



*Object-oriented & Graphical user interface*

---

# 面向对象 **和** 图形用户界面

---

Department of Computer Science and Technology

Department of University Basic Computer Teaching

用Python玩转数据

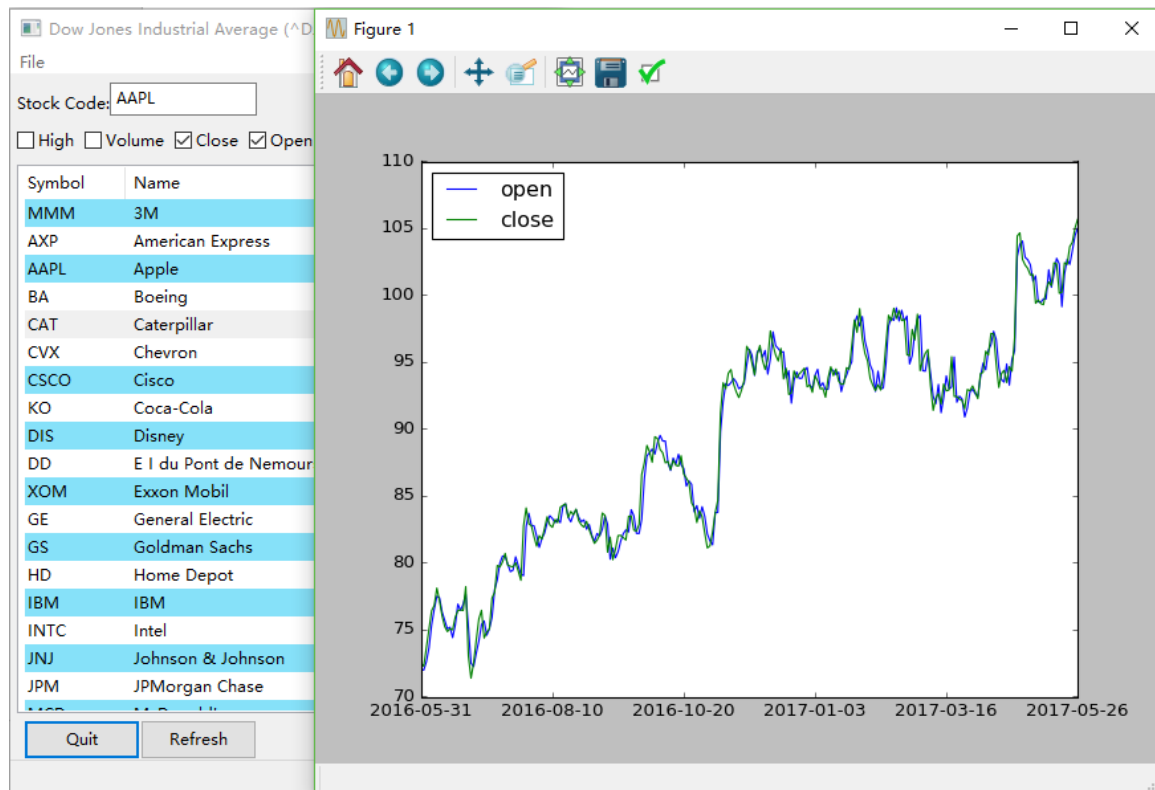
# 1 GUI与面向对象

# 字符用户界面

```
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
```

# 图形用户界面

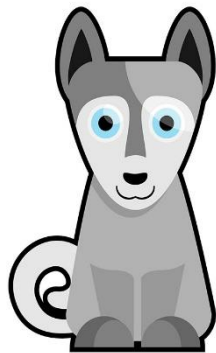


用Python玩转数据

2

# 抽象

- 对象（实例）
  - 由数据及能对其实施的操作所构成的封装体
- 类
  - 类描述了对对象的特征（数据和操作）



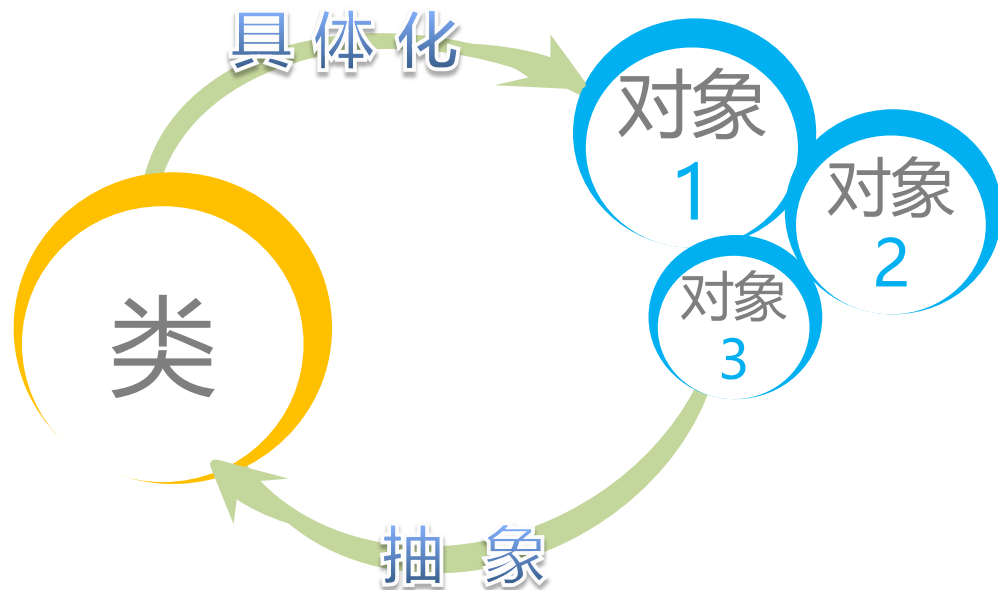


- 有名字
- 有矩形框
- 鼠标点击时有效果
- 功能不同：刷新、退出

## 面向对象 之 抽象

# 类与对象的关系

8





# 类的定义



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



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

**万类之源——object**

- 类的方法定义



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

# 实例 ( Instances )



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



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

- 实例的创建——通过调用类对象

- 1 定义类——Dog
- 2 创建一个实例——dog
- 3 通过实例使用属性或方法——dog.greet

# 实例属性 ( Instance Attributes )

F<sub>ile</sub>

```
# 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.

# 对象的初始化方法 `__init__()`

01

当类被调用后，Python将创建实例对象

02

创建完对象以后，Python自动调用的第一个方法为 `__init__()`

03

实例对象作为方法的第一个参数（`self`）被传递进去，调用类创建实例对象时的参数都传给 `__init__()`

# \_\_init\_\_() 举例

14



# 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.

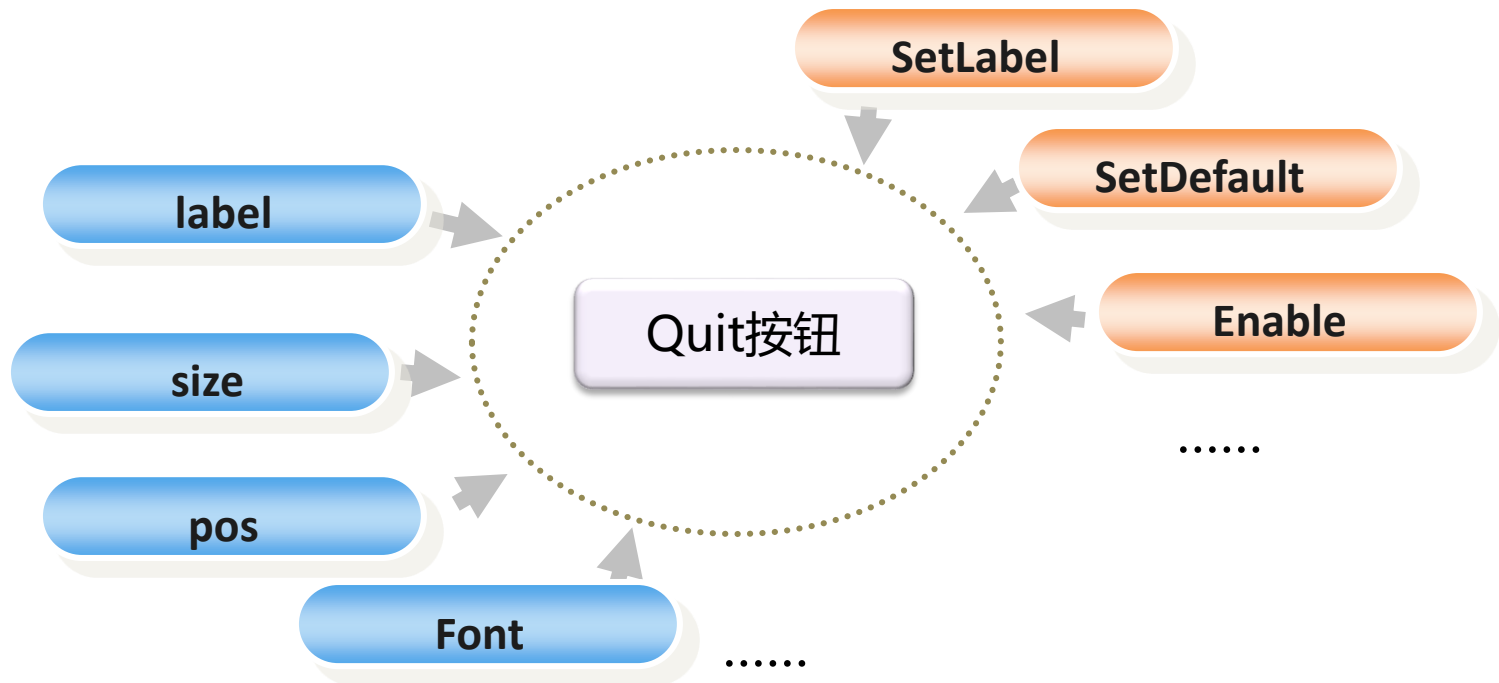
# 类属性 ( Class Attributes )

- 类的数据属性 ( 静态成员 )  
仅仅是所定义的类的变量
- 在类创建后被使用
- 可以由类中的方法来更新，也可以在主程序中更新
- 类属性和实例无关，修改类属性需要使用类名



```
# 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()
```

# 以按钮（ Button ）为例





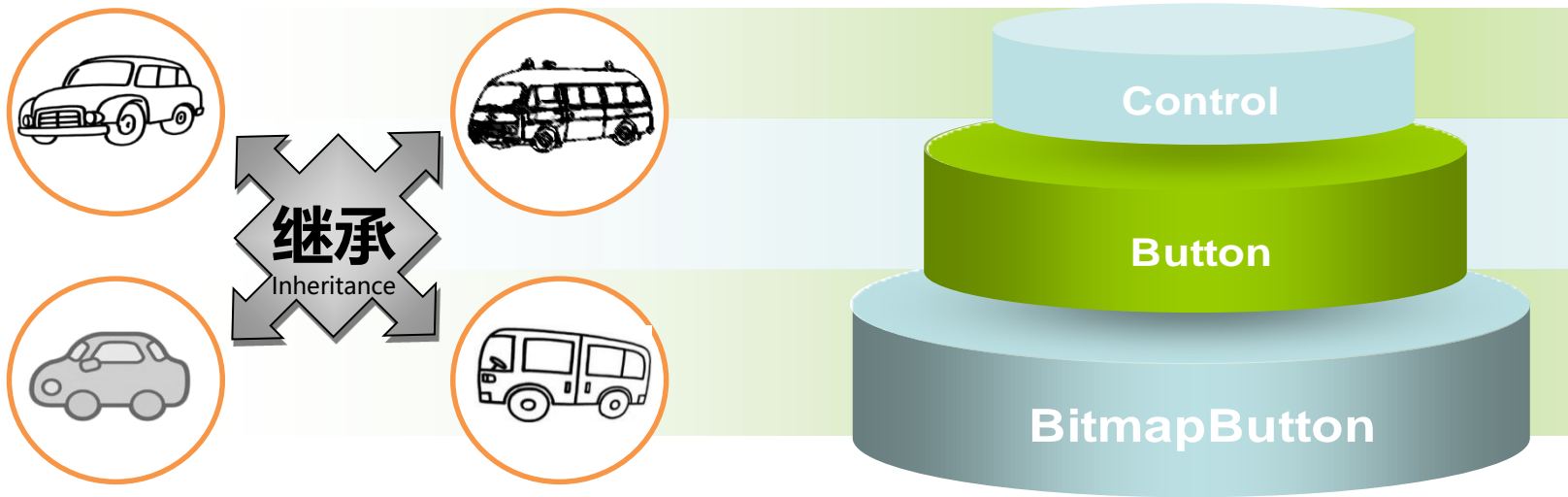
用Python玩转数据

# 3

## 继承

# 父类（基类）子类（派生类）

18



# 子类的定义

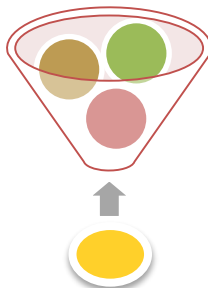


```
class SubClassName (ParentClass1[, ParentClass2, ...]):  
    'optional class documentation string'  
    class_suite
```

单  
继  
承



多  
继  
承



# 子类定义举例和重载

20

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()
```

## 私有属性和方法

- 默认情况下，Python 类的成员属性与方法都是 “public”
- 提供 “访问控制符” 来限定成员函数的访问
  - 双下划线(\_\_\_\_)
  - `__var`属性会被 `__classname_var` 替换，将防止父类与子类中的同名冲突
  - 单下划线(\_)
  - 在属性名前使用一个单下划线字符，防止模块的属性用 `“from mymodule import *”` 来加载

用Python玩转数据

# 4

## GUI的基本框架

# 创建一个简单的wxPython程序

23



# Filename: firstwxPython.py

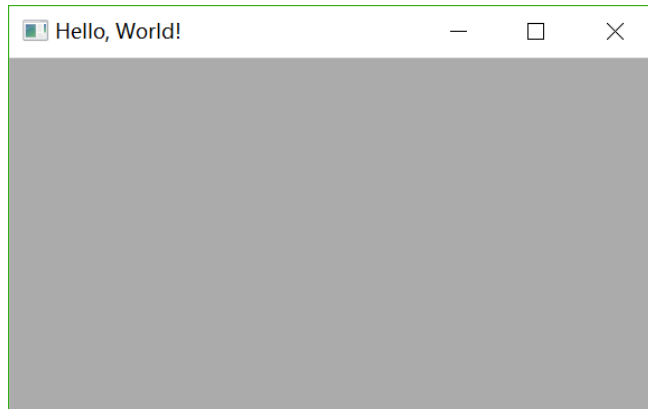
```
import wx
```

```
app = wx.App()
```

```
frame = wx.Frame(None, title = "Hello, World!")
```

```
frame.Show(True)
```

```
app.MainLoop()
```



上例也可改为：

```
# 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()
```

应用程序对象也可以是  
wx.App的子类的一个实例



- 组件容器 ( Containers ) ——用于容纳其他组件
  - 例 : wx.Panel等
- 动态组件 ( Dynamic Widgets ) ——可以被用户编辑
  - 例 : wx.Button、wx.TextCtrl、wx.ListBox等
- 静态组件 ( Static Widgets ) ——显示信息用 , 不能被用户编辑
  - 例 : wx.StaticBitmap、wx.StaticText、wxStaticLine等
- 其它组件
  - 例 : wx.ToolBar、wx.MenuBar、wx.StatusBar

# 又见 “Hello , World ! ”

26

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))
```

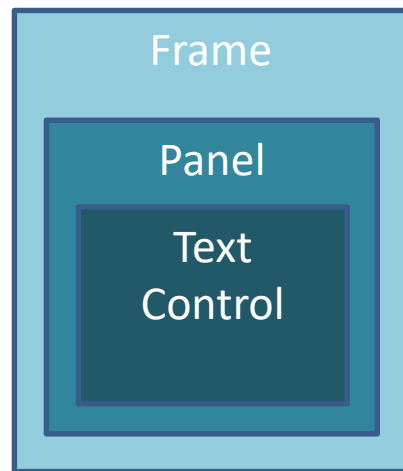
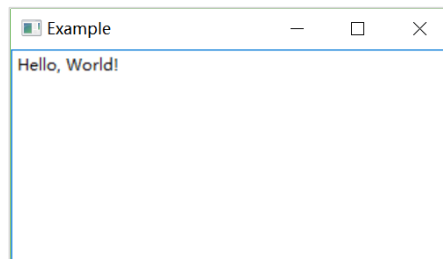
```
if __name__ == '__main__':
```

```
    app =wx.App()
```

```
    frame = Frame1(None)
```

```
    frame.Show(True)
```

```
    app.MainLoop()
```



- GUI程序工作的基本机制之一——事件处理
- 事件
  - 移动鼠标，按下鼠标左键、单击按钮等
  - 可以由用户操作触发产生，也可以在程序中创建对象产生
- wxPython程序将特定类型的事件关联到特定的一块代码（方法），当该类型的事件产生时，相关代码将响应事件被自动执行
  - 例：当产生鼠标移动事件时，OnMove()方法将被自动调用

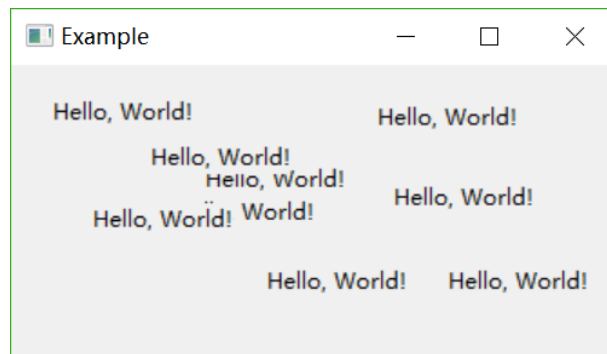
# 还是Hello, World!

28

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
```



用Python玩转数据

# 5

## GUI常用组件

# 应用程序示例

静态文本

菜单

输入框

列表框

按钮

Dow Jones Industrial Average (^DJII)

File

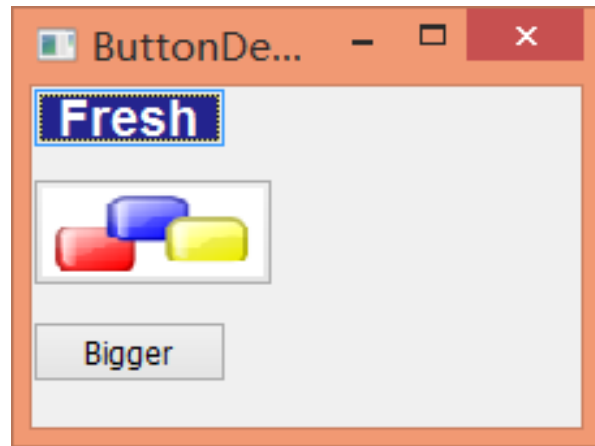
Stock Code:

☒ Open ☒ Close ☐ High ☐ Low ☐ Volume

Symbol	Name	Last Trade
MMM	3M	206.47
AXP	American Express	78.25
AAPL	Apple	153.71
BA	Boeing	190.67
CAT	Caterpillar	106.29
CVX	Chevron	102.95
CSCO	Cisco	31.88
KO	Coca-Cola	45.72
DIS	Disney	107.89
DD	E I du Pont de Nemours and Co	80.36
XOM	Exxon Mobil	79.66
GE	General Electric	27.71
GS	Goldman Sachs	213.96
HD	Home Depot	155.21
IBM	IBM	152.07
INTC	Intel	36.11
JNJ	Johnson & Johnson	129.25
JPM	JPMorgan Chase	82.26

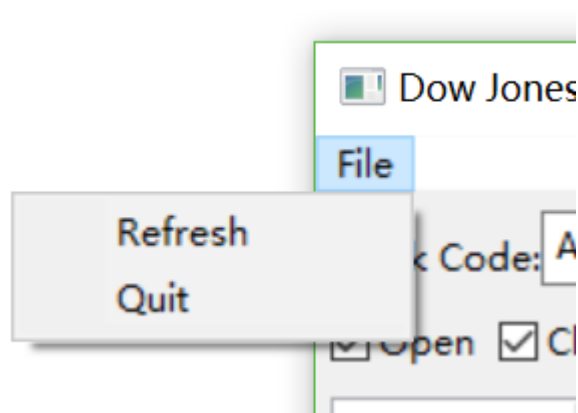
## 按钮（Button及其家族）

- 功能：接受用户的点击事件，触发相应的操作。
- 常用按钮：
  - wx.Button：文本按钮
  - wx.BitmapButton：位图按钮
  - wx.ToggleButton：开关按钮
- 绑定处理按钮点击的事件



# 菜单 ( Menu及其组件 )

- 菜单
  - 菜单栏
  - 菜单
  - 菜单项命令
- wxPython用于创建菜单的类：
  - wx.MenuBar
  - wx.Menu
  - wx.MenuItem





# 菜单常用事件

- 菜单事件
  - wx.EVT\_MENU



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

```
...
```

```
#绑定事件处理器
```

```
self.Bind(wx.EVT_MENU,self.OnClickBigger,biggerItem)  
self.Bind(wx.EVT_MENU,self.OnClickQuit,id=wx.ID_EXIT)
```

```
...
```

```
#事件处理器
```

```
def OnClickBigger(self,e):
```

```
    pass
```

```
def OnClickQuit(self,e):
```

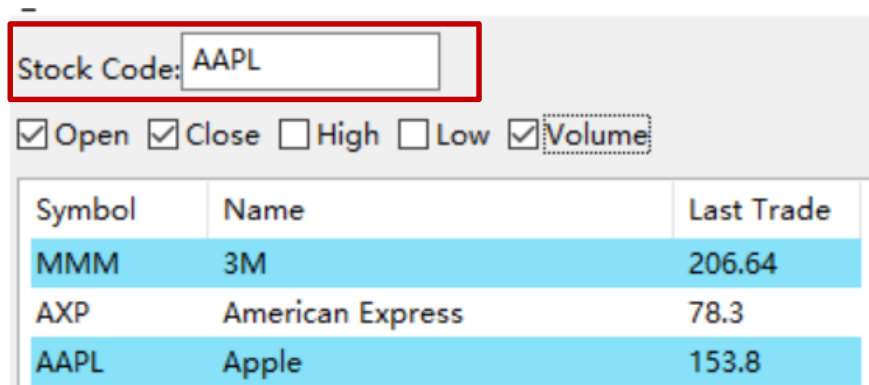
```
    self.Close()
```

```
...
```

# 静态文本（ StaticText ）和文本框（ TextCtrl

34

- 文本框用于接收用户在框内输入的信息，或显示由程序提供的信息
- 静态文本框（ 标签 ）：
  - 类：wx.StaticText
- 文本框：
  - 类：wx.TextCtrl
  - 常用形式：单行,多行,富文本框



The screenshot shows a user interface for a stock application. At the top, there is a text input field labeled 'Stock Code:' containing the text 'AAPL'. Below this, there are several checkboxes: 'Open', 'Close', 'High', 'Low', and 'Volume'. The 'Open', 'Close', and 'Volume' checkboxes are checked, while 'High' and 'Low' are unchecked. Below the checkboxes is a table with three columns: 'Symbol', 'Name', and 'Last Trade'. The table contains three rows of data: 'MMM' (3M) with a last trade of 206.64, 'AXP' (American Express) with a last trade of 78.3, and 'AAPL' (Apple) with a last trade of 153.8. The 'AAPL' row is highlighted in blue.

Symbol	Name	Last Trade
MMM	3M	206.64
AXP	American Express	78.3
AAPL	Apple	153.8

# 列表 ( ListCtrl )

- 列表用于显示多个条目并且可供用户选择
- 列表能够以下面四种不同模式建造：
  - wx.LC\_ICON ( 图标 )
  - wx.LC\_SMALL\_ICON ( 小图标 )
  - wx.LC\_LIST ( 列表 )
  - wx.LC\_REPORT ( 报告 )

Col #1	Col #2	Col #3
Row #1	aaaa	1
Row #2	bbbb	2
Row #3	cccc	3
Row #4	dddd	4
Row #5	eeee	5

Report 模式

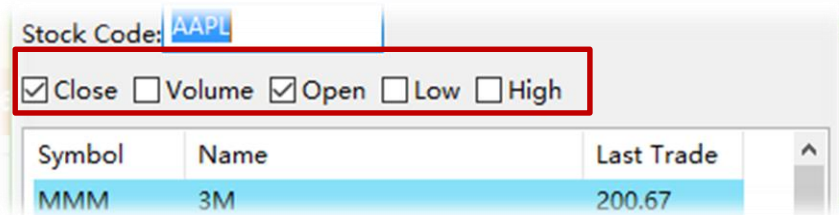
Row #1
Row #2
Row #3
Row #4
Row #5

List模式

# 单选 ( RadioBox ) 与复选框 ( CheckBox )

36

- 复选框用于从一组可选项中，同时选中多个选项
- 对应的，单选框用于从一组互斥的选项中，选取其一



Stock Code: AAPL

☒ Close ☐ Volume ☒ Open ☐ Low ☐ High

Symbol	Name	Last Trade
MMM	3M	200.67

# 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!")

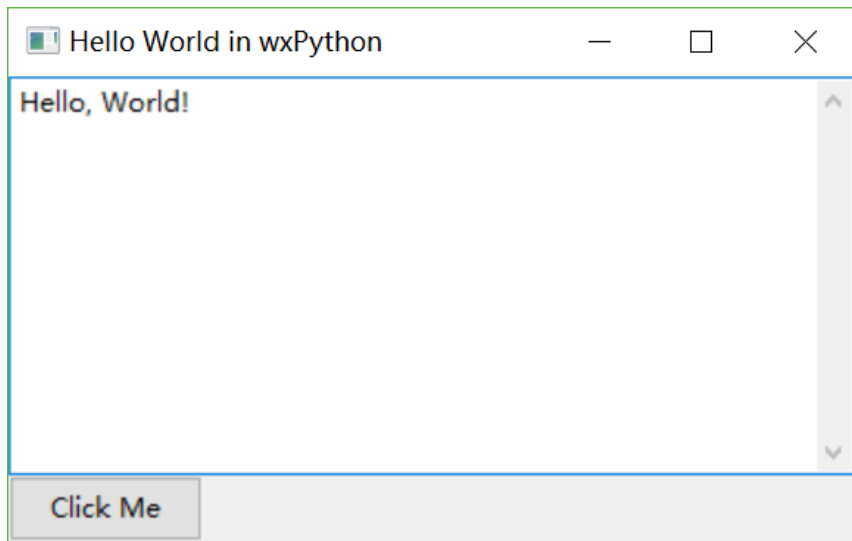
用Python玩转数据

# 布局管理

6

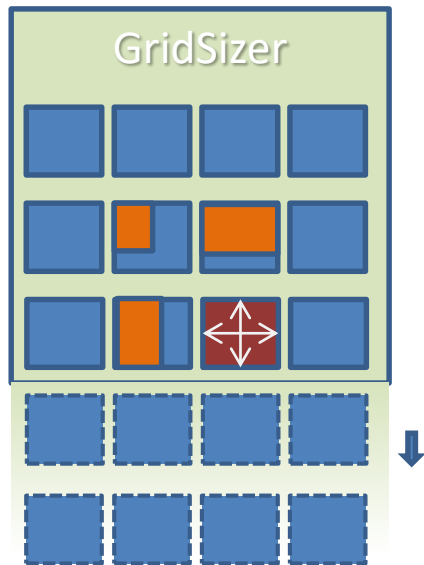
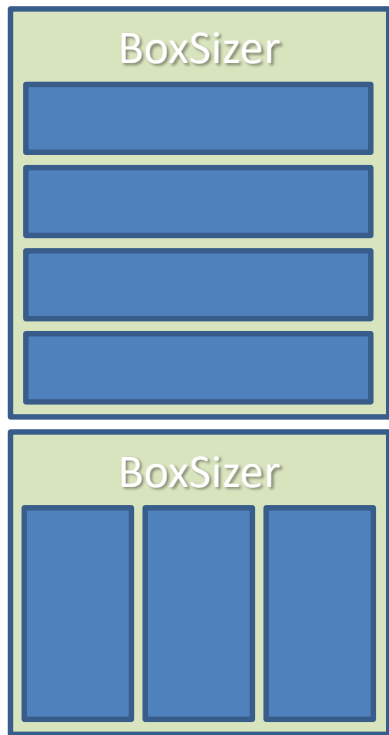
- 绝对定位 - 每个窗口组件被创建时可以显式地指定它的位置和大小
  - 缺点：定位不灵活
  - 调整大小困难
  - 受设备、操作系统甚至字体影响
- 灵活布局的解决方案 - sizer
  - 每个sizer有自己的定位策略
  - 开发者只需要选择合适策略的sizer将窗口组件放入，并且指定好需求即可

- sizer本身不是一个容器或一个窗口组件。它只是一个屏幕布局的算法
- sizer允许嵌套
- wxPython常用的sizer
  - wx.BoxSizer
  - wx.GridSizer
  - wx.GridBagSizer
  - wx.StaticBoxSizer

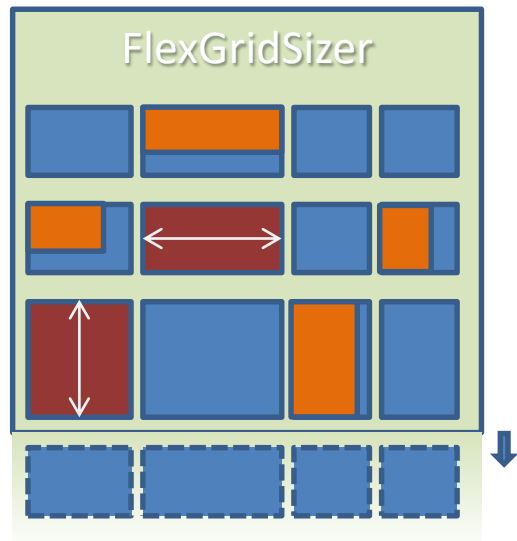




# 各种sizer示意

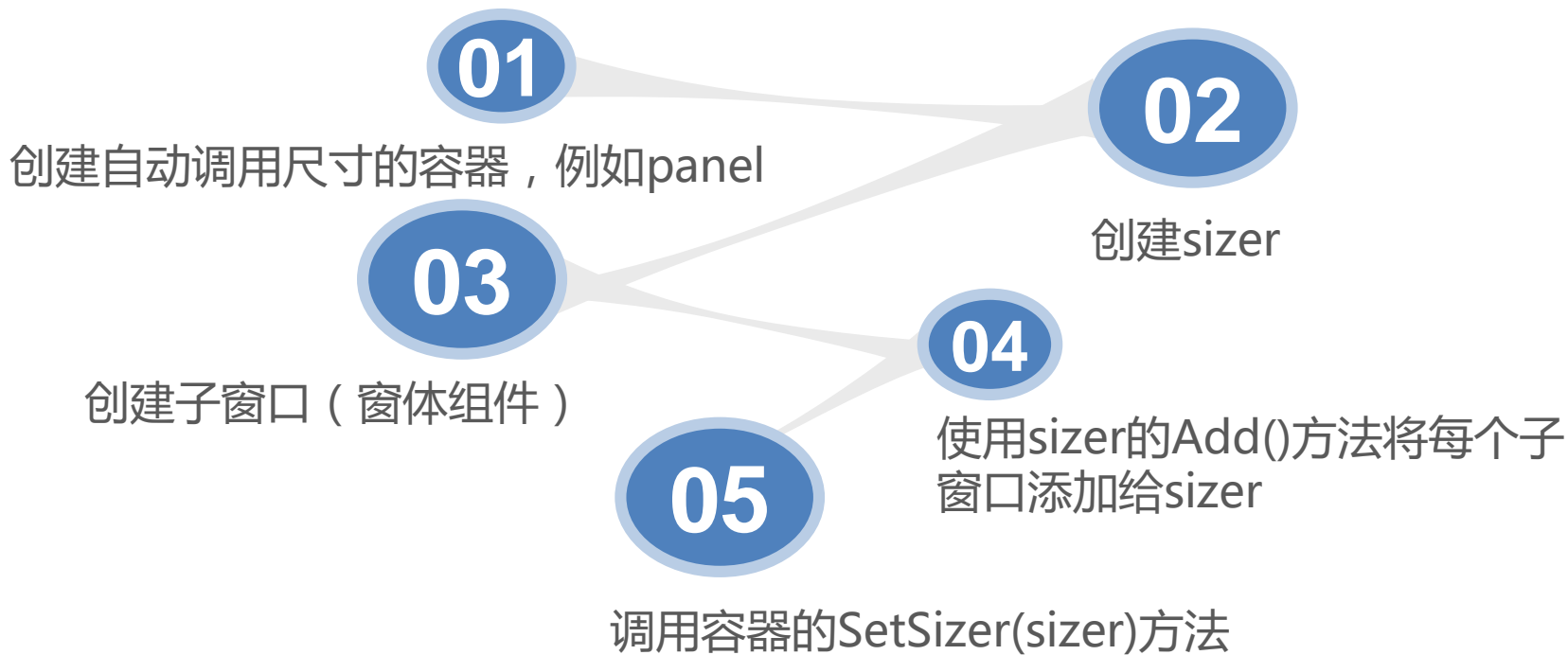


所有组件大小一致，  
固定一个方向，在  
另外一个方向生长



行高和列宽由最大的  
组件决定

# 使用sizer的步骤



# 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)

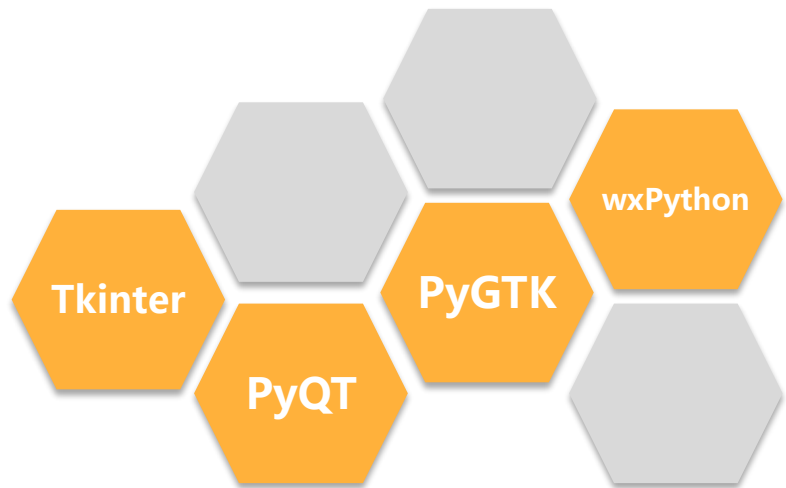
def OnClick(self, text):

self.text1.AppendText("\nHello, World!")

用Python玩转数据

7

# 其他GUI库



*wxPython*

开源软件，具有优秀的  
跨平台能力。官网：

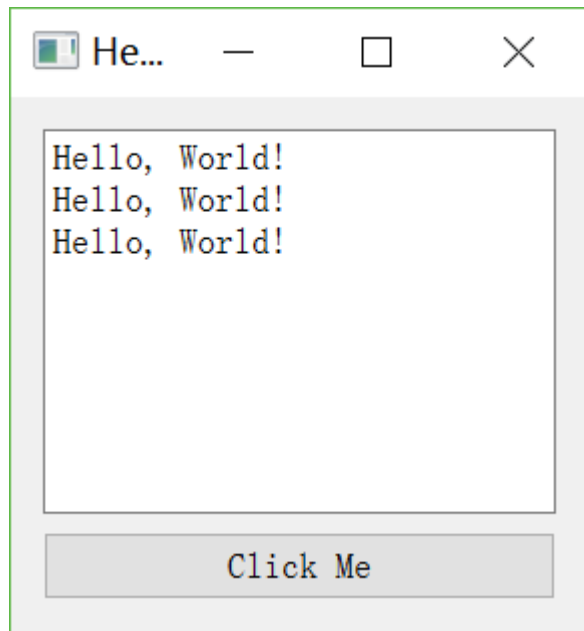
<http://wxpython.org>

- 是Python语言的GUI编程解决方案之一
- 提供了GPL与商业协议两种授权方式，可以免费地用于自由软件的开发
- 跨平台：可以运行于Microsoft Windows、Mac OS X、Linux 以及其它类Unix平台上

# PyQt简单示例

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_())
```



# PyQT的优缺点

## 优点

- 文档比其他GUI库丰富
- 与Qt、C++开发经验互通
- 可使用大多数为Qt开发的组件
- 有方便的周边工具支持PyQt，如QtDesigner，Eric4

## 缺点

- 要注意避免内存泄露
- 运行时庞大
- 需要学习一些C++知识



- Tkinter绑定了 Python 的 Tk GUI 工具集，通过内嵌在 Python 解释器内部的 Tcl 解释器实现
- Tkinter 的调用转换成 Tcl 命令，然后交给 Tcl 解释器进行解释，实现 Python 的 GUI 界面

# Tkinter简单示例

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)
```

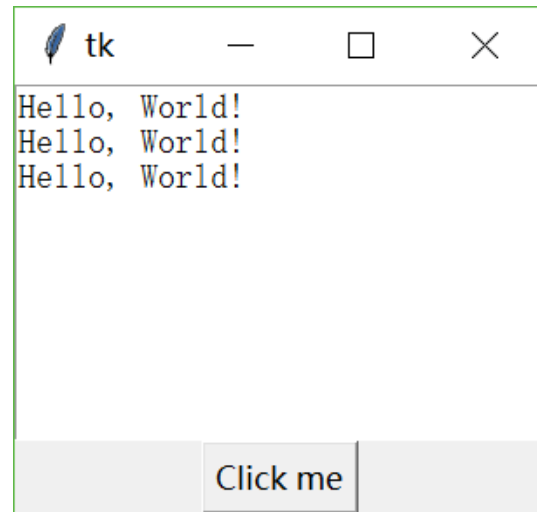
```
        self.button.pack()
```

```
    def sayhello(self):
```

```
        self.txt.insert(tk.INSERT,"Hello, World!\n")
```

```
d=Tkdemo()
```

```
d.root.mainloop()
```



# Tkinter的优缺点

51

- 历史最悠久，是Python 事实上的标准 GUI
  - Python 中使用 Tk GUI 工具集的标准接口，已包括在标准的Python Windows 安装中
  - 著名的 IDLE 用 Tkinter 实现 GUI
- 创建的 GUI 简单，学起来和用起来也简单

优点

缺点

- 性能不太好，执行速度慢

- PyGTK是一套 GTK+ GUI 库的Python封装
- pyGTK为创建桌面程序提供了一套综合的图形元素和其它使用的编程工具
- PyGTK是基于LGPL协议的免费软件
- 许多 Gnome 下的著名应用程序的 GUI 都是使用 PyGTK 实现的，比如 BitTorrent ， GIMP 和 Gedit 都有可选的实现

# PyGTK的简单示例

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()
```



# PyGTK的优缺点

54

- 底层的GTK+提供了各式的可视元素和功能
- 能开发在GNOME桌面系统运行的功能完整的软件

优点

缺点

- 在Windows平台表现不太好

用Python玩转数据

# 综合应用



# 图形用户界面

