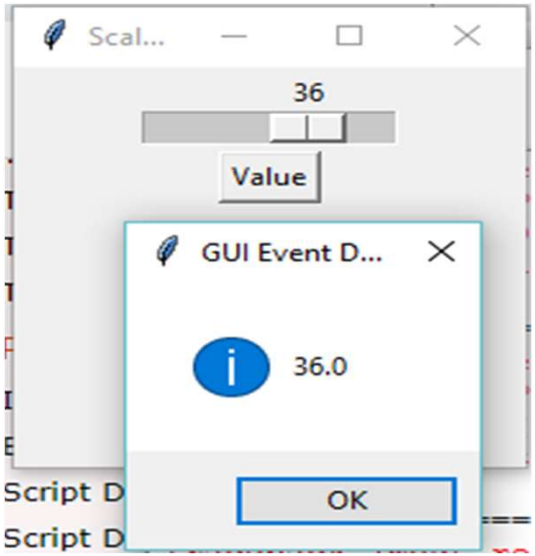
Scale

- Scale widget is used to implement the graphical slider to the python application so that the user can slide through the range of values shown on the slider and select the one among them. We can control the minimum and maximum values along with the resolution of the scale. It provides an alternative to the Entry widget when the user is forced to select only one value from the given range of values.
- **Syntax**

w = Scale(top, options)

Example:

```
from tkinter import messagebox
root= Tk()
root.title('Scale Demo')
root.geometry("200x200")
def slide():
   msg = messagebox.showinfo( "GUI Event Demo",v.get())
v = DoubleVar()
scale = Scale( root, variable = v, from_ = 1, to = 50, orient = HORIZONTAL)
scale.pack(anchor=CENTER)
btn = Button(root, text="Value", command=slide)
btn.pack(anchor=CENTER)
root.mainloop()
```



Spinbox

- The Spinbox widget is an alternative to the Entry widget. It provides the range of values to the user, out of which, the user can select the one.

Syntax

w = Spinbox(top, options)

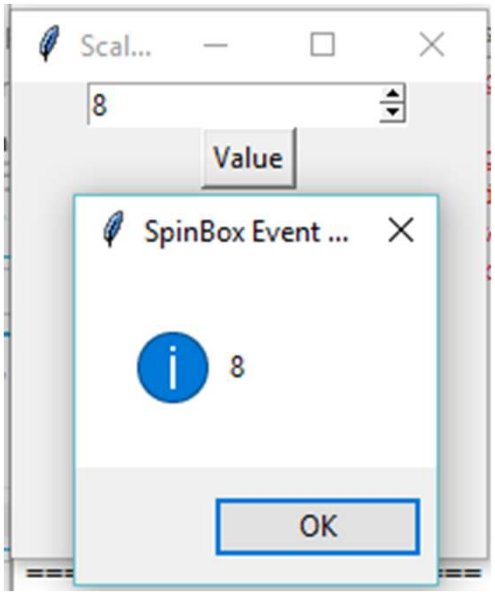
Example:

```
from tkinter import *
from tkinter import messagebox

root= Tk()
root.title('Scale Demo')
root.geometry("200x200")

def slide():
   msg = messagebox.showinfo( "SpinBox Event Demo",spin.get())

spin = Spinbox(root, from_= 0, to = 25)
spin.pack(anchor=CENTER)
btn = Button(root, text="Value", command=slide)
btn.pack(anchor=CENTER)
root.mainloop()
```



Menubutton

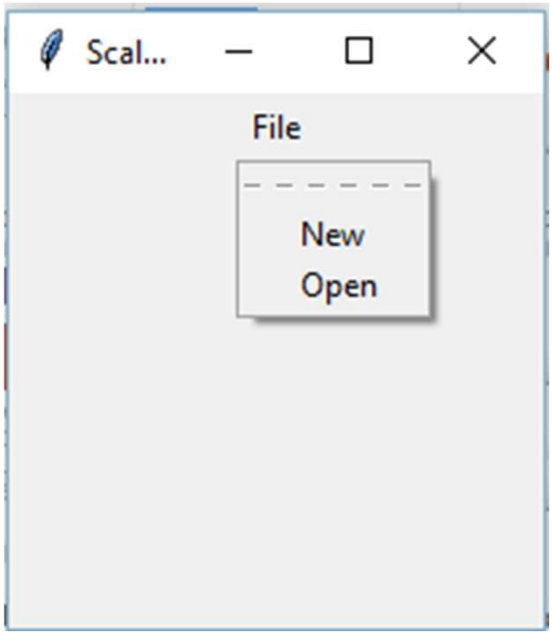
- Menubutton widget can be defined as the drop-down menu that is shown to the user all the time. It is used to provide the user a option to select the appropriate choice exist within the application.

Syntax

w = Menubutton(Top, options)

Example:

```
from tkinter import *
root= Tk()
root.title('Scale Demo')
root.geometry("200x200")
menubutton = Menubutton(root, text = "File", relief = FLAT)
menubutton.grid()
menubutton.menu = Menu(menubutton)
menubutton["menu"]=menubutton.menu
menubutton.menu.add_checkbutton(label = "New", variable=IntVar())
menubutton.menu.add_checkbutton(label = "Open", variable = IntVar())
menubutton.pack()
root.mainloop()
```



Menubutton

- Menubutton widget can be defined as the drop-down menu that is shown to the user all the time. It is used to provide the user a option to select the appropriate choice exist within the application.

Syntax

w = Menubutton(Top, options)

Example:

```
from tkinter import *
from tkinter import messagebox
root= Tk()
root.title('Menu Demo')
root.geometry("200x200")
def new():
   print("New Menu!")
def disp():
   print("Open Menu!")
menubutton = Menubutton(root, text="File")
menubutton.grid()
menubutton.menu = Menu(menubutton, tearoff = 0)
menubutton["menu"] = menubutton.menu
menubutton.menu.add_command(label="Create
new",command=new)
menubutton.menu.add_command(label="Open",command=dis
p)
menubutton.menu.add_separator()
menubutton.menu.add_command(label="Exit",command=root.
quit)
menubutton.pack()
```