Data Processing Using Python

Object-Oriented and Graphical User Interface

ZHANG Dazhuang

Department of Computer Science and Technology

Department of University Basic Computer Teaching

Data Processing Using Python

1

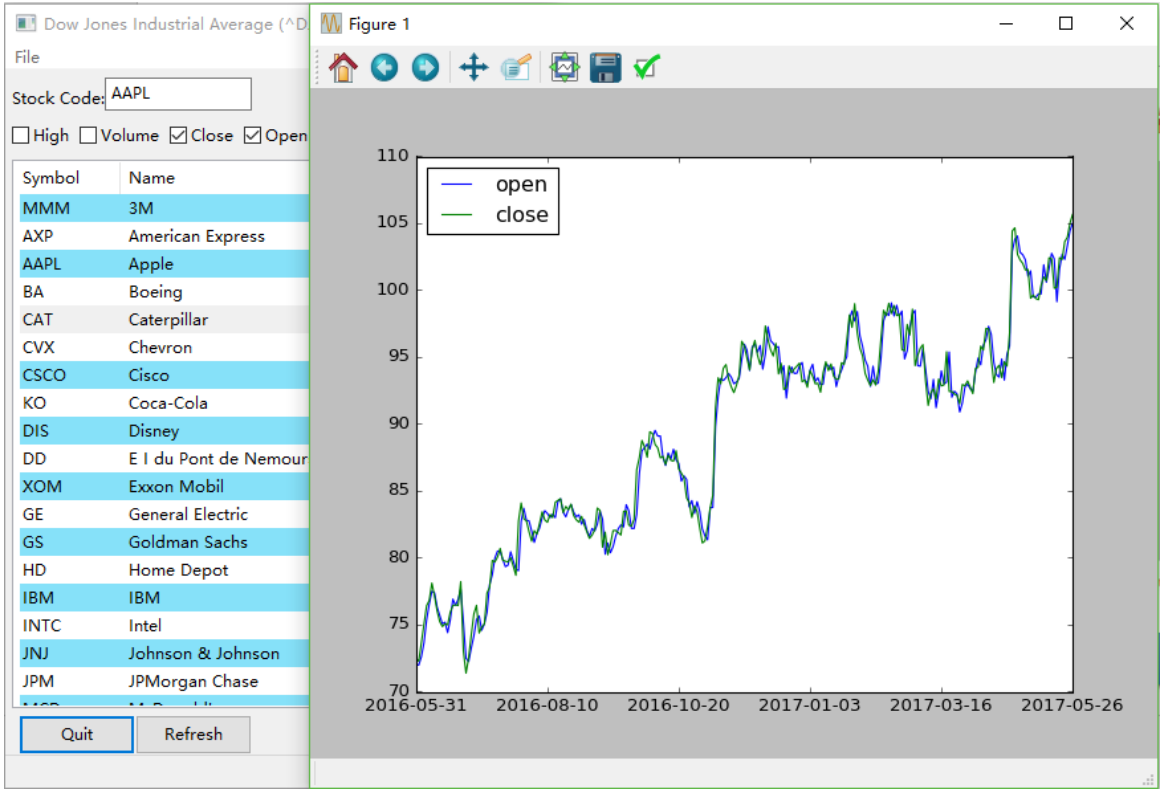
GUI AND OBJECT-ORIENTED

Character User Interface (CUI)

```
def foo():
    '''add function'''
    listA = []
    print('input the numbers: ')
    while True:
        num = input()
        if num == '.':
            break
        listA.append(eval(num))
    sumList = sum(listA)
    return sumList
```

```
>>> foo()
input the numbers:
3
5
6
7
.
21
```

Graphical User Interface (GUI)

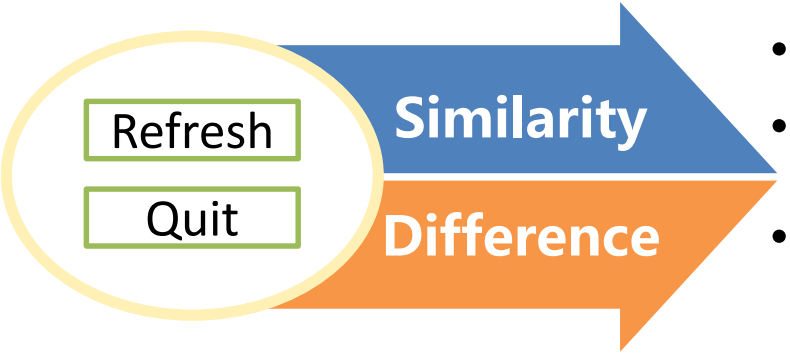


Data Processing Using Python

2 ABSTRACTION

Object-Oriented

- Object (Instance)
  - Data and operations on specific data
- Class
  - describe the feature of object (data & operation)

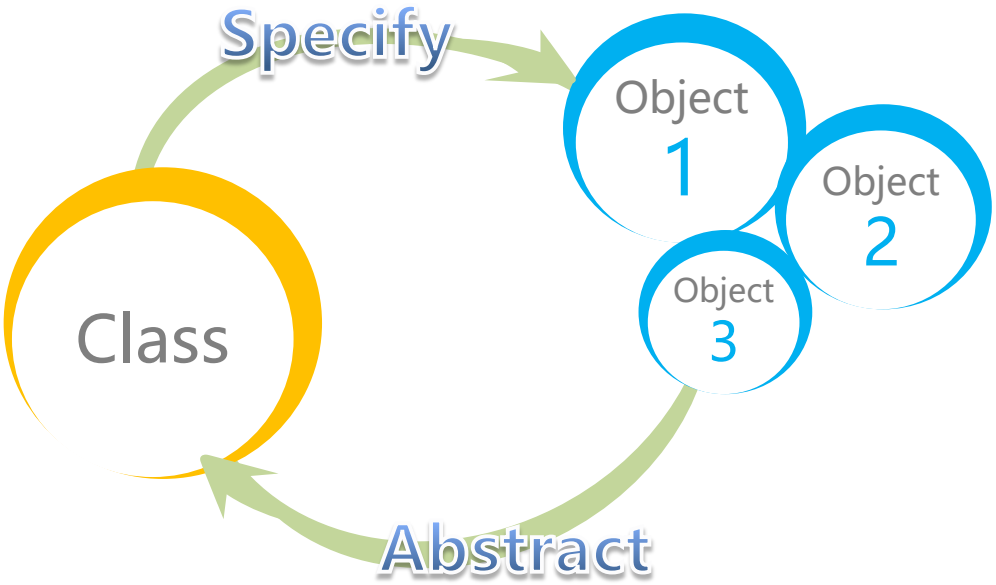


- Have a name
- Have a square frame
- React when clicked

- Different functions：Refresh、Quit

Abstraction

Relationship between Class and Object



Definition of Class

Source

```
class ClassName(object):
    'define ClassName class'
    class_suite
```

Source

```
class MyDate(object):
    'this is a  very simple
    example class'
    pass
```

Origin of all class——object

Methods of Class

- Method Definition

Source

```
>>> class Dog(object):
        def greet(self):
            print('Hi!')
```

Instances

Source

```
>>> class Dog(object):
        def greet(self):
            print('Hi!')
```

Source

```
>>> dog = Dog()
>>> dog.greet()
```

- Creation of instance——By calling the class object
  1. Define a class——Dog
  2. Create an instance——dog
  3. Use attributes or methods by instance——dog.greet

Instance Attributes

File

```
# Filename: doginsta.py
class Dog(object):
    "define Dog class"
    def setName(self, name):
        self.name = name
    def greet(self):
        print("Hi, I am called %s." % self.name)
if __name__ == '__main__':
    dog = Dog()
    dog.setName("Paul")
    dog.greet()
```

Output:
Hi, I am called Paul.

Initializing Method of Object __init__()

01 When a class is called, Python will create an instance

02 After creation, the first method Python automatically calls is __init__()

03 The instance will be passed as the first parameter (self) of the method, and all parameters in creation will be passed to __init__()

__init__() Example

File

```
# Filename: doginsta.py
class Dog(object):
    "define Dog class"
    def __init__(self, name):
        self.name = name
    def greet(self):
        print("Hi, I am called %s." % self.name)
if __name__ == '__main__':
    dog = Dog("Sara")
    dog.greet()
```

Output:
Hi, I am called Sara.

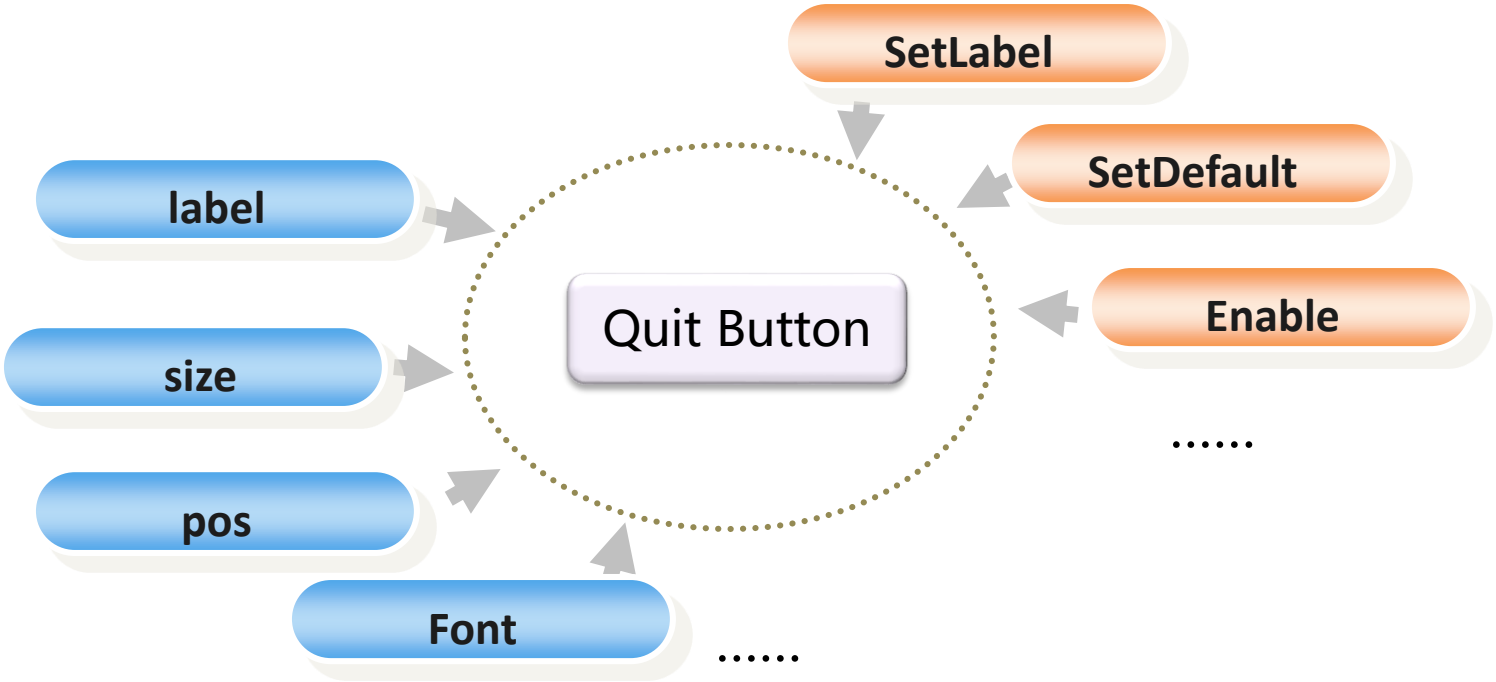
Class Attributes

- The data attributes (static members) of class are only variables for defined class
- Be used after creation of class
- Can be updated by both methods in class and main program
- Independent of instances, and the modification of class attributes should use the class name

File

```
# Filename: doginsta.py
class Dog(object):
   "define Dog class“
   counter = 0
   def __init__(self, name):
      self.name = name
      Dog.counter += 1
   def greet(self):
      print("Hi, I am %s, my number is %d" % (self.name,
Dog.counter))
if __name__ == '__main__':
    dog = Dog("Zara")
    dog.greet()
```

Use Button as an Example

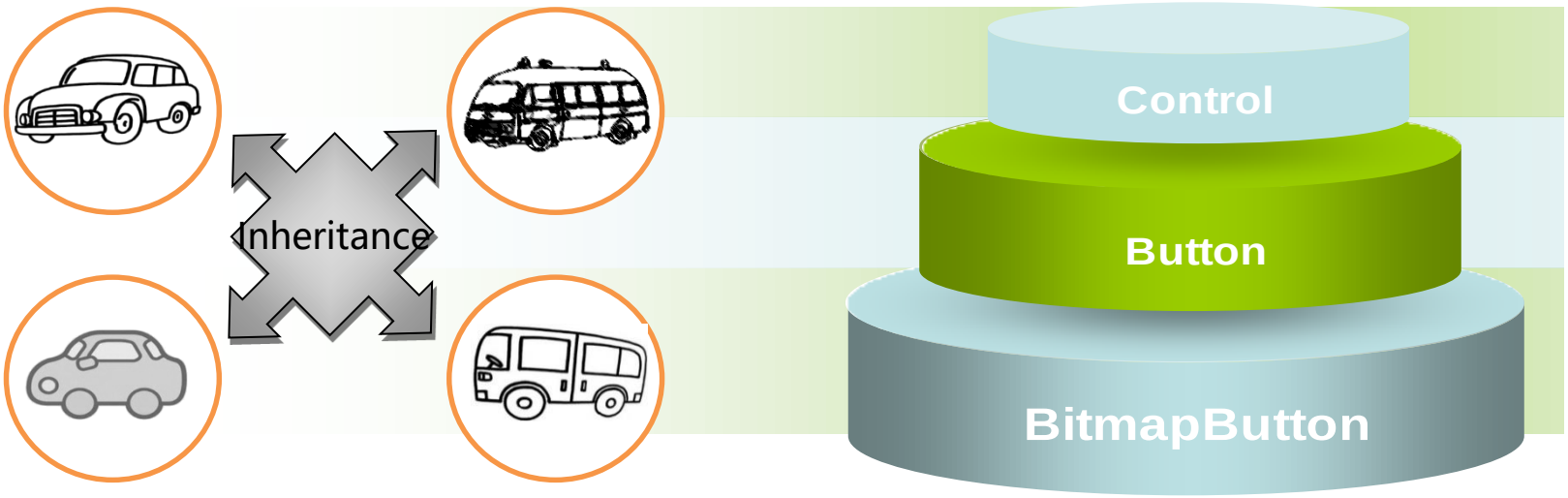


Data Processing Using Python

3

INHERIT

Base class & Derived Class

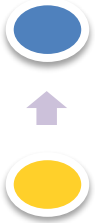


Derived Class/Subclass

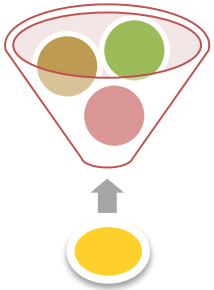
Source

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

Single Inheritance



Multiple Inheritance



Subclass Definition and Override

File

```
# Filename: doginsta.py
class Dog(object):
   "define Dog class“
   counter = 0
   def __init__(self, name):
      self.name = name
      Dog.counter += 1
   def greet(self):
      print("Hi, I am %s, my number is %d" %
            (self.name, Dog.counter))
```

File

```
# Filename: overridepro.py
class BarkingDog (Dog):
   "define subclass BarkingDog"
   def greet(self):
      "initial subclass"
       print("Woof! I am %s, my number is
%d" % (self.name, Dog.counter))
if __name__ == '__main__':
   dog = BarkingDog("Zoe")
   dog.greet()
```

Private Attribute and Method

- In default situation, the member attributes and methods are all "public"
- Python provide "access controller" to control the visit of member functions
  - Double underline (__)

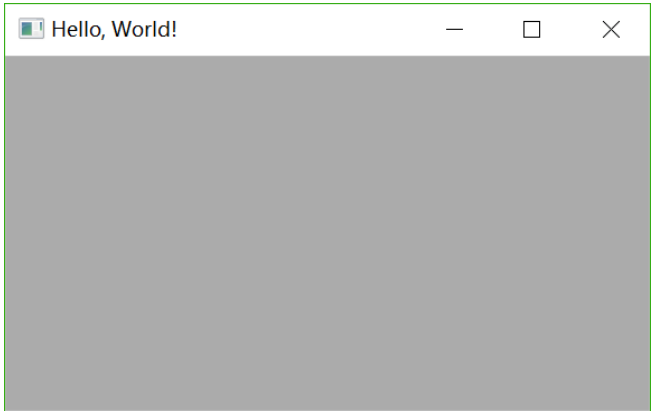
    __var attribute will be replaced by __classname_var, preventing the conflict of same name in base class and derived class
  - Single underline(_)

    Use a single underline before attributes to prevent attributes from being loaded by "from mymodule import *"

Data Processing Using Python

4 BASIC GUI FRAMEWORK

Create a simple wxPython Program



File

```
# Filename: firstwxPython.py
import wx
app = wx.App()
frame = wx.Frame(None, title = "Hello, World!")
frame.Show(True)
app.MainLoop()
```

The above case can also be modified as

```
# Filename: mouse.py
import wx

class MyApp(wx.App):
    def OnInit(self):
        frame = wx.Frame(None, title = "Hello, World!")
        frame.Show()
        return True
if __name__ == '__main__':
    app = MyApp()
    app.MainLoop()
```

The application object can also be an instance of wx.App' s subclass

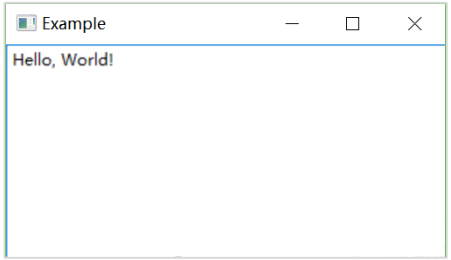
Widget

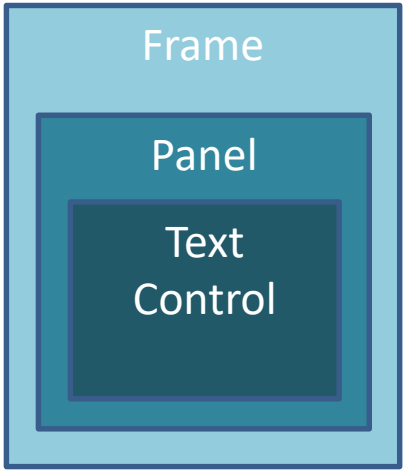
- Widget Containers——To contain other widgets
  - e.g. wx.Panel etc.
- Dynamic Widgets——Can be edited by users
  - e.g. wx.Button、wx.TextCtrl、wx.ListBox etc.
- Static Widgets——Can not be edited by users
  - e.g. wx.StaticBitmap、wx.StaticText、wxStaticLine etc.
- Others
  - e.g. wx.ToolBar、wx.MenuBar、wx.StatusBar

“Hello , World ! ” Again

File

```
# Filename: helloworld.py
import wx
class Frame1(wx.Frame):
    def __init__(self,superior):
        wx.Frame.__init__(self, parent = superior, title = "Example", pos=
(100,200), size= (350,200))
        panel = wx.Panel(self)
        text1= wx.TextCtrl(panel, value = "Hello, World!", size = (350,200))
if __name__ == '__main__':
    app =wx.App()
    frame = Frame1(None)
    frame.Show(True)
    app.MainLoop()
```





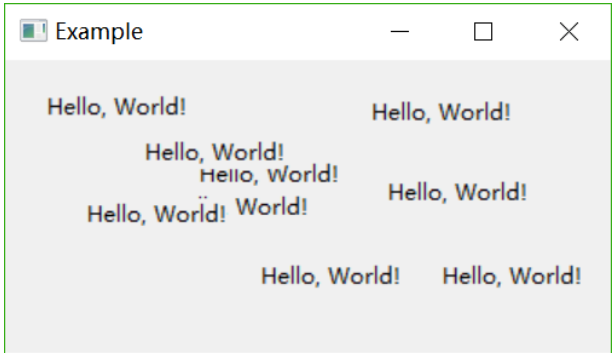
Event Handling

- Basic Mechanism of GUI programs——Event Handling
- Event
  - Move of mouse, left click, click on button, etc.
  - Can be created by user operations or programs
- wxPython associates certain kind of event with specific code (methods), when the event is created, related codes will be automatically executed.
  - E.g. : When a mouse move event is triggered, method OnMove() will be called

"Hello, World!" Again

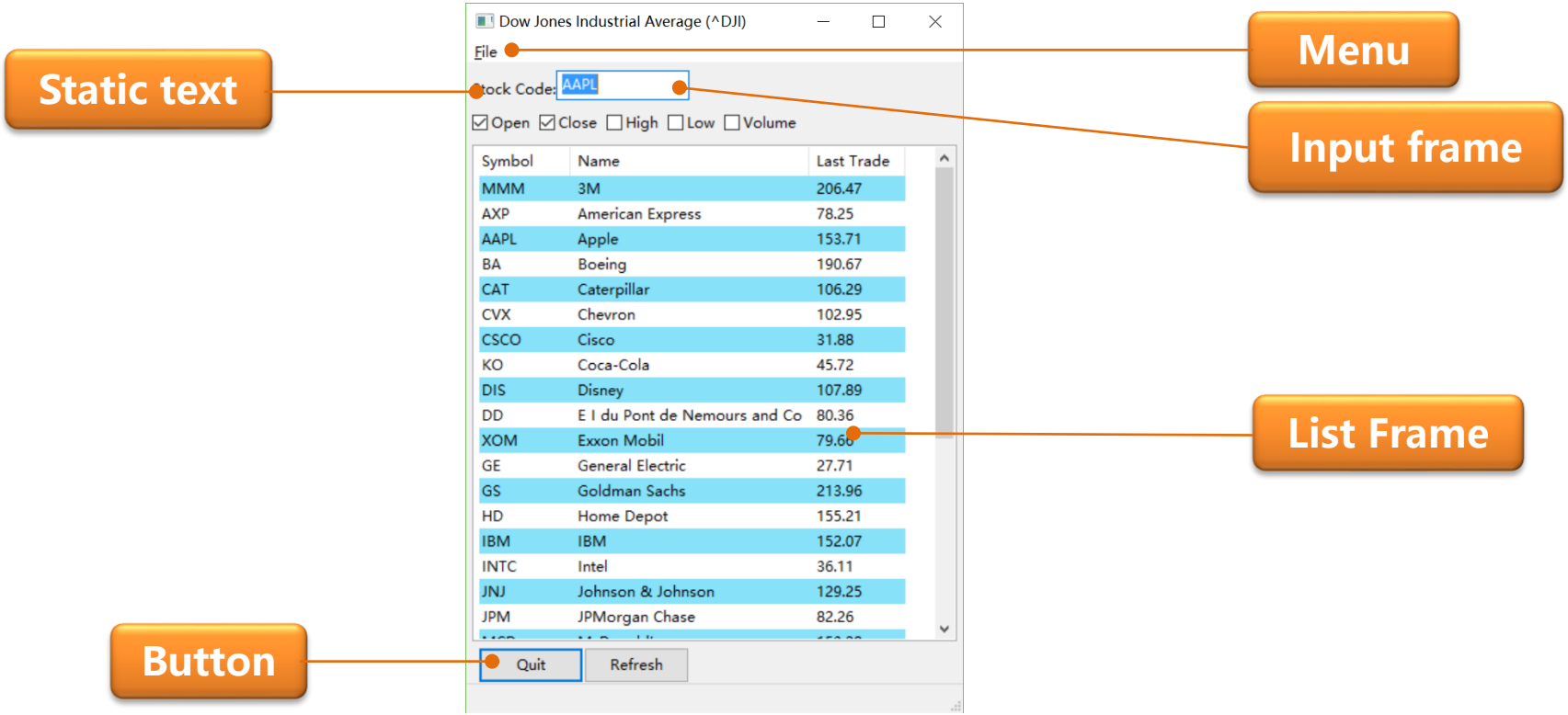
File

```
# Filename: mouse.py
import wx
class Frame1(wx.Frame):
    def __init__(self,superior):
        … …
        self.panel.Bind(wx.EVT_LEFT_UP, self.OnClick)
    def OnClick(self, event):
        posm = event.GetPosition()
        wx.StaticText(parent = self.panel,label = "Hello, World!",pos = (posm.x, posm.y))

…… #create app and frame, show and execute event loop
```



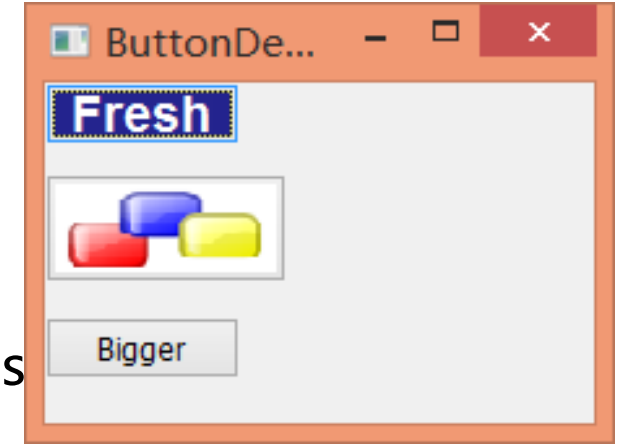
5 USEFUL GUI WIDGETS

Example of Application



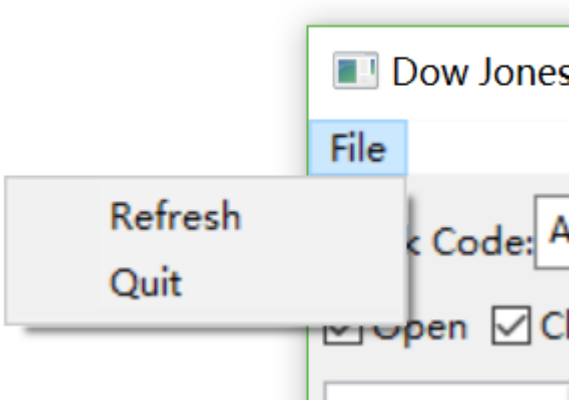
Button and Family

- Receive the click events and trigger corresponding operations
- Useful Button :
  - wx.Button : Text Button
  - wx.BitmapButton : Bitmap Button
  - wx.ToggleButton : Toggle Button
- Binding events with handling methods



Menu and Components

- Menu
  - Menu bar
  - Menu
  - Menu items
- wxPython classes for Menu :
  - wx.MenuBar
  - wx.Menu
  - wx.MenuItem



Useful Menu Events

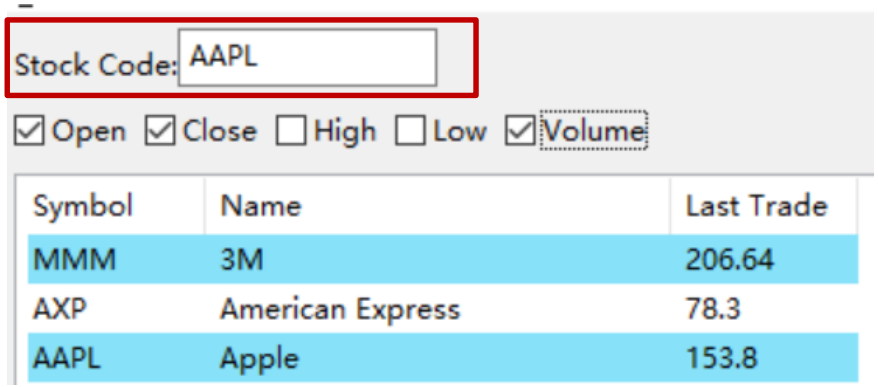
- Menu Events
  - wx.EVT_MENU

File

```
# Filename: menudemo.py
...

#Binding event handlers
    self.Bind(wx.EVT_MENU,self.OnClickBigger,biggerItem)
    self.Bind(wx.EVT_MENU,self.OnClickQuit,id=wx.ID_EXIT)
...
#Event handler
def OnClickBigger(self,e):
    pass
def OnClickQuit(self,e):
    self.Close()
...
```

StaticText and TextCtrl

- Textbox is used to receive user input or display information from programs
- Static text (label) :
  - Class : wx.StaticText
- Textbox :
  - Class : wx.TextCtrl
  - Useful setting : single line, multiple lines, rich text

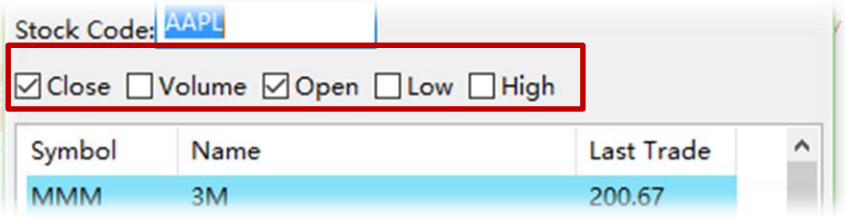


ListCtrl

- List is used to display multiple factors for user to choose
- List can be built by following 4 ways：
  - wx.LC_ICON (icon)
  - wx.LC_SMALL_ICON (small icon)
  - wx.LC_LIST (list)
  - wx.LC_REPORT （ report)

RadioBox and CheckBox

- Radiobox is used to select multiple objects from a selectable set
- Checkbox is used to choose from a mutually exclusive set.



Example

```
# Filename: helloworldbtn.py
import wx
class Frame1(wx.Frame):
    def __init__(self, superior):
        wx.Frame.__init__(self, parent = superior, title = "Hello World in wxPython")
        panel = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        self.text1= wx.TextCtrl(panel, value = "Hello, World!", size = (200,180), style = wx.TE_MULTILINE)
        sizer.Add(self.text1, 0, wx.ALIGN_TOP | wx.EXPAND)
        button = wx.Button(panel, label = "Click Me")
        sizer.Add(button)
        panel.SetSizerAndFit(sizer)
        panel.Layout()
        self.Bind(wx.EVT_BUTTON,self.OnClick,button)
    def OnClick(self, text):
        self.text1.AppendText("\nHello, World!")
```

Data Processing Using Python

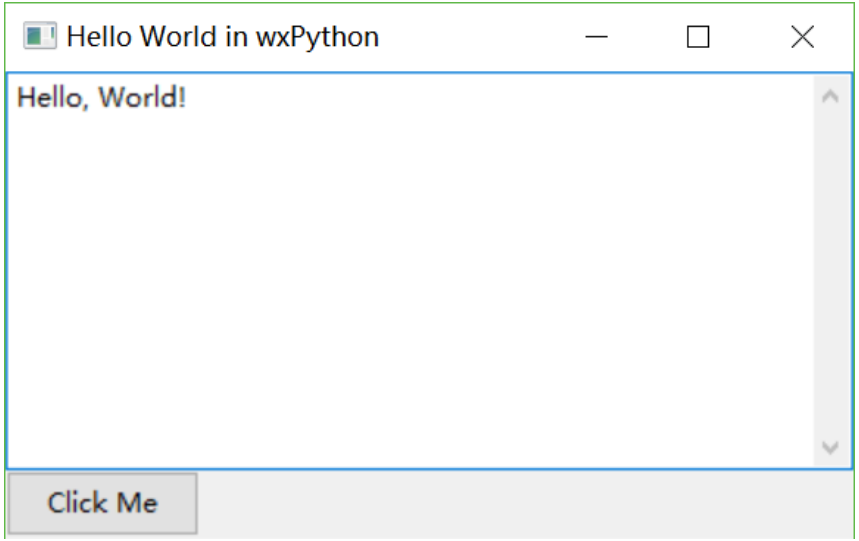
6 LAYOUT MANAGEMENT

Layout Management

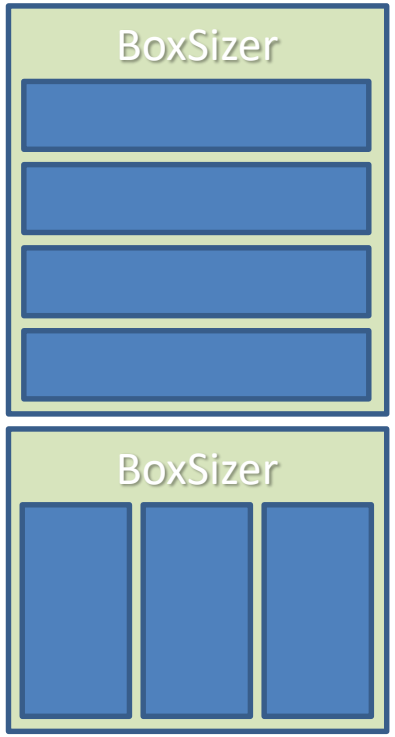
- Absolute position – Each window widget can explicitly appoint its position and size when created.
  - Weakness : Limited Flexibility
  - Hard to modify the size
  - Influenced by device, OS, even fonts
- Flexible layout solution - sizer
  - Every sizer has its own strategy.
  - Developer chooses sizer with proper strategy, inputs the widget and appoints the demands

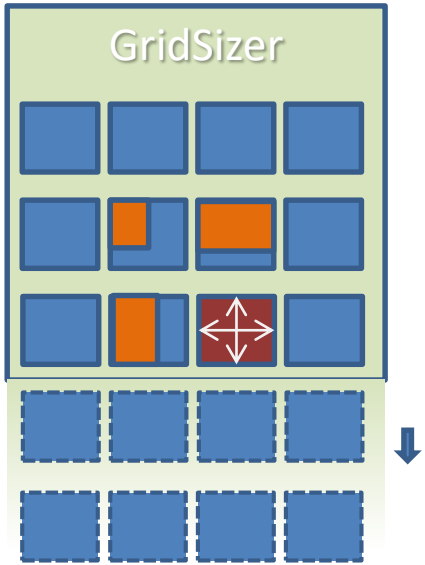
sizer

- Sizer is not a container or widget, but a layout algorithm
- Sizer allows nesting
- Useful sizer in wxPython
  - wx.BoxSizer
  - wx.FlexGridSizer
  - wx.GridSizer
  - wx.GridBagSizer
  - wx.StaticBoxSizer

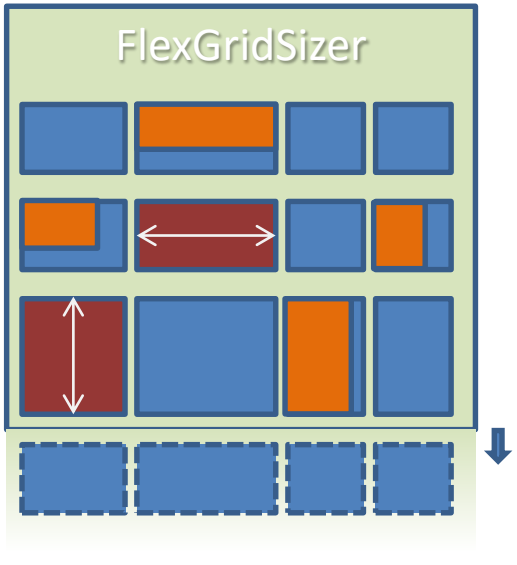


Example of sizer





All widgets are with the same size, one direction is fixed and the layout expands to the other side.



Height and width are decided by the largest widget.

Steps to use sizer

01

Create a container to automatically call sizer, like **panel**

02

Create sizer

03

Create sub-windows
(window widgets)

04

Use Add() method to add every sub-window to sizer

05

Call **SetSizer**(sizer) method of container

Example

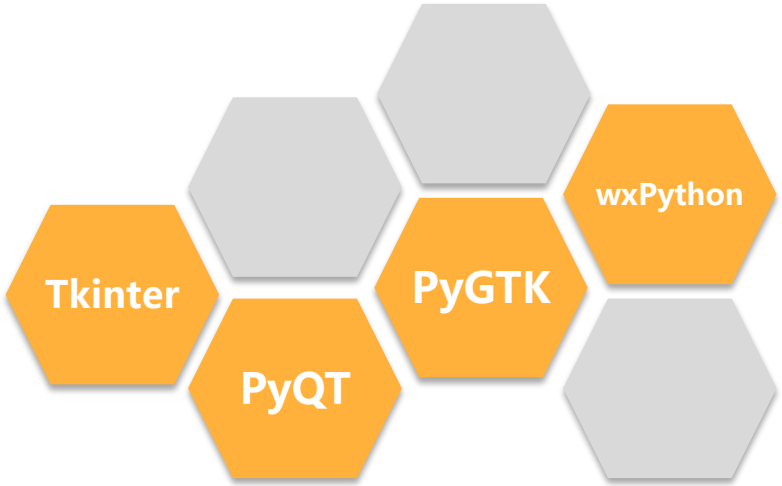
```
# Filename: helloworldbtn.py
import wx
class Frame1(wx.Frame):
    def __init__(self,superior):
        wx.Frame.__init__(self, parent = superior, title = "Hello World in wxPython")
        panel = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        self.text1= wx.TextCtrl(panel, value = "Hello, World!", size = (200,180), style = wx.TE_MULTILINE)
        sizer.Add(self.text1, 0, wx.ALIGN_TOP | wx.EXPAND)
        button = wx.Button(panel, label = "Click Me")
        sizer.Add(button)
        panel.SetSizerAndFit(sizer)
        panel.Layout()
        self.Bind(wx.EVT_BUTTON,self.OnClick,button)
    def OnClick(self, text):
        self.text1.AppendText("\nHello, World!")
```

Data Processing Using Python

7

OTHER GUI LIBRARIES

GUI Implement in Python



wxPython

Open source project with great cross platform performance.

http://wxpython.org

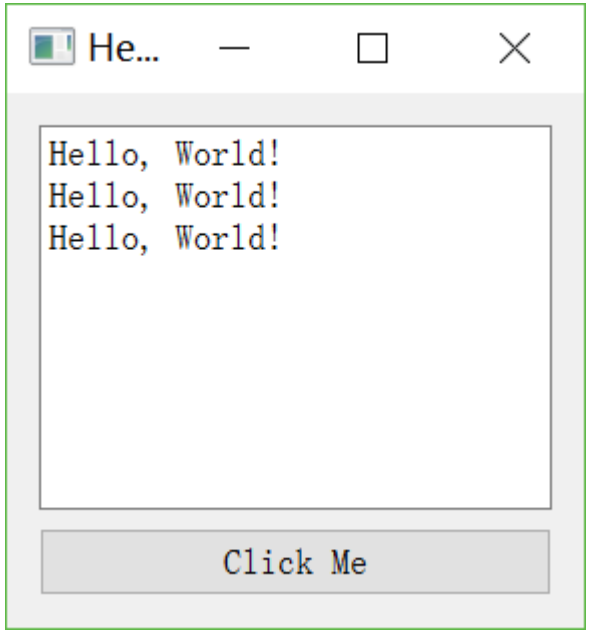
PyQt

- One GUI solution of Python language
- Provide two kinds of authorization, GPL and commercial agreements, and can be used without limit in free software.
- Cross Platform：Can run on Microsoft Windows、Mac OS X、Linux and other Unix-like platforms.

PyQt Example

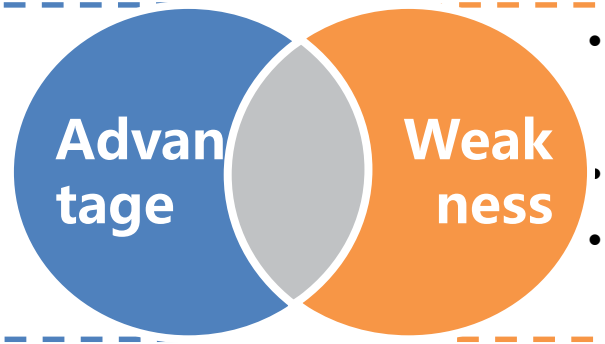
File

```
# Filename: PyQTdemo.py
import sys
from PyQt5 import QtWidgets
class TestWidget(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Hello World!")
        self.outputArea=QtWidgets.QTextBrowser()
        self.helloButton=QtWidgets.QPushButton("Click Me")
        self.layout=QtWidgets.QVBoxLayout()
        self.layout.addWidget(self.outputArea)
        self.layout.addWidget(self.helloButton)
        self.setLayout(self.layout)
        self.helloButton.clicked.connect(self.sayHello)
    def sayHello(self):
        self.outputArea.append("Hello, World!")
if __name__ == '__main__':
    app=QtWidgets.QApplication(sys.argv)
    testWidget=TestWidget()
    testWidget.show()
    sys.exit(app.exec_())
```



Advantage and Weakness of PyQT

- Rich document
- Experience similar to Qt、C++ development
- Most components for Qt are also available
- Convenient tools for PyQt , like QtDesigner , Eric4



- Be careful of memory leak
- Large runtime size
- C++ knowledge needed

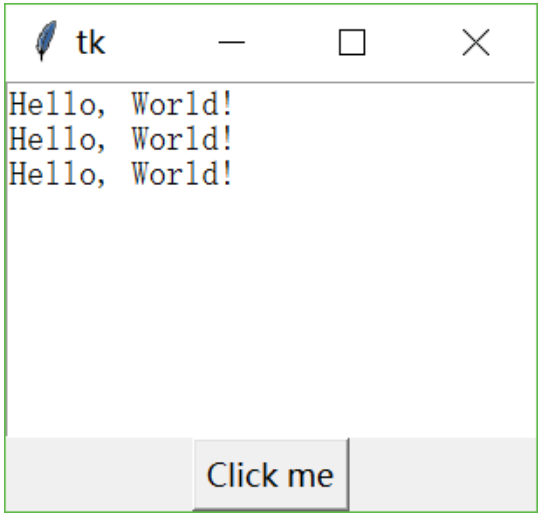
Tkinter

- Tkinter binds Tk GUI toolkit in Python, and is implemented by Tcl interpreter inside Python interpreter.
- Call of Tkinter is converted into Tcl instructions, and be interpreted by Tcl interpreter to build Python GUI.

Tkinter Example

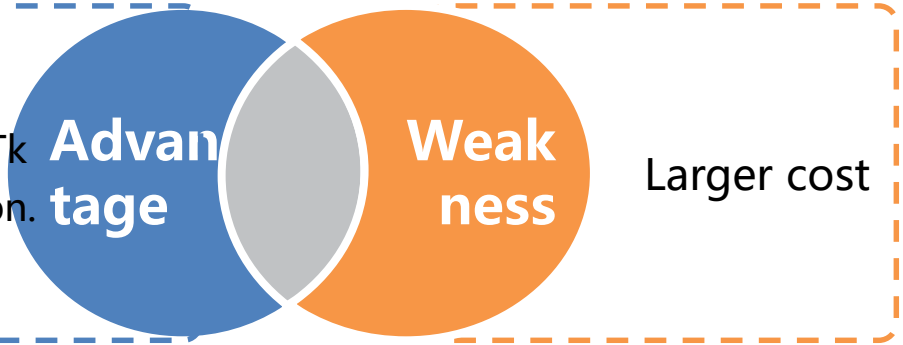
File

```
# Filename: Tkinterdemo.py
import tkinter as tk
class Tkdemo(object):
    def __init__(self):
        self.root=tk.Tk()
        self.txt=tk.Text(self.root,width=30,height=10)
        self.txt.pack()
        self.button=tk.Button(self.root,text='Click me',
                    command=self.sayhello)
        self.button.pack()
    def sayhello(self):
        self.txt.insert(tk.INSERT,"Hello, World!\n")
d=Tkdemo()
d.root.mainloop()
```



Advantage and Weakness of Tkinter

- With the longest history, the standard GUI for Python actually.
  - Python contains standard interface for Tk GUI toolkits in standard Windows version.
  - IDLE use Tkinter to implement GUI
- Simple to learn and use.



PyGTK

- PyGTK is a Python package of GTK+ GUI library
- pyGTK provides a set of comprehensive graphical elements and programming tools for desktop program
- PyGTK is a free software based on LGPL licence.
- Many famous GUI applications under GNOME are implemented by PyGTK, including BitTorrent, GIMP and Gedit

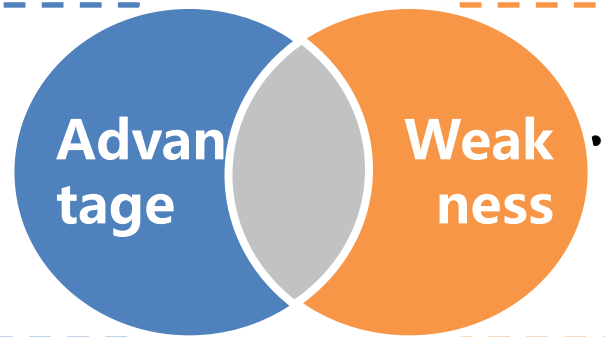
PyGTK Example

File

```
#PyGTKdemo.py
import pygtk
pygtk.require('2.0')
import gtk

class HelloWorld:
    def hello(self, widget, data=None):
        textbuffer = self.textview.get_buffer()
        startiter, enditer = textbuffer.get_bounds()
        content_text = textbuffer.get_text(startiter, enditer)
        content_text += "Hello, World!\n"
        textbuffer.set_text(content_text)

    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_title("A Simple Example of PyGtk")
        self.window.connect("delete_event", self.delete_event)
        self.window.connect("destroy", self.destroy)
        self.window.set_border_width(10)
        box1 = gtk.VBox(False, 0)
        self.window.add(box1)
        box1.show()
        sw = gtk.ScrolledWindow()
        sw.set_policy(gtk.POLICY_AUTOMATIC,
gtk.POLICY_AUTOMATIC)
        self.textview = gtk.TextView()
        textbuffer = self.textview.get_buffer()
        sw.add(self.textview)
        sw.show()
        self.textview.show()
        box1.pack_start(sw)

        self.button = gtk.Button("Click Me")
        self.button.connect("clicked", self.hello, None)
        self.button.show()
        box1.pack_start(self.button, expand=False, fill=False)
        self.window.show()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    hello = HelloWorld()
    hello.main()
```

Advantage and Weakness of PyGTK

- Bottom GTK+ provides several kinds of elements and functions
- Can be used to develop software for GNOME system.



- Bad performance on Windows platform

Data Processing Using Python

8 COMPREHENSIVE APPLICATION

Graphical User Interface (GUI)