Brief Tour of the Standard Library:

Python’s standard library is very extensive, offering a wide range of facilities. The library contains built-in modules that provide access to system functionality such as I/O that would otherwise be inaccessible to the python programmers.

Operating system interface:

- The OS module in Python provides a way of using operating system dependent functionality.
- The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.
- You can find important information about your location or about the process.

OS functions

1. Executing a shell command
   os.system()
2. Returns the current working directory.
   os.getcwd()
3. Return the real group id of the current process.
   os.getgid()
4. Return the current process’s user id.
   os.getuid()
5. Returns the real process ID of the current process.
   os.getpid()
6. Set the current numeric umask and return the previous umask.
   os.umask(mask)
7. Return information identifying the current operating system.
   os.uname()
8. Change the root directory of the current process to path.
   os.chroot(path)
9. Return a list of the entries in the directory given by path.
   os.listdir(path)
10. Create a directory named path with numeric mode mode.
    os.mkdir(path)
11. Remove (delete) the file path.
    os.remove(path)
12. Remove directories recursively.
    os.removedirs(path)
13. Rename the file or directory src to dst.
    os.rename(src, dst)

String Pattern Matching:

The **re** module provides regular expression tools for advanced string processing. For complex matching and manipulation, the regular expressions offer succinct, optimized solutions.

re Functions:

1. match Function

re.match(pattern, string, flags=0)

Here is the description of the parameters:

Parameter	Description
Pattern	This is the regular expression to be matched.
String	This is the string, which would be searched to match the pattern at the beginning of string.
Flags	You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

The *re.match* function returns a **match** object on success, **None** on failure. We use*group(num)* or *groups()* function of **match** object to get matched expression.

Match Object Methods	Description
group(num=0)	This method returns entire match (or specific subgroup num)
groups()	This method returns all matching subgroups in a tuple (empty if there weren't any)

```
import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)

if matchObj:
   print "matchObj.group() : ", matchObj.group()
   print "matchObj.group(1) : ", matchObj.group(1)
   print "matchObj.group(2) : ", matchObj.group(2)
else:
   print "No match!!"
```

Output:

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

2. *search* Function

This function searches for first occurrence of RE *pattern* within *string* with optional *flags*. Here is the syntax for this function:

re.search(pattern, string, flags=0)

Here is the description of the parameters:

Parameter	Description
pattern	This is the regular expression to be matched.
string	This is the string, which would be searched to match the pattern anywhere in the string.
flags	You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

The *re.search* function returns a **match** object on success, **none** on failure. We use *group(num)* or *groups()* function of **match** object to get matched expression.

Match Object Methods	**Description**
group(num=0)	This method returns entire match (or specific subgroup num)
groups()	This method returns all matching subgroups in a tuple (empty if there weren't any)

3. sub function:

One of the most important **re** methods that use regular expressions is sub.

re.sub(pattern, repl, string, max=0)

This method replaces all occurrences of the RE pattern in string with repl, substituting all occurrences unless max provided. This method returns modified string.

Example:

```
import re

phone = "2004-959-559 # This is Phone Number"

# Delete Python-style comments
num = re.sub(r'#.*$', "", phone)
print "Phone Num : ", num

# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print "Phone Num : ", num
```

Output:

```
Phone Num :  2004-959-559
Phone Num :  2004959559
```

Regular Expression Patterns

Except for control characters, **(+ ? . * ^ $ ( ) [ ] { } | \)**, all characters match themselves. You can escape a control character by preceding it with a backslash.

Following table lists the regular expression syntax that is available in Python −

Pattern	**Description**
^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character except newline. Using m option allows it to match newline as well.
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
re{ n}	Matches exactly n number of occurrences of preceding expression.
re{ n,}	Matches n or more occurrences of preceding expression.

re{ n, m}	Matches at least n and at most m occurrences of preceding expression.
a| b	Matches either a or b.
(?#...)	Comment.
(?= re)	Specifies position using a pattern. Doesn't have a range.
(?! re)	Specifies position using pattern negation. Doesn't have a range.
(?> re)	Matches independent pattern without backtracking.
\w	Matches word characters.
\W	Matches nonword characters.
\s	Matches whitespace. Equivalent to [\t\n\r\f].
\S	Matches nonwhitespace.
\d	Matches digits. Equivalent to [0-9].
\D	Matches nondigits.
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.
\G	Matches point where last match finished.
\B	Matches nonword boundaries.
\n, \t, etc.	Matches newlines, carriage returns, tabs, etc.
\1...\9	Matches nth grouped subexpression.
\10	Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

Mathematical Functions:

The **math** module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using **import math**.

Function	Description
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair (m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
ldexp(x, i)	Returns x * (2**i)
modf(x)	Returns the fractional and integer parts of x

trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e**x
expm1(x)	Returns e**x - 1
log(x[, base])	Returns the logarithm of x to the base (defaults to e)
log10(x)	Returns the base-10 logarithm of x
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
atan2(y, x)	Returns atan(y / x)
cos(x)	Returns the cosine of x
hypot(x, y)	Returns the Euclidean norm, sqrt(x*x + y*y)
sin(x)	Returns the sine of x
tan(x)	Returns the tangent of x
degrees(x)	Converts angle x from radians to degrees
radians(x)	Converts angle x from degrees to radians
acosh(x)	Returns the inverse hyperbolic cosine of x
asinh(x)	Returns the inverse hyperbolic sine of x
atanh(x)	Returns the inverse hyperbolic tangent of x
cosh(x)	Returns the hyperbolic cosine of x
sinh(x)	Returns the hyperbolic cosine of x
tanh(x)	Returns the hyperbolic tangent of x
erf(x)	Returns the error function at x
erfc(x)	Returns the complementary error function at x
gamma(x)	Returns the Gamma function at x
lgamma(x)	Returns the natural logarithm of the absolute value of the Gamma function at x
pi	Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...)
e	mathematical constant e (2.71828...)

Internet Access:

- Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.
- Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.
- Here is a simple syntax to create one SMTP object, which can later be used to send an e-mail:

```
import smtplib
smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

- Here is the detail of the parameters:
  - **host:** This is the host running your SMTP server. You can specify IP address of the host or a domain name like tutorialspoint.com. This is optional argument.
  - **port:** If you are providing *host* argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.
  - **local_hostname**: If your SMTP server is running on your local machine, then you can specify just *localhost* as of this option.

An SMTP object has an instance method called **sendmail**, which is typically used to do the work of mailing a message. It takes three parameters:

- The *sender* - A string with the address of the sender.
- The *receivers* - A list of strings, one for each recipient.
- The *message* - A message as a string formatted as specified in the various RFCs.

Example: Write a program to send email to any mail address.

```
import smtplib
from email.mime.text import MIMEText
body="The message you want to send........"
msg=MIMEText(body)
fromaddr="fromaddress@gmail.com"
toaddr="toaddress@gmail.com"
msg['From']=fromaddr
msg['To']=toaddr
msg['Subject']="Subject of mail"
server=smtplib.SMTP('smtp.gmail.com',587)
server.starttls()
server.login(fromaddr,"fromAddressPassword")
server.sendmail(fromaddr,toaddr,msg.as_string())
print "Mail Sent......."
server.quit()
```

Output:

```
Mail Sent..........
```

Note: To send a mail to others you have to change "**Allow less secure apps: ON**" in from address mail. Because Google has providing security for vulnerable attacks

Dates and Times:

A Python program can handle date and time in several ways. Converting between date formats is a common chore for computers. Python's time and calendar modules help track dates and times.

The *time* Module

There is a popular **time** module available in Python which provides functions for working with times and for converting between representations. Here is the list of all available methods:

Sr. No.	Function with Description
1	**time.ctime([secs])** Like asctime(localtime(secs)) and without arguments is like asctime( )
2	**time.localtime([secs])** Accepts an instant expressed in seconds since the epoch and returns a time-tuple t with the local time (t.tm_isdst is 0 or 1, depending on whether DST applies to instant secs by local rules).
3	**time.sleep(secs)** Suspends the calling thread for secs seconds.
4	**time.time( )** Returns the current time instant, a floating-point number of seconds since the epoch.
5	**time.clock( )** The method returns the current processor time as a floating point numberexpressed in seconds on **Unix**.
6	**time.asctime([tupletime])** Accepts a time-tuple and returns a readable 24-character string such as 'Tue Dec 11 18:07:14 2008'.

Example:

```
import time
print "time: ",time.time()
print "ctime: ",time.ctime()
time.sleep(5)
print "ctime: ",time.ctime()
print "localtime: ",time.localtime()
print "asctime: ",time.asctime( time.localtime(time.time()) )
print "clock: ",time.clock()
```

Output:

```
time:  1506843198.01
ctime:  Sun Oct 01 13:03:18 2017
ctime:  Sun Oct 01 13:03:23 2017
localtime:   time.struct_time(tm_year=2017, tm_mon=10, tm_mday=1, tm_hour=13,
tm_min=3, tm_sec=23, tm_wday=6, tm_yday=274, tm_isdst=0)
asctime:  Sun Oct 01 13:03:23 2017
clock:  1.14090912202e-06
```

The *calendar* Module:

- The calendar module supplies calendar-related functions, including functions to print a text calendar for a given month or year.
- By default, calendar takes Monday as the first day of the week and Sunday as the last one. To change this, call calendar.setfirstweekday() function.

Sr. No.	Function with Description
1	**calendar.calendar(year,w=2,l=1,c=6)** Returns a multiline string with a calendar for year formatted into three columns separated by c spaces. w is the width in characters of each date; each line has length 21*w+18+2*c. l is the number of lines for each week.
2	**calendar.isleap(year)** Returns True if year is a leap year; otherwise, False.
3	**calendar.setfirstweekday(weekday)** Sets the first day of each week to weekday code weekday. Weekday codes are 0 (Monday) to 6 (Sunday).

4	**calendar.leapdays(y1,y2)** Returns the total number of leap days in the years within range(y1,y2).
5	**calendar.month(year,month,w=2,l=1)** Returns a multiline string with a calendar for month of year, one line per week plus two header lines. w is the width in characters of each date; each line has length 7*w+6. l is the number of lines for each week.

Example:

```
import calendar
print "Here it is the calendar:"
print calendar.month(2017,10)
calendar.setfirstweekday(6)
print calendar.month(2017,10)
print "Is 2017 is leap year?",calendar.isleap(2017)
print "No.of Leap days",calendar.leapdays(2000,2013)
print "1990-November-12 is",calendar.weekday(1990,11,12)
```

Output:

```
Here it is the calendar:
   October 2017
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31

   October 2017
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

Is 2017 is leap year? False
No.of Leap days 4
1990-November-12 is 0
```

Data Compression

Common data archiving and compression formats are directly supported by the modules including: zlib, gzip, bz2, lzma, zipfile and tarfile.

Example: write a program to zip the three files into one single ".zip" file

```
import zipfile
FileNames=['README.txt','NEWS.txt','LICENSE.txt']
with zipfile.ZipFile('reportDir1.zip', 'w') as myzip:
    for f in FileNames:
        myzip.write(f)
```

Multithreading

Running several threads is similar to running several different programs concurrently, but with the following benefits:

- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
- Threads sometimes called light-weight processes and they do not require much memory overhead; they care cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

- It can be pre-empted (interrupted).
- It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

Starting a New Thread

- To spawn another thread, you need to call following method available in *thread* module:

  thread.start_new_thread (function, args[, kwargs])
- This method call enables a fast and efficient way to create new threads in both Linux and Windows.
- The method call returns immediately and the child thread starts and calls function with the passed list of *agrs*. When function returns, the thread terminates.
- Here, *args* is a tuple of arguments; use an empty tuple to call function without passing any arguments. *kwargs* is an optional dictionary of keyword arguments.

Example:

```
import thread
import time
def print_time(tname,delay):
    count=0
    while count<5:
        count+=1
        time.sleep(delay)
        print tname,time.ctime(time.time())

thread.start_new_thread( print_time, ("Thread-1", 2 ) )
thread.start_new_thread( print_time, ("Thread-2", 5 ) )
```

Output:

```
Thread-1 Sun Oct 01 22:15:08 2017
Thread-1 Sun Oct 01 22:15:10 2017
Thread-2 Sun Oct 01 22:15:11 2017
Thread-1 Sun Oct 01 22:15:12 2017
Thread-1 Sun Oct 01 22:15:14 2017
Thread-1Thread-2  Sun Oct 01 22:15:16 2017Sun Oct 01 22:15:16 2017
Thread-2 Sun Oct 01 22:15:21 2017
Thread-2 Sun Oct 01 22:15:26 2017
Thread-2 Sun Oct 01 22:15:31 2017
```

The Threading Module:

The *threading* module exposes all the methods of the *thread* module and provides some additional methods:

- **threading.activeCount():** Returns the number of thread objects that are active.
- **threading.currentThread():** Returns the number of thread objects in the caller's thread control.
- **threading.enumerate():** Returns a list of all thread objects that are currently active.

In addition to the methods, the threading module has the *Thread* class that implements threading. The methods provided by the *Thread* class are as follows:

- **run():** The run() method is the entry point for a thread.
- **start():** The start() method starts a thread by calling the run method.
- **join([time]):** The join() waits for threads to terminate.
- **isAlive():** The isAlive() method checks whether a thread is still executing.
- **getName():** The getName() method returns the name of a thread.
- **setName():** The setName() method sets the name of a thread.

Creating Thread Using *Threading* Module:

To implement a new thread using the threading module, you have to do the following:

- Define a new subclass of the *Thread* class.
- Override the *__init_(self [,args])* method to add additional arguments.
- Then, override the run(self [,args]) method to implement what the thread should do when started.

Once you have created the new *Thread* subclass, you can create an instance of it and then start a new thread by invoking the *start()*, which in turn calls *run()* method.

Example:

```
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 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"
```

Output:

```
Starting Thread-1Starting Thread-2Exiting Main Thread
Thread-1 Sun Oct 01 22:26:17 2017
Thread-1 Sun Oct 01 22:26:18 2017
Thread-2 Sun Oct 01 22:26:18 2017
Thread-1 Sun Oct 01 22:26:19 2017
Thread-1Thread-2  Sun Oct 01 22:26:20 2017Sun Oct 01 22:26:20 2017

Thread-1 Sun Oct 01 22:26:21 2017
Exiting Thread-1
Thread-2 Sun Oct 01 22:26:22 2017
Thread-2 Sun Oct 01 22:26:24 2017
Thread-2 Sun Oct 01 22:26:26 2017
Exiting Thread-2
```

Synchronizing Threads

- The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the *Lock()* method, which returns the new lock.
- The *acquire(blocking)* method of the new lock object is used to force threads to run synchronously. The optional *blocking* parameter enables you to control whether the thread waits to acquire the lock.
- If *blocking* is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and waits for the lock to be released.
- The *release()* method of the new lock object is used to release the lock when it is no longer required.

Example:

```
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
        threadLock.acquire()
        print_time(self.name, self.counter, 5)
        threadLock.release()
        print "Exiting " + self.name
def print_time(threadName, delay, counter):
    while counter:
        if exitFlag:
            thread.exit()
        time.sleep(delay)
        print 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()

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

Output:

```
Starting Thread-1Starting Thread-2
Thread-1 Sun Oct 01 22:32:54 2017
Thread-1 Sun Oct 01 22:32:55 2017
Thread-1 Sun Oct 01 22:32:56 2017
Thread-1 Sun Oct 01 22:32:57 2017
Thread-1 Sun Oct 01 22:32:58 2017
Exiting Thread-1
Thread-2 Sun Oct 01 22:33:00 2017
Thread-2 Sun Oct 01 22:33:02 2017
Thread-2 Sun Oct 01 22:33:04 2017
Thread-2 Sun Oct 01 22:33:06 2017
Thread-2 Sun Oct 01 22:33:08 2017
Exiting Thread-2
Exiting Main Thread
```

GUI Programming

Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below:

- **Tkinter:** Tkinter is the Python interface to the Tk GUI toolkit shipped with Python..
- **wxPython:** This is an open-source Python interface for wxWindows **http://wxpython.org**.
- **JPython:** JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine **http://www.jython.org**.

There are many other interfaces available, which you can find them on the net.

Tkinter Programming

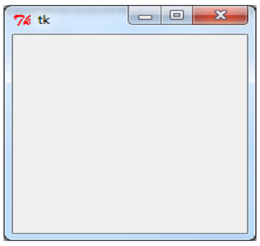
Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps:

- ✓ Import the *Tkinter* module.
- ✓ Create the GUI application main window.
- ✓ Add one or more of the above-mentioned widgets to the GUI application.
- ✓ Enter the main event loop to take action against each event triggered by the user.

Example:

```
import Tkinter
top = Tkinter.Tk()
# Code to add widgets will go here...
top.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.
- There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table:

Operator	Description
Button	The Button widget is used to display buttons in your application.
Canvas	The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
Checkbutton	The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
Frame	The Frame widget is used as a container widget to organize other widgets.
Label	The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
Listbox	The Listbox widget is used to provide a list of options to a user.
Menubutton	The Menubutton widget is used to display menus in your application.
Menu	The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.

Message	The Message widget is used to display multiline text fields for accepting values from a user.
Radiobutton	The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time. Scale The Scale widget is used to provide a slider widget.
Scrollbar	The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
Text	The Text widget is used to display text in multiple lines.
Toplevel	The Toplevel widget is used to provide a separate window container.
Spinbox	The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
PanedWindow	A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
LabelFrame	A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
tkMessageBox	This module is used to display message boxes in your applications.

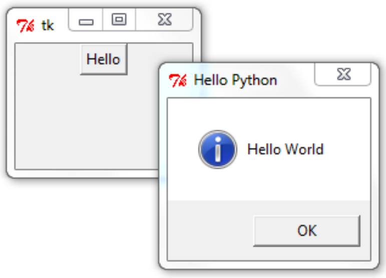
Button:

The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons. You can attach a function or a method to a button which is called automatically when you click the button.

Example:

```
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()
```

Output:



Hello World

Entry

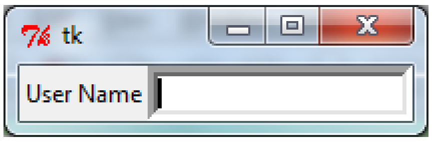
The Entry widget is used to accept single-line text strings from a user.

- If you want to display multiple lines of text that can be edited, then you should use the *Text* widget.
- If you want to display one or more lines of text that cannot be modified by the user, then you should use the *Label* widget.

Example:

```
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()
```

Output:



User Name

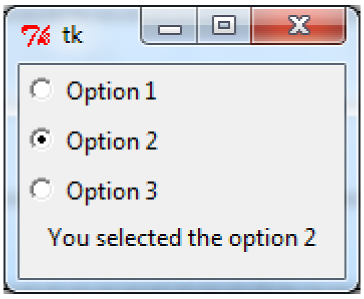
Radiobutton

- This widget implements a multiple-choice button, which is a way to offer many possible selections to the user and lets user choose only one of them.
- In order to implement this functionality, each group of radiobuttons must be associated to the same variable and each one of the buttons must symbolize a single value. You can use the Tab key to switch from one radionbutton to another.

Example:

```
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()
```

Output:



Option 1 [✓ Option 2] Option 3

You selected the option 2

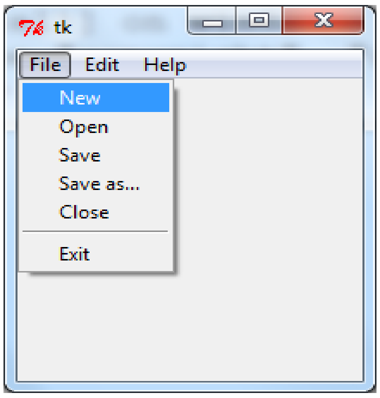
Menu

- The goal of this widget is to allow us to create all kinds of menus that can be used by our applications. The core functionality provides ways to create three menu types: pop-up, toplevel and pull-down.
- It is also possible to use other extended widgets to implement new types of menus, such as the *OptionMenu* widget, which implements a special type that generates a pop-up list of items within a selection.

Example:

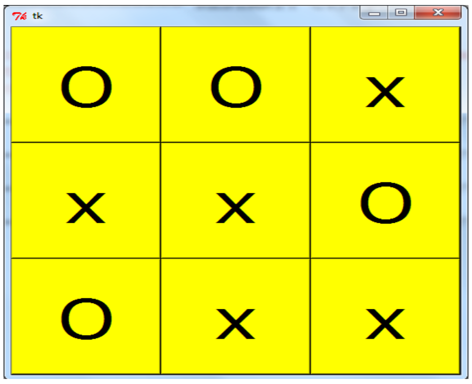
```
from Tkinter import *
def donothing():
   filewin = Toplevel(root)
   button = Button(filewin, text="Do nothing button")
   button.pack()

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()
```

Output:



Example: Write a program for Tic-Tac-Toe Game

```
from Tkinter import *
def callback(r,c):
    global player
    if player=='X' and states[r][c]==0:
        b[r][c].configure(text='x')
        states[r][c]='X'
        player='O'
    if player=='O' and states[r][c]==0:
        b[r][c].configure(text='O')
        states[r][c]='O'
        player='X'
root=Tk()
states=[[0,0,0],[0,0,0],[0,0,0]]
b=[[0,0,0],[0,0,0],[0,0,0]]
for i in range(3):
    for j in range(3):

b[i][j]=Button(font=('verdana',56),width=3,bg='yellow',command=lambda
r=i,c=j:callback(r,c))
        b[i][j].grid(row=i,column=j)
player='X'
root.mainloop()
```

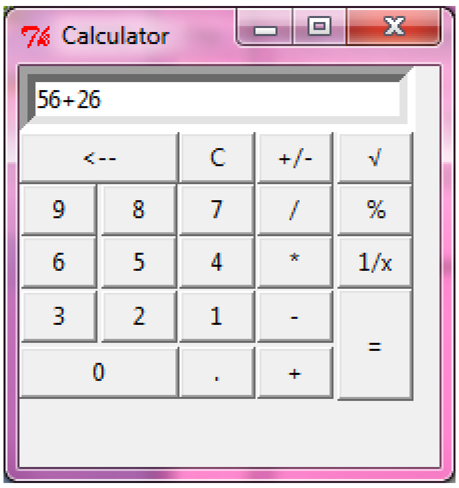


Example: Write a GUI for an Expression Calculator using tk

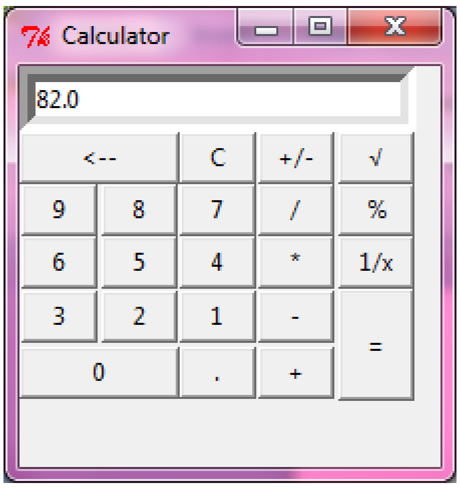
```
from Tkinter import *
from math import *
root=Tk()
root.title("Calculator")
root.geometry("210x200")
e=Entry(root,bd=8,width=30)
e.grid(row=0,column=1,columnspan=5)
def setText(txt):
   l=len(e.get())
   e.insert(l,txt)
def clear1():
   txt=e.get()
   e.delete(0,END)
   e.insert(0,txt[:-1])
def clear():
   e.delete(0,END)
def sqroot():
   txt=sqrt(float(e.get()))
   e.delete(0,END)
   e.insert(0,txt)
def negation():
   txt=e.get()
   if txt[0]=="-":
      e.delete(0,END)
      e.insert(0,txt[1:])
   elif txt[0]=="+":
      e.delete(0,END)
      e.insert(0,"-"+txt[1:])
   else:
      e.insert(0,"-")

def equals():
   try:
      s=e.get()
      for i in range(0,len(s)):
         if s[i]=="+" or s[i]=="-" or s[i]=="*" or s[i]=="/" or s[i]=="%":
            expr=str(float(s[:i]))+s[i:]
            break
         elif s[i]==".":
            expr=s
            break
      e.delete(0,END)
      e.insert(0,eval(expr))
```

```
    except Exception:
        e.delete(0,END)
        e.insert(0,"INVALID EXPRESSION")

back1=Button(root,text="<--",command=lambda:clear1(),width=10)
back1.grid(row=1,column=1,columnspan=2)

sqr=Button(root,text=u'\u221A',command=lambda:sqroot(),width=4)
sqr.grid(row=1,column=5)

can=Button(root,text="C",command=lambda:clear(),width=4)
can.grid(row=1,column=3)

neg=Button(root,text="+/-",command=lambda:negation(),width=4)
neg.grid(row=1,column=4)

nine=Button(root,text="9",command=lambda:setText("9"),width=4)
nine.grid(row=2,column=1)

eight=Button(root,text="8",command=lambda:setText("8"),width=4)
eight.grid(row=2,column=2)

seven=Button(root,text="7",command=lambda:setText("7"),width=4)
seven.grid(row=2,column=3)

six=Button(root,text="6",command=lambda:setText("6"),width=4)
six.grid(row=3,column=1)

five=Button(root,text="5",command=lambda:setText("5"),width=4)
five.grid(row=3,column=2)

four=Button(root,text="4",command=lambda:setText("4"),width=4)
four.grid(row=3,column=3)

three=Button(root,text="3",command=lambda:setText("3"),width=4)
three.grid(row=4,column=1)

two=Button(root,text="2",command=lambda:setText("2"),width=4)
two.grid(row=4,column=2)

one=Button(root,text="1",command=lambda:setText("1"),width=4)
one.grid(row=4,column=3)
zero=Button(root,text="0",command=lambda:setText("0"),width=10)
zero.grid(row=5,column=1,columnspan=2)
```

```
dot=Button(root,text=".",command=lambda:setText("."),width=4)
dot.grid(row=5,column=3)

div=Button(root,text="/",command=lambda:setText("/"),width=4)
div.grid(row=2,column=4)

mul=Button(root,text="*",command=lambda:setText("*"),width=4)
mul.grid(row=3,column=4)

minus=Button(root,text="-",command=lambda:setText("-"),width=4)
minus.grid(row=4,column=4)

plus=Button(root,text="+",command=lambda:setText("+"),width=4)
plus.grid(row=5,column=4)

mod=Button(root,text="%",command=lambda:setText("%"),width=4)
mod.grid(row=2,column=5)

byx=Button(root,text="1/x",command=lambda:setText("%"),width=4)
byx.grid(row=3,column=5)

equal=Button(root,text="=",command=lambda:equals(),width=4,height=3)
equal.grid(row=4,column=5,rowspan=2)

root.mainloop()
```



56+26



82.0

Turtle Graphics

- Graphics is the discipline that underlies the representation and display of geometric shapes in two and three-dimensional space.
- A Turtle graphics library provides an enjoyable and easy way to draw shapes in a window and gives you an opportunity to run several functions with an object.
- Turtle graphics were originally developed as part of the children's programming language called Logo, created by Seymour Papert and his colleagues at MIT in the late 1960s.
- Imagine a turtle crawling on a piece of paper with a pen tied to its tail.
- Commands direct the turtle as it moves across the paper and tells it to lift or lower its tail, turn some number of degrees left or right and move a specified distance.
- Whenever the tail is down, the pen drags along the paper, leaving a trail.
- In the context of computer, of course, the sheet of paper is a window on a display screen and the turtle is an invisible pen point.
- At any given moment of time, the turtle coordinates. The position is specified with (x, y) coordinates.
- The coordinate system for turtle graphics is the standard Cartesian system, with the origin (0, 0) at the centre of a window. The turtle's initial position is the origin, which is also called the home.

Turtle Operations:

Turtle is an object; its operations are also defined as methods. In the below table the list of methods of Turtle class.

Turtle Methods	WHAT IT DOES
home	Moves the turtle to the origin – coordinates (0, 0) – and set its heading to its start-orientation.
fd | forward	Moves the turtle forward for a specified distance, in the direction where the turtle is headed.
bk | backward	Moves the turtle backward for a specified distance, in the direction where the turtle is headed. Do not change the turtle's heading.
right | rt	Turns the turtle right by angle units. Units are by default degrees, but can be set via the degrees ( ) and radians ( ) functions.
left | lt	Turns the turtle left by angle units. Units are by default degrees, but can be set via the degrees ( ) and radians ( ) functions.
setx	Set the turtle's first coordinate to x, leaves the second coordinate unchanged.
sety	Set the turtle's second coordinate to y, leaves the first coordinate unchanged.
goto	Moves the turtle to an absolute position. If the pen is down, draws a line. Do not change the turtle's orientation.
degrees	Set the angle measurement unit to radians. Equivalent to degrees (2 * math.pi)
radians	Set the angle measurement unit, i.e., set the number of degrees for a full circle. The default value is 360^0.
seth	Sets the orientation of the turtle to to_angle.

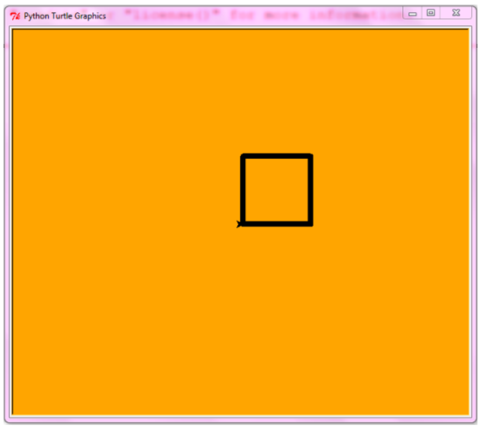
Turtle Object:

t=Turtle( ) creates a new turtle object and open sits window. The window's drawing area is 200 pixels wide and 200 pixels high.

t=Turtle(width, height) creates a new turtle object and open sits window. The window's drawing area has given width and height.

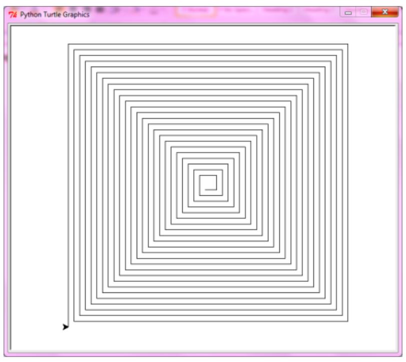
Example-1: Write a program to draw square.

```
import turtle
turtle.bgcolor('orange')
turtle.pensize(8)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
```



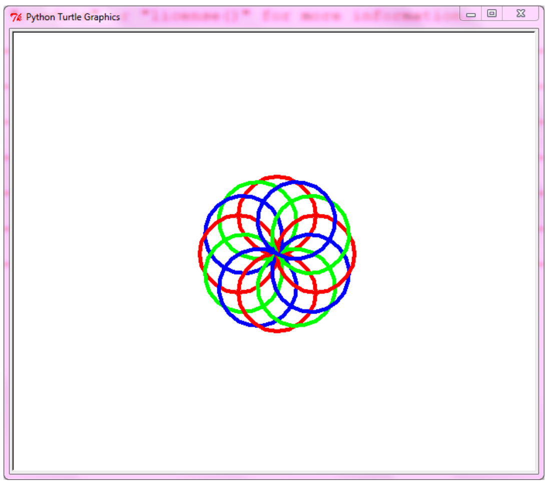
Example-2:

```
import turtle
for i in range(20,500,5):
    turtle.forward(i)
    turtle.left(90)
```



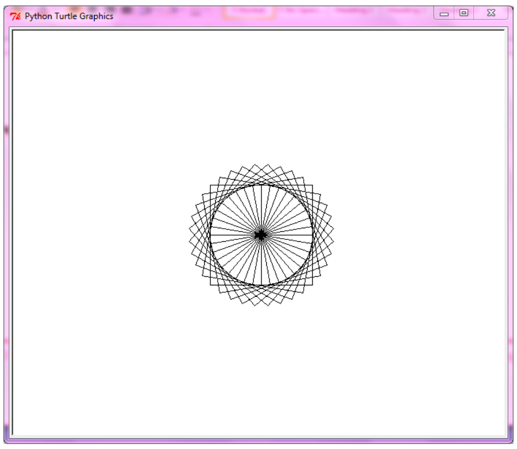
Example-3:

```
import turtle
c=["red","green","blue"]
i=0
turtle.pensize(5)
for angle in range(0,360,30):
    if i>2:
        i=0
    turtle.color(c[i])
    turtle.seth(angle)
    turtle.circle(50)
    i=i+1
```



Example-4:

```
import turtle
for i in range(36):
    for j in range(4):
        turtle.forward(70)
        turtle.left(90)
    turtle.left(10)
```



Testing: Why testing is required?

Software testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong-humans make mistakes all the time.

Software testing is very important because of the following reasons:

1. Software testing is really required to point out the defects and errors that were made during the development phases.
2. It’s essential since it makes sure of the customer’s reliability and their satisfaction in the application.
3. It is very important to ensure the quality of the product. Quality product delivered to the customers helps in gaining their confidence.
4. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
5. Testing is required for an effective performance of software application or product.
6. It’s important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.
7. It’s required to stay in the business.

Basic concepts of testing:

Basics	**Summary**
Software Quality	Learn how software quality is defined and what it means. Software quality is the degree of conformance to explicit or implicit requirements and expectations.
Dimensions of Quality	Learn the dimensions of quality. Software quality has dimensions such as Accessibility, Compatibility, Concurrency, Efficiency …
Software Quality Assurance	Learn what it means and what its relationship is with Software Quality Control. Software Quality Assurance is a set of activities for ensuring quality in software engineering processes.
Software Quality Control	Learn what it means and what its relationship is with Software Quality Assurance. Software Quality Control is a set of activities for ensuring quality in software products.
SQA and SQC Differences	Learn the differences between Software Quality Assurance and Software Quality Control. SQA is process-focused and prevention-oriented but SQC is product-focused and detection-oriented.
Software Development Life Cycle	Learn what SDLC means and what activities a typical SDLC model comprises of. Software Development Life Cycle defines the steps/stages/phases in the building of software.

Software Testing Life Cycle	Learn what STLC means and what activities a typical STLC model comprises of. Software Testing Life Cycle (STLC) defines the steps/ stages/ phases in testing of software.
Definition of Test	Learn the various definitions of the term ‘test’. Merriam Webster defines Test as “a critical examination, observation, or evaluation”.
Software Testing Myths	Just as every field has its myths, so does the field of Software Testing. We explain some of the myths along with their related facts.

Unit testing in Python:

The first unit testing framework, JUnit was invented by Kent Back and Erich Gamma in 1997, for testing Java programs. It was so successful that the framework has been implemented again in every major programming language. Here we discuss the python version, unit test.

Unit testing is nothing but testing individual ‘units’, or functions of a program. It does not have a lot to say about system integration, whether the various parts of a program fit together. That’s a separate issue.

The goals of unit testing framework are:

- To make it easy to write tests. All a ‘test’ needs to do is to say that, for this input, the function should give that result. The framework takes care of running the tests.
- To make it easy to run tests. Usually this is done by clicking a single button or typing a single keystroke (F5 in IDLE). Ideally, you should be comfortable running tests after every change in the program, however minor.
- To make it easy to tell if the tests passed. The framework takes care of reporting results; it either simply indicates that all tests passed, or it provides a detailed list of failures.

Example:

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())
if __name__ == '__main__':
    unittest.main()
```

Output:

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.016s

OK
```

Writing and Running Test cases

- Your object is to write test and not to prove that your program works, it's to try to find out where it doesn't! Test every 'extreme' case you can think of.
- For example, if you were to write and test a function to sort a list, then the first and last elements get moved to correct position? Can you sort a 1-element list without getting an error? How about an empty list?
- While you can put as many tests as you like into one test method that you shouldn't test methods should be short and single-purpose. If you are testing different aspects of a function, they should be in separate tests.
- Here are the rules for writing test methods:
    - The name of a test method must start with the letters 'test', otherwise it will be ignored.
    - This is so that you can write 'helper' methods you can call from your tests, but are not directly called by the test framework.
    - Every test method must have exactly one parameter, which is nothing but 'self'. You must put self in front of every built-in assertion method you call.
    - The tests must be independent of one another, because they may be run in any order.
    - Do not assume they will execute in the order they occur in the program.
- Here are some of the built-in test methods you can call. Each has an optional message parameter, to be printed if the test fails.

Method	Checks that
`assertEqual(a, b)`	`a == b`
`assertNotEqual(a, b)`	`a != b`
`assertTrue(x)`	`bool(x) is True`
`assertFalse(x)`	`bool(x) is False`
`assertIs(a, b)`	`a is b`
`assertIsNot(a, b)`	`a is not b`
`assertIsNone(x)`	`x is None`
`assertIsNotNone(x)`	`x is not None`
`assertIn(a, b)`	`a in b`
`assertNotIn(a, b)`	`a not in b`
`assertIsInstance(a, b)`	`isinstance(a, b)`
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`

Example: Unittest for addition of two numbers.

```
import unittest
def add(a,b):
    if isinstance(a,int) and isinstance(b,int):
        return a+b
    elif isinstance(a,str) and isinstance(b,str):
        return int(a)+int(b)
    else:
        raise Exception('Invalid arguments')
```

```
class TestAdd(unittest.TestCase):
    def test_add(self):
        self.assertEqual(5,add(2,3))
        self.assertEqual(15,add(-6,21))
        self.assertRaises(Exception,add,4.0,5.0)
unittest.main()
```

Output:

```
.
----------------------------------------------------------------------
Ran 1 test in 0.008s

OK
```