Python的行

多行语句

- **Python中没有强制的语句终止字符**
- **Python语句中一般以新行（换行）作为语句的结束符**

In [28]:
```
thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable1 = 1
thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable2 = 2
thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable3 = 3

#这一条语句写在一行太长了，不方便看也不美观
plusResult = thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable1 +thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable2 +

print(plusResult)
```

```
6
```

多行语句

- 直接把一条长语句中间断开成多行可不行

In [31]:
```
#这样直接分成多行可是不行，要报错的
plusResult = thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable1 +
thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable2 +
thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable3

print(plusResult)
```

```
  File "<ipython-input-31-68e8f0b54b24>", line 2
    plusResult = thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable1 +
                                                                           ^
SyntaxError: invalid syntax
```

多行语句

- 可以使用斜杠（\）将一行的语句分为多行显示

In [30]:
```
#这样就对了可以使用斜杠（ \）将一行的语句分为多行显示
plusResult = thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable1 +\
thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable2 +\
thisIsaVeryLongVariableNameSoCantWriteInOneLineVariable3

print(plusResult)
```

```
6
```

多行语句

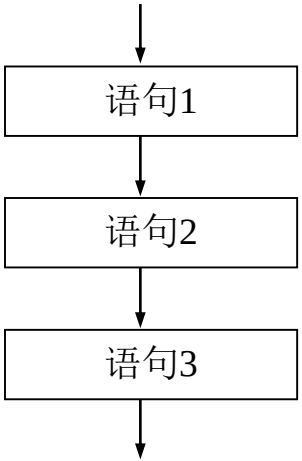
- 语句中包含 [], {} 或 () 括号就不需要使用多行连接符

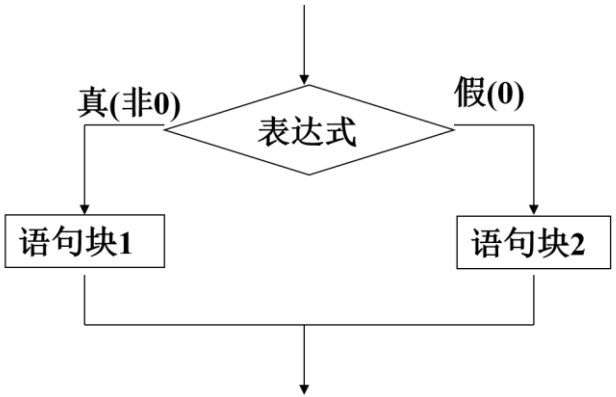
```
In [1]: days = ('Monday', 'Tuesday', 'Wednesday',
                'Thursday', 'Friday', 'Saturday', 'Sunday')
```

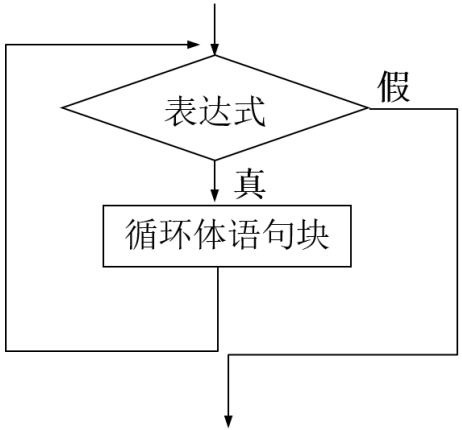
流程控制

流程控制

- 三种基本结构：顺序、条件（选择、分支）、循环



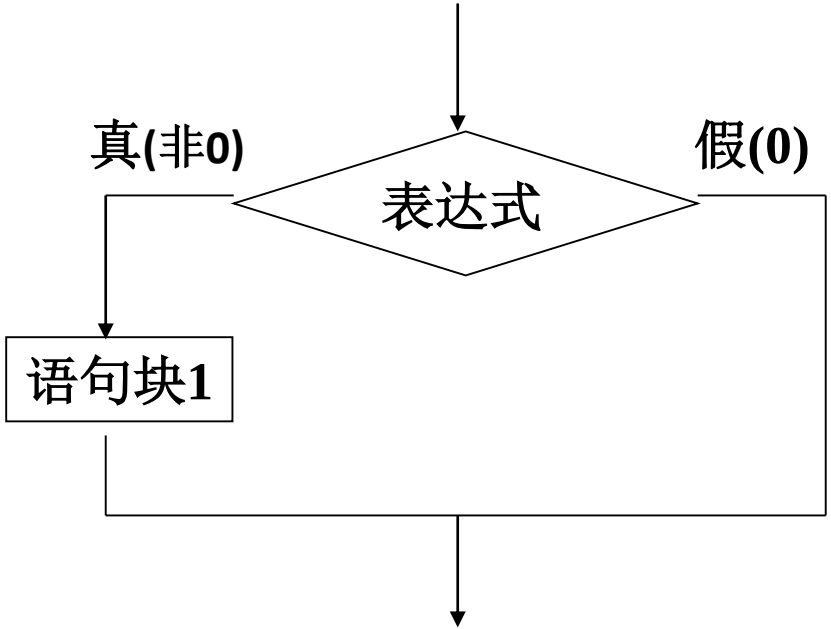




条件语句

条件语句

```
if 条件表达式:
    语句块1
```



- 代码块是通过缩进来指示的
- 缩进表示一个代码块的开始，逆缩进则表示一个代码块的结束
- 声明以冒号：字符结束，并且开启一个缩进级别

条件语句


```
print("请输入体重（kg）：")
weight = float(input())

if weight > 90:
    print("胖子，该减肥啦！！！")
```

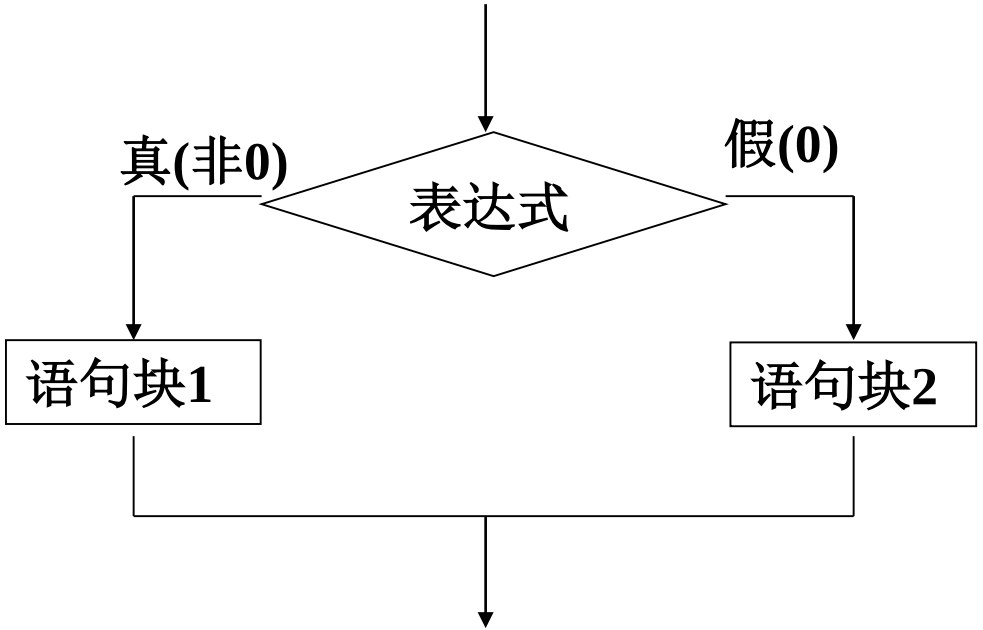
```
请输入体重（kg）：
100
胖子，该减肥啦！！！
```

```
print("请输入体重（kg）：")
weight = float(input())

if weight > 90:
    print("胖子，该减肥啦！！！")
```

```
请输入体重（kg）：
70
```

条件语句

```
if 条件表达式:
    语句块1
else:
    语句块2
```



条件语句


```
print("请输入体重（kg）：")
weight = float(input())

if weight > 90:
    print("胖子，该减肥啦！！！")
else:
    print("小样，身材保持的不错嘛！")
```

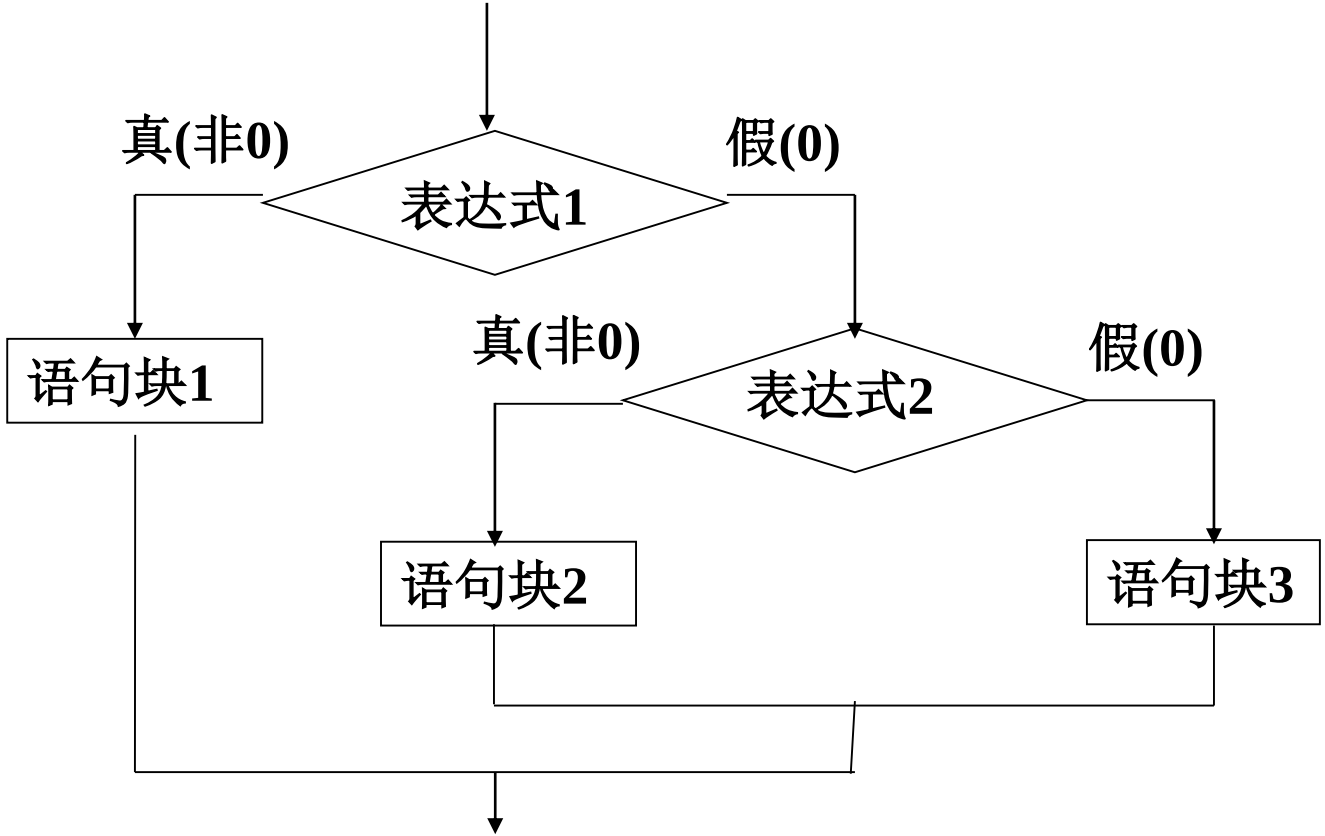
```
请输入体重（kg）：
100
胖子，该减肥啦！！！
```

```
print("请输入体重（kg）：")
weight = float(input())

if weight > 90:
    print("胖子，该减肥啦！！！")
else:
    print("小样，身材保持的不错嘛！")
```

```
请输入体重（kg）：
70
小样，身材保持的不错嘛！
```

条件语句

```
if  条件表达式1:
        语句块1
elif 条件表达式2 :
        语句块2
else:
        语句块3
```



条件语句：计算BMI指数

In [145]:
```
# 计算BMI指数

print("请输入体重（kg）：")
weight = float(input())
print("请输入身高（m）：")
height = float(input())

BMI = weight / height**2

if BMI < 20:
    print("你的BMI指数是%.2f,太轻了哟！" %BMI)
elif BMI > 25:
    print("你的BMI指数是%.2f,太重了哟！" %BMI)
else:
    print("你的BMI指数是%.2f,非常正常，请保持！" %BMI)
```

```
请输入体重（kg）：
72
请输入身高（m）：
1.76
你的BMI指数是23.24,非常正常，请保持！
```

需要严格遵守缩进格式

In [26]:
```
# 计算BMI指数

print("请输入体重（kg）：")
weight = float(input())
print("请输入身高（m）：")
height = float(input())

BMI = weight / height**2

if BMI < 20:
    print("你的BMI指数是%.2f,太轻了哟！" %BMI)
  elif BMI > 25:  #这里没有严格遵守缩进格式，执行时会报错
    print("你的BMI指数是%.2f,太重了哟！" %BMI)
else:
    print("你的BMI指数是%.2f,非常正常，请保持！" %BMI)
```

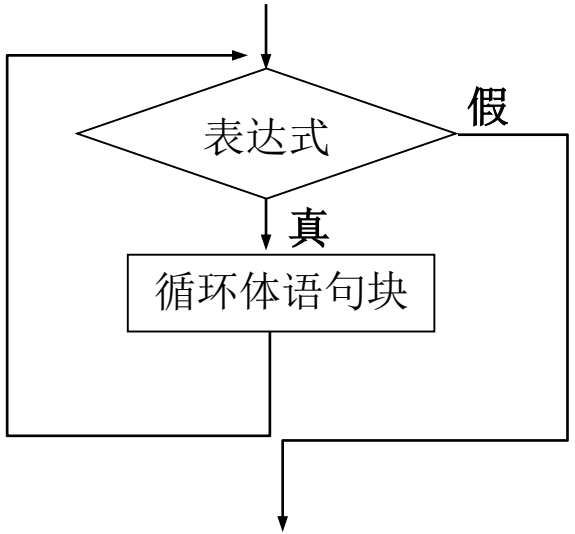
```
  File "<ipython-input-26-1bee6dde9c50>", line 12
    elif BMI > 25:  #这里没有严格遵守缩进格式，执行时会报错
                                                  ^
IndentationError: unindent does not match any outer indentation level
```

循环语句

循环语句

循环语句允许执行一条语句或语句块多次

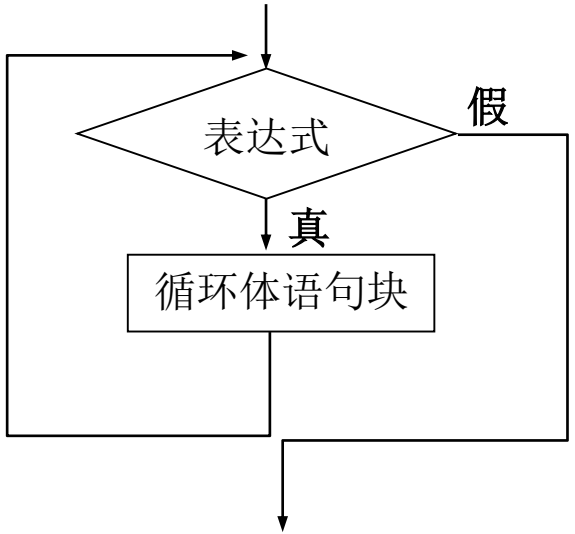
Python提供了while循环和for循环



while 循环语句

while 循环语句

while 条件表达式:
    语句块



统计6出现在2的100次方中的次数

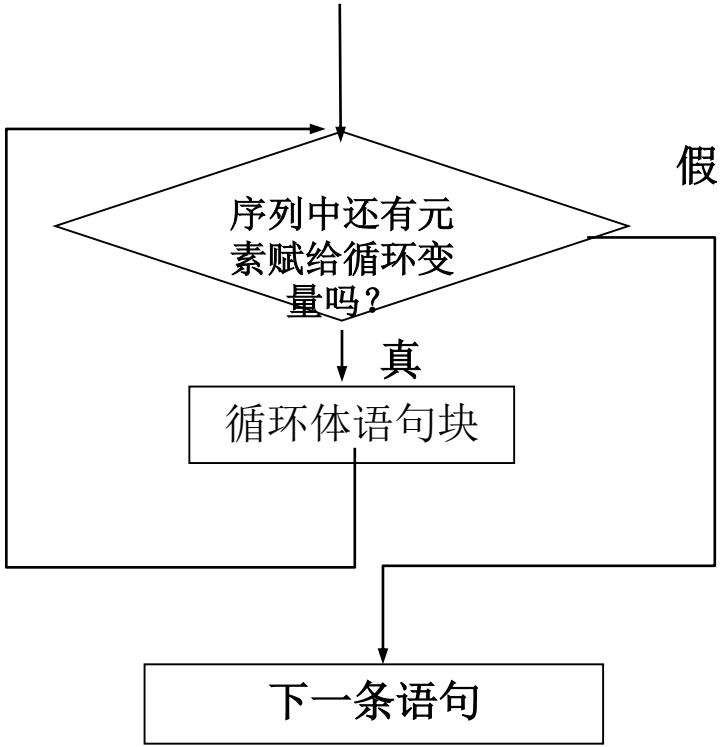
In [147]:
```
# 统计6出现在2的100次方中的次数:

num = 2**100
print(num)

count = 0

while num > 0:
    if num % 10 == 6:
        count = count + 1
    num = num // 10

print(count)
```

```
1267650600228229401496703205376
5
```

for 循环语句

for循环语句

for 循环变量 in 序列:
**　　语句块**



统计6出现在2的100次方中的次数

In [148]:
```
# 统计6出现在2的100次方中的次数:

num = 2**100
count = 0
for digit in str(num):
    if digit == "6":
        count = count + 1

print(count)
```

```
5
```

使用for和range来枚举列表中的元素

In [149]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [150]:
```
for x in range(1,10):
    print(x)                #结果不包含最后的10
```

```
1
2
3
4
5
6
7
8
9
```

列表推导式

列表推导式

- 列表推导式(List Comprehension)提供了一个创建和操作列表的有力工具
- 列表推导式由一 个表达式以及紧跟着这个表达式的for语句构成，for语句还可以跟0个或多个if或for语句

```
In [153]: lst1 = [1, 2, 3]
          lst2 = [3, 4, 5]
          [x * y for x in lst1 for y in lst2]

Out[153]: [3, 4, 5, 6, 8, 10, 9, 12, 15]
```

列表推导式

- 数值判断可以链接使用，例如 1<x<3 能够判断变量 x是否在1和3之间

```
In [154]: [x for x in lst1 if 4 > x > 1]
Out[154]: [2, 3]
```

循环嵌套（多重循环）

多重循环

In [45]:
```
# 9*9乘法口诀表

for i in range(1, 10):
    for j in range(1, i + 1):
        result = j * i
        print('%s x %s = %-5s' % (j, i, result), end='' )
    print()
```

```
1 x 1 = 1
1 x 2 = 2    2 x 2 = 4
1 x 3 = 3    2 x 3 = 6    3 x 3 = 9
1 x 4 = 4    2 x 4 = 8    3 x 4 = 12   4 x 4 = 16
1 x 5 = 5    2 x 5 = 10   3 x 5 = 15   4 x 5 = 20   5 x 5 = 25
1 x 6 = 6    2 x 6 = 12   3 x 6 = 18   4 x 6 = 24   5 x 6 = 30   6 x 6 = 36
1 x 7 = 7    2 x 7 = 14   3 x 7 = 21   4 x 7 = 28   5 x 7 = 35   6 x 7 = 42   7 x 7 = 49
1 x 8 = 8    2 x 8 = 16   3 x 8 = 24   4 x 8 = 32   5 x 8 = 40   6 x 8 = 48   7 x 8 = 56   8 x 8 = 64
1 x 9 = 9    2 x 9 = 18   3 x 9 = 27   4 x 9 = 36   5 x 9 = 45   6 x 9 = 54   7 x 9 = 63   8 x 9 = 72   9 x 9 = 81
```

break 语句

Break 语句

- break语句用在while和for循环中
- break语句用来终止循环语句，即循环条件没有False或者序列还没被完全递归完，也会停止执行循环语句

In [39]:
```
# 统计第1个9在2的100次方中出现的位置:

num = 2**100
pos = 0
for digit in str(num):
    pos = pos + 1
    if digit == "9":
        break
print("2**100 is: %d \nthe first posion of 9 is Pos.%d" %(num,pos))
```

```
2**100 is: 1267650600228229401496703205376
the first posion of 9 is Pos.16
```

Break 语句

如果在嵌套循环中，break语句将停止执行本层的循环

In [43]:
```
#求2到100之间的素数

i = 2
while(i < 100):
    flag = 0
    j = 2
    while(j <= (i/j)):
        if i%j==0:
            flag = 1
            break
        j = j + 1
    if (flag == 0) :
        print (i, " 是素数")
    i = i + 1
```

```
2  是素数
3  是素数
5  是素数
7  是素数
11  是素数
13  是素数
17  是素数
19  是素数
```

continue 语句

continue 语句

continue 语句用来跳过当前循环的剩余语句，然后继续进行下一轮循环

In [47]:
```
# 求在2的100次方中删除所有的9后的数字:

num = 2**100
without9 = ''
for digit in str(num):
    if digit == "9":
        continue
    without9 += digit
print("2**100 is: %d \nwithout 9 is :%s" %(num,without9))
```

```
2**100 is: 1267650600228229401496703205376
without 9 is :126765060022822401467032053 76
```

pass 语句

pass 语句

pass 语句是空语句，是为了保持程序结构的完整性，一般用做占位语句

In [48]:
```
# 求在2的100次方中删除所有的9后的数字:

num = 2**100
without9 = ''
for digit in str(num):
    if digit == "9":
        pass
    else:
        without9 += digit
print("2**100 is: %d \nwithout 9 is :%s" %(num,without9))
```

```
2**100 is: 1267650600228229401496703205376
without 9 is :126765060022822401467032053 76
```

函数

函数

- 函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段
- 函数能提高应用的模块性，和代码的重复利用率
- Python提供了许多内建函数，比如print()
- 开发者也可以自己创建函数，这被叫做用户自定义函数

函数定义语法：

```
def functionname( parameters ):
    "函数_文档字符串"
    function_suite
    return [expression]
```

函数

- 函数通过“def”关键字进行声明，后接函数标识符名称和圆括号()
- 任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数
- return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None

In [7]:
```
# 定义一个求 n!的函数
def fact(n):
    result=1
    for i in range(1,n+1):
        result=result*i
    return result
```

In [8]:
```
fact(5)
```

Out[8]: 120

函数

In [12]:
```
# 打印一张从0到9的阶层表，每2个一行

for i in range(0,9+1):
    item = fact(i)
    print("%i!=%-10d" % (i,item), end="")
    if i%2==1:
        print()
```

```
0!=1          1!=1
2!=2          3!=6
4!=24         5!=120
6!=720        7!=5040
8!=40320      9!=362880
```

函数

- 可选参数以集合的方式出现在函数声明中并紧跟着必选参数，可选参数可以在函数声明中被赋予一个默认值。已命名的参数需要赋值。
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明
- 函数可以返回一个元组（使用元组拆包可以有效返回多个值）

函数

In [151]:
```
# intp 和 stringp 是可选参数，它们有默认值
# 如果调用 fun_example 时只指定一个参数，那么 intp 缺省为 0 ，stringp 缺省为 A default string。
# 如果调用 fun_example 时指定了前面两个参数，stringp 仍缺省为 A default string。
# listp 是必备参数，因为它没有指定缺省值。
def fun_example(listp, intp=0, stringp="A default string"):
    listp.append("A new item")
    intp += 1
    return listp, intp, stringp
```

In [152]:
```
my_list = [1, 2, 3]
my_int = 10
print (fun_example(my_list, my_int))
print (my_list)
```

```
([1, 2, 3, 'A new item'], 11, 'A default string')
[1, 2, 3, 'A new item']
```

全局变量与局部变量

全局变量与局部变量

- 全局变量在函数之外声明
- 局部变量在函数内容声明
- 函数参数也是局部变量，不需要在函数内部重复定义！！！
- 全局变量可以不需要任何特殊的声明即能读取，但如果想要修改全局变量的值，就必须在函数开始之处用global关键字进行声明，否则Python会将此变量按照新的局部变量处理（请注意，这点很容易被坑）

全局变量

In [155]:
```
number = 5

def func1():
    print (number)

def func2():
    number = 3
    print (number)

def func3():
    global number
    number = 3
    print (number)
```

In [156]:
```
print(number)
func1()
print(number)
```

```
5
5
5
```

In [157]:
```
func2()
print(number)
```

```
3
5
```

In [158]:
```
func3()
print(number)
```

```
3
3
```

类 Class

类 Class

- 类（Class）用来描述具有相同的属性和方法的对象的集合
- 它定义了该集合中每个对象所共有的属性和方法
- 对象是类的实例

使用 class 语句来创建一个新类，class 之后为类的名称并以冒号结尾:

```
class ClassName:
        '类的帮助信息'   #类文档字符串
        class_suite      #类体
```

class_suite 由类成员，方法，数据属性组成

定义类


```python
# 定义一个叫做DeepLearner的类

class DeepLearner(object):
    'DeepLearner是深度学习者的类，这是有关这个类的帮助文档'

    # learnerCount 变量是一个类的属性，它的值将在这个类的所有实例之间共享。你可以在内部类或外部类使用 DeepLearner.learnerCount 访问
    learnerCount = 0

    #__init__()方法是一种特殊的方法，被称为类的构造函数或初始化方法，当创建了这个类的实例时就会调用该方法
    # 类的方法与普通的函数只有一个特别的区别—它们必须有一个额外的第一个参数名称，按照惯例它的名称是 self
    # self 代表类的实例，参数self 在定义类的方法时是必须要的，虽然在调用时不必传入相应的参数
    def __init__(self,name,schoolName):
        self.name = name
        self.schoolName = schoolName
        DeepLearner.learnerCount = DeepLearner.learnerCount + 1

    def getName(self):
        return self.name

    def getSchoolName(self):
        return self.schoolName

    def displayCount(self):
        print("Total DeepLearner count is %d" % DeepLearner.learnerCount)

    def displayLearner(self):
        print("Name: %s, School: %s" % (self.name,self.schoolName))
```

打印类帮助文档信息

In [2]:
```
# 打印类帮助文档信息

print(DeepLearner.__doc__)
```

```
DeepLearner是深度学习者的类，这是有关这个类的帮助文档
```

实例化和调用

In [3]:
```
# 实例化类其他编程语言中一般用关键字 new，但是在 Python 中并没有这个关键字，类的实例化类似函数调用方式
# 新建一个对象

newLearner1 = DeepLearner('giggle','Zhejiang University')
newLearner2 = DeepLearner('sherry','Zhejiang University of Technology')
```

In [4]:
```
# 使用点号 . 来访问对象的属性和方法

print(DeepLearner.learnerCount)
newLearner1.displayCount()
newLearner2.displayCount()

print(newLearner1.getName(), newLearner1.getSchoolName())
print(newLearner2.getName(), newLearner2.getSchoolName())

newLearner1.displayLearner()
newLearner2.displayLearner()
```

```
2
Total DeepLearner count is 2
Total DeepLearner count is 2
giggle Zhejiang University
sherry Zhejiang University of Technology
Name: giggle, School: Zhejiang University
Name: sherry, School: Zhejiang University of Technology
```

文件

文件

Python针对文件的处理有很多内建的函数库可以调用

In [5]:
```
# 写文件
with open("test.txt", "wt") as out_file:
    out_file.write("该文本会写入到文件中\n看到我了吧！")

# Read a file
with open("test.txt", "rt") as in_file:
    text = in_file.read()

print(text)
```

```
该文本会写入到文件中
看到我了吧！
```

异常

异常

Python中的异常由 try-except [exceptionname] 块处理

In [7]:
```
def except_function():
    try:
        # 故意除零
        10 / 0
    except ZeroDivisionError:
        print ("发生除零异常啦.")
    else:
        # 正常情况.
        print ("一切正常啦.")
        pass
    finally:
        # 无论是否发生异常都将执行最后的代码
        print ("finally必须被执行，不管有没有发生异常.")
```

In [8]:
```
except_function()
```

```
发生除零异常啦.
finally必须被执行，不管有没有发生异常.
```

导入外部库

异常

外部库可以使用 import [libname] 关键字来导入

可以用 from [libname] import [funcname] 来导入所需要的函数

In [9]:
```
import random
from time import clock
import numpy as np
import matplotlib.pyplot as plt

randomint = random.randint(1, 100)
print(randomint)
```

```
77
```

获取帮助信息

dir()

调用dir()来显示该对象的所有方法

```
In [10]: dir(1)

Out[10]: ['__abs__',
          '__add__',
          '__and__',
          '__bool__',
          '__ceil__',
          '__class__',
          '__delattr__',
          '__dir__',
          '__divmod__',
          '__doc__',
          '__eq__',
          '__float__',
          '__floor__',
          '__floordiv__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getnewargs__',
          '__gt__',
```

help()

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

调用help()会显示其文档

```
In [11]: help(int)
```

```
Help on class int in module builtins:

class int(object)
 |  int(x=0) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
 |
 |  If x is not a number or if base is given, then x must be a string,
 |  bytes, or bytearray instance representing an integer literal in the
 |  given base.  The literal can be preceded by '+' or '-' and be surrounded
 |  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
 |  Base 0 means to interpret the base from the string as an integer literal.
 |  >>> int('0b100', base=0)
 |  4
 |
 |  Methods defined here:
```

```
In [25]: a = 12345
         a.__neg__()

Out[25]: -12345
```