Chaptere 6

Standard Library, Testing

6.1 Operating System Interface

The OS module in Python gives a method for utilizing working framework subordinate usefulness. The capacities that the OS module gives enables you to interface with the hidden working framework that Python is running on – be that Windows, Mac or Linux.

The os module gives many capacities to interfacing with the working framework:

```
>>> import os
>>> os.getcwd()        # Return the current working directory
'C:\\Python34'
>>> os.chdir('/server/accesslogs')   # Change current work-
ing directory
>>> os.system('mkdir today')   # Run the command mkdir in
the system shell
0
```

Make sure to utilize the import os style rather than from os import *. This will keep os.open() from shadowing the implicit open() work which works much in an unexpected way.

The implicit dir() and help() capacities are valuable as intelligent guides for working with vast modules like os:

```
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's
docstrings>
```

For day by day document and index administration errands, the shutil module gives a more elevated amount interface that is less demanding to utilize:

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
'archive.db'
>>> shutil.move('/build/executables', 'installdir')
'installdir'
```

6.2 String Pattern Matching

The re module gives consistent expression devices to cutting edge string handling. For complex coordinating and control, normal expressions offer brief, improved arrangements:

```
>>> import re
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
'cat in the hat'
```

At the point when just straightforward capacities are required, string strategies are favored on the grounds that they are simpler to peruse and troubleshoot:

```
>>> 'tea for too'.replace('too', 'two')
'tea for two'
```

6.3 Mathematics

The math module offers access to the basic C library capacities for coasting point math:

```
>>> import math
>>> math.cos(math.pi / 4)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

The arbitrary module gives apparatuses to making irregular choices:

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(range(100), 10)   # sampling without replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random()    # random float
0.17970987693706186
```

```
>>> random.randrange(6)      # random integer chosen from
range(6)
4
```

6.4 Internet Access

There are various modules for getting to the web and preparing web conventions. Two of the least difficult are urllib.request for recovering information from URLs and smtplib for sending letters:

```
>>> from urllib.request import urlopen
>>> for line in urlopen('http://tycho.usno.navy.mil/cgi-
bin/timer.pl'):
...     line = line.decode('utf-8')  # Decoding the binary
data to text.
...     if 'EST' in line or 'EDT' in line:  # look for
Eastern Time
...         print(line)

<BR>Nov. 25, 09:43:32 PM EST

>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org',
'jcaesar@example.org',
... """To: jcaesar@example.org
... From: soothsayer@example.org
...
... Beware the Ides of March.
... """)
>>> server.quit()
```

6.5 Dates and Times

The datetime module supplies classes for controlling dates and times in both straightforward and complex ways. While date and time number juggling is upheld, the concentrate of the usage is on effective part extraction for yield arranging and control. The module likewise underpins objects that are timezone mindful..

```
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
```

```
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day
of %B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of Decem-
ber.'

>>> # dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```

6.6 Data Compression

Normal information chronicling and pressure positions are straightforwardly bolstered by modules including: zlib, gzip, bz2, lzma, zipfile and tarfile.

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

6.7 Multi Threading

Running a few strings is like running a few unique projects simultaneously, however with the accompanying advantages:

- Multiple strings inside a procedure share similar information space with the fundamental string and can in this way share data or speak with each other more effortlessly than if they were separate procedures.
- Threads now and again called light-weight procedures and they don't require much memory overhead; they mind less expensive than forms.

A thread has a starting, an execution succession, and a conclusion. It has a guideline pointer that monitors where inside its setting it is at present running..

- It can be pre-empted (interrupted)
- It can incidentally be put on hold (otherwise called resting) while different strings are running - this is called yielding.

Starting a New Thread

To generate another string, you have to call following strategy accessible in thread module:

```
thread.start_new_thread ( function, args[, kwargs] )
```

This technique call empowers a quick and proficient approach to make new strings in both Linux and Windows.

The technique call returns quickly and the tyke string begins and calls work with the passed rundown of agrs. At the point when work restores, the string ends.

Here, args is a tuple of contentions; utilize an exhaust tuple to call work without passing any contentions. kwargs is a discretionary lexicon of watchword contentions.

Example

```
#!/usr/bin/python
import thread

import time
# Define a function for the thread
def print_time( threadName, delay):
count = 0
while count < 5:
time.sleep(delay)
count += 1
print "%s: %s" % ( threadName, time.ctime(time.time()) )
# Create two threads as follows
try:
thread.start_new_thread( print_time, ("Thread-1", 2, ) )
thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
print "Error: unable to start thread"
while 1:
pass
```

At the point when the above code is executed, it creates the accompanying outcome:

Thread-1: Thu Jan 22 15:42:17 2009
Thread-1: Thu Jan 22 15:42:19 2009
Thread-2: Thu Jan 22 15:42:19 2009
Thread-1: Thu Jan 22 15:42:21 2009
Thread-2: Thu Jan 22 15:42:23 2009

Thread-1: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:25 2009
Thread-2: Thu Jan 22 15:42:27 2009
Thread-2: Thu Jan 22 15:42:31 2009
Thread-2: Thu Jan 22 15:42:35 2009

In spite of the fact that it is exceptionally viable for low-level threading, yet the string module is extremely constrained contrasted with the more current threading module.

The *Threading* Module:

The more up to date threading module included with Python 2.4 gives substantially more effective, abnormal state bolster for strings than the string module examined in the past area.

The threading module uncovered every one of the strategies for the string module and gives some extra techniques:

- **threading.activeCount():** Returns the quantity of thread objects that are dynamic..
- **threading.currentThread():** Returns the quantity of thread objects in the guest's string control.
- **threading.enumerate():** Returns a rundown of all thread objects that are at present dynamic.

Notwithstanding the techniques, the threading module has the Thread class that executes threading. The strategies given by the Thread class are as per the following:

- **run():** The run() technique is the passage point for a thread.
- **start():** The begin() technique begins a thread by calling the run strategy..
- **join([time]):** The join() sits tight for thread to end..
- **isAlive():** The isAlive() technique checks whether a thread is as yet executing..
- **getName():** The getName() technique restores the name of a thread.
- **setName():** The setName() technique sets the name of a thread.

Creating Thread Using *Threading* Module:

To execute another string utilizing the threading module, you need to do the accompanying:

- Define a new subclass of the *Thread* class.
- Override the *__init_(self [,args])* technique to include extra contentions.
- Then, override the run(self [,args]) strategy to execute what the string ought to do when began.

When you have made the new Thread subclass, you can make a case of it and after that begin another string by conjuring the begin(), which thusly calls run() strategy.

Example

```
#!/usr/bin/python
import threading
import time
exitFlag = 0
class myThread (threading.Thread):
def __init__(self, threadID, name, counter):
threading.Thread.__init__(self)
self.threadID = threadID
self.name = name
self.counter = counter
def run(self):
print "Starting " + self.name
print_time(self.name, self.counter, 5)
print "Exiting " + self.name
def print_time(threadName, delay, counter):
while counter:
if exitFlag:

thread.exit()
time.sleep(delay)
print "%s: %s" % (threadName, time.ctime(time.time()))
counter -= 1
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
# Start new Threads
thread1.start()
thread2.start()
print "Exiting Main Thread"
```

At the point when the above code is executed, it delivers the accompanying outcome:

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1: Thu Mar 21 09:10:03 2013
Thread-1: Thu Mar 21 09:10:04 2013
Thread-2: Thu Mar 21 09:10:04 2013
Thread-1: Thu Mar 21 09:10:05 2013
Thread-1: Thu Mar 21 09:10:06 2013
Thread-2: Thu Mar 21 09:10:06 2013
Thread-1: Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2: Thu Mar 21 09:10:08 2013
Thread-2: Thu Mar 21 09:10:10 2013
Thread-2: Thu Mar 21 09:10:12 2013
Exiting Thread-2
```

Synchronizing Threads

The threading module given Python incorporates an easy to-execute locking system that enables you to synchronize strings. Another bolt is made by calling theLock() technique, which restores the new bolt.

The acquire(blocking) strategy for the new bolt protest is utilized to drive strings to run synchronously. The discretionary blocking parameter empowers you to control whether the string holds up to secure the bolt.

On the off chance that blocking is set to 0, the string returns instantly with a 0 esteem if the bolt can't be obtained and with a 1 if the bolt was gained. In the event that blocking is set to 1, the string squares and sits tight for the bolt to be discharged.

The release() technique for the new bolt protest is utilized to discharge the bolt when it is never again required.

Example

```
#!/usr/bin/python
import threading
```

```
import time
class myThread (threading.Thread):
def __init__(self, threadID, name, counter):
threading.Thread.__init__(self)
self.threadID = threadID
self.name = name
self.counter = counter
def run(self):
print "Starting " + self.name
# Get lock to synchronize threads
threadLock.acquire()
print_time(self.name, self.counter, 3)
# Free lock to release next thread
threadLock.release()
def print_time(threadName, delay, counter):
while counter:
time.sleep(delay)
print "%s: %s" % (threadName, time.ctime(time.time()))
counter -= 1
threadLock = threading.Lock()
threads = []
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list

threads.append(thread1)
threads.append(thread2)
# Wait for all threads to complete
for t in threads:
t.join()
print "Exiting Main Thread"
```

At the point when the above code is executed, it delivers the accompanying outcome:

Starting Thread-1

Starting Thread-2

Thread-1: Thu Mar 21 09:11:28 2013

Thread-1: Thu Mar 21 09:11:29 2013

Thread-1: Thu Mar 21 09:11:30 2013

Thread-2: Thu Mar 21 09:11:32 2013

Thread-2: Thu Mar 21 09:11:34 2013

Thread-2: Thu Mar 21 09:11:36 2013

Exiting Main Thread

Multithreaded Priority Queue

The Queue module enables you to make another line question that can hold a particular number of things. There are following strategies to control the Queue:

- **get():**The get() expels and restores a thing from the line.
- **put():**The put adds thing to a line.
- **qsize() :** The qsize() restores the quantity of things that are at present in the line.
- **empty():**The empty( ) returns True if line is unfilled; something else, False.
- **full():**the full() returns True if line is full; generally, False.

Example

```
#!/usr/bin/python
import Queue
import threading
import time
exitFlag = 0
class myThread (threading.Thread):
def __init__(self, threadID, name, q):
threading.Thread.__init__(self)
self.threadID = threadID
self.name = name
self.q = q
def run(self):
print "Starting " + self.name
process_data(self.name, self.q)
print "Exiting " + self.name
def process_data(threadName, q):
while not exitFlag:
queueLock.acquire()
if not workQueue.empty():
data = q.get()
queueLock.release()
print "%s processing %s" % (threadName, data)
```

```
else:
queueLock.release()
time.sleep(1)
threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
threads = []
threadID = 1
# Create new threads
for tName in threadList:
thread = myThread(threadID, tName, workQueue)
thread.start()
threads.append(thread)
threadID += 1
# Fill the queue
queueLock.acquire()
for word in nameList:
workQueue.put(word)
queueLock.release()
# Wait for queue to empty
while not workQueue.empty():
pass
# Notify threads it's time to exit
exitFlag = 1
# Wait for all threads to complete
for t in threads:
t.join()
print "Exiting Main Thread"
```

At the point when the above code is executed, it delivers the accompanying outcome:

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-1 processing One
Thread-2 processing Two
Thread-3 processing Three
Thread-1 processing Four
Thread-2 processing Five
Exiting Thread-3
```

Exiting Thread-1
Exiting Thread-2
Exiting Main Thread

6.8 GUI Programming

Python gives different alternatives to creating graphical UIs (GUIs). Most vital are recorded underneath:

- **Tkinter:** Tkinter is the Python interface to the Tk GUI toolbox dispatched with Python. We would look this alternative in this part.
- **wxPython:** This is an open-source Python interface for wxWindowshttp:// wxpython.org.
- **JPython:** JPython is a Python port for Java which gives Python scripts consistent access to Java class libraries on the nearby machine http:// www.jython.org.

There are numerous different interfaces accessible, which you can discover them on the net.

Tkinter Programming

Tkinter is the standard GUI library for Python. Python when consolidated with Tkinter gives a quick and simple approach to make GUI applications. Tkinter gives an intense question situated interface to the Tk GUI toolbox.

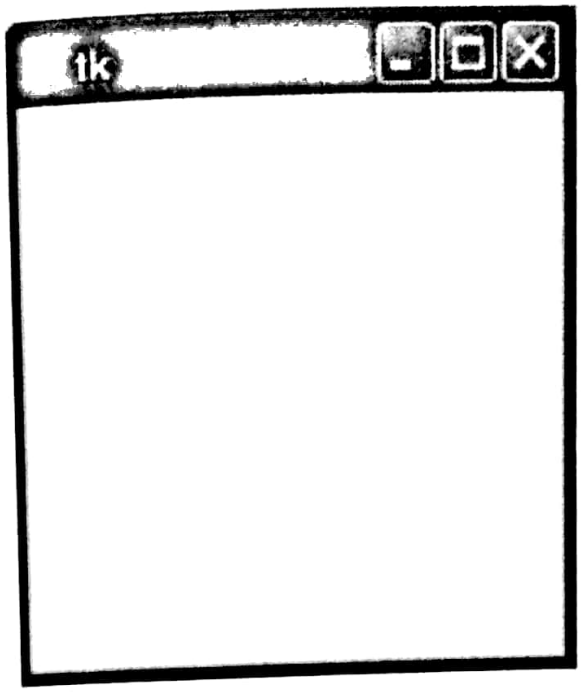
Making a GUI application utilizing Tkinter is a simple assignment. You should simply play out the accompanying strides:

- Import the *Tkinter* module.
- Create the GUI application principle window.
- Add at least one of the previously mentioned gadgets to the GUI application.
- Enter the headliner circle to make a move against every occasion activated by the client.

Example

```
#!/usr/bin/python
import Tkinter
top = Tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

This would make a following window:



Tkinter Widgets

Tkinter gives different controls, for example, catches, marks and content boxes utilized as a part of a GUI application. These controls are regularly called gadgets.

There are at present 15 sorts of gadgets in Tkinter. We introduce these gadgets and additionally a concise depiction in the accompanying table:

Operator	Description
Button	The Button widget is utilized to show catches in your application.
Canvas	The Canvas gadget is utilized to draw shapes, for example, lines, ovals, polygons and rectangles, in your application.
Checkbutton	The Checkbutton gadget is utilized to show various choices as checkboxes. The client can choose different choices at once.
Entry	The Entry gadget is utilized to show a solitary line content field for tolerating esteems from a client.
Frame	The Frame gadget is utilized as a compartment gadget to sort out different gadgets.
Label	The Label gadget is utilized to give a solitary line inscription to different gadgets. It can likewise contain pictures.
Listbox	The Listbox gadget is utilized to give a rundown of alternatives to a client.

Operator	Description
Menubutton	The Menubutton widget is used to display menus in your application.
Menu	The Menu gadget is utilized to give different orders to a client. These summonses are contained inside Menubutton.
Message	The Message gadget is utilized to show multiline content fields for tolerating esteems from a client.
Radiobutton	The Radiobutton gadget is utilized to show various choices as radio catches. The client can choose just a single choice at any given moment.
Scale	The Scale widget is utilized to give a slider gadget.
Scrollbar	The Scrollbar gadget is utilized to add looking over ability to different gadgets, for example, list boxes.
Text	The Text gadget is utilized to show message in numerous lines.
Toplevel	The Toplevel gadget is utilized to give a different window holder.
Spinbox	The Spinbox gadget is a variation of the standard Tkinter Entry gadget, which can be utilized to choose from a settled number of qualities.
PanedWindow	A PanedWindow is a compartment gadget that may contain any number of sheets, organized evenly or vertically.
LabelFrame	A labelframe is a basic holder gadget. Its main role is to go about as a spacer or holder for complex window designs.
tkMessageBox	This module is utilized to show message encloses your applications.

Give us a chance to examine these widgets in detail:

1. Button

The Button gadget is utilized to include catches in a Python application. These catches can show content or pictures that pass on the motivation behind the catches. You can append a capacity or a strategy to a catch which is called naturally when you tap the catch.

Syntax

Here is the simple structure to make this widget:

```
w = Button ( master, option=value, ... )
```

Parameters

- **master:** This speaks to the parent window..
- **options:** Here is the rundown of most normally utilized choices for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

Option	Description
activebackground	Foundation shading when the catch is under the cursor.
activeforeground	Frontal area shading when the catch is under the cursor
bd	Outskirt width in pixels. Default is 2.
bg	Typical foundation shading.
command	Capacity or strategy to be called when the catch is clicked.
fg	Ordinary closer view (content) shading.
font	Content text style to be utilized for the catch's mark.
height	Stature of the catch in content lines (for literary catches) or pixels (for pictures).
highlightcolor	The shade of the concentration highlight when the gadget has center.
image	Picture to be shown on the catch (rather than content).
justify	Instructions to demonstrate various content lines: LEFT to left-legitimize each line; CENTER to focus them; or RIGHT to right-legitimize.
padx	Extra cushioning left and right of the content.
pady	Extra cushioning above and beneath the content.
relief	Help determines the kind of the fringe. A portion of the qualities are SUNKEN, RAISED, GROOVE, and RIDGE.
state	Set this alternative to DISABLED to dark out the catch and make it lethargic. Has the esteem ACTIVE when the mouse is over it. Default is NORMAL.

Option	Description
underline	Default is - 1, implying that no character of the content on the catch will be underlined. In the event that nonnegative, the relating content character will be underlined.
width	Width of the catch in letters (if showing content) or pixels (if showing a picture).
wraplength	In the event that this esteem is set to a positive number, the content lines will be wrapped to fit inside this length.

Methods

Following are usually utilized strategies for this widget:

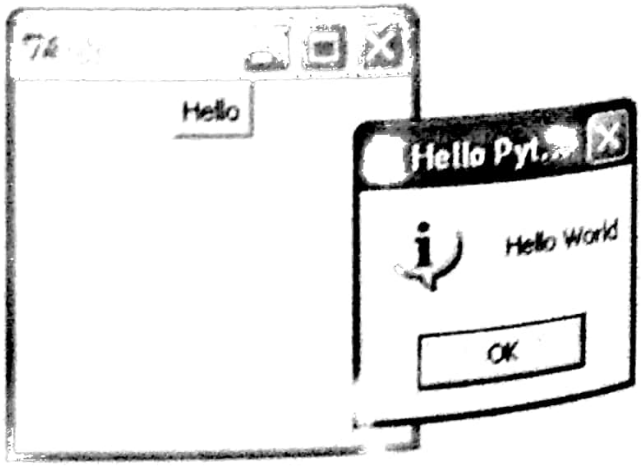
Method	Description
flash()	Causes the catch to streak a few times amongst dynamic and ordinary hues. Leaves the catch in the state it was in initially. Disregarded if the catch is handicapped.
invoke()	Calls the catch's callback, and returns what that capacity returns. Has no impact if the catch is debilitated or there is no callback.

Example

Attempt the accompanying case yourself:

```
import Tkinter
import tkMessageBox
top = Tkinter.Tk()
def helloCallBack():
tkMessageBox.showinfo( "Hello Python", "Hello World")
B = Tkinter.Button(top, text ="Hello", command =
helloCallBack)
B.pack()
top.mainloop()
```

When the above code is executed, it produces the following result:



2. Canvas

The Canvas is a rectangular region planned for drawing pictures or other complex formats. You can put designs, content, widgets2. or, on the other hand outlines on a Canvas.

Syntax

Here is the basic sentence structure to make this gadget:

```
w = Canvas ( master, option=value, ... )
```

Parameters

- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most regularly utilized choices for this gadget. These alternatives can be utilized as key-esteem sets isolated by commas.

Option	Description
bd	Outskirt width in pixels. Default is 2.
bg	Typical foundation shading.
confine	Assuming genuine (the default), the canvas can't be looked outside of the scrollregion..
cursor	Cursor utilized as a part of the canvas like bolt, circle, spot and so on.
height	Size of the canvas in the Y measurement.
highlightcolor	Shading appeared in the concentration highlight.
relief	Alleviation determines the kind of the fringe. A portion of the qualities are SUNKEN, RAISED, GROOVE, and RIDGE.
scrollregion	A tuple (w, n, e, s) that characterizes over how vast a zone the canvas can be looked over, where w is the left side, n the top, e the correct side, and s the base.
width	Size of the canvas in the X measurement.
xscrollincrement	On the off chance that you set this choice to some positive measurement, the canvas can be situated just on products of that separation and the esteem will be utilized for looking by looking over units, for example, when the client taps on the bolts at the closures of a scrollbar.

Option	Description
xscrollcommand	In the event that the canvas is scrollable, this property ought to be the .set() technique for the level scrollbar.
yscrollincrement	Works like xscroll increment, however administers vertical development.
yscrollcommand	On the off chance that the canvas is scrollable, this characteristic ought to be the .set() strategy for the vertical scrollbar.

The Canvas widget can bolster the accompanying standard things::

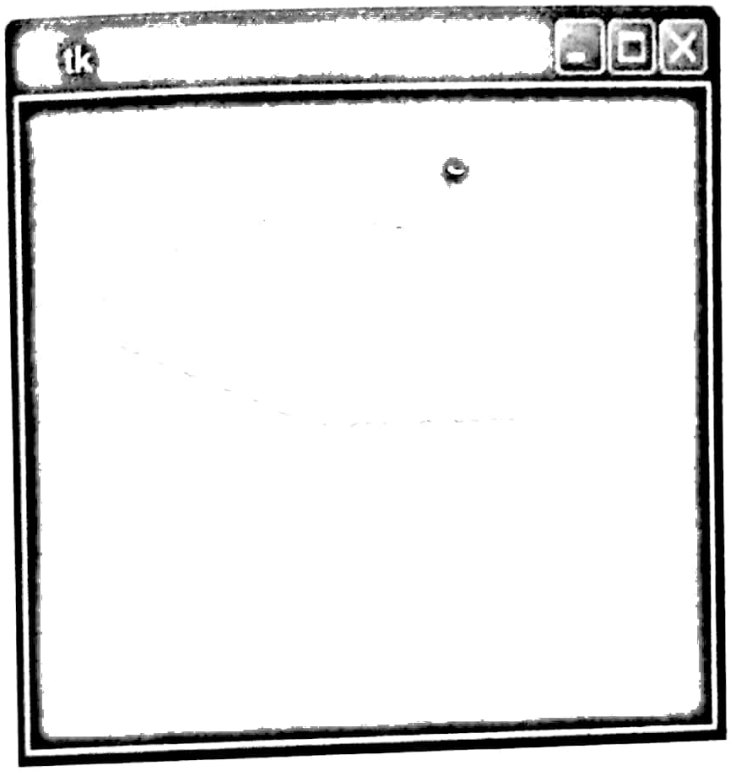
```
arc . Makes a bend thing, which can be a harmony, a
pieslice or a basic curve.
coord = 10, 50, 240, 210
arc = canvas.create_arc(coord, start=0, extent=150,
fill="blue")
image . Makes a picture thing, which can be an example of
either the BitmapImage or the PhotoImage classes.
filename = PhotoImage(file = "sunshine.gif")
image = canvas.create_image(50, 50, anchor=NE,
image=filename)
line . Creates a line item.
line = canvas.create_line(x0, y0, x1, y1, ..., xn, yn,
options)
oval . Makes a circle or an oval at the given directions.
It takes two sets of directions; the upper left and base
right corners of the bouncing rectangle for the oval.
oval = canvas.create_oval(x0, y0, x1, y1, options)
polygon . Makes a polygon thing that must have no less than
three vertices
oval = canvas.create_polygon(x0, y0, x1, y1,...xn, yn,
options)
```

Example

Attempt the accompanying case yourself:

```
import Tkinter
import tkMessageBox
top = Tkinter.Tk()
C = Tkinter.Canvas(top, bg="blue", height=250, width=300)
coord = 10, 50, 240, 210
arc = C.create_arc(coord, start=0, extent=150, fill="red")
C.pack()
top.mainloop()
```

At the point when the above code is executed, it creates the accompanying outcome:



3. Checkbutton

The Checkbutton widget is utilized to show various choices to a client as flip catches. The client would then be able to choose at least one alternative by tapping the catch relating to every choice.

You can likewise show pictures set up of content.

Syntax

Here is the basic sentence structure to make this widdget:

```
w = Checkbutton ( master, option, ... )
```

Parameters

- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most usually utilized alternatives for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

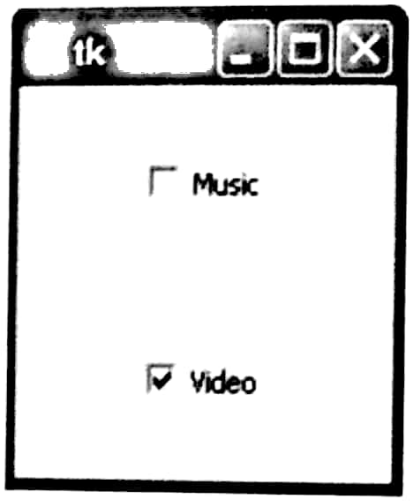
Example

Try the following example yourself:

```
from Tkinter import *
import tkMessageBox
import Tkinter
top = Tkinter.Tk()
CheckVar1 = IntVar()
```

```
CheckVar2 = IntVar()
C1 = Checkbutton(top, text = "Music", variable = CheckVar1,
\
onvalue = 1, offvalue = 0, height=5, \
width = 20)
C2 = Checkbutton(top, text = "Video", variable = CheckVar2,
\
onvalue = 1, offvalue = 0, height=5, \
width = 20)
C1.pack()
C2.pack()
top.mainloop()
```

When the above code is executed, it produces the following result:



4. Entry

The Entry gadget is utilized to acknowledge single-line content strings from a client.

- If you need to show different lines of content that can be altered, at that point you should utilize the Text widget.

If you need to show at least one lines of content that can't be altered by the client, at that point you should utilize the Label widget.

Syntax

Here is the straightforward language structure to make this widget:

```
w = Entry( master, option, ... )
```

Parameters

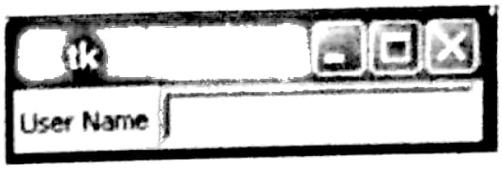
- **master:** This speaks to the parent window.

Example

Try the following example yourself:

```
from Tkinter import *
top = Tk()
L1 = Label(top, text="User Name")
L1.pack( side = LEFT)
E1 = Entry(top, bd =5)
E1.pack(side = RIGHT)
top.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



5. Frame

The Frame widget is essential for the way toward gathering and sorting out different gadgets in a by one means or another cordial way. It works like a holder, which is in charge of orchestrating the position of different gadgets.

It utilizes rectangular zones in the screen to compose the design and to give cushioning of these gadgets. An edge can likewise be utilized as an establishment class to execute complex widgets.

Syntax

Here is the straightforward language structure to make this widget:

```
w = Frame ( master, option, ... )
```

Parameters:

- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most usually utilized choices for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

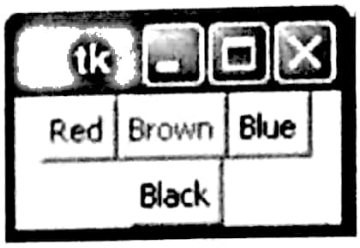
Example

Try the following example yourself:

```
from Tkinter import *
root = Tk()
frame = Frame(root)
```

```
frame.pack()
bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )
redbutton = Button(frame, text="Red", fg="red")
redbutton.pack( side = LEFT)
greenbutton = Button(frame, text="Brown", fg="brown")
greenbutton.pack( side = LEFT )
bluebutton = Button(frame, text="Blue", fg="blue")
bluebutton.pack( side = LEFT )
blackbutton = Button(bottomframe, text="Black", fg="black")
blackbutton.pack( side = BOTTOM)
root.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



6. Label

This widget executes a show box where you can put content or pictures. The content shown by this gadget can be refreshed whenever you need.

It is additionally conceivable to underline some portion of the content (jump at the chance to distinguish a console alternate route) and traverse the content over numerous lines.

Syntax

Here is the straightforward sentence structure to make this gadget:

```
w = Label ( master, option, ... )
```

Parameters

- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most generally utilized choices for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

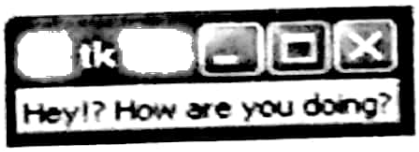
Example

Try the following example yourself:

```
from Tkinter import *
root = Tk()
```

```
var = StringVar()
label = Label( root, textvariable=var, relief=RAISED )
var.set("Hey!? How are you doing?")
label.pack()
root.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



7. Listbox

The Listbox widget is utilized to show a rundown of things from which a client can choose various things.

Syntax

Here is the straightforward linguistic structure to make this gadget:

```
w = Listbox ( master, option, ... )
```

Parameters

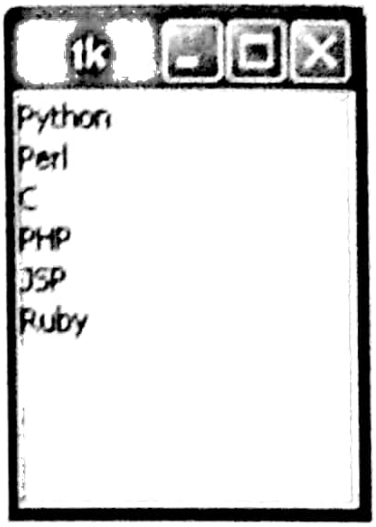
- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most usually utilized choices for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

Example

Try the following example yourself:

```
from Tkinter import *
import tkMessageBox
import Tkinter
top = Tk()
Lb1 = Listbox(top)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.insert(4, "PHP")
Lb1.insert(5, "JSP")
Lb1.insert(6, "Ruby")
Lb1.pack()
top.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



8. MenuButton

A menubutton is the piece of a drop-down menu that stays on the screen constantly. Each menubutton is related with a Menu gadget that can show the decisions for that menubutton when the client taps on it.

Syntax

Here is the straightforward linguistic structure to make this widget:

```
w = Menubutton ( master, option, ... )
```

Parameters

- **master:** This speaks to the parent window
- **options:** Here is the rundown of most ordinarily utilized alternatives for this gadget. These alternatives can be utilized as key-esteem sets isolated by commas.

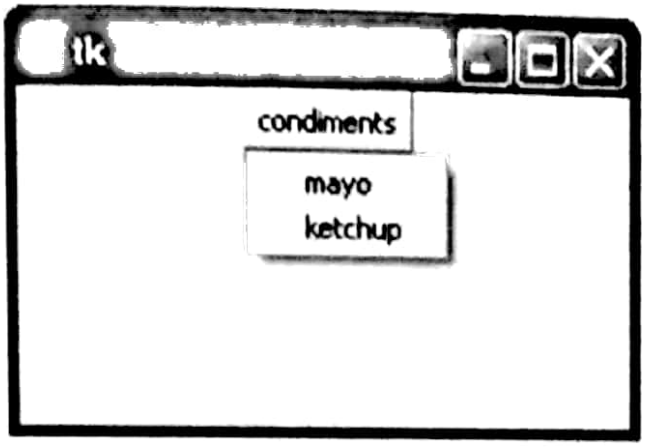
Example

Try the following example yourself:

```
from Tkinter import *
import tkMessageBox
import Tkinter
top = Tk()
mb= Menubutton ( top, text="condiments", relief=RAISED )
mb.grid()
mb.menu = Menu ( mb, tearoff = 0 )
mb["menu"] = mb.menu
mayoVar = IntVar()
ketchVar = IntVar()
```

```
mb.menu.add_checkbutton ( label="mayo",
variable=mayoVar )
mb.menu.add_checkbutton ( label="ketchup",
variable=ketchVar )
mb.pack()
top.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



9. Menu

The objective of this gadget is to enable us to make a wide range of menus that can be utilized by our applications. The center usefulness gives approaches to make three menu sorts: fly up, toplevel and pull-down.

It is additionally conceivable to utilize other stretched out gadgets to actualize new sorts of menus, for example, the OptionMenu gadget, which executes an uncommon sort that creates a fly up rundown of things inside a choice.

Syntax

Here is the straightforward linguistic structure to make this widget:

```
w = Menu ( master, option, ... )
```

Parameters

- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most usually utilized alternatives for this gadget. These alternatives can be utilized as key-esteem sets isolated by commas.

Example

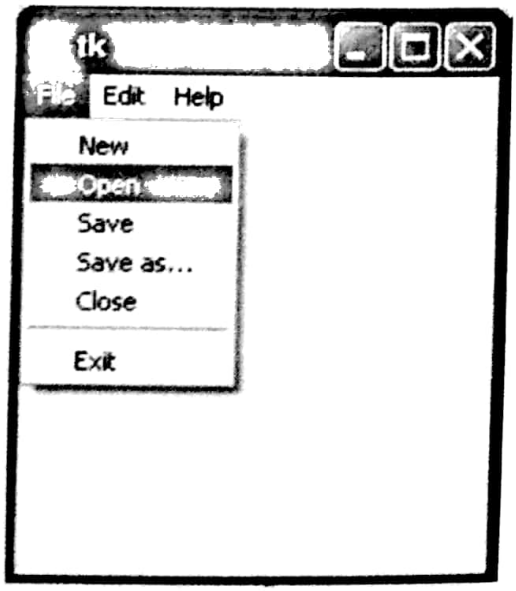
Try the following example yourself:

```
from Tkinter import *
def donothing():
```

```
filewin = Toplevel(root)
button = Button(filewin, text="Do nothing button")
button.pack()
root = Tk()
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New", command=donothing)
filemenu.add_command(label="Open", command=donothing)
filemenu.add_command(label="Save", command=donothing)
filemenu.add_command(label="Save as...", command=donothing)
filemenu.add_command(label="Close", command=donothing)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="File", menu=filemenu)
editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Undo", command=donothing)
editmenu.add_separator()
editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
editmenu.add_command(label="Paste", command=donothing)
editmenu.add_command(label="Delete", command=donothing)
editmenu.add_command(label="Select All", command=donothing)
menubar.add_cascade(label="Edit", menu=editmenu)
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)
menubar.add_cascade(label="Help", menu=helpmenu)

root.config(menu=menubar)
root.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



10. Message

This widget gives a multiline and noneditable question that presentations writings, naturally breaking lines and defending their substance.

Its usefulness is fundamentally the same as the one given by the Label gadget, with the exception of that it can likewise consequently wrap the content, keeping up a given width or viewpoint proportion.

Syntax

Here is the straightforward punctuation to make this widget:

```
w = Message ( master, option, ... )
```

Parameters

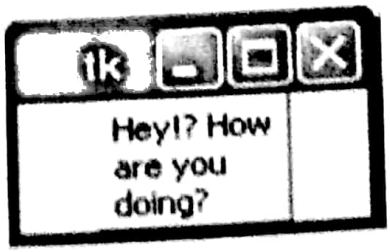
- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most usually utilized alternatives for this widget. These choices can be utilized as key-esteem sets isolated by commas.

Example

Try the following example yourself:

```
from Tkinter import *
root = Tk()
var = StringVar()
label = Message( root, textvariable=var, relief=RAISED )
var.set("Hey!? How are you doing?")
label.pack()
root.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



11. Radiobutton

This widget executes a different decision catch, which is an approach to offer numerous conceivable determinations to the client and gives client a chance to pick just a single of them.

Keeping in mind the end goal to execute this usefulness, each gathering of radiobuttons must be related to a similar variable and every one of the catches must symbolize a solitary esteem. You can utilize the Tab key to change starting with one radionbutton then onto the next.

Syntax

Here is the basic language structure to make this widget:

```
w = Radiobutton ( master, option, ... )
```

Parameters

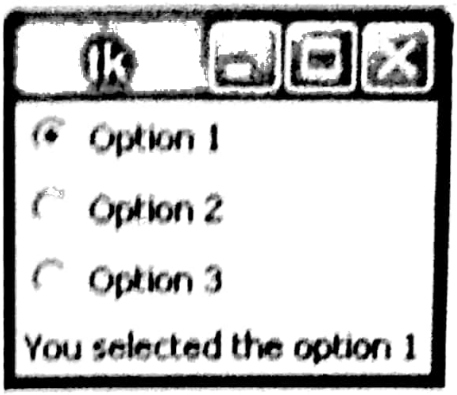
- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most normally utilized alternatives for this gadget. These alternatives can be utilized as key-esteem sets isolated by commas.

Example:

Try the following example yourself:

```
from Tkinter import *
def sel():
selection = "You selected the option " + str(var.get())
label.config(text = selection)
root = Tk()
var = IntVar()
R1 = Radiobutton(root, text="Option 1", variable=var,
value=1,
command=sel)
R1.pack( anchor = W )
R2 = Radiobutton(root, text="Option 2", variable=var,
value=2,
command=sel)
R2.pack( anchor = W )
R3 = Radiobutton(root, text="Option 3", variable=var,
value=3,
command=sel)
R3.pack( anchor = W)
label = Label(root)
label.pack()
root.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



12. Scale

The Scale gadget gives a graphical slider question that enables you to choose esteems from a particular scale.

Syntax

Here is the straightforward grammar to make this gadget:

```
w = Scale ( master, option, ... )
```

Parameters

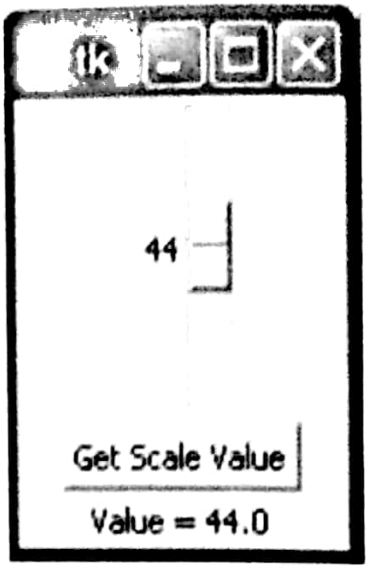
- **master:** This represents the parent window.
- **options:** Here is the rundown of most regularly utilized choices for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

Example

Try the following example yourself:

```
from Tkinter import *
def sel():
selection = "Value = " + str(var.get())
label.config(text = selection)
root = Tk()
var = DoubleVar()
scale = Scale( root, variable = var )
scale.pack(anchor=CENTER)
button = Button(root, text="Get Scale Value", command=sel)
button.pack(anchor=CENTER)
label = Label(root)
label.pack()
root.mainloop()
```

At the point when the above code is executed, it delivers the accompanying outcome:



13. Scrollbar

This widget gives a slide controller that is utilized to actualize vertical looked over gadgets, for example, Listbox, Text and Canvas. Note that you can likewise make level scrollbars on Entry gadgets.

Syntax:

Here is the basic sentence structure to make this widget:

```
w = Scrollbar ( master, option, ... )
```

Parameters

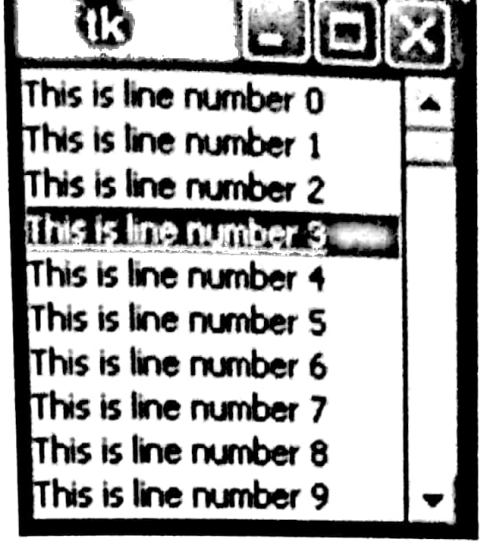
- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most normally utilized choices for this gadget. These alternatives can be utilized as key-esteem sets isolated by commas.

Example

Try the following example yourself:

```
from Tkinter import *
root = Tk()
scrollbar = Scrollbar(root)
scrollbar.pack( side = RIGHT, fill=Y )
mylist = Listbox(root, yscrollcommand = scrollbar.set )
for line in range(100):
mylist.insert(END, "This is line number " + str(line))
mylist.pack( side = LEFT, fill = BOTH )
scrollbar.config( command = mylist.yview )
mainloop()
```

When the above code is executed, it produces the followi g result:



14. Text

Content widgets give propelled capacities that enable you to alter a multiline content and configuration the way it must be shown, for example, changing its shading and text style.

You can likewise utilize rich structures like tabs and imprints to find particular segments of the content, and apply changes to those regions. Additionally, you can implant windows and pictures in the content since this gadget was intended to deal with both plain and designed content.

Syntax

Here is the straightforward linguistic structure to make this widget:

```
w = Text ( master, option, ... )
```

Parameters

- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most ordinarily utilized choices for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

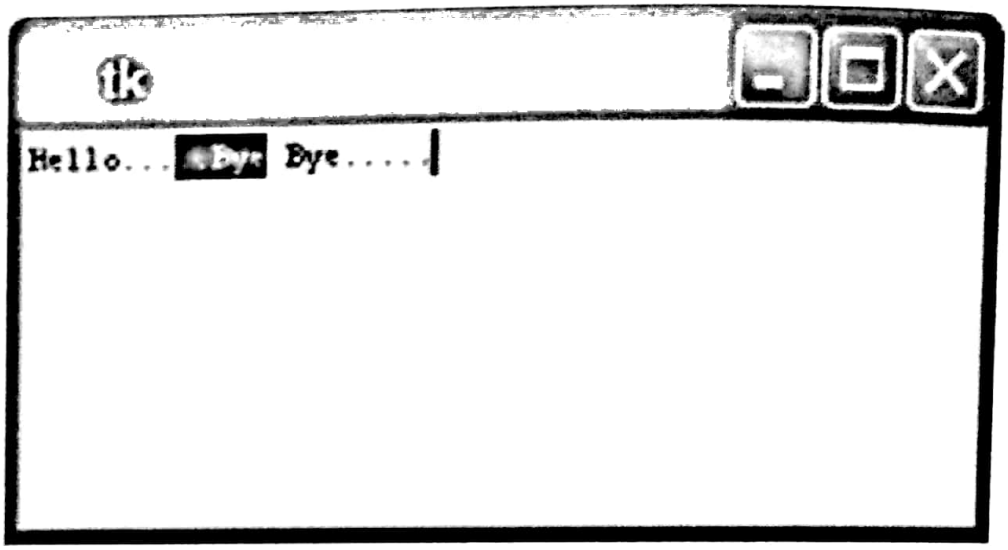
Example

Try the following example yourself:

```
from Tkinter import *
def onclick():
pass
root = Tk()
text = Text(root)
text.insert(INSERT, "Hello.....")
text.insert(END, "Bye Bye.....")
text.pack()
```

```
text.tag_add("here", "1.0", "1.4")
text.tag_add("start", "1.8", "1.13")
text.tag_config("here", background="yellow",
foreground="blue")
text.tag_config("start", background="black",
foreground="green")
root.mainloop()
```

When the above code is executed, it produces the following result:



15. TopLevel

Toplevel gadgets fill in as windows that are specifically overseen by the window director. They don't really have a parent gadget on top of them.

Your application can utilize any number of top-level windows.

Syntax

Here is the straightforward punctuation to make this gadget:

```
w = Toplevel ( option, ... )
```

Parameters:

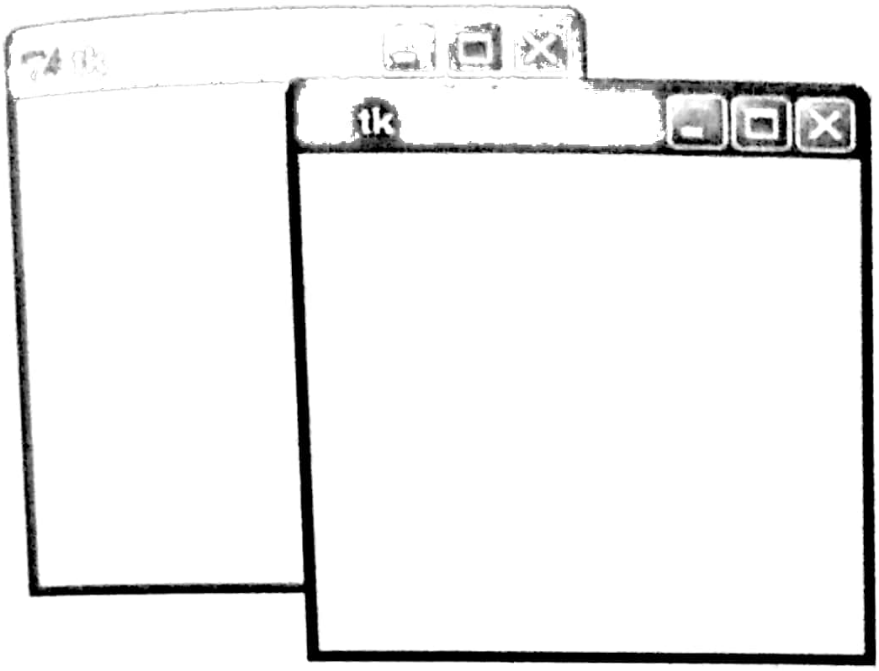
- **options:** Here is the rundown of most ordinarily utilized alternatives for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

Example

Try following example yourself:

```
from Tkinter import *
root = Tk()
top = Toplevel()
top.mainloop()
```

When the above code is executed, it produces the following result:



16. SpinBox

The Spinbox gadget is a variation of the standard Tkinter Entry gadget, which can be utilized to choose from a settled number of qualities.

Syntax

Here is the straightforward linguistic structure to make this widget:

```
w = Spinbox( master, option, ... )
```

Parameters

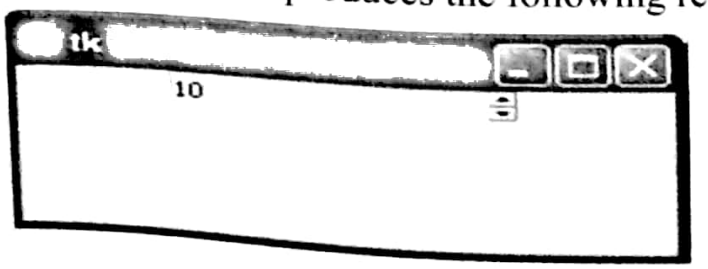
- master: This speaks to the parent window.
- options: Here is the rundown of most regularly utilized choices for this gadget. These choices can be utilized as key-esteem sets isolated by commas.

Example

Try the following example yourself:

```
from Tkinter import *
master = Tk()
w = Spinbox(master, from_=0, to=10)
w.pack()
mainloop()
```

When the above code is executed, it produces the following result:



17. PanelWindow

A PanedWindow is a holder widget that may contain any number of sheets, organized evenly or vertically.

Every sheet contains one gadget and each combine of sheets is isolated by a moveable (by means of mouse developments) band. Moving a scarf causes the gadgets on either side of the band to be resized.

Syntax

Here is the straightforward language structure to make this gadget:

```
w = PanedWindow( master, option, ... )
```

Parameters

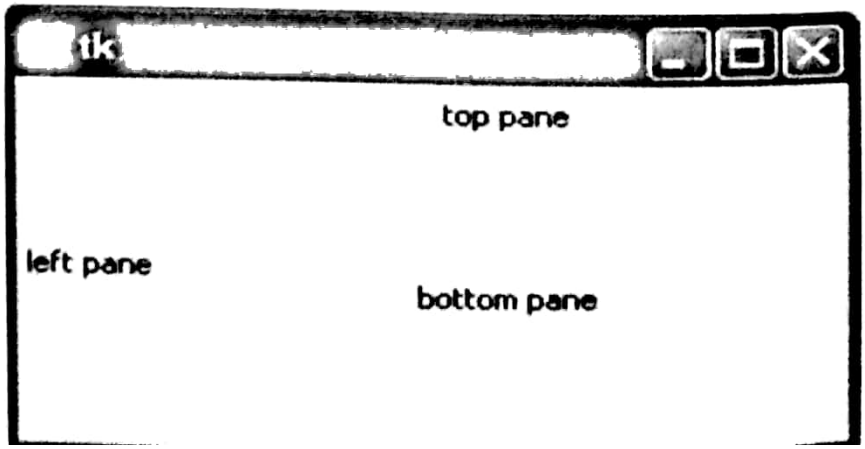
- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most normally utilized choices for this gadget. These alternatives can be utilized as key-esteem sets isolated by commas.

Example

Try the following example yourself. Here's how to create a 3-pane widget:

```
from Tkinter import *
m1 = PanedWindow()
m1.pack(fill=BOTH, expand=1)
left = Label(m1, text="left pane")
m1.add(left)
m2 = PanedWindow(m1, orient=VERTICAL)
m1.add(m2)
top = Label(m2, text="top pane")
m2.add(top)
bottom = Label(m2, text="bottom pane")
m2.add(bottom)
mainloop()
```

When the above code is executed, it produces the following result:



20. LabelFrame

A labelframe is a basic holder gadget. Its basic role is to go about as a spacer or compartment for complex window designs.

This widget has the components of an edge in addition to the capacity to show a name.

Syntax

Here is the basic sentence structure to make this widget:

```
w = LabelFrame( master, option, ... )
```

Parameters

- **master:** This speaks to the parent window.
- **options:** Here is the rundown of most generally utilized choices for this gadget. These alternatives can be utilized as key-esteem sets isolated by commas.

Example

Try the following example yourself. Here is how to create a labelframe widget:

```
from Tkinter import *
root = Tk()
labelframe = LabelFrame(root, text="This is a LabelFrame")
labelframe.pack(fill="both", expand="yes")
left = Label(labelframe, text="Inside the LabelFrame")
left.pack()
root.mainloop()
```

When the above code is executed, it produces the following result:

23. tkMessageBox

The tkMessageBox module is utilized to show message confines your applications. This module gives various capacities that you can use to show a proper message.

Some of these capacities are showinfo, showwarning, showerror, askquestion, askokcancel, askyesno, and askretryignore.

Syntax:

Here is the basic punctuation to make this widget:

```
tkMessageBox.FunctionName(title, message [, options])
```

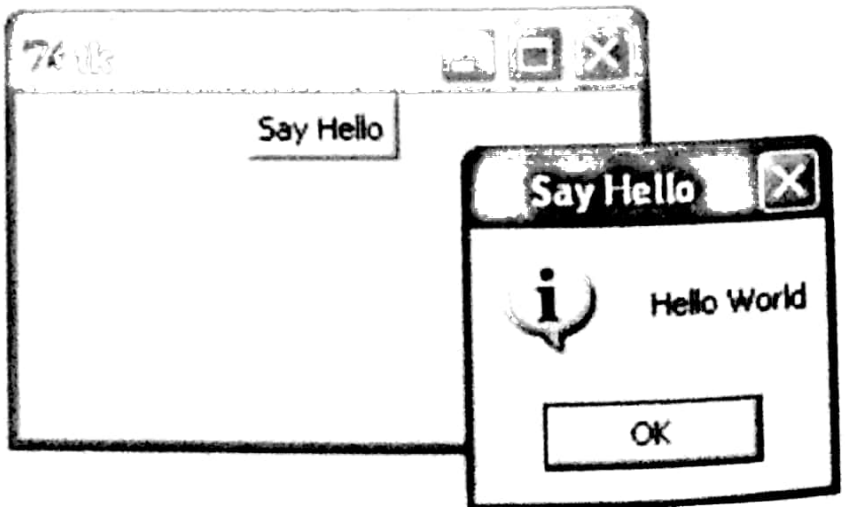
Parameters

- **FunctionName:** This is the name of the suitable message box work.
- **title:** This is the content to be shown in the title bar of a message box.
- **message:** This is the content to be shown as a message.
- **options:** choices are elective decisions that you may use to tailor a standard message box. A portion of the choices that you can utilize are default and parent. The default choice is utilized to determine the default catch, for example, ABORT, RETRY, or IGNORE in the message box. The parent choice is utilized to indicate the window on top of which the message box is to be shown.

Attempt the accompanying illustration yourself:

```
import Tkinter
import tkMessageBox
top = Tkinter.Tk()
def hello():
tkMessageBox.showinfo("Say Hello", "Hello World")
B1 = Tkinter.Button(top, text = "Say Hello", command =
hello)
B1.pack()
top.mainloop()
```

At the point when the above code is executed, it creates the accompanying outcome:



6.9 Turtle Graphics

There are numerous Python packages that can be utilized to make illustrations and GUI's. Two illustrations modules, called turtle and tkinter, come as a piece of Python's standard library. tkinter is principally intended for making GUI's. Indeed, IDLE is assembled utilizing tkinter. In any case, we will concentrate on the turtle module that is basically utilized as a straightforward illustrations bundle however can likewise be utilized to make basic GUI's.

Turtle Basics

In addition to other things, the strategies in the turtle module enable us to draw pictures. The thought behind the turtle some portion of "turtle illustrations" depends on an analogy. Envision you have a turtle on a canvas that is holding a pen. The pen can be either up (not touching the canvas) or down (touching the canvas). Presently think about the turtle as a robot that you can control by issuing charges. At the point when the pen it holds is down, the turtle leaves a trail when you instruct it to move to another area. At the point when the pen is up, the turtle moves to another position however no trail is cleared out. Notwithstanding position, the turtle likewise has a heading, i.e., a course, of forward development. The turtle module gives orders that can set the turtle's position and heading, control its forward and in reverse development, indicate the kind of pen it is holding, and so on. By controlling the development and introduction of the turtle and also the pen it is holding, you can make drawings from the trails the turtle clears out.

Importing Turtle Graphics

With a specific end goal to begin utilizing turtle illustrations, we have to import the turtle module. Begin Python/IDLE and sort the accompanying:

1. **Importing the turtle module.**

```
>>> import turtle as t
```

This imports the turtle module utilizing the identifier t. By bringing in the module thusly we get to the strategies inside the module utilizing t.<object> rather than turtle.<object>.

To guarantee that the module was appropriately transported in, utilize the dir() work as appeared in beneath 2.

2. Using dir() to view the turtle module's methods and attributes.

```
>>> dir(t)
['Canvas', 'Pen', 'RawPen', 'RawTurtle', 'Screen',
'ScrolledCanvas',
'Shape', 'TK', 'TNavigator', 'TPen', 'Tbuffer', 'Termi-
nator',
'Turtle', 'TurtleGraphicsError', 'TurtleScreen',
'TurtleScreenBase',
'Vec2D', '_CFG', '_LANGUAGE', '_Root', '_Screen',
'_TurtleImage',
'__all__', '__builtins__', '__cached__', '__doc__',
'__file__',
... <<MANY LINES OF OUTPUT DELETED>>
'window_width', 'write', 'write_docstringdict', 'xcor',
'ycor']
```

The rundown returned by the dir() work in 2 has been truncated—on the whole, there are 173 things in this rundown. To take in more about any of these characteristics or strategies, you can utilize the assistance() work. For instance, to find out about the forward() technique, enter the announcement appeared in line 1 of beneath 3.

3. Learning about the forward() method using help().

```
>>> help(t.forward)
Help on function forward in module turtle:

forward(distance)
Move the turtle forward by the specified distance.

Aliases: forward | fd

Argument:
distance – a number (integer or float)

Move the turtle forward by the specified distance, in
the direction
the turtle is headed.
...
<<REMAINING OUTPUT DELETED>>
```

From this we learn, as appeared in lines 12 and 13, that this technique moves "the turtle forward by the predetermined separation, toward the path the turtle is going." We likewise discover that there is a shorter name for this technique: fd().

Your First Drawing

How about we start by advising our turtle to draw a line. Try entering the command shown in 4.

4. Drawing a line with a length of 100 units.

```
>>> t.fd(100)
```

As appeared in Fig. 1, an illustrations window ought to show up in which you see a little bolt 100 units to one side of the focal point of the window. A thin dark line is drawn from the focal point of the window to the tail of the bolt. The bolt speaks to our "turtle" and the bearing the bolt is pointing demonstrates the present heading. The fd() technique is a shorthand for the forward() strategy—the two strategies are indistinguishable. fd() takes one whole number contention that determines the quantity of units you need to advance the turtle toward the current heading.3 If you give a negative contention, the turtle moves in reverse the predefined sum. Then again, to go in reverse one can call either in reverse(), back(), or bk().

The default shape for our turtle is a bolt however in the event that we needed to have it resemble a turtle we could sort the summon appeared in 5.

5. Changing the shape of the turtle.

```
>>> t.shape("turtle")
```

This replaces the bolt with a little turtle. We can change the state of our turtle to various other worked in shapes utilizing the shape() technique. We can likewise make custom shapes in spite of the fact that we won't cover that here.

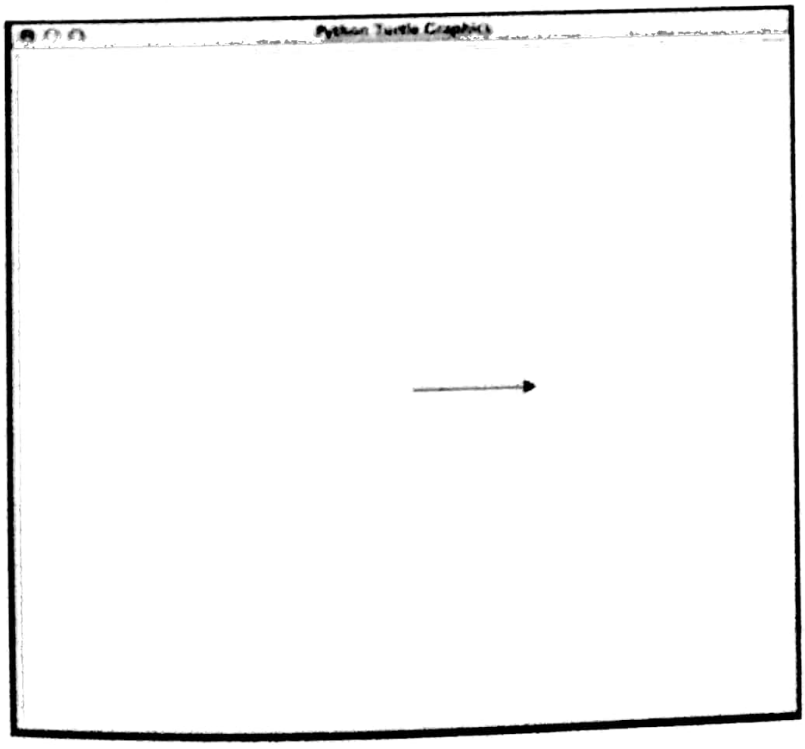


Figure 1: A line of length 100 created by the fd() technique..

6. **Command to hide the turtle's shape from the screen.**

```
>>> t.hideturtle()
```

7. **Making the turtle visible.**

```
>>> t.showturtle()
```

8. **Commands to change the turtle's heading and draw a square box.**

```
1  >>> t.left(90)
2  >>> t.fd(100)
3  >>> t.left(90)
4  >>> t.fd(100)
5  >>> t.left(90)
6  >>> t.fd(100)
```

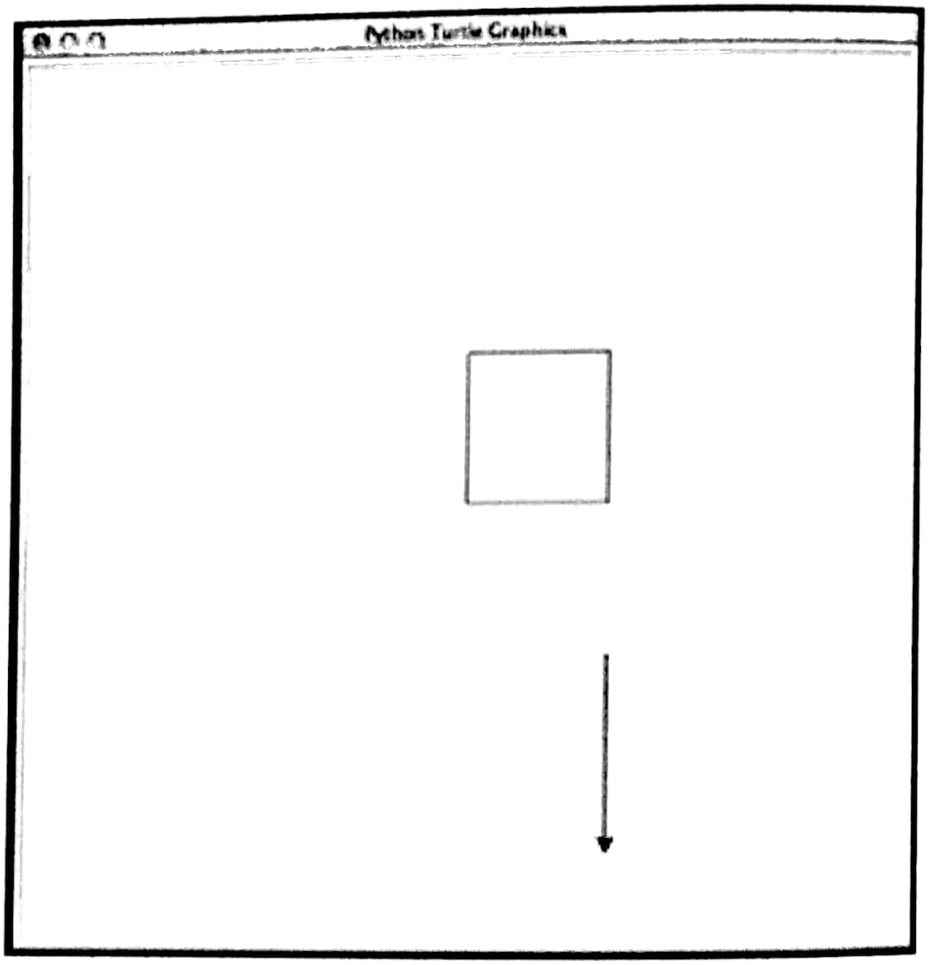


Figure 2: Aftereffect of moving the turtle without drawing a line and afterward, once at the new area, drawing a line.

9. **Using penup() and setposition() to move the turtle without making a line.**

```
1  >>> t.penup()
2  >>> t.setposition(100, -100)
3  >>> t.pendown()
4  >>> t.fd(130)
```

10. **Using the clear() method to clear the image.**

```
>>> t.clear()
```

11. **Resetting the turtle and clearing the image**

```
>>> t.reset()
```

Changing the shade of the turtle's pen.

```
1  >>> t.reset()
2  >>> t.color("blue")
```

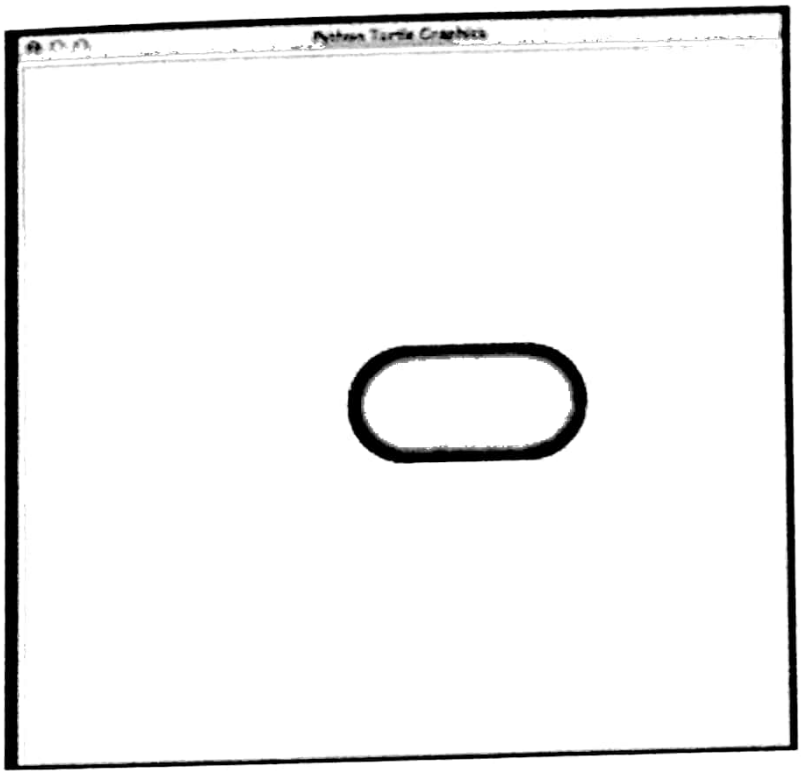


Figure 3: A blue line with a thickness and width of 100..

6.10 Testing

Why testing is required?

The following is given a few reasons which unmistakably demonstrate Why Software Testing is Necessary and what things we need to consider in testing.

Testing is Essential due to the accompanying reasons:

1. Testing is constantly required for accurately comprehend the blame mistakes in programming amid its improvement stages.
2. It is essential since it generally guarantees the clients or clients fulfillment and unwavering quality of the application.
3. It is required in programming advancement to expand the dependability and nature of the product.

4. Testing is expected to give the different offices to the clients like conveyance of great programming or application, bring down upkeep expenses and more precise and solid outcomes too.
5. Testing is fundamental in light of powerful ideal execution of framework and limit use, programming unwavering quality, quality, and framework or application affirmation.
6. It is vital in light of the fact that it is incorporating into the venture arrange, and to remain in business.
7. It is expected to demonstrate that product or application has no deficiencies, since disappointments can be extremely costly in nature.
8. Programming Testing is expected to take in more about the unwavering quality of the product and application.

Basic Concepts of Testing

What is Testing?

The way toward testing the application to ensure that the application is working as per the prerequisites.

What is Software Testing?

Software testing is the way toward executing a program/application under positive and negative conditions by manual or mechanized means. It checks for the Specification

- Functionality
- Performance

Why Software Testing ?

Software Testing is vital as it might cause mission disappointment, affect on operational execution and unwavering quality if not done appropriately.

Compelling programming testing conveys quality programming items fulfilling client's necessities, needs and desires.

Who should test?

- Developer
  - Understands the framework
  - But,will test tenderly

Independent Tester

- Must learn framework
- But, will endeavor to break it

What is an "Error","Bug","Fault" and "Failure"?

A man makes an Error

That makes a blame in programming

That can cause a disappointment in operation

Error : A mistake is a human activity that delivers the mistaken outcome that outcomes in a blame.

Bug : The nearness of mistake at the season of execution of the product.

Fault : State of programming caused by a mistake.

Failure : eviation of the product from its normal outcome. It is an occasion.

Who is a Software Tester?

Programming Tester is the person who performs testing and discovers bugs, in the event that they exist in the tried application.

The Testing Team

Program Manager

- The arranging and execution of the venture to guarantee the achievement of a venture limiting danger all through the lifetime of the venture.
- Responsible for composing the item detail, dealing with the calendar and settling on the basic choices and exchange offs.

QA Lead

- Coach and tutor other colleagues to help enhance QA adequacy
- Work with other office agents to team up on joint ventures and activities
- Implement industry best practices identified with testing mechanization and to streamline the QA Department.

Test Analyst / Lead

- Responsible for arranging, creating and executing mechanized test frameworks, manual test arrangements and relapses test arranges.
- Identifying the Target Test Items to be assessed by the test exertion.

- Defining the suitable tests required and any related Test Data.
- Gathering and dealing with the Test Data.
- Evaluating the result of each test cycle.

Test Engineer

- Writing and executing experiments and Reporting absconds.
- Test engineers are likewise in charge of deciding the most ideal way a test can be performed with a specific end goal to accomplish 100% test scope of all parts.

Importance of Testing In SDLC

SDLC remains for Software Development Life Cycle. Each product needs to experience an advancement procedure. Programming improvement philosophies are utilized for the PC based data frameworks. The development of the data needs to go through different stages/organizes, these stages are known as System Development Life Cycle (SDLC).

- Requirement
- Analysis
- Design
- Coding
- Testing
- Implementation
- Maintenance

Requirements

- This is the principal stage/phase of SDLC. In this stage, the required information's are accumulated from the client. The Analyst makes a study by gathering all the accessible data required for the framework components and designation of the prerequisites to the product.

Analysis

- In this stage, the framework build break down the necessity for the proposed framework. From the accessible data, the framework build builds up a rundown of utilization cases and framework level necessity for the venture. With the assistance of key client the rundown of utilization case and necessity is checked on. Refined and refreshed in an iterative way until the point that the client is fulfilled that it speaks to the pith of the proposed framework.

Design

- The configuration is the way toward outlining precisely how the determinations are to be executed. Examination and configuration are vital in the entire improvement cycle. Any blame in the plan could impact the item or could be extremely costly to unravel in the later phase of programming improvement.

Implementation or Coding:

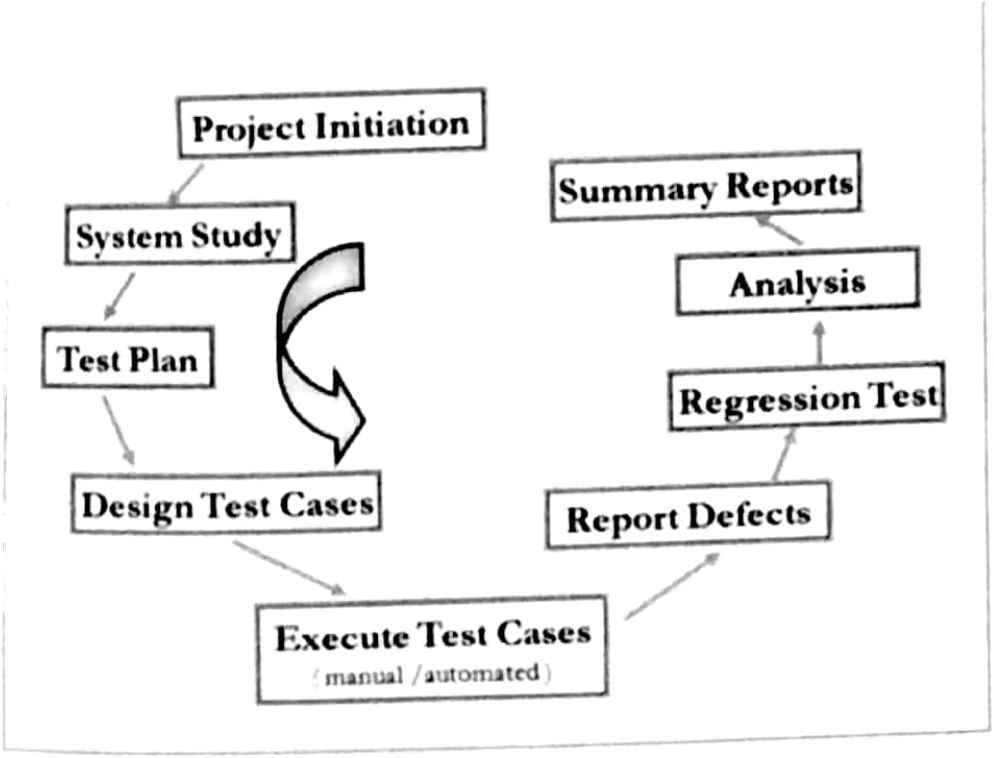
- In this segment, just designers are included for coding or programming.

Testing

- Once the coding is done, the examiner experiences it and test the framework for the item necessities.

Maintenance & Support

- This stage assumes a noteworthy part to the product item. Subsequent to discharging of the item, clients or customers give their criticism regarding the item execution and necessities. In the event that it is discovered any bug whenever then the item will go for the support.



Test Planning

Test Plan

A test arranges is a methodical way to deal with testing a framework i.e. programming. The arrangement normally contains an itemized comprehension of what the inevitable testing work process will be.

Test Case

An experiment is a particular technique of testing a specific prerequisite.

It will include:

- Identification of particular necessity tried
- Test case success/failure criteria
- Specific ventures to execute test
- Test Data

Levels of Testing

1. Unit Testing
2. Integration Testing
3. System Testing
4. Usability Testing
5. Functional Testing
6. Performance Testing
7. Security Testing
8. Smoke Testing
9. Alpha Testing
10. Acceptance Testing
11. Beta Testing
12. Regression Testing
13. Monkey Testing

Unit Testing

- Test every module independently.
- Follows a white box testing (Logic of the program)
- Done by Developers

Integration Testing

- After finishing the unit testing and ward modules improvement, developers interface the modules regarding HLD for Integration Testing through underneath approaches.

System Testing

- After finishing Unit and Integration testing through wh e box testing strategies advancement group discharge an .exe construct (all coordinated module) to perform discovery testing.

- Usability Testing
- Functional Testing
- Performance Testing
- Security Testing

Usability Testing

Amid this test, testing group focuses on the ease of use of fabricate interface. It comprises of following sub tests.

- **User Interface Test**: Ease of utilization (screens ought to be justifiable to work by End User.
- **Look & Feel** :- attractive
- **Speed in interface** :- Less number of assignment to finish undertaking.
- **Manual Support Test** :- Context affectability of client manual.

Functional Testing

- The procedure of checking the conduct of the application.
- It is adapted to utilitarian prerequisites of an application.
- To check the accuracy of yields.
- Data approval and Integration i.e. inputs are right or not.

Performance Testing

- LOAD TESTING – Also Known as Scalability Testing. Amid this test, test engineers execute application work under client anticipated that design and load would gauge execution.
- STRESS TESTING – During this test, Test engineers assesses the pinnacle stack. To discover the greatest number of clients for execution of our application client anticipated that arrangement would evaluate crest stack.
- PEAK LOAD > CUSTOMER LOAD (EXPECTED)
- DATA VOLUME TESTING — Testing group directs this test to locate the greatest furthest reaches of information volume of your application.

Security Testing

- Testing how well the framework ensures against unapproved inward or outside get to, obstinate harm, and so forth, may require refined testing systems.

Smoke testing

- Smoke testing is non-comprehensive programming testing, finding out that the most essential elements of a program work, yet not disturbing with better subtle elements.

Alpha Testing

- The application is tried by the client who doesn't think about the application.
- Done at engineer's site under controlled conditions
- Under the supervision of the engineers.

Acceptance Testing

- A formal test directed to decide if a framework fulfills its acknowledgment criteria and to empower the client to decide if to acknowledge the framework.
- It is the last test activity before conveying the product. The objective of acknowledgment testing is to confirm that the product is prepared and can be utilized by the end client to play out the capacities for which the product was assembled.

Beta Testing

- This Testing is done before the last arrival of the product to end-clients.
- Before the last arrival of the product is discharged to clients for testing where there will be no controlled conditions and the client here is sufficiently free to do whatever he needs to do on the framework to discover blunders.

Regression Testing

- Testing with the purpose of deciding whether bug fixes have been fruitful and have not made any new issues. Additionally, this sort of testing is done to guarantee that no debasement of standard usefulness has happened.

Monkey Testing

- Testing the application arbitrarily like hitting keys unpredictably and attempt to breakdown the framework there is no particular experiments and situations for monkey testing.

Verification & Validation

Verification

- Verification is the procedure affirming that - programming meets its determination, done through assessments and walkthroughs Use – To identify defects in the product early in the life cycle.
- Use – To distinguish surrenders in the item ahead of schedule in the life cycle

Validation

Validation is the procedure affirming that it meets the cli .t's prerequisites. It is the real testing.

Verification : Is the Product Right

Validation : Is it the Right Product

Unit Testing in Python

Unit testing is viewed as a basic piece of programming advancement. Through unit testing, we can assess each code part, discover how well it performs, and decide how well it responds to substantial or invalid information. A relapse suite of unit tests is additionally a great method for recognizing sudden changes in a code base caused by refactoring or composing new code.

The unittest Module

The unit test module began life as the outsider module PyUnit. PyUnit was a Python port of JUnit, the Java unit testing system. Composed by Steve Purcell, PyUnit turned into an official Python module beginning with form 2.5.

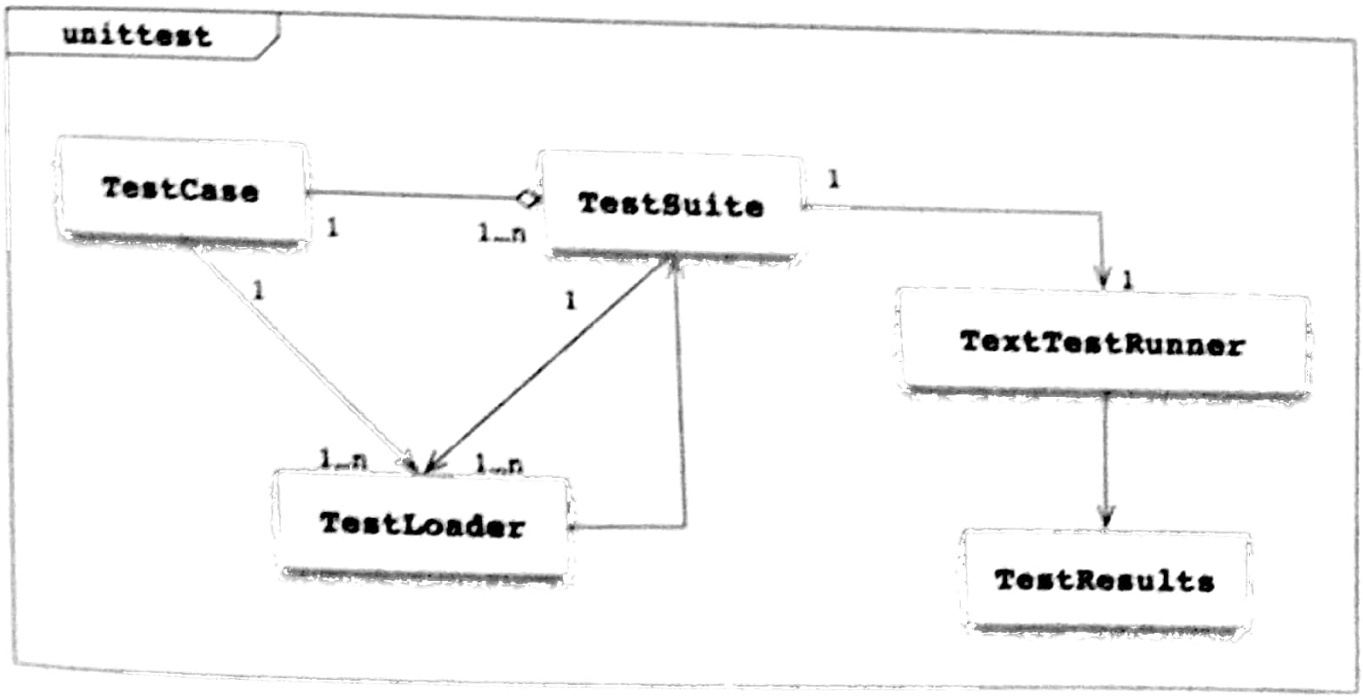


Figure 1: Core classes in unittest

As Figure 1 appears, there are five key classes in the unittest module. The TestCase class holds the test schedules and gives snares to setting up every standard and for tidying many. TheTestSuite class fills in as a gathering compartment. It can hold different TestCase objects and various TestSuite objects.

The TestLoader class loads test cases and suites characterized locally or from an outer document. It discharges a TestSuite question that holds those cases and suites. The TextTestRunner class gives a standard stage to run the tests. The TestResults class gives a standard compartment to the test outcomes.

Out of these five classes, just TestCase must be subclassed. The other four classes can likewise be subclassed, however they are for the most part utilized as may be.

Setting up a Test Case

Figure 2 demonstrates the structure of the TestCase class. In it are three arrangements of techniques that are utilized frequently in outlining the tests. In the main set are the pre-and post-test snares. ThesetUp() technique fires before each test schedule, the tearDown() after the schedule. Supersede these strategies when you make a custom experiment.

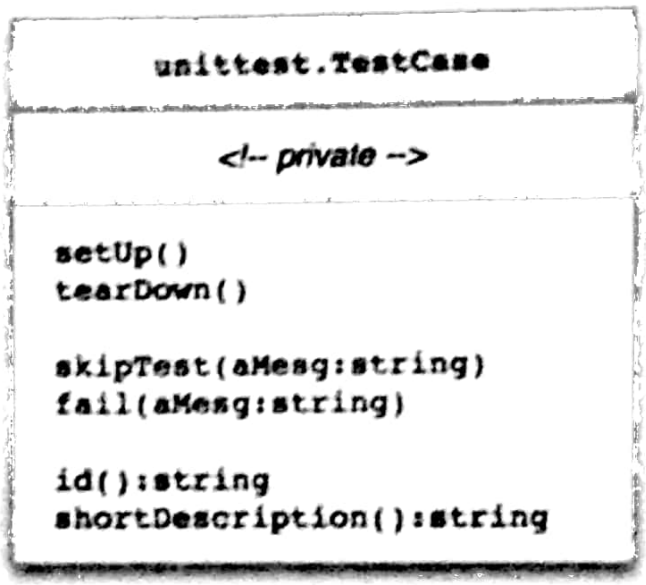


Figure 2: The formation of a Test Case class.

The second match of techniques control test execution. Both techniques take a message string as info, and both prematurely end a continuous test. Be that as it may, the skipTest() technique crosses out the present test, while the fail() strategy falls flat it expressly.

The third arrangement of strategies help distinguish the test. The method id() restores a string containing the name of the TestCase protest and of the test schedule. AJso, the methodshortDescription() restores the docstr remark toward the begin of each test schedule. On the off chance that the routine has no such remark, shortDescription() restores a None.

Posting One demonstrates the example no frills experiment FooTest. FooTest has two test routines:testA() and testB(). Both schedules get the required contention of self. Both have a docstrcomment for a first line.

Listing One: Sample program to show the sequence of unit test execution.

```
1.  #!/usr/bin/python
2.  import unittest
3.  class SampleTest(unittest.TestCase):
4.  # """Sample test case"""
5.  # preparing to test
6.  def setUp(self):
7.  #""" Setting up for the test """
8.  print "SampleTest setUp_begin"
9.  ## do something...
10. print "SampleTest setUp_end"
11. # ending the test
12. def tearDown(self):
13. #"""Cleaning up after the test"""
14. print "SampleTest tear Down_begin"
15. ## do something...
16. print "SampleTest tear Down_end"
17. # test routine A
18. def testA(self):
19. #"""Test routine A"""
20. print "SampleTest testA"
21. # test routine B
22. def testB(self):
23. #"""Test routine B"""
24. print "SampleTest testB"
```

Figure 3 indicates how FooTest acts when executed.

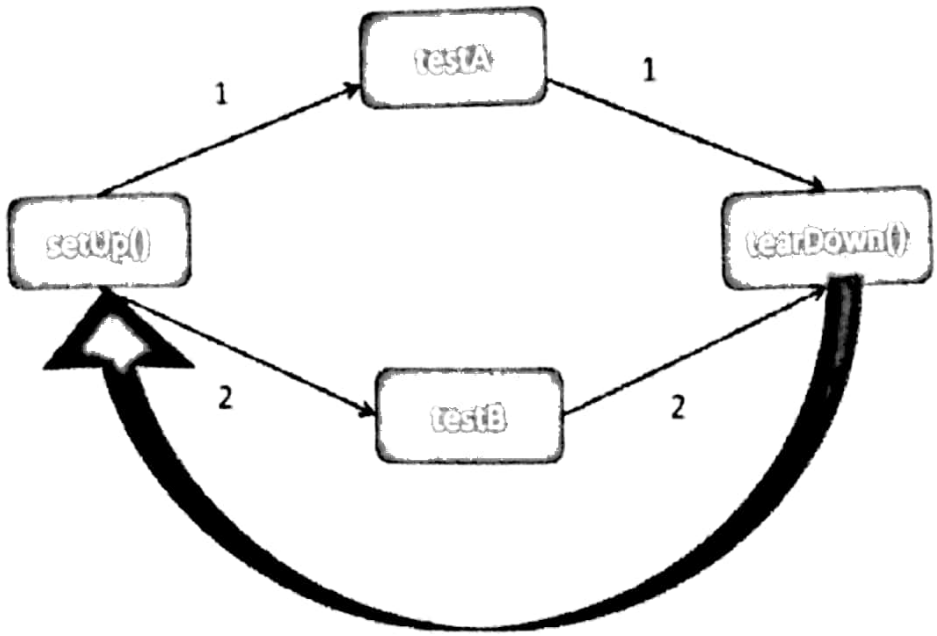


Figure 3: FooTest behavior

It is a similar setUp() and tearDown() strategies pursued earlier and each test schedule. So how would you let setUp() and tearDown() know which routine is being run? You should first recognize the routine by calling shortDescription() or id() (See Listing Two). At that point utilize anif-else square to course to the fitting code. In the specimen bit, FooTest callsshortDescription() to get the routine's docstr remark, at that point runs the prep and tidy up code for that schedule.

Posting Two: Using test depictions

```
1.  import unittest
2.  class SampleTest(unittest.TestCase):
3.  # """Sample test case"""
4.  # preparing to test
5.  def setUp(self):
6.  # """ Setting up for the test """
7.  print "SampleTest setUp_begin"
8.  testName = self.shortDescription()
9.  if (testName == "Test routine A"):
10.     print "setting up for test A"
11. elif (testName == "Test routine B"):
12.     print "setting up for test B"
13. else:
14.     print "UNKNOWN TEST ROUTINE"
15.     print "SampleTest setUp_end"
16. # ending the test
17. def tearDown(self):
18. #  """Cleaning up after the test"""
19. print "SampleTest tearDown_begin"
20. testName = self.shortDescription()
21. if (testName == "Test routine A"):
22.     print "cleaning up after test A"
23. elif (testName == "Test routine B"):
24.     print "cleaning up after test B"
25. else:
26.     print "UNKNOWN TEST ROUTINE"
27. print "SampleTest tearDown_end"
28. # see Listing One...
```