Overview of OOP Terminology

- **Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class.
  - The attributes are data members (class variables and instance variables) and methods, accessed via dot notation (**.**).
- **Class variable:** A variable that is shared by all instances of a class.
  - Class variables are defined within a class but also outside any of the class's methods.
  - Class variables aren't used as frequently as instance variables are.
- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.
- **Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.

Creating Classes

- The **class** statement creates a new class definition.

```
class ClassName:
    'Optional class documentation string'
    class_suite
```

- The class has a documentation string, which can be accessed via *ClassName.__doc__*.

- The *class_suite* consists of all the component statements defining class members, data attributes and functions.

EXAMPLE

- The variable *empCount* is a class variable whose value would be shared among all instances of a this class.
  - This can be accessed as *Employee.empCount* from inside the class or outside the class.
- The first method *__init__()* is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.
- You declare other class methods like normal functions with the exception that the first argument to each method is *self*.
  - Python adds the *self* argument to the list for you; you don't need to include it when you call the methods.

```python
class Employee:
    'Common base class for all employees'
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

    def displayEmployee(self):
        print "Name : ", self.name,  ", Salary: ", self.salary
```

Creating instance objects

- To create instances of a class, you call the class using class name and pass in whatever arguments its __*init*__ method accepts.

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

- Accessing attributes

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

Example

```python
#!/usr/bin/python

class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary

"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

Built-In Class Attributes

- Every Python class keeps following **built-in attributes** and they can be accessed using **dot (.)** operator like any other attribute:

- **__dict__ :** Dictionary containing the class's namespace.

- **__doc__ :** Class documentation string or None if undefined.

- **__name__ :** Class name.

- **__module__ :** Module name in which the class is defined.
  - This attribute is "__main__" in interactive mode.

- **__bases__ :** A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Example

```python
#!/usr/bin/python

class Employee:
    'Common base class for all employees'
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

    def displayEmployee(self):
        print "Name : ", self.name,  ", Salary: ", self.salary
```

```python
print "Employee.__doc__:", Employee.__doc__
print "Employee.__name__:", Employee.__name__
print "Employee.__module__:", Employee.__module__
print "Employee.__bases__:", Employee.__bases__
print "Employee.__dict__:", Employee.__dict__
```

```
Employee.__doc__: Common base class for all employees
Employee.__name__: Employee
Employee.__module__: __main__
Employee.__bases__: ()
Employee.__dict__: {'__module__': '__main__', 'displayCount':
<function displayCount at 0xb7c84994>, 'empCount': 2,
'displayEmployee': <function displayEmployee at 0xb7c8441c>,
'__doc__': 'Common base class for all employees',
'__init__': <function __init__ at 0xb7c846bc>}
```

Destroying Objects (Garbage Collection)

- Python deletes unneeded objects (built-in types or class instances) automatically to free memory space.

- The process by which Python periodically reclaims blocks of memory that no longer are in use is termed garbage collection.

- Python's garbage collector runs during program execution and is triggered <u>when an object's reference count reaches zero</u>.

  - An object's reference count changes as the number of aliases that point to it changes.

Destroying Objects

- An object's reference count increases when it's assigned a new name or placed in a container (list, tuple or dictionary).
  - The object's reference count decreases when it's deleted with *del*, its reference is reassigned, or its reference goes out of scope.
  - When an object's reference count reaches zero, Python collects it automatically.

```
a = 40          # Create object <40>
b = a           # Increase ref. count  of <40>
c = [b]         # Increase ref. count  of <40>

del a           # Decrease ref. count  of <40>
b = 100         # Decrease ref. count  of <40>
c[0] = -1       # Decrease ref. count  of <40>
```

EXAMPLE

- This __del__() destructor prints the class name of an instance that is about to be destroyed.

```python
#!/usr/bin/python

class Point:
    def __init( self, x=0, y=0):
        self.x = x
        self.y = y
    def __del__(self):
        class_name = self.__class__.__name__
        print class_name, "destroyed"

pt1 = Point()
pt2 = pt1
pt3 = pt1
print id(pt1), id(pt2), id(pt3) # prints the ids of the obejcts
del pt1
del pt2
del pt3
```

```
3083401324 3083401324 3083401324
Point destroyed
```

Class Inheritance

- You can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
    'Optional class documentation string'
    class_suite
```

- The child class inherits the attributes of its parent class
  - you can use those attributes as if they were defined in the parent class.
- A child class can also override data members and methods from the parent.

EXAMPLE

```python
#!/usr/bin/python

class Parent:           # define parent class
    parentAttr = 100
    def __init__(self):
        print "Calling parent constructor"

    def parentMethod(self):
        print 'Calling parent method'

    def setAttr(self, attr):
        Parent.parentAttr = attr

    def getAttr(self):
        print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
    def __init__(self):
        print "Calling child constructor"

    def childMethod(self):
        print 'Calling child method'

c = Child()              # instance of child
c.childMethod()          # child calls its method
c.parentMethod()         # calls parent's method
c.setAttr(200)           # again call parent's method
c.getAttr()              # again call parent's method
```

```
Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200
```

Multiple Inheritance

```
class A:          # define your class A
.....

class B:          # define your calss B
.....

class C(A, B):    # subclass of A and B
.....
```

- You can use **issubclass()** or **isinstance()** functions to check a relationships of two classes and instances.

- The **issubclass(sub, sup)** boolean function returns true if the given subclass **sub** is indeed a subclass of the superclass **sup**.

- The **isinstance(obj, Class)** boolean function returns true if *obj* is an instance of class *Class* or is an instance of a subclass of Class

Overriding Methods

- You can always override your parent class methods.

```python
#!/usr/bin/python

class Parent:              # define parent class
    def myMethod(self):
        print 'Calling parent method'

class Child(Parent):       # define child class
    def myMethod(self):
        print 'Calling child method'

c = Child()                # instance of child
c.myMethod()               # child calls overridden method
```

```
Calling child method
```

Base Overloading Methods

- Following table lists some generic functionality that you can override in your own classes.

SN	Method, Description & Sample Call
1	__init__ (self [,args...]) Constructor (with any optional arguments) Sample Call : *obj = className(args)*
2	__del__(self) Destructor, deletes an object Sample Call : *del obj*
3	__repr__(self) Evaluatable string representation Sample Call : *repr(obj)*
4	__str__(self) Printable string representation Sample Call : *str(obj)*
5	__cmp__ (self, x) Object comparison Sample Call : *cmp(obj, x)*

Overloading Operators

- You could define the __*add*__ method in your class to perform vector addition and then the <u>plus operator would behave as per expectation</u>

```python
#!/usr/bin/python

class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return 'Vector (%d, %d)' % (self.a, self.b)

    def __add__(self,other):
        return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

```
Vector(7,8)
```

Data Hiding

- An object's attributes may or may not be visible outside the class definition.

- For these cases, you can name attributes with a double underscore prefix, and those attributes will not be directly visible to outsiders.

```python
#!/usr/bin/python

class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```

```
1
2
Traceback (most recent call last):
  File "test.py", line 12, in <module>
    print counter.__secretCount
AttributeError: JustCounter instance has no attribute '__secretCount'
```

Data Hiding

- Python protects those members by internally changing the name to include the class name.

- You can access such attributes as *object._className__attrName*.

- If you would replace your last line as following, then it would work for you:

```
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
print counter._JustCounter__secretCount
```

```
1
2
2
```

Python Modules

- A module allows you to logically organize your Python code.

- Grouping related code into a module makes the code easier to understand and use.

- A module is a Python object with arbitrarily named attributes that you can bind and reference.

- Simply, a module is a file consisting of Python code.

- A module can define functions, classes and variables. A module can also include runnable code.

The *import* Statement

- You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax:

```
import module1[, module2[,... moduleN]
```

- When the interpreter encounters an import statement, it imports the module if the module is present in the search path.

- A search path is a list of directories that the interpreter searches before importing a module.

Example

- To import the module hello.py, you need to put the following command at the top of the script:

```python
#!/usr/bin/python

# Import module support
import support

# Now you can call defined function that module as follows
support.print_func("Zara")
```

```
Hello : Zara
```

The *from...import* Statement

- Python's *from* statement lets you import specific attributes from a module into the current namespace.

- The *from...import* has the following syntax:

```
from modname import name1[, name2[, ... nameN]]
```

- For example, to import the function fibonacci from the module fib, use the following statement:

```
from fib import fibonacci
```

The *from…import *** Statement:

- It is also possible to import all names from a module into the current namespace by using the following import statement:

```
from modname import *
```

Locating Modules:

- When you import a module, the Python interpreter searches for the module in the following sequences:
  - The current directory.
  - If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
  - If all else fails, Python checks the default path.
    - On UNIX, this default path is normally /usr/local/lib/python/.

The *PYTHONPATH* Variable:

- The PYTHONPATH is an environment variable, consisting of a list of directories.

- The syntax of PYTHONPATH is the same as that of the shell variable PATH.

- Here is a typical PYTHONPATH from a Windows system:
  - set PYTHONPATH=c:\python27\lib;

- Here is a typical PYTHONPATH from a UNIX system:
  - set PYTHONPATH=/usr/local/lib/python

Namespaces and Scoping

- Variables are names (identifiers) that map to objects.

- A **namespace** is a dictionary of variable names (keys) and their corresponding objects (values).

- A Python statement can access variables in a *local namespace* and in the *global namespace*.
  - If a local and a global variable have the same name, the local variable shadows the global variable.

- Each function has its own local namespace.
  - Class methods follow the same scoping rule as ordinary functions.

- Python makes educated guesses on whether variables are local or global.
  - It assumes that any variable assigned a value in a function is local.

Namespaces and Scoping

- Therefore, in order to assign a value to a global variable within a function, you must first use the global statement.
- The statement *global VarName* tells Python that VarName is a global variable.
  - Python stops searching the local namespace for the variable.

```python
#!/usr/bin/python

Money = 2000
def AddMoney():
    # Uncomment the following line to fix the code:
    # global Money
    Money = Money + 1

print Money
AddMoney()
print Money
```
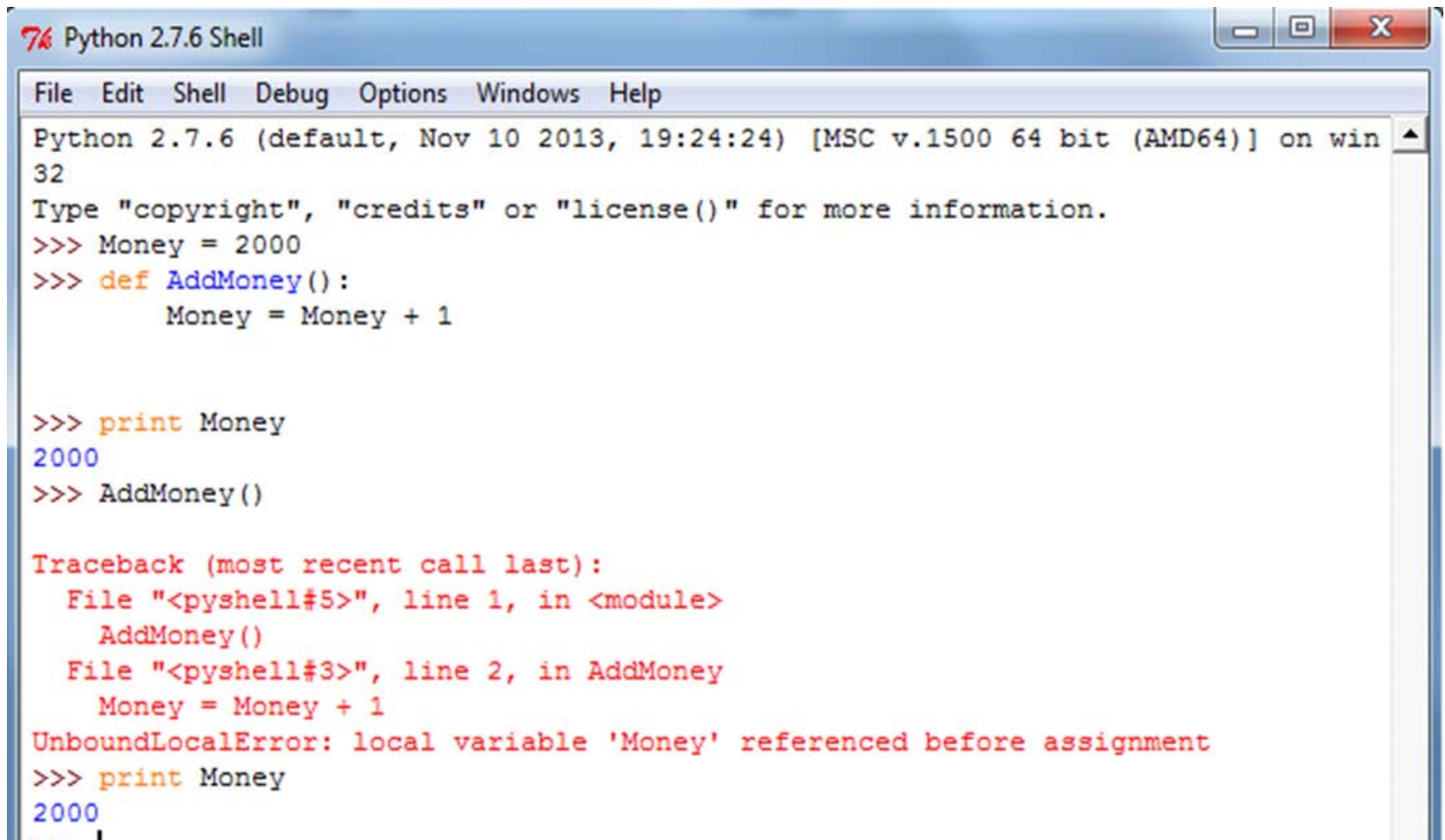
```
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    AddMoney()
  File "<pyshell#3>", line 2, in AddMoney
    Money = Money + 1
UnboundLocalError: local variable 'Money' referenced before assignment
```

Results


```
7 Python 2.7.6 Shell

File  Edit  Shell  Debug  Options  Windows  Help

Python 2.7.6 (default, Nov 10 2013, 19:24:24) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> Money = 2000
>>> def AddMoney():
        Money = Money + 1


>>> print Money
2000
>>> AddMoney()

Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    AddMoney()
  File "<pyshell#3>", line 2, in AddMoney
    Money = Money + 1
UnboundLocalError: local variable 'Money' referenced before assignment
>>> print Money
2000
```

Example

```
# sample.py
myGlobal = 5

def func1():
    myGlobal = 42

def func2():
    print myGlobal

func1()
func2()
```
5

```
def func1():
    global myGlobal
    myGlobal = 42
```

42

The dir( ) Function

- The dir() built-in function <u>returns a sorted list of strings containing the names defined by a module.</u>

- The list contains the names of all the modules, variables and functions that are defined in a module.

- Here, the special string variable __*name*__ is the **module's name**, and __*file*__ is the **filename** from which the module was loaded.

```
#!/usr/bin/python

# Import built-in module math
import math

content = dir(math)

print content;
```

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

The *globals()* and *locals()* Functions

- The *globals()* and *locals()* functions can be used to return the names in the global and local namespaces depending on the location from where they are called.

- If locals() is called from within a function, it will return all the names that can be accessed **locally** from that function.

- If globals() is called from within a function, it will return all the names that can be accessed **globally** from that function.

- The return type of both these functions is dictionary.
  - Therefore, names can be extracted using the keys() function.

Packages in Python

- A package is a hierarchical file directory structure
  - It defines a single Python application environment that consists of <u>modules</u> and <u>subpackages</u> and <u>sub-subpackages</u>, and so on.
- Consider a file *Pots.py* available in *Phone* directory
- We have another two files having different functions with the same directory as above:
  - *Phone/Isdn.py* file having function <u>Isdn()</u>
  - *Phone/G3.py* file having function <u>G3()</u>
- Now, create one more file __init__.py in *Phone* directory:
  - Phone/__init__.py

Packages in Python

- To make all of your functions available when you've imported Phone, you need to put explicit import statements in __init__.py as follows:
  - from Pots import Pots
  - from Isdn import Isdn
  - from G3 import G3

```python
#!/usr/bin/python

# Now import your Phone Package.
import Phone

Phone.Pots()
Phone.Isdn()
Phone.G3()
```

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

Python Image Library - Examples

Original image

- import Image
- global ext
- ext = ".jpg"
- imageFile = "test.jpg"
- im1 = Image.open(imageFile)
- Im1.show()

- **Python Imaging Library (PIL)**
- http://www.pythonware.com/products/pil/

Resize

- def imgResize(im):
-     div = 2
-     width = im.size[0] / div
-     height = im.size[1] / div

-     im2 = im.resize((width, height), Image.NEAREST) # use nearest neighbour
-     im3 = im.resize((width, height), Image.BILINEAR) # linear interpolation in a 2x2 environment
-     im4 = im.resize((width, height), Image.BICUBIC) # cubic spline interpolation in a 4x4 environment
-     im5 = im.resize((width, height), Image.ANTIALIAS) # best down-sizing filter

-     im2.save("NEAREST" + ext)
-     im3.save("BILINEAR" + ext)
-     im4.save("BICUBIC" + ext)
-     im5.save("ANTIALIAS" + ext)
- imgResize(im1)

Resize

Crop

- def imgCrop(im):
-     box = (50, 50, 200, 300)
-     region = im.crop(box)
-     region.save("CROPPED" + ext)
- imgCrop(im1)

Transpose

- def imgTranspose(im):
-   box = (50, 50, 200, 300)
-   region = im.crop(box)
-   region =region.transpose(Image.ROTATE_180)
-   im.paste(region, box)
-   im.save("TRANSPOSE"+ext)
- imgTranspose(im1)

Band Merge

- def bandMerge(im):
-    r, g, b = im.split()
-    im = Image.merge("RGB", (g,g,g))
-    im.save("MERGE" + ext)
- bandMerge(im1)

Blur

- import ImageFilter
- def filterBlur(im):
-    im1 = im.filter(ImageFilter.BLUR)
-    im1.save("BLUR" + ext)
- filterBlur(im1)

Find contours

- def filterContour(im):
-    im1 = im.filter(ImageFilter.CONTOUR)
-    im1.save("CONTOUR" + ext)
- filterContour(im1)

Find edges

- def filterFindEdges(im):
-    im1 = im.filter(ImageFilter.FIND_EDGES)
-    im1.save("EDGES" + ext)
- filterFindEdges(im1)

What is CGI?

- The Common Gateway Interface (CGI)
  - is a set of standards that define how information is exchanged between the web server and a custom script.
  - is a standard for external gateway programs to interface with information servers such as HTTP servers.
- The current version is CGI/1.1 and CGI/1.2 is under progress.
- Web Browsing
  - Your browser contacts the HTTP web server and demands for the URL i.e., filename.
  - Web Server will parse the URL and will look for the filename in if it finds that file then sends it back to the browser, otherwise sends an error message indicating that you have requested a wrong file.

CGI Architecture Diagram

– Web browser takes response from web server and displays either the received file or error message.
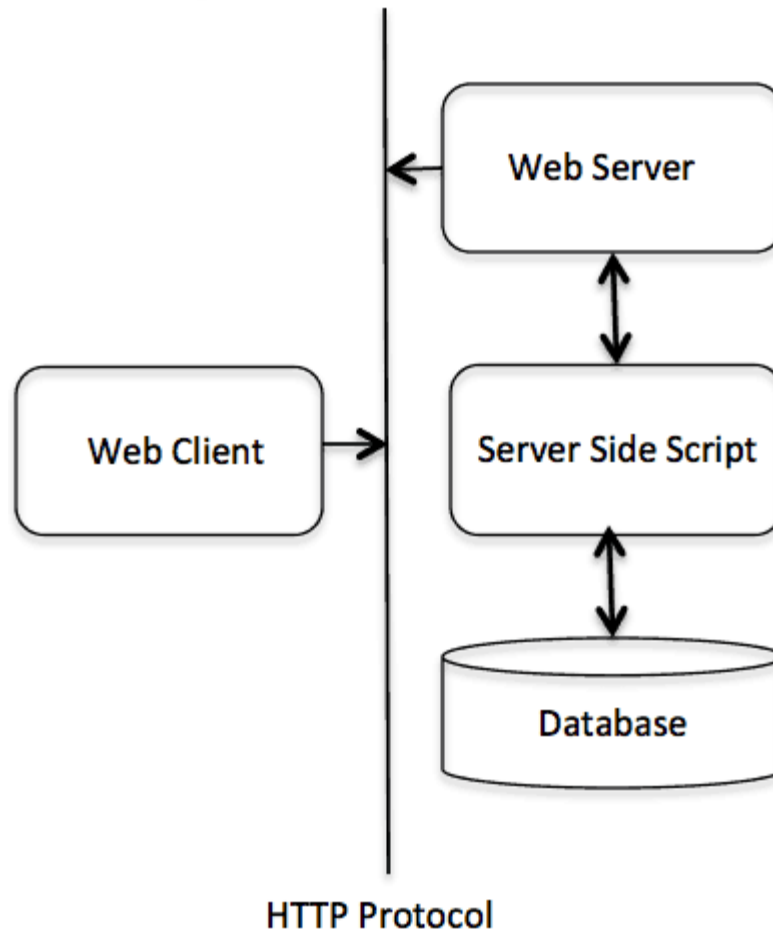
Web Server Support & Configuration

- Please make sure that your Web Server supports CGI and it is configured to handle CGI Programs.

- All the CGI Programs to be executed by the HTTP server are kept in a pre-configured directory.
  - This directory is called CGI Directory and by convention it is named as /var/www/cgi-bin

```
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>

<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

First CGI Program

```python
#!/usr/bin/python

print "Content-type:text/html\r\n\r\n"
print '<html>'
print '<head>'
print '<title>Hello Word - First CGI Program</title>'
print '</head>'
print '<body>'
print '<h2>Hello Word! This is my first CGI program</h2>'
print '</body>'
print '</html>'
```

- If you click hello.py, then this produces the following output:

Hello Word! This is my first CGI program

- There is one important and extra feature available which is first line to be printed **Content-type:text/html\r\n\r\n**.

- This line is sent back to the browser and specifiy the content type to be displayed on the browser screen.

HTTP Header

- All the HTTP header will be in the following form:

```
HTTP Field Name: Field Content

For Example
Content-type: text/html\r\n\r\n
```

- There are few other important HTTP headers, which you will use frequently in your CGI Programming.

Header	Description
Content-type:	A MIME string defining the format of the file being returned. Example is Content-type:text/html
Expires: Date	The date the information becomes invalid. This should be used by the browser to decide when a page needs to be refreshed. A valid date string should be in the format 01 Jan 1998 12:00:00 GMT.
Location: URL	The URL that should be returned instead of the URL requested. You can use this field to redirect a request to any file.
Last-modified: Date	The date of last modification of the resource.
Content-length: N	The length, in bytes, of the data being returned. The browser uses this value to report the estimated download time for a file.
Set-Cookie: String	Set the cookie passed through the *string*

CGI Environment Variables

- All the CGI program will have access to the following environment variables.

Variable Name	Description
CONTENT_TYPE	The data type of the content. Used when the client is sending attached content to the server. For example, file upload, etc.
CONTENT_LENGTH	The length of the query information. It's available only for POST requests.
HTTP_COOKIE	Returns the set cookies in the form of key & value pair.
HTTP_USER_AGENT	The User-Agent request-header field contains information about the user agent originating the request. Its name of the web browser.
PATH_INFO	The path for the CGI script.
QUERY_STRING	The URL-encoded information that is sent with GET method request.
REMOTE_ADDR	The IP address of the remote host making the request. This can be useful for logging or for authentication purpose.
REMOTE_HOST	The fully qualified name of the host making the request. If this information is not available then REMOTE_ADDR can be used to get IR address.
REQUEST_METHOD	The method used to make the request. The most common methods are GET and POST.
SCRIPT_FILENAME	The full path to the CGI script.
SCRIPT_NAME	The name of the CGI script.
SERVER_NAME	The server's hostname or IP Address
SERVER_SOFTWARE	The name and version of the software the server is running.

Python GUI Programming (Tkinter)

- Python provides various options for developing **graphical user interfaces (GUIs)**.

- Most important are listed below:

  1. **Tkinter:** Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.

  2. **wxPython:** This is an open-source Python interface for wxWindows http://wxpython.org.

  3. **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.

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.

- All you need to do is perform the following steps:

  1. Import the *Tkinter* module.

  2. Create the GUI application main window.

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

  4. Enter the <u>main event loop</u> to take action against each event triggered by the user.

Example

```
#!/usr/bin/python

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

Tkinter

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.

Standard Attributes

- Let's take a look at how some of their common attributes, such as sizes, colors and fonts are specified.
  - Dimensions
  - Colors
  - Fonts
  - Anchors
  - Relief styles
  - Bitmaps
  - Cursors

Geometry Management

- All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area.

- Tkinter exposes the following geometry manager classes: pack, grid, and place.

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

- The *grid()* Method - This geometry manager organizes widgets in a table-like structure in the parent widget.

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

Example

- 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)

Example

- bluebutton = Button(frame, text="Blue", fg="blue")
- bluebutton.pack(side = LEFT)
- blackbutton = Button(bottomframe, text="Black", fg="black")
- blackbutton.pack(side = BOTTOM)
- root.mainloop()

Example

- Import Tkinter
- class GUIDemo(Frame): # (inherit) Tkinter Frame
-    def __init__(self, master=None):
-       Frame.__init__(self, master)
-       self.grid()
-       self.createWidgets()
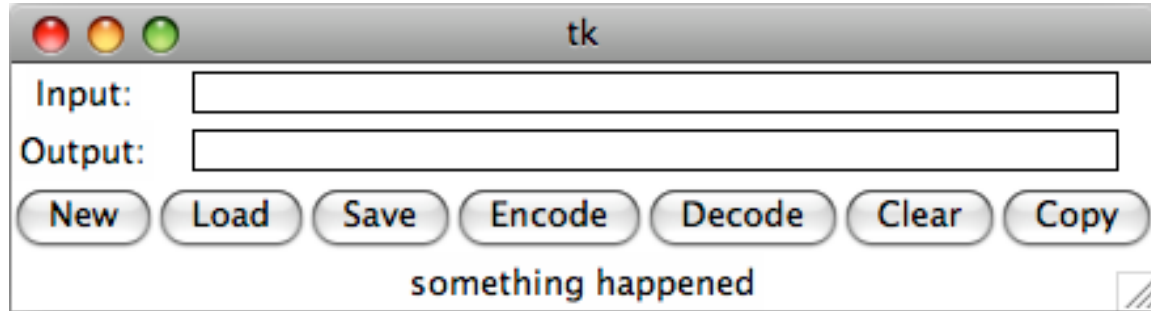
-    def createWidgets(self):
- # input
-       self.inputText = Label(self)
-       self.inputText["text"] = "Input:"
-       self.inputText.grid(row=0, column=0)
-       self.inputField = Entry(self)
-       self.inputField["width"] = 50
-       self.inputField.grid(row=0, column=1, columnspan=6)
- #output
-       self.outputText = Label(self)
-       self.outputText["text"] = "Output:"
-       self.outputText.grid(row=1, column=0)
-       self.outputField = Entry(self)
-       self.outputField["width"] = 50
-       self.outputField.grid(row=1, column=1, columnspan=6)

tk

Input:

Output:

New | Load | Save | Encode | Decode | Clear | Copy

something happened

```python
self.new = Button(self)
self.new["text"] = "New"
self.new.grid(row=2, column=0)


self.load = Button(self)
self.load["text"] = "Load"
self.load.grid(row=2, column=1)

self.save = Button(self)
self.save["text"] = "Save"
self.save.grid(row=2, column=2)


self.encode = Button(self)
self.encode["text"] = "Encode"
self.encode.grid(row=2, column=3)


self.decode = Button(self)
self.decode["text"] = "Decode"
self.decode.grid(row=2, column=4)
```

- self.clear = Button(self)
- self.clear["text"] = "Clear"
- self.clear.grid(row=2, column=5)

- self.copy = Button(self)
- self.copy["text"] = "Copy"
- self.copy.grid(row=2, column=6)

- self.displayText = Label(self)
- self.displayText["text"] = "something happened"
- self.displayText.grid(row=3, column=0, columnspan=7)

- if __name__ == '__main__':
-    root = Tk()
-    app = GUIDemo(master=root)
-    app.mainloop()

File   Edit   Format   Run   Options   Windows   Help
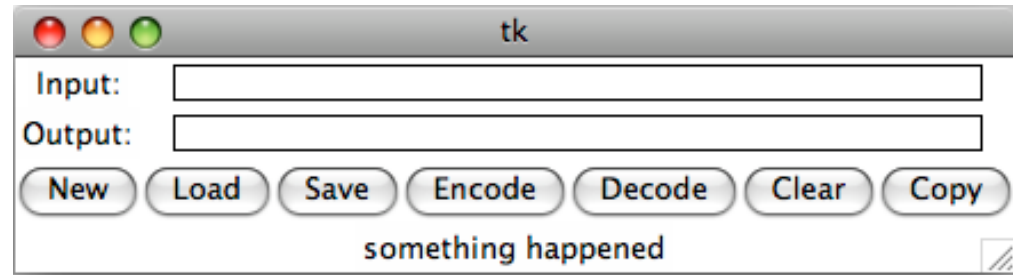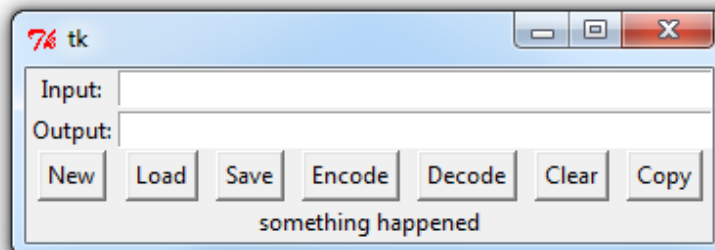
```python
from Tkinter import *

class GUIDemo(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.grid()
        self.createWidgets()

    def createWidgets(self):
        self.inputText = Label(self)
        self.inputText["text"] = "Input:"
        self.inputText.grid(row=0, column=0)
        self.inputField = Entry(self)
        self.inputField["width"] = 50
        self.inputField.grid(row=0, column=1, columnspan=6)

        self.outputText = Label(self)
        self.outputText["text"] = "Output:"
        self.outputText.grid(row=1, column=0)
        self.outputField = Entry(self)
        self.outputField["width"] = 50
        self.outputField.grid(row=1, column=1, columnspan=6)

        self.new = Button(self)
        self.new["text"] = "New"
        self.new.grid(row=2, column=0)
        self.load = Button(self)
        self.load["text"] = "Load"
        self.load.grid(row=2, column=1)
        self.save = Button(self)
        self.save["text"] = "Save"
        self.save.grid(row=2, column=2)
        self.encode = Button(self)
        self.encode["text"] = "Encode"
        self.encode.grid(row=2, column=3)
        self.decode = Button(self)
        self.decode["text"] = "Decode"
        self.decode.grid(row=2, column=4)
        self.clear = Button(self)
        self.clear["text"] = "Clear"
        self.clear.grid(row=2, column=5)
```
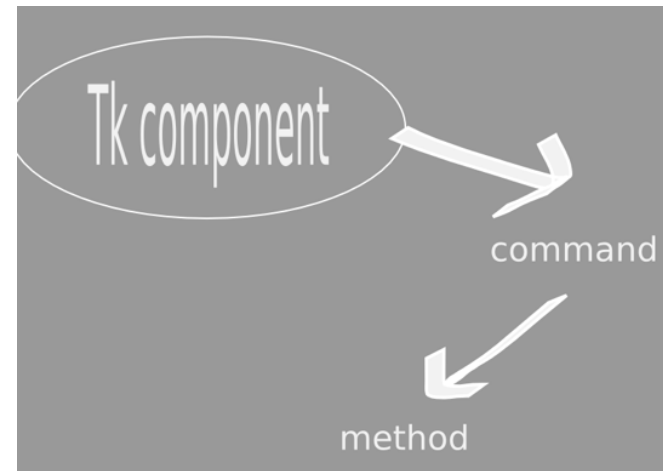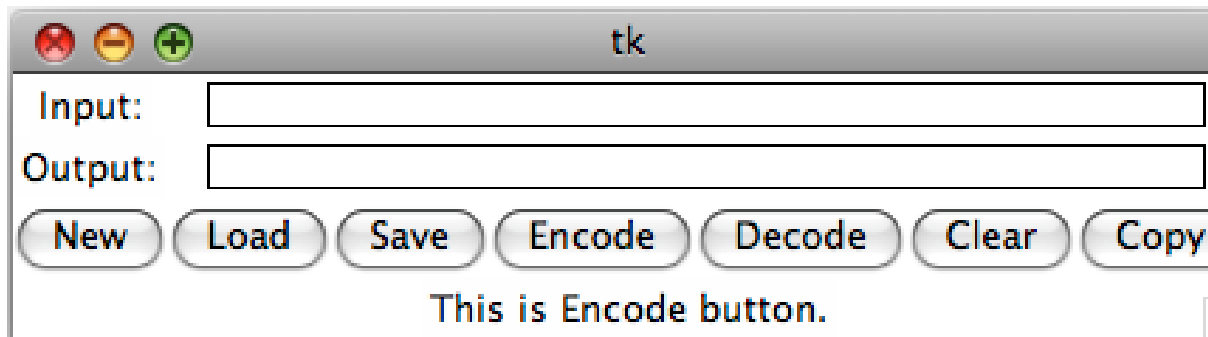
tk

Input:
Output:

New   Load   Save   Encode   Decode   Clear   Copy

something happened

Ln: 1   Col: 6

Command

Tk component → command → method

- self.new["command"] = self.newMethod

- def newMethod(self):

    self.displayText["text"] = "This is New button."


```
tk
Input:  [                              ]
Output: [                              ]
(New) (Load) (Save) (Encode) (Decode) (Clear) (Copy)
        This is Encode button.
```

- Add commands to New, Load, Save, Encode, Decode, Clear, and Copy

Command

- self.new = Button(self)
- self.new["text"] = "New"
- self.new.grid(row=2, column=0)
- self.new["command"] = self.newMethod
- self.load = Button(self)
- self.load["text"] = "Load"
- self.load.grid(row=2, column=1)
- self.load["command"] = self.loadMethod
- self.save = Button(self)
- self.save["text"] = "Save"
- self.save.grid(row=2, column=2)
- self.save["command"] = self.saveMethod

- self.encode = Button(self)
- self.encode["text"] = "Encode"
- self.encode.grid(row=2, column=3)
- self.encode["command"] = self.encodeMethod
- self.decode = Button(self)
- self.decode["text"] = "Decode"
- self.decode.grid(row=2, column=4)
- self.decode["command"] = self.decodeMethod
- self.clear = Button(self)
- self.clear["text"] = "Clear"
- self.clear.grid(row=2, column=5)
- self.clear["command"] = self.clearMethod
- self.copy = Button(self)
- self.copy["text"] = "Copy"
- self.copy.grid(row=2, column=6)
- self.copy["command"] = self.copyMethod
- self.displayText = Label(self)
- self.displayText["text"] = "something happened"
- self.displayText.grid(row=3, column=0, columnspan=7)

```python
def newMethod(self):
    self.displayText["text"] = "This is New button."
def loadMethod(self):
    self.displayText["text"] = "This is Load button."
def saveMethod(self):
    self.displayText["text"] = "This is Save button."
def encodeMethod(self):
    self.displayText["text"] = "This is Encode button."
def decodeMethod(self):
    self.displayText["text"] = "This is Decode button."
def clearMethod(self):
    self.displayText["text"] = "This is Clear button."
def copyMethod(self):
    self.displayText["text"] = "This is Copy button."
```
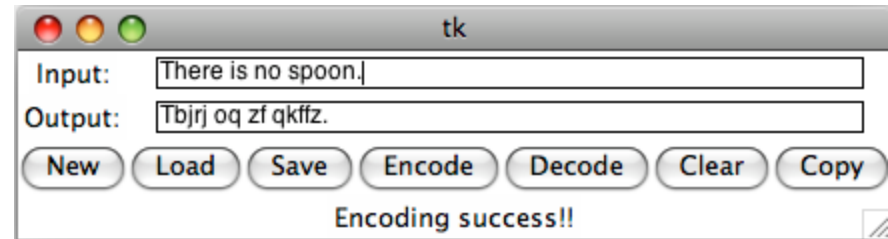
Encrypt

- Import Tkinter
- import Encrypt
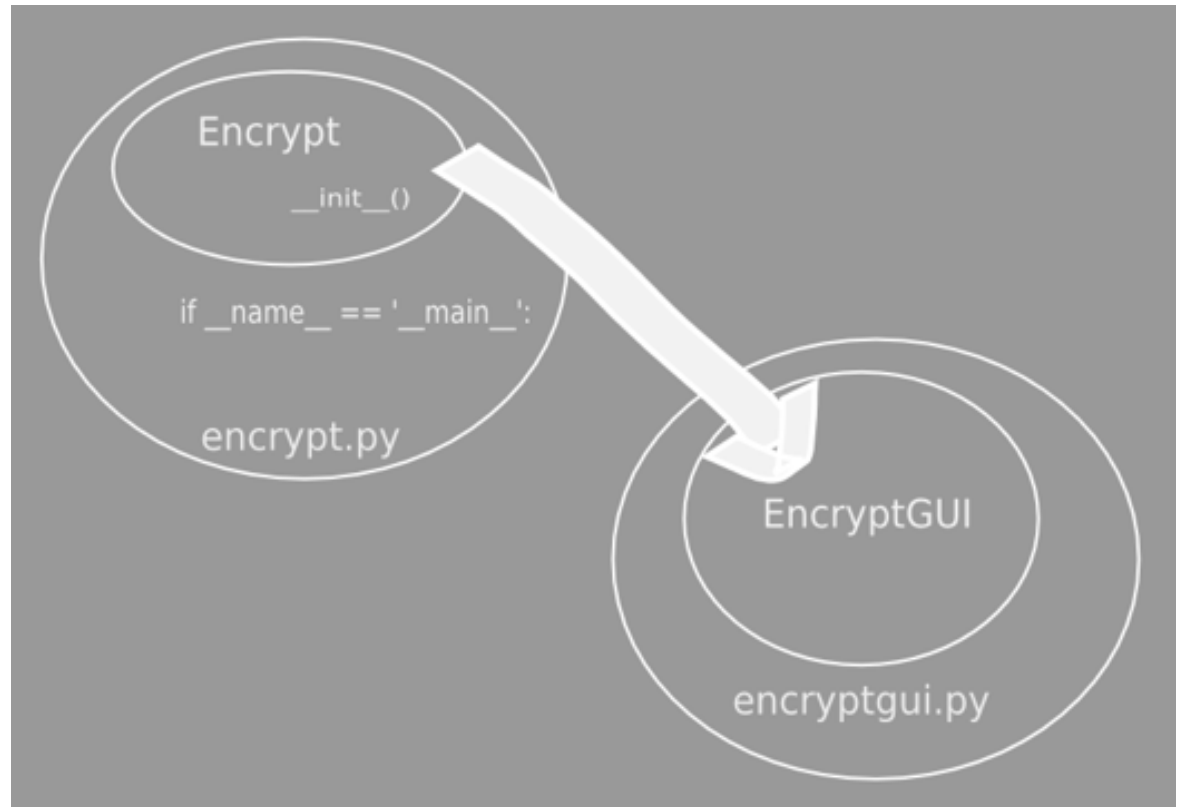
encodeMethod

- def encodeMethod(self):
-     self.userinput = self.inputField.get()

-     if self.userinput == "":
-       self.displayText["text"] = "No input string!!"
-     else:
-       if self.e == None:
-         self.displayText["text"] = "No encrypt object!!"
-       else:
-         self.result = self.e.toEncode(self.userinput)
-         self.outputField.delete(0, 200)
-         self.outputField.insert(0, self.result)
-         self.displayText["text"] = "Encoding success!!"

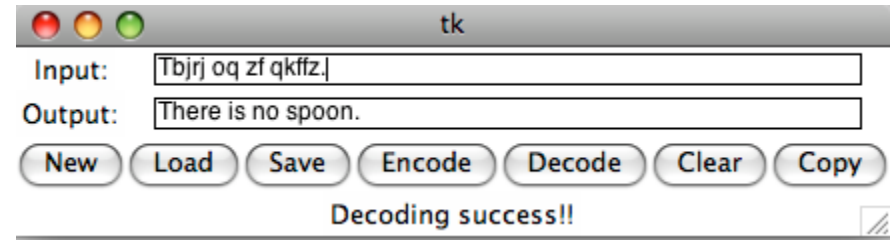# *decodeMethod*

- def decodeMethod(self):
-     self.userinput = self.inputField.get()

-     if self.userinput == "":
-       self.displayText["text"] = "No input string!!"
-     else:
-       if self.e == None:
-         self.displayText["text"] = "No encrypt object!!"
-       else:
-         self.result = self.e.toDecode(self.userinput)
-         self.outputField.delete(0, 200)
-         self.outputField.insert(0, self.result)
-         self.displayText["text"] = "Decoding success!!"

Save

```python
def saveMethod(self):
    if self.e == None:
        self.displayText["text"] = "No Encrypt object can save!!"
    else:
        f = open('./code.txt', 'w')
        f.write(self.e.getCode())
        f.closed
        self.displayText["text"] = "The code is saved."
```

Load

- def loadMethod(self):
-     if os.path.exists("./code.txt"):
-         f = open('./code.txt', 'r')
-         code = f.readline()
-         self.e = Encrypt()
-         self.e.setCode(code)
-         self.displayText["text"] = "code: " + self.e.getCode()
-     else:
-         self.displayText["text"] = "Load denied!!"

Clear

- def clearMethod(self):
-     self.e = None
-     self.userinput = ""
-     self.result = ""
-     self.inputField.delete(0, 200)
-     self.outputField.delete(0, 200)
-     self.displayText["text"] = "It's done."

Copy

- def copyMethod(self):
-    if self.result == "":
-       self.displayText["text"] = "Copy denied!!"
-    else:
-       self.clipboard_clear()
-       self.clipboard_append(self.result)
-       self.displayText["text"] = "It is already copied to the clipboard."

New

- def newMethod(self):
-       self.e = Encrypt()
-       self.displayText["text"] = self.e

encrypt.py

```python
import random
class Encrypt:
    def __init__(self):
        self.code = [chr(i) for i in range(97, 123)]
        random.shuffle(self.code)
        self.alph = [chr(i) for i in range(97, 123)]

    def __str__(self):
        return "code: " + "".join(self.code)

    def setCode(self, data):
        self.code = list(data)

    def getCode(self):
        return "".join(self.code)

    def toEncode(self, s):
        result = ""
        for i in s:
            if i in self.code:
                j = self.alph.index(i)
                result += self.code[j]
            else:
                result += i

        return result

    def toDecode(self, s):
        result = ""
        for i in s:
            if i in self.code:
                j = self.code.index(i)
                result += self.alph[j]
            else:
                result += i

        return result
```

encrypt.py

- if __name__ == '__main__':
-     e = Encrypt()
-     print()
-     print(e)
-     s1 = "There is no spoon."
-     print("input: " + s1)
-     s2 = e.toEncode(s1)
-     print("encode: " + s2)
-     s3 = e.toDecode(s2)
-     print("decode: " + s3)
-     print()

```python
from Tkinter import *
import Tkinter

class GUIDemo(Tkinter.Frame):
    def __init__(self, master=None):
        Tkinter.Frame.__init__(self, master)
        self.grid()
        self.createWidgets()
        self.num = 0
        self.inputNumber = 0
        self.answer = 0
        self.inputFlag = 0
        self.op = 0
        self.opFlag = 0
        self.dotFlag = 0
        self.content = 0
        self.idx = -1

    def createWidgets(self):
        self.outputField = Entry(self)
        self.outputField["width"] = 32
        self.outputField.insert(0,"0")
        self.outputField.grid(row=0, column=0, columnspan=4)

        self.back = Button(self)
        self.back["width"] = 4
        self.back["text"] = "<-"
        self.back.grid(row=0, column=4)
        self.back["command"]= self.backMethod

        self.zero = Button(self)
        self.zero["width"] = 12
        self.zero["text"] = "0"
        self.zero.grid(row=4, column=0, columnspan=2)
        self.zero["command"]= self.zeroMethod

        self.one = Button(self)
        self.one["width"] = 4
        self.one["text"] = "1"
        self.one.grid(row=3, column=0)
```
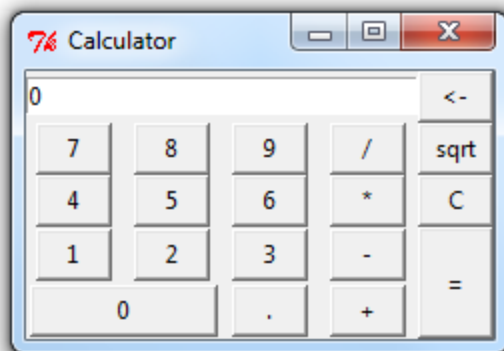
Calculator

```
0                          <-

  7      8      9      /    sqrt

  4      5      6      *     C

  1      2      3      -
                             =
      0           .    +
```

createWidgets

- self.outputField = Entry(self)
- self.outputField["width"] = 32
- self.outputField.insert(0,"0")
- self.outputField.grid(row=0, column=0, columnspan=4)

- self.back = Button(self)
- self.back["width"] = 4
- self.back["text"] = "<-"
- self.back.grid(row=0, column=4)
- self.back["command"]= self.backMethod

- self.zero = Button(self)
- self.zero["width"] = 12
- self.zero["text"] = "0"
- self.zero.grid(row=4, column=0, columnspan=2)
- self.zero["command"]= self.zeroMethod

createWidgets

- self.one = Button(self)
- self.one["width"] = 4
- self.one["text"] = "1"
- self.one.grid(row=3, column=0)
- self.one["command"]= self.oneMethod

- self.two = Button(self)
- self.two["width"] = 4
- self.two["text"] = "2"
- self.two.grid(row=3, column=1)
- self.two["command"]= self.twoMethod

command

- def numberMethod(self):
-    if self.dotFlag == 1: # float situation
-       self.content = self.outputField.get() + str(self.inputNumber)
-       self.num = float(self.content)
-    elif self.inputFlag == 0:
-       self.num = self.inputNumber
-    else: # non-float
-       self.num = 10 * self.num + self.inputNumber
-    self.inputFlag = 1
-    self.outputField.delete(0, 40)
-    self.outputField.insert(0, self.num)

command

- def zeroMethod(self):
- self.inputNumber = 0
- self.numberMethod()

- def oneMethod(self):
- self.inputNumber = 1
- self.numberMethod()

- def twoMethod(self):
- self.inputNumber = 2
- self.numberMethod()

- def threeMethod(self):
- self.inputNumber = 3
- self.numberMethod()

equalMethod

```python
def equalMethod(self):
    if self.op == '+':
        self.answer = self.answer + self.num
    elif self.op == '-':
        self.answer = self.answer - self.num
    elif self.op == '*':
        self.answer = self.answer * self.num
    elif self.op == '/':
        if self.num == 0:
            self.answer = 'NAN'
        else:
            self.answer = self.answer / self.num
    else:
        self.answer = self.num
        self.dotFlag = 0
    if self.answer == 0.0:
        self.answer = 0
        self.dotFlag = 0
    self.outputField.delete(0, 40)
    self.outputField.insert(0, self.answer)
    self.inputFlag = 0
    self.opFlag = 0
```

sqrtMethod

- def sqrtMethod(self):
-     if self.opFlag == 0:
-       if self.answer < 0:
-         self.outputField.delete(0, 40)
-         self.outputField.insert(0, "invalid")
-         return
-       self.answer = self.answer ** 0.5
-       self.outputField.delete(0, 40)
-       self.outputField.insert(0, self.answer)
-     else:
-       if self.num < 0:
-         self.outputField.delete(0, 40)
-         self.outputField.insert(0, "invalid")
-         return
-       self.num = self.num ** 0.5
-       self.c()
-     self.inputFlag = 1
-     self.op = 'sqrt'

dotMethod

- def dotMethod(self):
- if self.opFlag == 0 and self.dotFlag == 0 and self.inputFlag == 0:
- self.answer = 0
- self.outputField.delete(0, 40)
- self.outputField.insert(0, str(self.answer) + ".")
-
- elif self.inputFlag == 0:
- self.num = 0.0
- else:
- self.outputField.delete(0, 40)
- self.outputField.insert(0, str(self.num) + ".")
- self.dotFlag = 1

Python Network Programming

- Python provides two levels of access to network services.

- At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

- Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

What is Sockets?

- Sockets are the endpoints of a bidirectional communications channel.

- Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

- Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on.

- The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Term	Description
domain	The family of protocols that will be used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
type	The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: ◻ A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation ◻ A string "<broadcast>", which specifies an INADDR_BROADCAST address. ◻ A zero-length string, which specifies INADDR_ANY, or ◻ An Integer, interpreted as a binary address in host byte order.
port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The *socket* Module

- To create a socket, you must use the *socket.socket()* function available in *socket* module, which has the general syntax:

  s = socket.socket (socket_family, socket_type, protocol=0)

- Here is the description of the parameters:

  - **socket_family:** This is either AF_UNIX or AF_INET, as explained earlier.

  - **socket_type:** This is either SOCK_STREAM or SOCK_DGRAM.

  - **protocol:** This is usually left out, defaulting to 0.

- Once you have *socket* object, then you can use required functions to create your client or server program.

Server /Client Socket Methods

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

Method	Description
s.connect()	This method actively initiates TCP server connection.

General Socket Methods

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
socket.gethostname()	Returns the hostname.

A Simple Server

- To write Internet servers, we use the **socket** function available in socket module to create a socket object.
  - A socket object is used to call other functions to setup a socket server.
- Now call **bind(hostname, port)** function to specify a *port* for your service on the given host.
- Next, call the *accept* method of the returned object.
  - This method waits until a client connects to the port you specified, and then returns a *connection* object that represents the connection to that client.

```python
#!/usr/bin/python              # This is server.py file

import socket                  # Import socket module

s = socket.socket()            # Create a socket object
host = socket.gethostname()    # Get local machine name
port = 12345                   # Reserve a port for your service.
s.bind((host, port))           # Bind to the port

s.listen(5)                    # Now wait for client connection.
while True:
   c, addr = s.accept()        # Establish connection with client.
   print 'Got connection from', addr
   c.send('Thank you for connecting')
   c.close()                   # Close the connection
```

A Simple Client

- Now we will write a very simple client program which will open a connection to a given port 12345 and given host.

- This is very simple to create a socket client using Python's *socket* module function.

- The **socket.connect(hosname, port)** opens a TCP connection to *hostname* on the *port*.

  - Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

- The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits:

```python
#!/usr/bin/python              # This is client.py file

import socket                  # Import socket module

s = socket.socket()           # Create a socket object
host = socket.gethostname()   # Get local machine name
port = 12345                  # Reserve a port for your service.

s.connect((host, port))
print s.recv(1024)
s.close                       # Close the socket when done
```

```
# Following would start a server in background.
$ python server.py &

# Once server is started run client as follows:

$ python client.py
```

```
Got connection from ('127.0.0.1', 48437)
Thank you for connecting
```

Python Internet modules

Protocol	Common function	Port No	Python module
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib