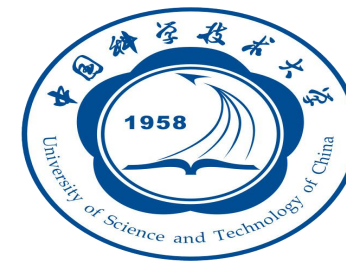




# Python 程序开发技术

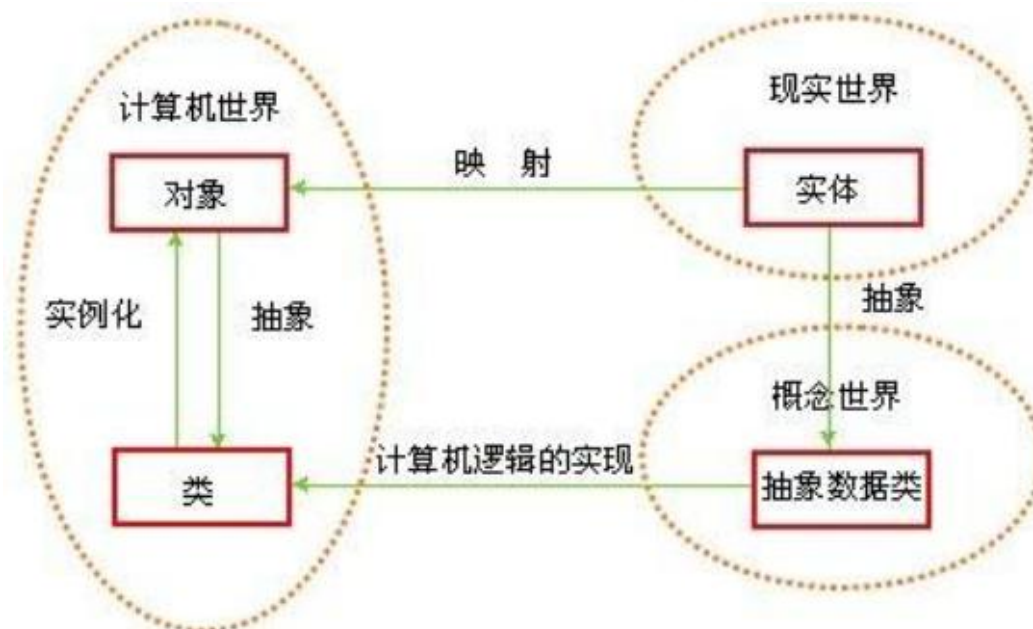
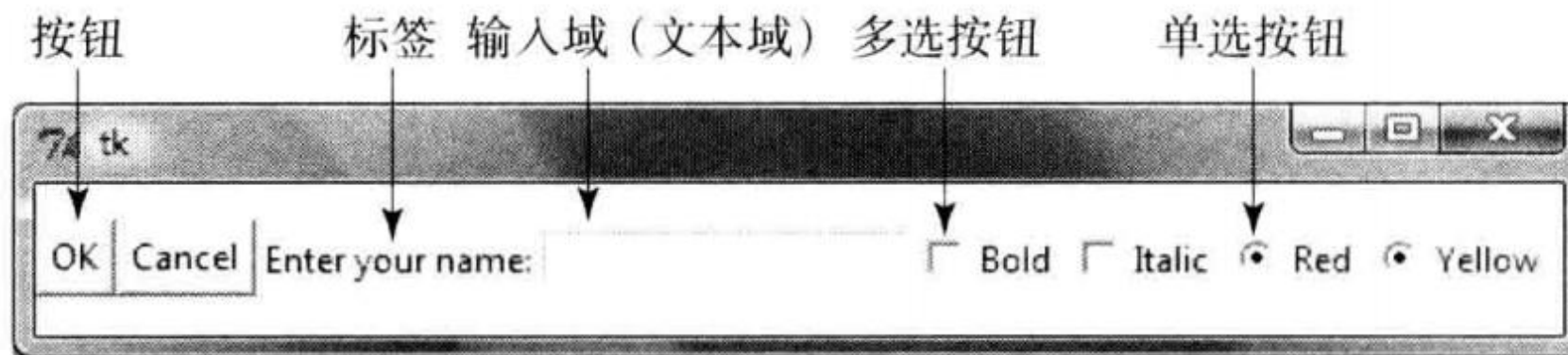
## ---第二章:面向对象程序设计



# 本章节目录

- 对象和类
- Str类：更多字符串和特殊方法
- 实践：用Tkinter进行GUI设计
- List类与列表
- 多维列表
- 继承与多态
- 文件对象及异常处理

## 面向对象：如何使用对象创建程序





# 对象和类

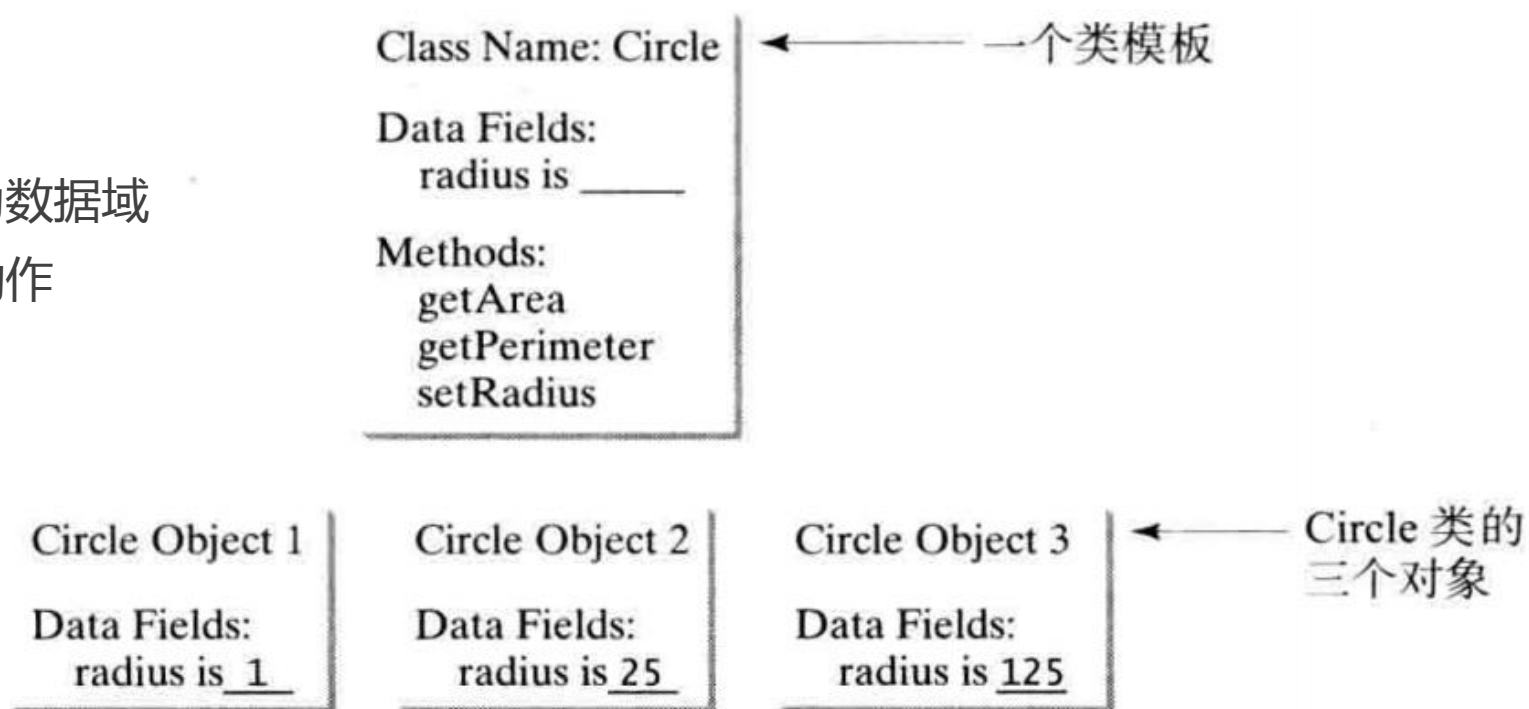
## ■ 为对象定义类

### ■ 对象：

特性（每个对象的id）

状态：对象的属性，称为数据域

行为：方法，完成某个动作





# 对象和类

## ■ 为对象定义类

### ■ 定义类：

```
class ClassName:  
    initializer  
    methods
```

### □ 约定：

类名首字母大写；

变量和方法的首字母小写

## 例程：

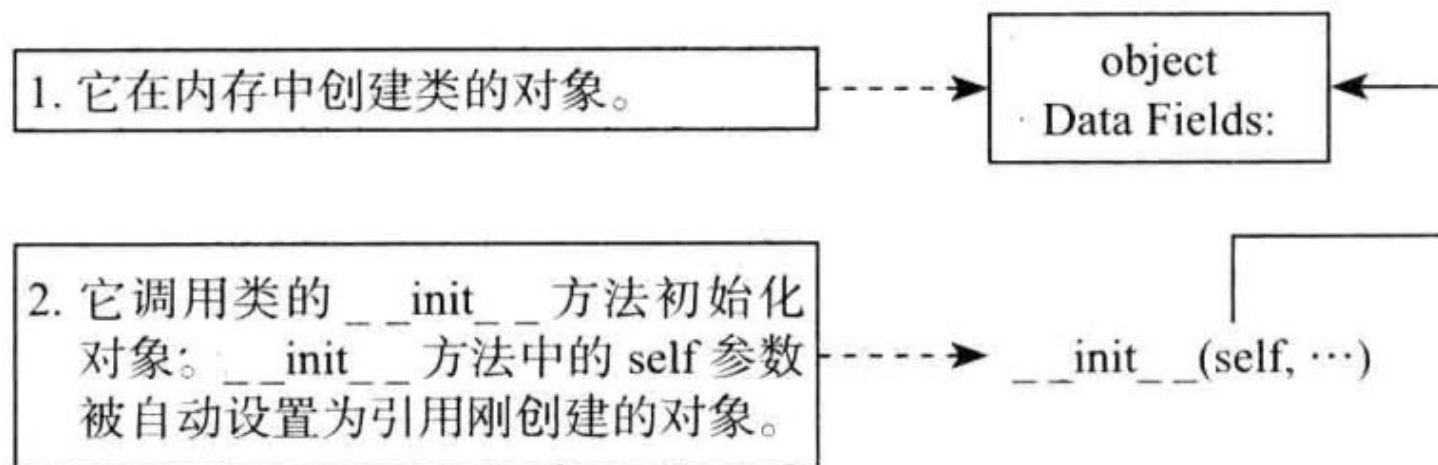
```
import math  
  
class Circle:  
    # Construct a circle object  
    def __init__(self, radius = 1):  
        self.radius = radius  
  
    def getPerimeter(self):  
        return 2 * self.radius * math.pi  
  
    def getArea(self):  
        return self.radius * self.radius * math.pi  
  
    def setRadius(self, radius):  
        self.radius = radius
```

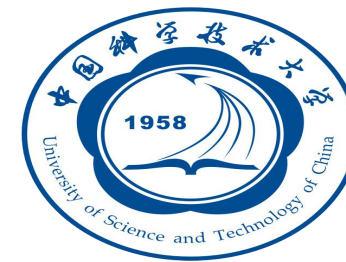


# 对象和类

## ■ 为对象定义类

- 构造对象：用构造方法来创建类 **类名 (参数)**
  - `__init__` 方法
  - `self` 参数





# 对象和类

## ■ 构造方法：

Circle()

Circle(5)

1. 创建一个 Circle 对象



Circle 对象

2. 调用 `__init__(self, radius)`



Circle 对象  
radius: 5



# 对象和类

- 访问对象成员：

数据域  $\leftrightarrow$  实例变量

方法  $\leftrightarrow$  实例方法

例程：

```
>>> from Circle import Circle
>>> c = Circle(5)
>>> c.radius
5
>>> c.getPerimeter()
31.41592653589793
>>> c.getArea()
78.53981633974483
```





# 对象和类

- self参数：指向对象本身

```
def ClassName:

    def __init__(self, ...):
        self.x = 1 # Create/modify x
        ...

    def m1(self, ...):
        self.y = 2 # Create/modify y
        ...
        z = 5 # Create/modify z
        ...

    def m2(self, ...):
        self.y = 3 # Create/modify y
        ...
        u = self.x + 1 # Create/modify u
        self.m1(...) # Invoke m1
```

访问实例变量x →

self.x 和 self.y 的作用域

z 的作用域

调用对象的实例方法 →

# 对象和类

- 例：类的使用

```
from Circle import Circle

def main():
    # Create a circle with radius 1
    circle1 = Circle()
    print("The area of the circle of radius",
          circle1.radius, "is", circle1.getArea())

    # Create a circle with radius 25
    circle2 = Circle(25)
    print("The area of the circle of radius",
          circle2.radius, "is", circle2.getArea())

    # Create a circle with radius 125
    circle3 = Circle(125)
    print("The area of the circle of radius",
          circle3.radius, "is", circle3.getArea())

    # Modify circle radius
    circle2.radius = 100 # or circle2.setRadius(100)
    print("The area of the circle of radius",
          circle2.radius, "is", circle2.getArea())

main() # Call the main function
```

类的客户端





# 对象和类

## ■ UML类图

- ① 为用户端描述如何创建对象以及如何调用其中的方法
- ② 不包含特殊的self参数
- ③ `__init__`不需要罗列在UML中

数据域被表示为: `dataFieldName: dataType`

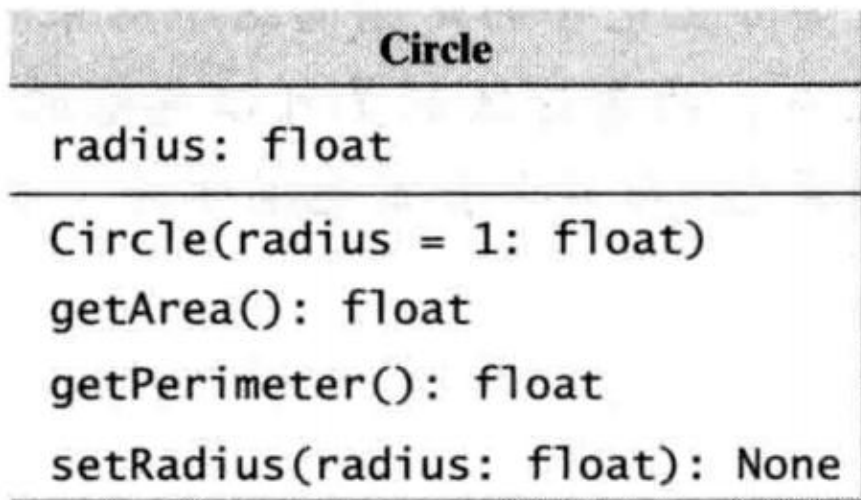
构造方法 `ClassName(parameterName: parameterType)`

方法被表示为: `methodName(parameterName: parameterType): returnType`

# 对象和类



UML 类图

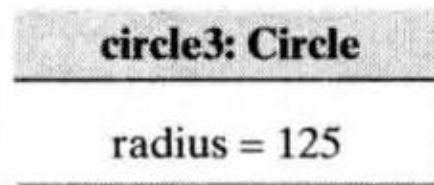
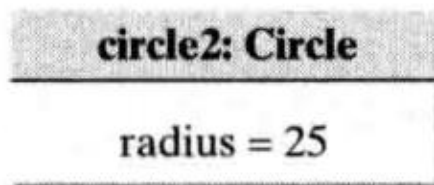
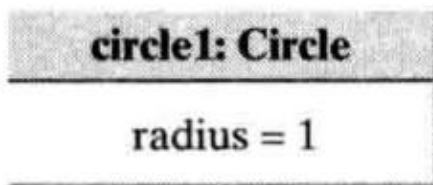


← 类图

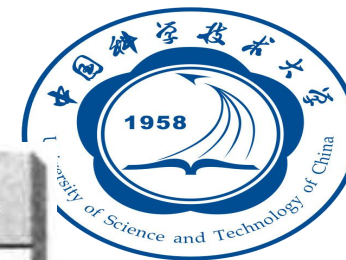
← 数据域

← 构造方法

← 方法

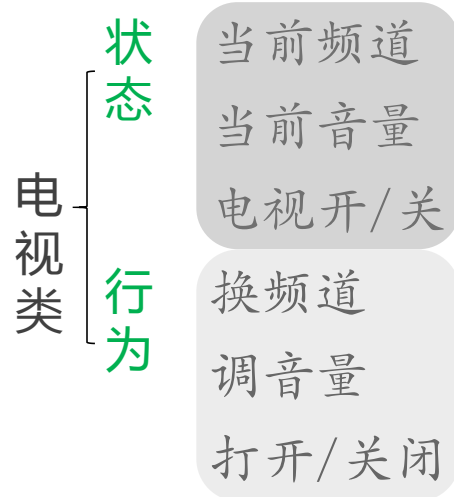


← 对象的  
UML 符号



# 对象和类

## ■ 以TV作为例子



TV	
channel: int	
volumeLevel: int	
on: bool	
TV()	
turnOn(): None	
turnOff(): None	
getChannel(): int	
setChannel(channel: int): None	
getVolume(): int	
setVolume(volumeLevel: int): None	
channelUp(): None	
channelDown(): None	
volumeUp(): None	
volumeDown(): None	

# 对象和类



TV类的部分Python代码：

```
class TV:
    def __init__(self):
        self.channel = 1    # Default channel is 1
        self.volumeLevel = 1    # Default volume level is 1
        self.on = False    # Initially, TV is off

    def setChannel(self, channel):
        if self.on and 1 <= self.channel <= 120:
            self.channel = channel

    def setVolume(self, volumeLevel):
        if self.on and \
            1 <= self.volumeLevel <= 7:
            self.volumeLevel = volumeLevel
```



# 对象和类



用TV类创建两个对象：

```
from TV import TV

def main():
    tv1 = TV()
    tv1.turnOn()
    tv1.setChannel(30)
    tv1.setVolume(3)
    tv2 = TV()
    tv2.turnOn()
    tv2.channelUp()
    tv2.channelUp()
    tv2.volumeUp()

    print("tv1's channel is", tv1.getChannel() ,
          "and volume level is", tv1.getVolumeLevel())
    print("tv2's channel is", tv2.getChannel(),
          "and volume level is", tv2.getVolumeLevel())

main() # Call the main function
```

# 对象和类

## ■ 不可变对象和可变对象

- 可变对象：函数可能会改变对象的内容
- 不可变对象：数字或字符串
- 传递可变对象和不可变对象
  - 可变对象：圆
  - 不可变对象：数字、字符串

创建一个新的int对象，并赋  
给c.radius

```
from Circle import Circle
```

```
def main():
```

```
    # Create a Circle object with radius 1  
    myCircle = Circle()
```

```
    # Print areas for radius 1, 2, 3, 4, and 5  
    n = 5  
    printAreas(myCircle, n)
```

```
    # Display myCircle.radius and times  
    print("\nRadius is", myCircle.radius)  
    print("n is", n)
```

myCircle和c指向同一对象

```
    # Print a table of areas for radius
```

```
    def printAreas(c, times):
```

```
        print("Radius \t\tArea")
```

```
        while times >= 1:
```

```
            print(c.radius, "\t\t", c.getArea())
```

```
            c.radius = c.radius + 1
```

```
            times = times - 1
```

```
main() # Call the main function
```





7.11 给出下面程序的输出结果:

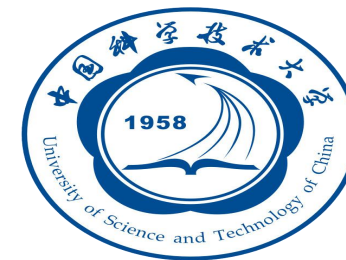
```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    times = 0
    for i in range(100):
        increment(c, times)

    print("count is", c.count)
    print("times is", times)

def increment(c, times):
    c.count += 1
    times += 1

main() # Call the main function
```



# 对象和类

## ■ 隐藏数据域：私有数据域

```
>>> c = Circle(5)
>>> c.radius = 5.4 # Access instance variable directly
>>> print(c.radius) # Access instance variable directly
5.4
```

- ① 数据被篡改。
- ② 类难以维护。



# 对象和类

- Python中的私有数据域和私有方法

- 以\_\_开头
- 只在类内被访问
- 客户端访问需要 get set 方法

get方法 { `def getProperty(self):`  
          如果返回类型是布尔型, `def isPropertyName(self):`

set方法 `def setPropertyName(self, propertyValue):`

# 对象和类



■ 例程：`import math`

```
class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def getPerimeter(self):
        return 2 * self.__radius * math.pi

    def getArea(self):
        return self.__radius * self.__radius * math.pi
```



# 对象和类

```
>>> from CircleWithPrivateRadius import Circle
>>> c = Circle(5)
>>> c.__radius
AttributeError: no attribute '__radius'
>>> c.getRadius()
5
```

- 如果这个类只在程序内部使用，就没必要隐藏数据域
- 私有数据域和方法不要同时也以1个以上下划线结尾。



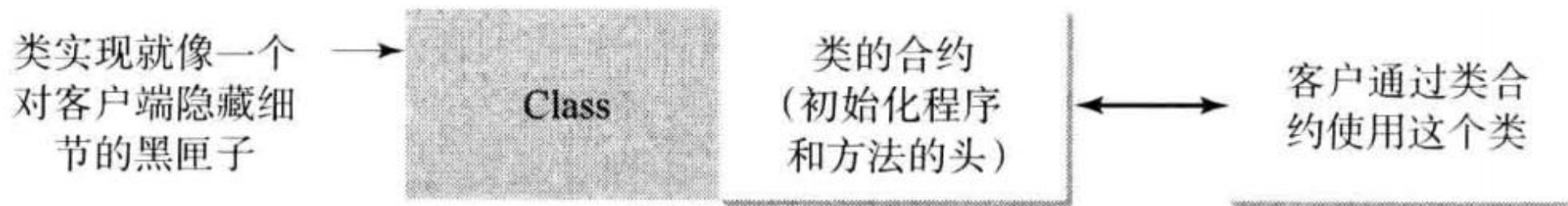
# 对象和类

## ■ 类的抽象与封装

将类的实现和类的使用分离

类的实现细节对用户不可见

### ■ 类：抽象数据类型



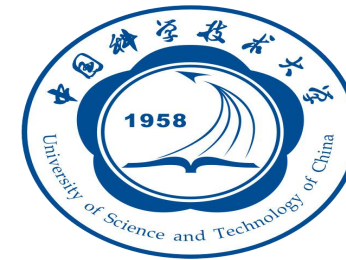
# 对象和类

- 例程：一个贷款程序

Loan类的UML图

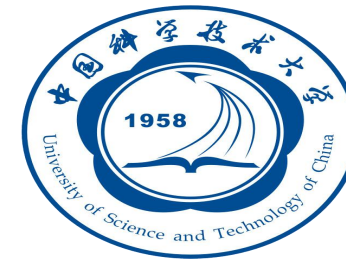
```
-annualInterestRate: float  
-numberOfYears: int  
-loanAmount: float  
-borrower: str
```

```
Loan(annualInterestRate: float,  
      numberOfYears: int, loanAmount  
      float, borrower: str)  
getAnnualInterestRate(): float  
getNumberOfYears(): int  
getLoanAmount(): float  
getBorrower(): str  
setAnnualInterestRate(  
    annualInterestRate: float): None  
setNumberOfYears(  
    numberOfYears: int): None  
setLoanAmount(  
    loanAmount: float): None  
setBorrower(borrower: str): None  
setMonthlyPayment(): float  
getTotalPayment(): float
```





# 对象和类



客户端使用Loan类：

```
# Create a Loan object
loan = Loan(annualInterestRate, numberOfYears,
            loanAmount, borrower)

# Display loan date, monthly payment, and total payment
print("The loan is for", loan.getBorrower() )
print("The monthly payment is",
      format(loan.getMonthlyPayment() , ".2f"))
print("The total payment is",
      format(loan.getTotalPayment() , ".2f"))
```

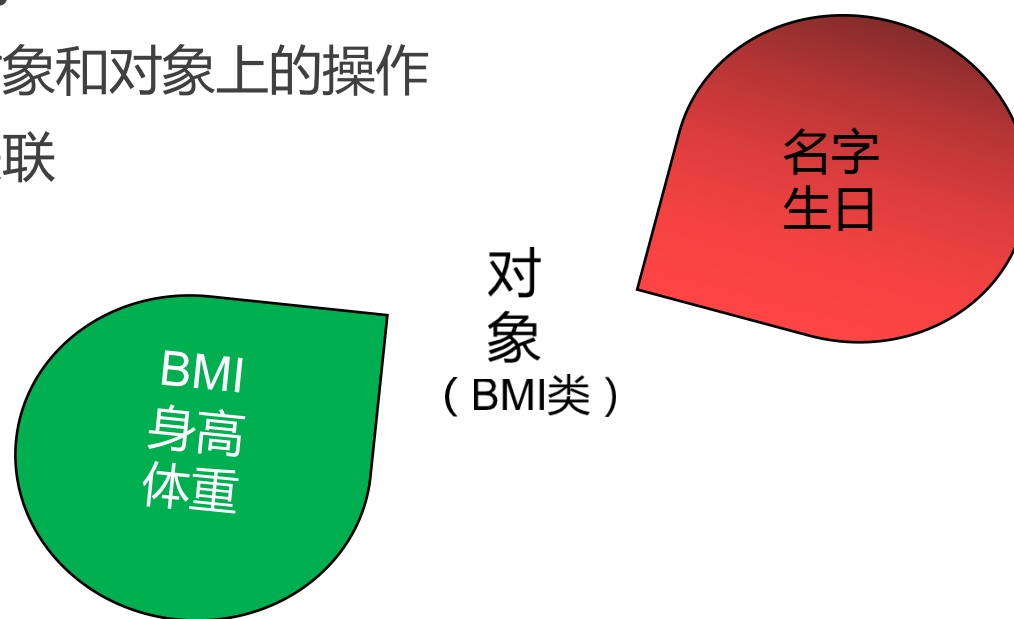




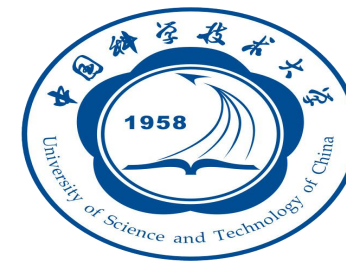
# 对象和类

## ■ 面向对象的思考

- 设计重点是对象和对象上的操作
- 数据的恰当关联



Python 程序可被视为相互作用的对象的集合。



# Str类：更多字符串和特殊方法

## ■ Str类

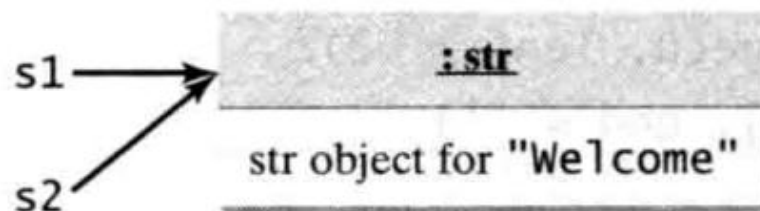
### ■ 创建字符串

```
s1 = str() # Create an empty string object 用构造函数  
s2 = str("Welcome") # Create a string object for Welcome
```

Python中 

```
s1 = "" # Same as s1 = str() 用字符串值  
s2 = "Welcome" # Same as s2 = str("Welcome")
```

```
>>> s1 = "Welcome"  
>>> s2 = "Welcome"  
>>> id(s1)  
505408902  
>>> id(s2)  
505408902
```



Python使用一个对象表示相同内容的字符串；类似，int也如此。



# Str类：更多字符串和特殊方法

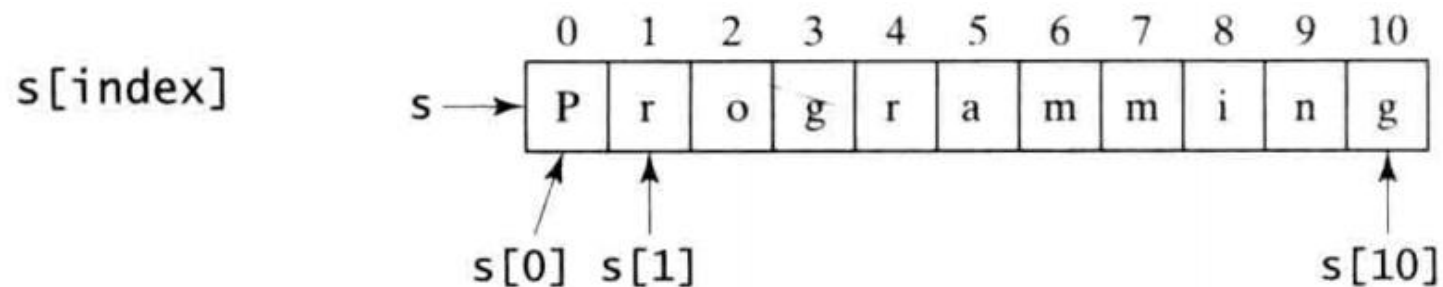
- 处理字符串的函数

```
>>> s = "Welcome"
>>> len(s)
7
>>> max(s)
'o'
>>> min(s)
'W'
>>>
```



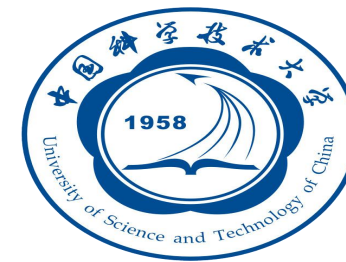
# Str类：更多字符串和特殊方法

- 下标运算符[]



应用：

```
>>> s = "Welcome"
>>> s[-1]
'e'
>>> s[-2]
'm'
>>>
```



# Str类：更多字符串和特殊方法

- 截取运算符[start:end]

例程1：

```
>>> s = "Welcome"
>>> s[ : 6]
'Welcom'
>>> s[4 : ]
'ome'
>>> s[1 : -1]
'elcom'
>>>
```

1. 若start或者end为负，则用len(s)+index替换
2. 若j>len(s),则被替换成len(s)

例程2：

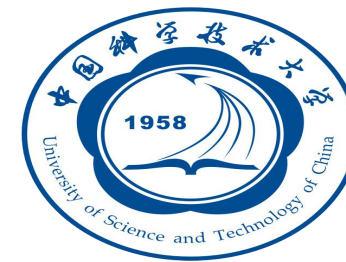
```
Python 3.6 (64-bit)
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 201
on win32
Type "help", "copyright", "credits" or "
>>> s="welcome"
>>> s
'welcome'
>>> s1=s[-1:9]
>>> s1
'e'
>>> s2=s[-3:-5]
>>> s2
''
>>> s[-5]
'l'
>>> s3=s[-5:-3]
>>> s3
'lc'
>>> _
```



# Str类：更多字符串和特殊方法

- 连接运算符+ 和 复制运算符\*

```
>>> s1 = "Welcome"
>>> s2 = "Python"
>>> s3 = s1 + " to " + s2
>>> s3
'Welcome to Python'
>>> s4 = 3 * s1
>>> s4
'WelcomeWelcomeWelcome'
>>> s5 = s1 * 3
>>> s5
'WelcomeWelcomeWelcome'
>>>
```



# Str类：更多字符串和特殊方法

- in 和 not in 运算符

```
>>> s1 = "Welcome"
>>> "come" in s1
True
>>> "come" not in s1
False
>>>
```

- 比较运算符

==、!=、>、<、>= 和 <=



# Str类：更多字符串和特殊方法

- 迭代字符串：用for循环访问整个字符串

```
for ch in s:  
    print(ch)
```

```
for i in range(0, len(s), 2):  
    print(s[i])
```

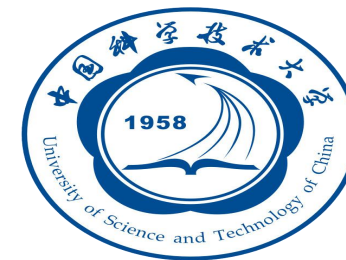




# Str类：更多字符串和特殊方法

## ■ 测试字符串

str	
<code>isalnum(): bool</code>	如果这个字符串中的字符是字母数字且至少有一个字符则返回 True
<code>isalpha(): bool</code>	如果这个字符串中的字符是字母且至少有一个字符则返回 True
<code>isdigit(): bool</code>	如果这个字符串中只含有数字字符则返回 True
<code>isidentifier(): bool</code>	如果这个字符串是 Python 标识符则返回 True
<code>islower(): bool</code>	如果这个字符串中的所有字符全是小写的且至少有一个字符则返回 True
<code>isupper(): bool</code>	如果这个字符串中的所有字符全是大写的且至少有一个字符则返回 True
<code>isspace(): bool</code>	如果这个字符串中只包含空格则返回 True



# Str类：更多字符串和特殊方法

## ■ 搜索子串

str
endswith(s1: str): bool
startswith(s1: str): bool
find(s1): int
rfind(s1): int
count(substring): int

```
>>> s = "welcome to python"
>>> s.endswith("thon")
True
>>> s.startswith("good")
False
>>> s.find("come")
3
>>> s.find("become")
-1
>>> s.rfind("o")
15
>>> s.count("o")
3
>>>
```

ue  
ue  
如果字符串中不存  
  
如果字符串中不存  
  
出现的次数



# Str类：更多字符串和特殊方法

- 转换字符串：创建了新的字符串

str
<code>capitalize(): str</code>
<code>lower(): str</code>
<code>upper(): str</code>
<code>title(): str</code>
<code>swapcase(): str</code>
<code>replace(old, new): str</code>

返回这个复制的字符串并只大写第一个字符

返回这个复制的字符串并将所有字母转换为小写的

返回这个复制的字符串并将所有字母转换为大写的

返回这个复制的字符串并大写每个单词的首字母

返回这个复制的字符串，将小写字母转换成大写，将大写字母转换成小写

返回一个新的字符串，它用一个新字符串替换旧字符串所有出现的地方

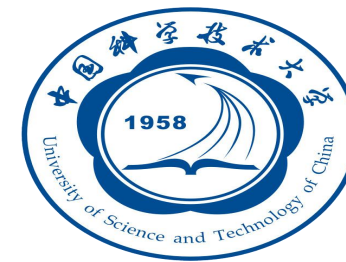


# Str类：更多字符串和特殊方法

- 删除字符串中的空格（空白字符串）

str
<code>lstrip(): str</code>
<code>rstrip(): str</code>
<code>strip(): str</code>

返回去掉前端空白字符的字符串
返回去掉末端空白字符的字符串
返回去掉两端空白字符的字符串



# Str类：更多字符串和特殊方法

例程：

```
>>> s2 = s.rstrip()
>>> s2
' Welcome to Python' 不能删除单词之间的空白字符
>>> s3 = s.strip()
>>> s3
'Welcome to Python'
```

建议：在输入的字符串用strip()来确保删除输入末尾任何不需要的字符

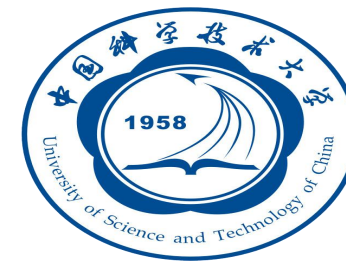


# Str类：更多字符串和特殊方法

- 格式化字符串

str	
center(width): str	返回在给定宽度域上居中的字符串副本
ljust(width): str	返回在给定宽度域上左对齐的字符串文本
rjust(width): str	返回在给定宽度域上右对齐的字符串文本
format(items): str	格式化一个字符串



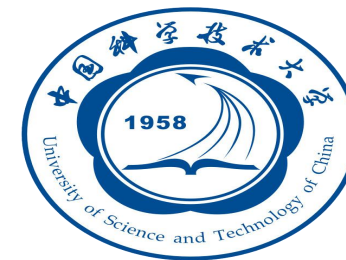


# Str类：更多字符串和特殊方法

## ■ 运算符重载和特殊方法

- 使用内嵌的运算符为用户定义方法
- 以\_\_开头，以\_\_结尾

```
s1 = "Washington"  
s2 = "California"  
print("The first character in s1 is", s1.__getitem__(0))  
print("s1 + s2 is", s1.__add__(s2))  
print("s1 < s2?", s1.__lt__(s2))
```



# Str类：更多字符串和特殊方法

运算符	方法	描述	运算符	方法	描述
+	<code>__add__(self, other)</code>	加法	<code>!=</code>	<code>__ne__(self, other)</code>	不等于
*	<code>__mul__(self, other)</code>	乘法	<code>&gt;</code>	<code>__gt__(self, other)</code>	大于
-	<code>__sub__(self, other)</code>	减法	<code>&gt;=</code>	<code>__ge__(self, other)</code>	大于等于
/	<code>__truediv__(self, other)</code>	除法	<code>[index]</code>	<code>__getitem__(self, index)</code>	下标运算符
%	<code>__mod__(self, other)</code>	求余	<code>in</code>	<code>__contains__(self, value)</code>	检查其成员资格
<code>&lt;</code>	<code>__lt__(self, other)</code>	小于	<code>len</code>	<code>__len__(self)</code>	元素个数
<code>&lt;=</code>	<code>__le__(self, other)</code>	小于等于	<code>str</code>	<code>__str__(self)</code>	字符串表示
<code>==</code>	<code>__eq__(self, other)</code>	等于			





# Str类：更多字符串和特殊方法

- 不改变原有的运算符的含义及参数的意义
- 对自定义对象将运算符赋予新的规则
- 许多特殊的运算符被定义为Python的内置类型

如 int和float类型

$i.\_\_add\_\_(j) \longleftrightarrow i+j$

$print(x) \longleftrightarrow print(str(x))$

```
class Mylist:
    def __init__(self, iterable=()):
        self.data = list(iterable)

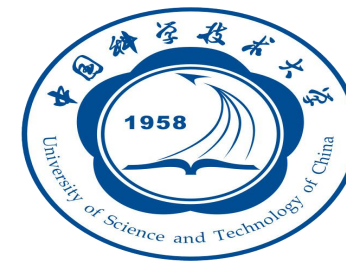
    def __repr__(self):
        return 'Mylist(%s)' % self.data

    def __add__(self, lst):

        return Mylist(self.data + lst.data)

    def __mul__(self, rhs):
        # rhs为int类型, 不能用rhs.data
        return Mylist(self.data * rhs)

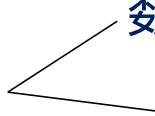
L1 = Mylist([1, 2, 3])
L2 = Mylist([4, 5, 6])
L3 = L1 + L2
print(L3) # Mylist([1,2,3,4,5,6])
L4 = L2 + L1
print(L4) # Mylist([4,5,6,1,2,3])
L5 = L1 * 3
print(L5) # Mylist([1,2,3,1,2,3,1,2,3])
```



# Str类：更多字符串和特殊方法

## ■ 实例研究：Rational类

表示和处理有理数

**Rational类**  **数据域：** numerator、denominator  
**行为：** 加减乘除、比较  
转换为整数、浮点数、字符串  
返回分子、分母



# Str类：更多字符串和特殊方法

## ■ Rational类的UML类图

Rational
<pre>-numerator: int -denominator: int  Rational(numerator = 0: int,           denominator = 1: int)  __add__(secondRational:          Rational): Rational __sub__(secondRational:          Rational): Rational __mul__(secondRational:          Rational): Rational __truediv__(secondRational:             Rational): Rational __lt__(secondRational:         Rational): bool  Also __le__, __eq__, __ne__,       __gt__, __ge__ are supported  __int__(): int __float__(): float __str__(): str  __getitem__(i)</pre>

这个有理数的分子  
这个有理数的分母

创建带有特定分子（默认为 0）和分母（默认为 1）的有理数

返回这个有理数和其他有理数的加法结果

返回这个有理数和其他有理数的减法结果

返回这个有理数和其他有理数的乘法结果

返回这个有理数和其他有理数的除法结果

将这个有理数和另一个有理数进行比较

返回分子除以分母的整数结果

返回分子除以分母的浮点数结果

返回形式为“分子 / 分母”的字符串。如果分母为 1，返回分子

使用 [0] 返回分子，使用 [1] 返回分母



# Str类：更多字符串和特殊方法

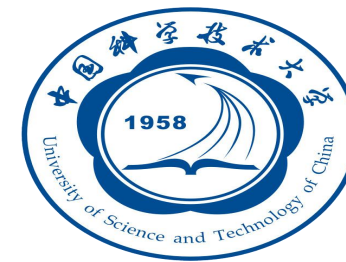
- Rational类定义的部分分析：

```
class Rational:
    def __init__(self, numerator = 1, denominator = 0):
        divisor = gcd(numerator, denominator)
        self.__numerator = (1 if denominator > 0 else -1) \
            * int(numerator / divisor)
        self.__denominator = int(abs(denominator) / divisor)
    def __getitem__(self, index):
    def __lt__(self, secondRational):
    def gcd(n, d):
```

私有数据域 → 最简化形式

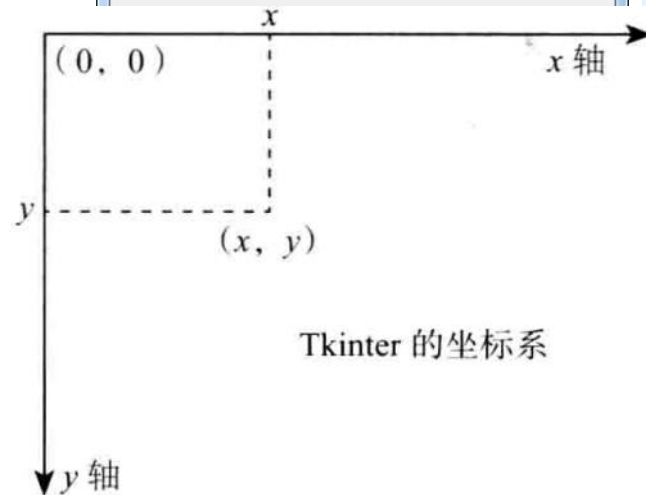
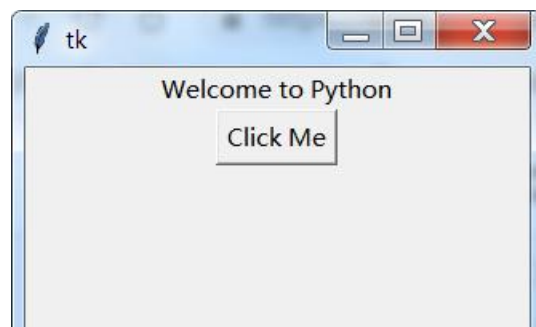
Rational类中的成员方法

定义在Rational类中的函数，求最大公约数



# 实践：用Tkinter进行GUI设计

## ■ Tkinter使用介绍



```
Tkinter_tst.py - D:/Python_tst/Tkinter_tst.py (3.6.4)
File Edit Format Run Options Window Help

import tkinter

window = tkinter.Tk()
label = tkinter.Label(window, text="Welcome to Python")
button=tkinter.Button(window, text = "Click Me")
label.pack()
button.pack()

window.mainloop()
```

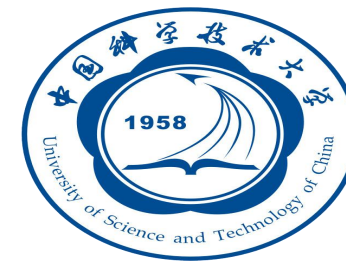
```
管理员: 命令提示符 - python tkinter_tst.py

File "tkinter_tst.py", line 3, in <module>
    window = TK()
NameError: name 'TK' is not defined

D:\Python_tst>python tkinter_tst.py
Traceback (most recent call last):
  File "tkinter_tst.py", line 1, in <module>
    from Tkinter import *
ModuleNotFoundError: No module named 'Tkinter'

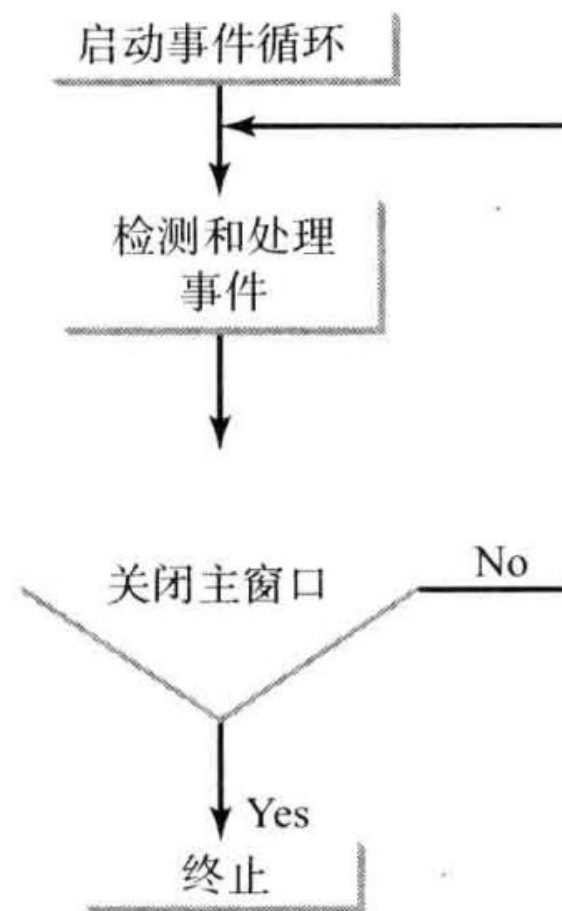
D:\Python_tst>python tkinter_tst.py
Traceback (most recent call last):
  File "tkinter_tst.py", line 4, in <module>
    label = Label(window, text="Welcome to Python")
NameError: name 'Label' is not defined
```

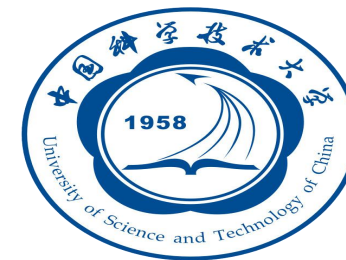




# 实践：用Tkinter进行GUI设计

- Tkinter needs mainloop to work (to call all functions behind).
- `window.mainloop()` 创建了一个循环
- 基于事件驱动



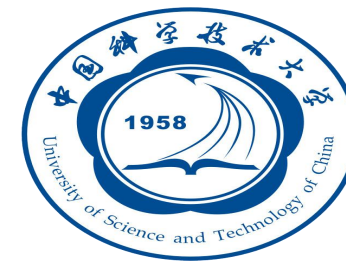


# 实践：用Tkinter进行GUI设计

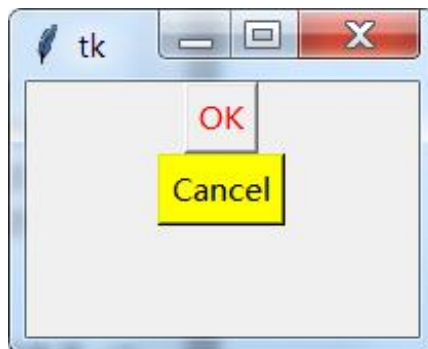
## ■ 事件处理

```
*ProcessButtonEvent.py - D:/Python_tst/ProcessButtonEvent.py (3.6.4)*
File Edit Format Run Options Window Help
import tkinter
def processOK(): ← 回调函数，被绑定到按钮
    print("OK button is clicked")
def processCancel():
    print("Cancel button is clicked")
window = tkinter.Tk()
btOK=tkinter.Button(window, text="OK", fg="red", command = processOK)
btCancel=tkinter.Button(window, text="Cancel", bg="yellow", command = processCancel)
btOK.pack()
btCancel.pack()
window.mainloop()
```

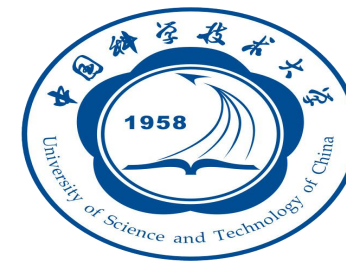




# 实践：用Tkinter进行GUI设计



```
D:\Python_tst>python ProcessButtonEvent.py  
OK button is clicked  
Cancel button is clicked  
Cancel button is clicked  
OK button is clicked
```



# 实践：用Tkinter进行GUI设计

- 定义一个类来创建GUI和处理GUI事件：

```
import tkinter

class ProcessButtonEvent:
    def __init__(self):
        window = tkinter.Tk()
        btOK = tkinter.Button(window, text = "OK", command = self.processOK)
        btCancel = tkinter.Button(window, text = "Cancel", command = self.processCancel)

        btOK.pack()
        btCancel.pack()

        window.mainloop()

    def processOK(self):
        print("OK button is clicked")

    def processCancel(self):
        print("Cancel button is clicked")

ProcessButtonEvent()
```

类中的实例方法

用类被



# 实践：用Tkinter进行GUI设计

## ■ 小构件类

小构件类	描述
Button	一个用来执行一条命令的简单按钮
Canvas	结构化的图形，用于绘制图形、创建图形编辑器以及实现自定制的小构件类
Checkbutton	单击复选按钮在值之间切换
Entry	一个文本输入域，也被称为文本域或文本框
Frame	包含其他小构件的一个容器小构件
Label	显示文本或图像
Menu	用来实现下拉和弹出菜单的菜单栏
Menubutton	用来实现下拉菜单的菜单按钮
Message	显示文本，类似于标签小构件，但能自动将文本放在给定的宽度或宽高比内
Radiobutton	单击单选按钮设置变量为那个值，同时清除所有和同一个变量相关联的其他单选按钮
Text	格式化的文本显示，允许用不同的风格和属性显示和编辑文本，也支持内嵌的图片和窗口



# 实践：用Tkinter进行GUI设计

## ■ 例：widgetsDemo小程序

父容器, text, bg, variable, command, grid()方法

父容器  
text, bg, variable, command, value  
grid()方法

checkboxbutton radiobutton

grid()方法 label entry grid()方法 button grid()方法

message

text insert()方法

frame pack布局方法

title属性  
frame方法

window=Tk()

Widgets Demo

Bold ☐ Yellow ☐

Enter your name:  Get Name

It is a widgets demo

Tip

The best way to learn Tkinter is to read these carefully designed examples and use them to create your applications.

### 输入域类型：

IntVar、DoubleVar 和 StringVar

### 调用方法：

```
def processCheckbutton(self):  
def processRadiobutton(self):  
def processButton(self):
```



# 实践：用Tkinter进行GUI设计

## ■ 画布

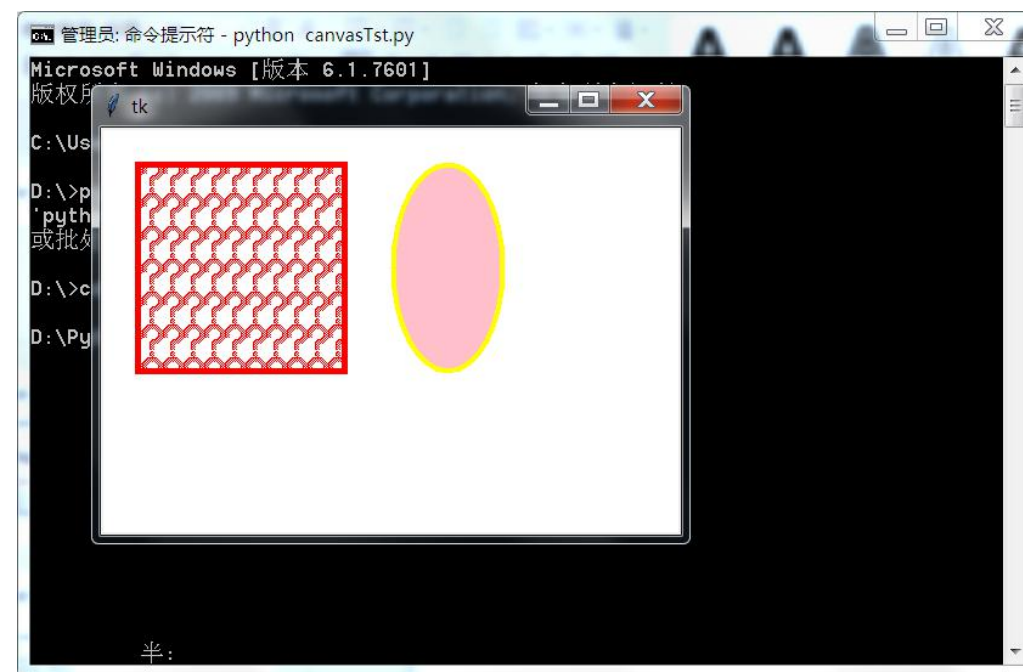
- 使用canvas小构件显示图形
- canvas :
  - create\_arc : 绘制弧。
  - create\_bitmap : 绘制位图。
  - create\_image : 绘制图片。
  - create\_line() : 绘制直线。
  - create\_polygon : 绘制多边形。
  - create\_text : 绘制文字。
  - create\_window : 绘制组件。

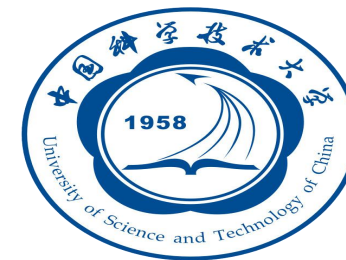




# 实践：用Tkinter进行GUI设计

```
canvasTst.py - D:\Python_tst\canvasTst.py (3.6.4)
File Edit Format Run Options Window Help
from tkinter import *
# 创建窗口
root = Tk()
# 创建并添加Canvas
cv = Canvas(root, background='white')
cv.pack(fill=BOTH, expand=YES)
cv.create_rectangle(30, 30, 200, 200,
    outline='red', # 边框颜色
    stipple = 'question', # 填充的位图
    fill="red", # 填充颜色
    width=5 # 边框宽度
)
cv.create_oval(240, 30, 330, 200,
    outline='yellow', # 边框颜色
    fill='pink', # 填充颜色
    width=4 # 边框宽度
)
root.mainloop()
```





# 实践：用Tkinter进行GUI设计

```
canvasTstList.py - D:\Python_tst\canvasTstList.py (3.6.4)
File Edit Format Run Options Window Help
from tkinter import *

#创建窗口对象背景
root = Tk()
root.title('画布')

#创建两个列表
li = ['c', 'python', 'php', 'html', 'sql', 'java']
movie = ['CSS', 'JQuery', 'Bootstrap']

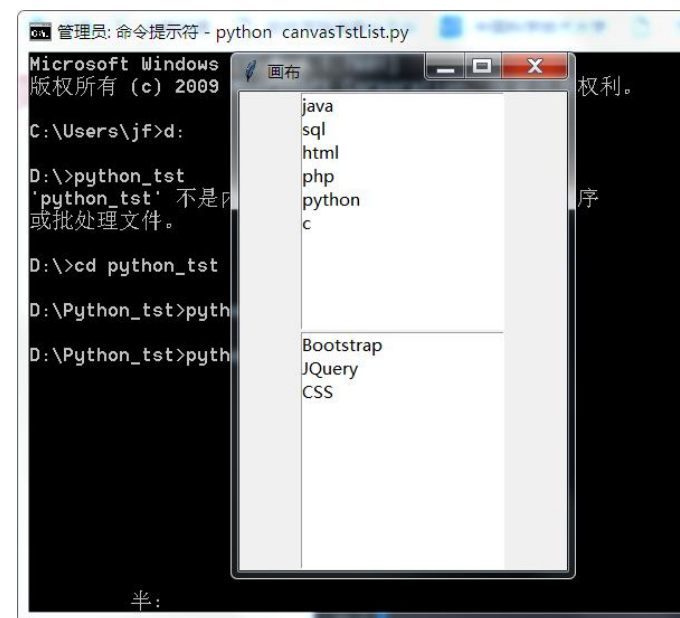
#创建两个列表组件
listb = Listbox(root)
listb2 = Listbox(root)

#第一个小部件插入数据
for item in li:
    listb.insert(0, item)

#第二个小部件插入数据
for item in movie:
    listb2.insert(0, item)

#将小部件放置到主窗口中
listb.pack()
listb2.pack()

root.mainloop()
```





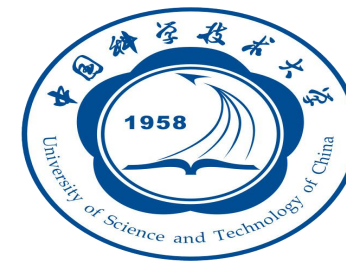


# 实践：用Tkinter进行GUI设计

## ■ 几何管理器

- 网格管理器：将小构件放入不可见网格的一个单元内
  - 参数：rowspan colspan

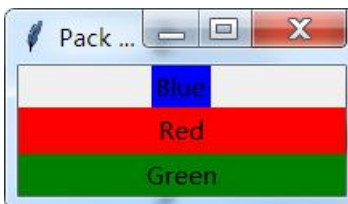
```
message = Message(window, text =  
    "This Message widget occupies three rows and two columns")  
message.grid(row = 1, column = 1, rowspan = 3, colspan = 2)  
Label(window, text = "First Name:").grid(row = 1, column = 3)  
Entry(window).grid(row = 1, column = 4, padx = 5, pady = 5)  
Label(window, text = "Last Name:").grid(row = 2, column = 3)  
Entry(window).grid(row = 2, column = 4)  
Button(window, text = "Get Name").grid(row = 3,  
    padx = 5, pady = 5, column = 4, sticky = E)
```



# 实践：用Tkinter进行GUI设计

- 包管理器：将小组件依次放置在另一个的顶部或者一个挨一个的放置

```
from tkinter import *  
  
class PackManagerDemo:  
    def __init__(self):  
        window = Tk()  
        window.title("Pack Manager Demo 1")  
  
        Label(window, text = "Blue", bg="blue").pack()  
        # fill通过X, Y, BOTH 来填充水平, 垂直, 或者两个方向的空间  
        # expand告诉管理器分配额外的空间给小构件  
        Label(window, text = "Red", bg = "red").pack(fill = BOTH, expand = 1)  
        Label(window, text = "Green", bg = "green").pack(fill = BOTH)  
  
        window.mainloop()  
  
PackManagerDemo()
```

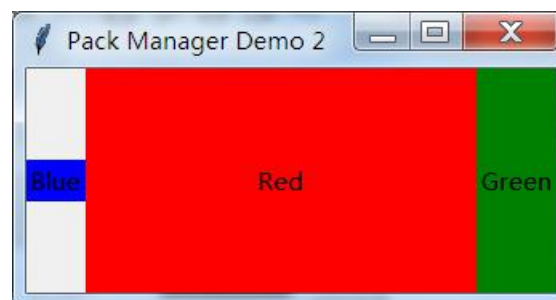




# 实践：用Tkinter进行GUI设计

#side可以是LEFT, RIGHT, TOP, BOTTOM,默认是TOP。

```
Label(window, text = "Blue", bg="blue").pack(side = LEFT)
Label(window, text = "Red", bg = "red").pack(side = LEFT, fill = BOTH, expand = 1)
Label(window, text = "Green", bg = "green").pack(side = LEFT, fill = BOTH)
```



pack 在使用上更加简单，适用于少量组件的排列。创建相对复杂的布局结构，可以使用多个框架（Frame）结构构成

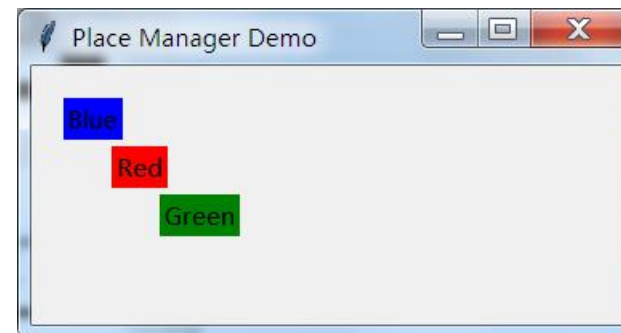
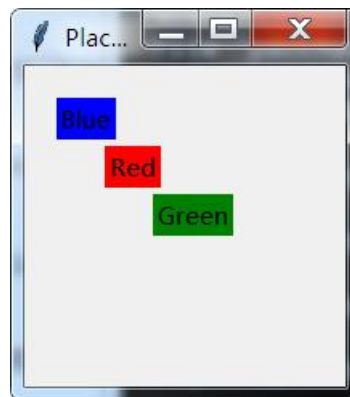


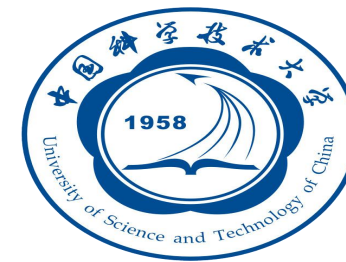
# 实践：用Tkinter进行GUI设计

- 位置管理器：将小构件放在绝对位置

```
Label(window, text = "Blue", bg = "blue").place(  
    x = 20, y = 20)  
Label(window, text = "Red", bg = "red").place(  
    x = 50, y = 50)  
Label(window, text = "Green", bg = "green").place(  
    x = 80, y = 80)
```

位置管理器不能兼容所有计算机。





# 实践：用Tkinter进行GUI设计

## ■ 显示图像

- 用PhotoImage类创建图像

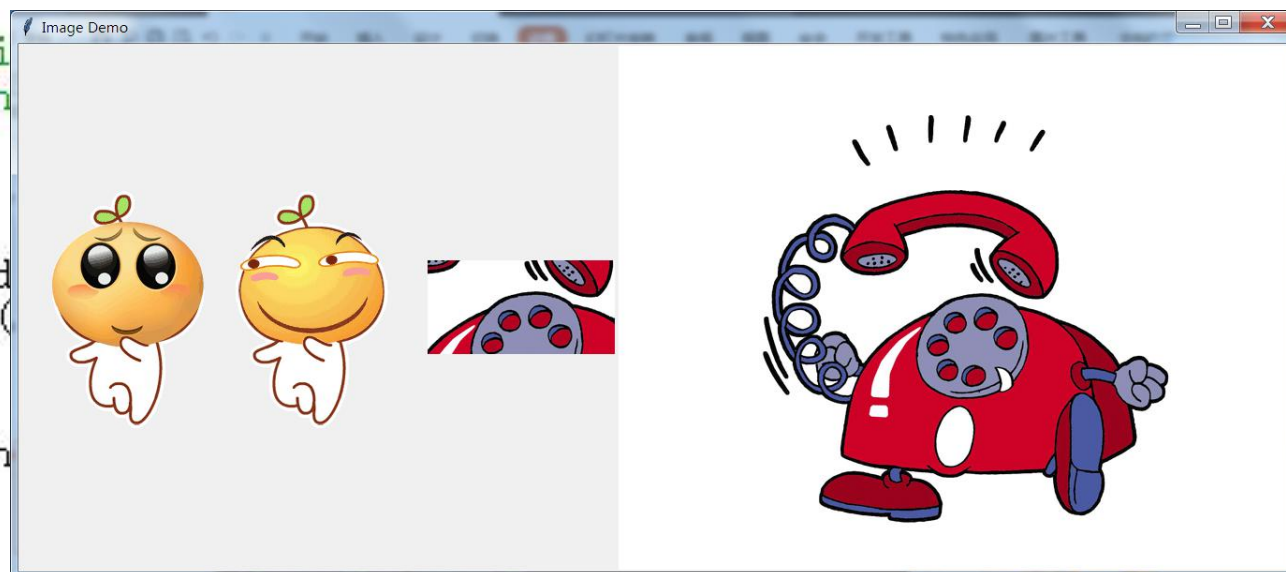
```
photo = PhotoImage(file = imagefilename)
```

- creat\_image方法

```
caImage = PhotoImage(file="pic/ca.gif")  
phoneImage = PhotoImage(file="pic/phone.gif")
```

```
frame1=Frame(window)  
frame1.pack()  
Label(frame1, image=caImage).pack(side=LEFT)  
Label(frame1, image=phoneImage).pack(side=LEFT)
```

```
canvas = Canvas(frame1)  
canvas.create_image(90, 50, image=phoneImage)  
canvas["width"] = 200  
canvas["height"] = 100  
canvas.pack(side = LEFT)
```







# 实践：用Tkinter进行GUI设计

## ■ 菜单

- 使用menu类创建菜单栏和菜单
- add\_command方法给菜单添加条目

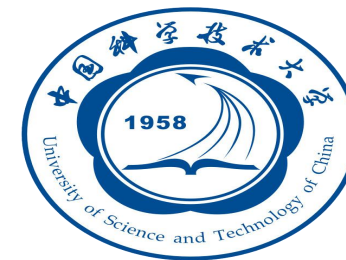
```
# 定义顶级菜单
menubar = Menu(root)

#定义子菜单
#add_command 中的command 指被点击时调用的方法，accelerator指快捷键
fmenu = Menu(menubar)
for item in ['新建', '打开', '保存', '另存为']:
    fmenu.add_command(label = item)
```

```
# 级联，menu 指明要把哪个菜单级联到该菜单项上
menubar.add_cascade(label = "文件", menu = fmenu)
```

```
#或者 root.config(menu=menubar)，在窗口显示menubar
root['menu'] = menubar
```



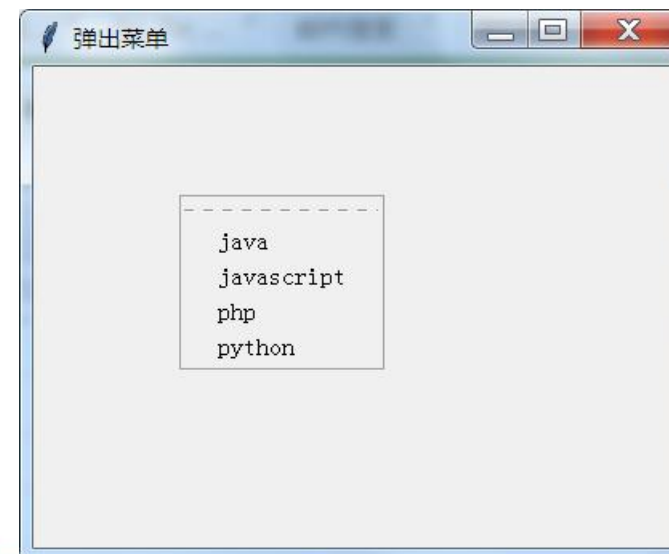


# 实践：用Tkinter进行GUI设计

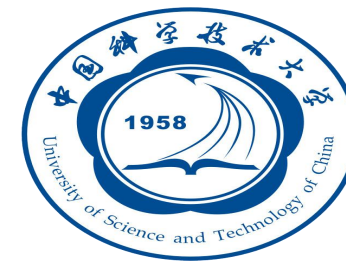
## ■ 弹出菜单

- 步骤：创建一个menu的实例  
添加条目  
将一个小构件与一个时间绑定

```
def popLabel():  
    global top  
    Label(top, text="I love python").pack()  
  
# 创建第四个菜单项，并绑定事件  
menubar.add_command(label='python', command=popLabel)  
  
# 创建弹出方法  
def pop(event):  
    # Menu 类里面的post 方法，接收x 和y 坐标，在相应的位置弹出菜单。  
    menubar.post(event.x_root, event.y_root)  
  
# 鼠标右键是用的<Button-3>，绑定pop 方法  
top.bind("<Button-3>", pop)
```







# 实践：用Tkinter进行GUI设计

## ■ 鼠标、按键事件和绑定

- 用bind方法将鼠标事件和回调函数绑定：

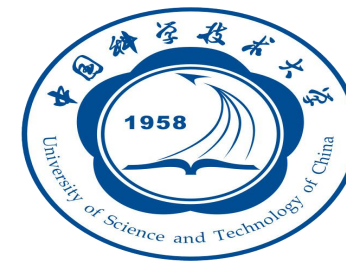
```
widget.bind(event, handler)
```

- 每个处理器都将一个事件作为参数：

```
def popup(event):  
    menu.post(event.x_root, event.y_root)
```

- 例程：

```
# Bind with <Button-1> event  
canvas.bind("<Button-1>", self.processMouseEvent)  
  
# Bind with <Key> event  
canvas.bind("<Key>", self.processKeyEvent)
```



# 实践：用Tkinter进行GUI设计

## ■ 绑定

- 一个 Tkinter 应用生命周期中的大部分时间都处在一个消息循环 (event loop) 中，等待事件的发生。
- Tkinter 提供了用以处理相关事件的机制. 处理函数可以被绑定给各个控件的各种事件.

`widget.bind(event, handler)`

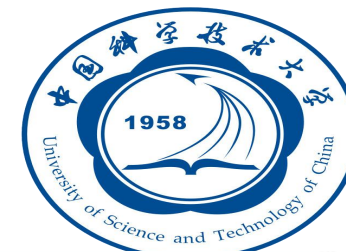
↓  
回调函数



# 实践：用Tkinter进行GUI设计

## ■ 事件

- 在Tkinter中，事件的描述格式为：<[modifier-]type[-detail]>
  - modifier：事件修饰符。如：Alt、Shit组合键和Double事件。
  - type：事件类型。如：按键（Key）、鼠标（Button/Motion/Enter/Leave/Release）、Configure等。
  - detail：事件细节。如：鼠标左键（1）、鼠标中键（2）、鼠标右键（3）。



# 实践：用Tkinter进行GUI设计

Event	Description
<Button>	某个鼠标按键在控件上被点击. <b>detail</b> 指定了哪一个按键被点击了, 比如, 鼠标左键点击为 <Button-1>, 鼠标中键点击为 <Button-2>, 鼠标右键点击为 <Button-3>, 向上滚动滑轮为 <Button-4>, 向下滚动滑轮为 <Button-5>. 如果在控件上按下鼠标的某个键并保持按下, Tkinter 将“抓住”该事件. 之后的鼠标事件, 比如 鼠标移动 或 鼠标按键释放 事件, 会被自动发送给该控件处理, 即使鼠标移动出该控件时依然如此. 鼠标相对当前控件的位置会被存储在 event 对象中的 x 和 y 字段中传递给回调函数.
<Motion>	鼠标在某个按键被按下时的移动事件. 鼠标左键点击为 <B1-Motion>, 鼠标中键点击为 <B2-Motion>, 鼠标右键点击为 <B3-Motion>. 鼠标相对当前控件的位置会被存储在 event 对象中的 x 和 y 字段中传递给回调函数.
<ButtonRelease>	按钮点击释放事件. 鼠标左键点击为 <ButtonRelease-1>, 鼠标中键点击为 <ButtonRelease-2>, 鼠标右键点击为 <ButtonRelease-3>. 鼠标相对当前控件的位置会被存储在 event 对象中的 x 和 y 字段中传递给回调函数.
<Double-Button>	鼠标双击事件. 鼠标左键点击为 <Double-Button-1>, 鼠标中键点击为 <Double-Button-2>, 鼠标右键点击为 <Double-Button-3>. Double 和 Triple 都可以被用作前缀. 注意: 如果同时绑定单击事件 (<Button-1>) 和双击事件 (<Double-Button-1>), 则两个回调都会被调用.
<Enter>	鼠标移入控件事件. 注意: 这个事件不是 Enter 键按下事件, Enter 键按下事件是 <Return>.
<Leave>	鼠标移出控件事件.



# 实践：用Tkinter进行GUI设计

<FocusIn>	控件或控件的子空间获得键盘焦点.
<FocusOut>	控件丢失键盘焦点 (焦点移动到另一个控件).
<Return>	Enter 点击事件. 键盘上的所有键位都可以被绑定. 特殊键位名称包括 Cancel, BackSpace, Tab, Return (Enter), Shift_L (任意 Shift), Control_L (任意 Control), Alt_L (任意 Alt), Pause, Caps_Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num_Lock, and Scroll_Lock
<Key>	键盘按键点击事件. 键值被存储在 event 对象中传递. (特殊键位会传递空键值).
a	"a" 键被点击. 其他字符也可以如此定义. 特殊情况包括 空格 (<space>) 和 小于号 (<less>). 注意 "1" 是绑定键盘键位, 而 <1> 则是按钮绑定.
<Shift-Up>	在 shift 被按下时点击 up 键. 同样的, 也有 Alt-Up, Control-Up 事件.
<Configure>	控件大小改变事件. 新的控件大小会存储在 event 对象中的 width 和 height 属性传递. 有些平台上该事件也可能代表控件位置改变.





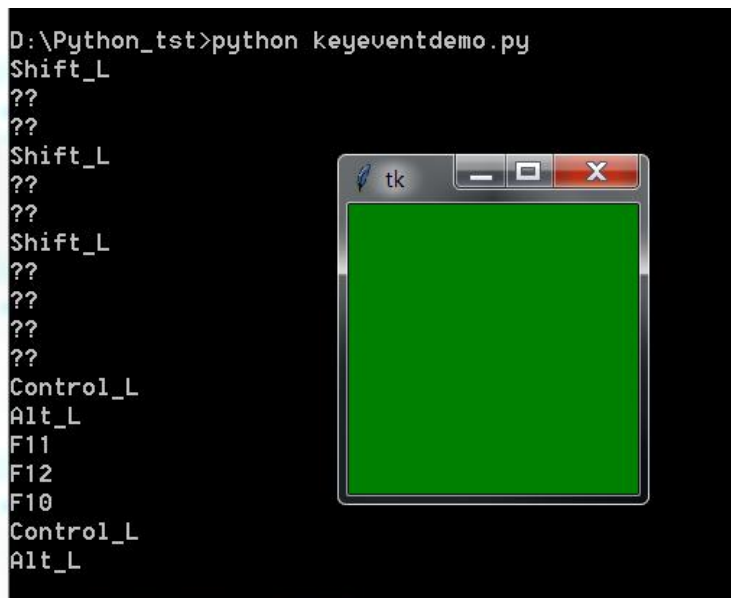
# 实践：用Tkinter进行GUI设计

事件属性	描述
char	从键盘输入的和按键事件相关的字符
keycode	从键盘输入的和按键事件相关的键的键代码（即统一码）
keysym	从键盘输入的和按键事件相关的键的键符号（即字符）
num	按键数字（1、2、3）表明按下的是哪个鼠标键
widget	触发这个事件的小构件对象
x 和 y	当前鼠标在小构件中以像素为单位的位置
x_root 和 y_root	当前鼠标相对于屏幕左上角的以像素为单位的位置



# 实践：用Tkinter进行GUI设计

- 例程：输出键盘的键值



```
# keysym属性的测试

from tkinter import *

def call_back(event):
    print(event.keysym)

def main():
    root = Tk()

    frame = Frame(root,
                    width=200, height=200,
                    background='green')

    # 输出键盘特殊按键的keysym。
    frame.bind("<KeyPress>", call_back)
    frame.pack()

    # 当前框架有效，键盘触发
    frame.focus_set()

    mainloop()

if __name__ == '__main__':
    main()
```



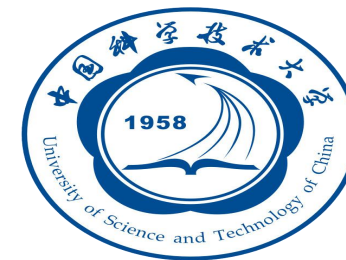


# 实践：用Tkinter进行GUI设计

## ■ 动画

- canvas类用来开发动画
- move(tags,dx,dy)方法移动图片

```
def animate(self): # Move the message
    while not self.isStopped:
        self.canvas.move("text", self.dx, 0) # Move text
        self.canvas.after(self.sleepTime) # Sleep
        self.canvas.update() # Update canvas
        if self.x < self.width:
            self.x += self.dx # Set new position
        else:
            self.x = 0 # Reset string position to beginning
            self.canvas.delete("text")
            self.canvas.create_text(self.x, 30,
                                    text = "Message moving?", tags = "text")
```



# 实践：用Tkinter进行GUI设计

## ■ 滚动条

- Scrollbar与Text、Canvas或者Listbox一起使用，可以在垂直或水平方向展开控件中的内容

```
frame1 = Frame(window)
frame1.pack()
scrollbar = Scrollbar(frame1)
scrollbar.pack(side = RIGHT, fill = Y)
text = Text(frame1, width = 40, height = 10, wrap = WORD,
            1. yscrollcommand = scrollbar.set)
text.pack()
2. scrollbar.config(command = text.yview)
```

添加一个水平方向的scrollbar，只需对应设置 xscrollcommand 和 xview 即可。



# 实践：用Tkinter进行GUI设计

- 三种标准对话框模块：messagebox、filedialog、colorchooser

```
import tkinter.messagebox
import tkinter.simpledialog
import tkinter.colorchooser
```

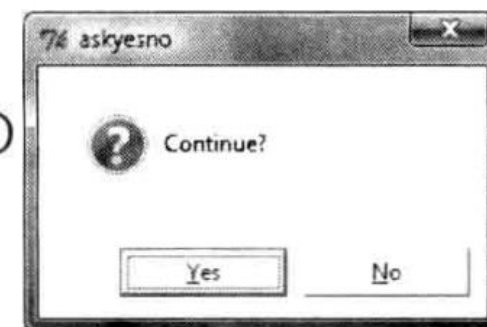
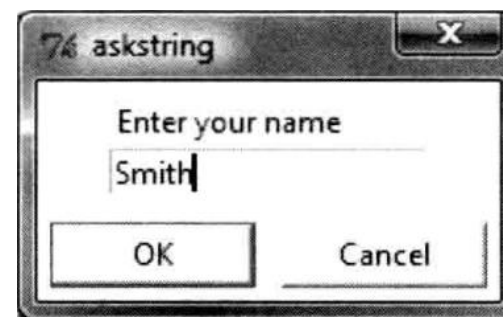
```
tkinter.messagebox.showinfo("showinfo", "This is an info msg")
```

```
isYes = tkinter.messagebox.askyesno("askyesno", "Continue?")
```

```
print(isYes) 显示一个问题. 选择 ok 则返回 True
```

```
name = tkinter.simpledialog.askstring(
    "askstring", "Enter your name")
print(name)
```

所有对话框都是模态窗口



# 程序作业



- \*9.23 （按钮和单选按钮）编写程序使用单选按钮选择文本的背景色，如图所示 变量色彩是红色、黄色、灰色和绿色。程序使用按钮 “<=” 和 “=>” 将文本向左或向右移动。

