Python 开发基础教程

1 开发环境

Python 开发工具繁多，下面介绍两种常用的开发环境：

1.1 Anaconda

推荐 Anaconda ，自带基础开发环境及常用科学计算包，包括： Python 环境、pip 包管理工具、常用的库、配置好环境路径等等。

1.2 PyCharm

PyCharm 是 JetBrains 开发的 Python IDE。PyCharm 具备如下功能：调试、语法高亮、Project 管理、代码跳转、智能提示、自动完成、单元测试、版本控制……另外，PyCharm 还提供了一些很好的功能用于 Django 开发，同时支持 Google App Engine。

两种开发环境安装过程同普通工具，配置过程见配置手册。

2 基础语法

2.1基础知识

1）编码

为支持中文显示，Python 脚本文件都应在文件头标上 #-*- coding:utf-8 -*- 。设置编辑器，默认保存为 utf-8 格式。

2）注释

业界普遍认同 Python 的注释分为两种的概念，一种是由 # 开头的单行注释,用法及含义同其他语言；另一种是文档描述 docstrings（''' 具体说明内容 '''），用于说明如何使用这个包、模块、类、函数（方法），甚至包括使用示例和单元测试。实际使用中，推荐对每一个包、模块、类、函数（方法）写 docstrings，除非代码一目了然，非常简单。

3）缩进

Python 依赖缩进来确定代码块的层次，行首空白符主要有两种：tab 和空格，但严禁两者混用。**如果使用 tab 缩进，设定 tab 为 4 个空格**。

一般规范推荐使用 4 个空格的 tab 进行缩进。

4）空格

空格在 Python 代码中是有意义的，因为 Python 的语法依赖于缩进，在行首的空格称为前导空格，会导致错误。在这一节不讨论前导空格相关的内容，只讨论非前导空格。非前导空格在 Python 代码中没有意义，但适当地加入非前导空格可以增进代码的可读性。

1） 在二元算术、逻辑运算符前后加空格：如 a = b + c；

2） 在一元前缀运算符后不加空格，如 if !flg: pass；

3） “:”用在行尾时前后皆不加空格，如分枝、循环、函数和类定义语言；用在非行尾时两端加空格，如 dict 对象的定义 d = {‘key’ : ’value’}。

4） 括号（含圆括号、方括号和花括号）前后不加空格，如 do_something(arg1, arg2)，而不是 do_something(arg1, arg2)；

5） 逗号后面加一个空格，前面不加空格；

5）断行

以斜杠\表示续行：

```
very_very_very_long_variable_name = Edit(parent, \
width, \
font, \
color)
```

2.2 入门程序

1) 示例程序 1

```
#这是第一个 Python 示例程序
print("Hello, Python!")
```

2）示例程序 2

```
#这是第二个 Python 示例程序
a=3;b=65.1;c="Test";d=14
print(a,b,c,d/a)
print(a+b,str(b)+c)
```

3）示例程序 3

```
#这是第三个 Python 示例程序
#有 1/2/3/4 四个数字，可以组成多少个互不相同且无重复数字的三位数？各是多少？
count = 0
for i in range(1,5):
    for j in range(1,5):
        for k in range(1,5):
            if i!=j and i!=k and j!=k:
                print(i*100+j*10+k)
```

```
                count += 1
print(count)
```

2.2数据类型

命名规则：第一个字符必须是字母或者下划线“_”；剩下的部分可以是字母、下划线“_”或数字（0-9）；变量名称对大小写敏感，myname 和 myName 不是同一个变量。

1）示例程序 1

```
#数据类型示例程序 1—简单数据类型
name = 'Crossin'
myVar = 123
price = 5.99
visible = True
print(name,myVar,price,visible);
myVar = "中文内容"
price = 5+2j;
visible = 3.14159
print(name,myVar,price,visible);
```

2）示例程序 2

```
#数据类型示例程序 2—列表类型
TestList = [10001,'中文内容','En Info',[1,2,3,4,5,6]]
print(TestList)
print("当前列表长度为：%d \n" % len(TestList))
for info in TestList:
    print(info) ;

TestList.append(1.234) ;
TestList.append("最后一个元素")

print(TestList)
print("当前列表长度为：%d \n" % len(TestList))

TestList.append("string")
print(TestList)
```

备注：与 list 类似的类型有 tuple 也称元组或切片，定义规则同 list，除了把中括号[]换成小括号(),定义后则不能改变内容，可以理解为常量 list。示例：

```
tup1 = ('Google', 'Runoob', 1997, 2000)
```

3）示例程序 3

```
#数据类型示例程序 3—字典类型
#新建一个字典
testDict={'time':'时间','machine':'机器','time machine':'时间机器'}
print(testDict['time'])
#do it in another way
#另一种构造字典的方式
newDict ={}
newDict['stuff']='start'
print(newDict)
#aperson in Python from thebook
#字典的属性可以是字典
rec= {'name': {'first': 'Bob','last': 'Smith'},'job': ['dev', 'mgr'],'age': 40.5}
print(rec)
SortRec = sorted(rec)
print(SortRec)
rec= 0 #memory freed whenthe last reference is gone
#sort adictionary
#字典排序
D ={ 'b':1,'a': 2, 'c':3}
if not D.__contains__('d'):
    print('没有关键字 d')
print(D)
for i in sorted(D):
    print(i)
    print(D[i])
```

2.3程序控制

控制语句包括分支控制和循环控制两类：

1）示例程序 1

```
#程序控制示例程序 1—简单组合
# Fibonacci series: 斐波纳契数列
#  两个元素的总和确定了下一个数
a, b = 0, 1
while b < 10:
    print(b)
    a, b = b, a+b
else:
    print("over")
```

2）示例程序 2

```
#程序控制示例程序 2—分支组合
number = 7
guess = -1
print("数字猜谜游戏!")
while guess != number:
    guess = int(input("请输入你猜的数字："))

    if guess == number:
        print("恭喜，猜对了！")
    elif guess < number:
        print("猜的小了...")
    else:
        print("猜的大了...")
```

3）示例程序 3

```
#程序控制示例程序 3—循环控制
#打印三角形
for i in range(0, 5):
    for j in range(0, i+1):
        print('*', end='')
    print()

#奇偶性判断
numbers = [12,37,5,42,8,3,24,13,28]
even = []
odd = []
print(numbers)
while len(numbers) > 0:
    curValue = numbers.pop() ;
    if( curValue % 2 == 0 ):
        even.append(curValue)
    else:
        odd.append(curValue)

print(even)
print(odd)
```

4）示例程序 4

```
#程序控制示例程序 4—循环嵌套
#判断是否是素数
i = 2
```

```
while(i < 20):
    j = 2
    while(j <= (i/j)):
        if not(i%j):
            break
        j = j + 1
    if (j > i/j) :
        print(i, " 是素数")
    i = i + 1

print("处理完成!")
```

2.4文件操作

1）基本概念

文件操作：
os.mknod("test.txt") 创建空文件
fp = open("test.txt",w) 直接打开一个文件，如果文件不存在则创建文件
关于 open 模式：
w 以写方式打开，
a 以追加模式打开 (从 EOF 开始, 必要时创建新文件)
r+ 以读写模式打开
w+ 以读写模式打开 (参见 w)
a+ 以读写模式打开 (参见 a)
rb 以二进制读模式打开
wb 以二进制写模式打开 (参见 w)
ab 以二进制追加模式打开 (参见 a)
rb+ 以二进制读写模式打开 (参见 r+)
wb+ 以二进制读写模式打开 (参见 w+)
ab+ 以二进制读写模式打开 (参见 a+)

2）示例程序 1

```
#文件操作示例程序 1—读写文件
fileHandle = open('test1.py' )
fileList = fileHandle.readlines()
for fileLine in fileList:
    print(fileLine)
fileHandle.close()

fileHandle = open ( 'test.txt', 'w' )
fileHandle.write ( 'This is a test.\nReally, it is.' )
fileHandle.close()
```

print("文件操作结束")

2）示例程序 2

```
#文件操作示例程序 2—遍历目录里面文件信息
import os
import os.path
#  指明被遍历的文件夹
rootdir = os.getcwd()

for parent,dirnames,filenames in os.walk(rootdir):
#三个参数：分别返回 1.父目录  2.所有文件夹名字（不含路径）  3.所有文件名字
    for dirname in   dirnames:                                  #输出文件夹信息
        print("parent is:"+parent)
        print("dirname is" + dirname)

    for filename in filenames:                                  #输出文件信息
        print("parent is:" + parent)
        print("filename is:" + filename)
        print("the full name of the file is:" + os.path.join(parent,filename)) #输出文件路径信息
print("遍历结束")
```

2.5基本语法总结

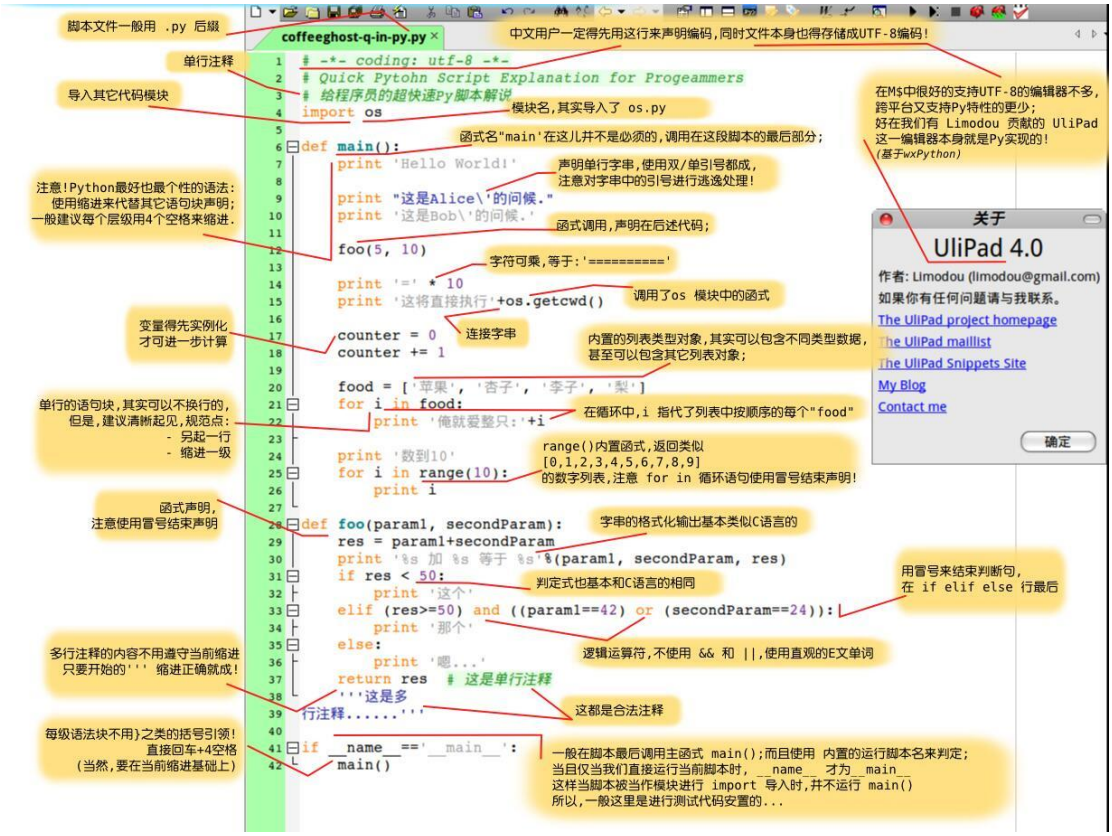

图 1 Python 语法图

2.6 程序练习

1）统计文本文件中各字母（区分大小写）的出现频率
2）试卷生成器：从网上资源构建任意年级/科目题库，从题库文件中随机挑选出不同的题目生成一份试卷，从对应的答案文件中抽取对应的答案构成标准答案。
3）文件批量处理：程序批量处理一些文件，将根据用户提供的过滤器为它们重命名。比如整理手机图片数据，用户输入的过滤器为 myimage###.jpg，那么会生成至少包含 3 位数的文件名，如 myimage001.jpg、myimage145.jpg 等，数字次序为处理次序。更进一步，imgDate-###.jpg 表示为 img20180516-001.jpg，同时不同类型的文件单独排序编码。
备注：1 必做，2/3 任选一个。

3 高级语法

3.1 函数

1）函数基本概念

函数用 def 语句定义，格式如下：
def <name>(<parameters>):
<body>
说明：函数名<name>：任何有效的 Python 标识符
参数列表<parameters>：调用函数时传递给函数的值（个数 >= 0，多个参数用”,”分隔）
形式参数（形参）：定义函数时给函数传递的参数形参只在函数内部有效
实际参数（实参）：调用函数时给函数传递的参数（可以是实数，变量，函数；必须要初始化）
函数体<body>：函数被调用时执行的代码，由一至多个语句组成
函数调用的一般形式：<name>(<parameters>)
函数返回：1）return 语句 — 结束函数调用并返回结果；2）return 语句是可选的，可以出现在函数体的任意位置；3）若无 return 语句，函数结束后将控制权返回给调用方
函数接口 — 返回值和参数
函数传递信息的主要途径：1）通过函数返回值的方式传递信息；2）通过参数传递信息
注意：所有参数（自变量）在 Python 里都是按**引用**传递。即：如果在函数里修改了参数，那么在调用这个函数的函数里，原始的参数也被改变了。

2）示例程序 1

```
#函数示例程序 1
def changeInfo( mylist ):
    "修改传入的列表"
    mylist.append([1,2,3,4]);
```

```
    print("函数内取值: ", mylist)
    return

# 调用 changeInfo 函数
mylist = [10,20,30];
changeme( mylist );
print("函数外取值: ", mylist)
```

3）示例程序 2

```
#函数示例程序 2
#打印《生日快乐歌》歌词
def happy():
    print("Happy birthday to you!")
def sing(person):
    happy()
    happy()
    print("Happy birthday to", person + "!")
def main():
    sing("Mike")
    print()
main()
```

4）示例程序 3

```
#设计两点距离函数，求三角形周长
import math
def square(x):
    return x*x
def distance(x1,y1,x2,y2):
    dist = math.sqrt(square(x1 - x2) + square(y1 - y2))
    return dist
def isTriangle(x1,y1,x2,y2,x3,y3):
    flag = ((x1 - x2) * (y3 - y2) - (x3 - x2) * (y1 - y2)) != 0
    return flag
def main():
    print("Please enter (x,y) of three points in turn:")
    #获取用户输入的三个坐标点
    x1, y1 = eval(input("Point1: (x, y) = "))
    x2, y2 = eval(input("Point2: (x, y) = "))
    x3,   y3 = eval(input("Point3: (x, y) = "))
```

```
    # 判断三个点是否构成三角形
    if (isTriangle(x1,y1,x2,y2,x3,y3)):
        # 计算三角形周长
        perim = distance(x1,y1,x2,y2) + distance(x2,y2,x3,y3) + distance(x1,y1,x3,y3)
        print("The perimeter of the triangle is : {0:0.2f}".format(perim))
    else:
        print("Kidding me? This is not a triangle!")
main()
```

3.2 模块

模块可以更有效、更有逻辑地组织 Python 代码段。把相关的代码分配到一个模块里能使代码更好用，更易懂。模块也是 Python 对象，具有随机的名字属性用来绑定或引用。简单地说，模块就是一个保存了 Python 代码的文件。模块能定义函数，类和变量。模块里也能包含可执行的代码。

1）基本概念

import 语句：import module1[, module2[,... moduleN]

当解释器遇到 import 语句，如果模块在当前的搜索路径就会被导入。

```
#suppot.py
def print_func( par ):
print "Hello : ", par
return
```

搜索路径是一个解释器会先进行搜索的所有目录的列表。如想要导入模块 hello.py，需要把命令放在脚本的顶端：

```
# 导入模块
import support
# 现在可以调用模块里包含的函数了
support.print_func("Zara")
```

备注：一个模块只会被导入一次，不管执行了多少次 import。

From…import 语句从模块中导入一个指定的部分到当前命名空间中：

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

from modname import *把一个模块的所有内容全都导入到当前的命名空间。

dir()函数返回一个列表，容纳了在模块里定义的所有模块，变量和函数：

```
import math
content = dir(math)
print(content)
```

2）示例程序

```
# quadratic.py
# 计算二次方程的实数根程序
import math
def main():
    print("This program finds the real solutions to a quadratic\n")
    a,b,c = eval(input("Please enter the coefficients(a,b,c): "))
    delta = b*b - 4*a*c
    if delta >= 0:
        discRoot = math.sqrt(delta)
        root1 = (-b + discRoot) / (2*a)
        root2 = (-b - discRoot) / (2*a)
        print("\nThe solutions are:", root1, root2)
    else:
        print("The equation has no real roots!")
main()
```

3.3 正则表达式

1）基本概念

正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

正则表达式的大致匹配过程是：

1.依次拿出表达式和文本中的字符比较，

2.如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。

3.如果表达式中有量词或边界，这个过程会稍微有一些不同。

Python 的 re 模块提供了对正则表达式的支持。主要用到的方法列举如下：

```
#返回 pattern 对象
re.compile(string[,flag])
#以下为匹配所用函数
re.match(pattern, string[, flags])
re.search(pattern, string[, flags])
re.split(pattern, string[, maxsplit])
re.findall(pattern, string[, flags])
re.finditer(pattern, string[, flags])
re.sub(pattern, repl, string[, count])
re.subn(pattern, repl, string[, count])
```

pattern 可以理解为一个匹配模式，需要利用 re.compile 方法获得这个匹配模式。

参数 flag 是匹配模式，取值可以使用按位或运算符'|'表示同时生效，比如 re.I | re.M。
可选值有：
? re.I(全拼：IGNORECASE): 忽略大小写（括号内是完整写法，下同）
? re.M(全拼：MULTILINE): 多行模式，改变'^'和'$'的行为（参见上图）
? re.S(全拼：DOTALL): 点任意匹配模式，改变'.'的行为
? re.L(全拼：LOCALE): 使预定字符类 \w \W \b \B \s \S 取决于当前区域设定
? re.U(全拼：UNICODE): 使预定字符类 \w \W \b \B \s \S \d \D 取决于 unicode 定义的字符属性
? re.X(全拼：VERBOSE): 详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释

2）示例程序 1

```
# -*- coding: utf-8 -*-
#导入 re 模块
import re

# 将正则表达式编译成 Pattern 对象，注意 hello 前面的 r 的意思表示“原生字符串”
pattern = re.compile(r'hello')

# 使用 re.match 匹配文本，获得匹配结果，无法匹配时将返回 None
result1 = re.match(pattern,'hello')
result2 = re.match(pattern,'helloo CQC!')
result3 = re.match(pattern,'helo CQC!')
result4 = re.match(pattern,'hello CQC!')

#如果 1 匹配成功
if result1:
    # 使用 Match 获得分组信息
    print(result1.group())
else:
    print('1 匹配失败！')

#如果 2 匹配成功
if result2:
    # 使用 Match 获得分组信息
    print(result2.group())
else:
    print('2 匹配失败！')

#如果 3 匹配成功
if result3:
    # 使用 Match 获得分组信息
    print(result3.group())
else:
```

```
    print('3 匹配失败!')

#如果 4 匹配成功
if result4:
    # 使用 Match 获得分组信息
    print(result4.group())
else:
    print('4 匹配失败!')
```

3）示例程序 2

正则库中 Match 对象是一次匹配的结果，包含了很多关于此次匹配的信息，可以使用 Match 提供的可读属性或方法来获取这些信息。

属性：

1.string: 匹配时使用的文本。

2.re: 匹配时使用的 Pattern 对象。

3.pos: 文本中正则表达式开始搜索的索引。值与 Pattern.match()和 Pattern.seach()方法的同名参数相同。

4.endpos: 文本中正则表达式结束搜索的索引。值与 Pattern.match()和 Pattern.seach()方法的同名参数相同。

5.lastindex: 最后一个被捕获的分组在文本中的索引。如果没有被捕获的分组，将为 None。

6.lastgroup: 最后一个被捕获的分组的别名。如果这个分组没有别名或者没有被捕获的分组，将为 None。

方法：

1.group([group1, …]):

获得一个或多个分组截获的字符串；指定多个参数时将以元组形式返回。group1 可以使用编号也可以使用别名；编号 0 代表整个匹配的子串；不填写参数时，返回 group(0)；没有截获字符串的组返回 None；截获了多次的组返回最后一次截获的子串。

2.groups([default]):

以元组形式返回全部分组截获的字符串。相当于调用 group(1,2,…last)。default 表示没有截获字符串的组以这个值替代，默认为 None。

3.groupdict([default]):

返回以有别名的组的别名为键、以该组截获的子串为值的字典，没有别名的组不包含在

内。default 含义同上。

4.start([group]):

返回指定的组截获的子串在 string 中的起始索引（子串第一个字符的索引）。group 默认值为 0。

5.end([group]):

返回指定的组截获的子串在 string 中的结束索引（子串最后一个字符的索引+1）。group 默认值为 0。

6.span([group]):

返回(start(group), end(group))。

7.expand(template):

将匹配到的分组代入 template 中然后返回。template 中可以使用\id 或\g、\g 引用分组，但不能使用编号 0。\id 与\g 是等价的；但\10 将被认为是第 10 个分组，如果你想表达\1 之后是字符'0'，只能使用\g0。

下面我们用一个例子来体会一下

```
#一个简单的 match 实例

import re
# 匹配如下内容：单词+空格+单词+任意字符
m = re.match(r'(\w+) (\w+)(?P.*)', 'hello world!')

print("m.string:", m.string)
print("m.re:", m.re)
print("m.pos:", m.pos)
print("m.endpos:", m.endpos)
print("m.lastindex:", m.lastindex)
print("m.lastgroup:", m.lastgroup)
print("m.group():", m.group())
print("m.group(1,2):", m.group(1, 2))
print("m.groups():", m.groups())
print("m.groupdict():", m.groupdict())
print("m.start(2):", m.start(2))
print("m.end(2):", m.end(2))
print("m.span(2):", m.span(2))
print(r"m.expand(r'\g \g\g'):", m.expand(r'\2 \1\3'))
```

4）其他函数

re.search(pattern, string[, flags]) search 方法与 match 方法极其类似，区别在于 match()函数只检测 re 是不是在 string 的开始位置匹配，search()会扫描整个 string 查找匹配，match（）只有在 0 位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，match()就返

回 None。同样，search 方法的返回对象同样 match()返回对象的方法和属性。

re.split(pattern, string[, maxsplit])

按照能够匹配的子串将 string 分割后返回列表。maxsplit 用于指定最大分割次数，不指定将全部分割。

re.findall(pattern, string[, flags]) 搜索 `string`，以列表形式返回全部能匹配的子串。

re.finditer(pattern, string[, flags])

搜索 string，返回一个顺序访问每一个匹配结果（Match 对象）的迭代器。

re.sub(pattern, repl, string[, count])

使用 repl 替换 string 中每一个匹配的子串后返回替换后的字符串。

当 repl 是一个字符串时，可以使用\id 或\g、\g 引用分组，但不能使用编号 0。

当 repl 是一个方法时，这个方法应当只接受一个参数（Match 对象），并返回一个字符串用于替换（返回的字符串中不能再引用分组）。

count 用于指定最多替换次数，不指定时全部替换。

re.subn(pattern, repl, string[, count])

返回 (sub(repl, string[, count]), 替换次数)。

5）另一种调用方法

Python Re 模块的另一种使用方式

上面介绍了 7 个工具方法，例如 match，search 等等，不过调用方式都是 re.match，re.search 的方式，其实还有另外一种调用方式，可以通过 pattern.match，pattern.search 调用，这样调用便不用将 pattern 作为第一个参数传入了，怎样调用皆可。

3.3 网络操作

Socket 是网络编程的一个抽象概念。通常我们用一个 Socket 表示“打开了一个网络链接”，而打开一个 Socket 需要知道目标计算机的 IP 地址和端口号，再指定协议类型即可。

1）TCP 开发

（1）客户端

大多数连接都是可靠的 TCP 连接。创建 TCP 连接时，主动发起连接的叫客户端，被动响应连接的叫服务器。举个例子，当在浏览器中访问新浪时，我们自己的计算机就是客户端，浏览器会主动向新浪的服务器发起连接。如果一切顺利，新浪的服务器接受了连接，一个 TCP 连接就建立起来了，后面的通信就是发送网页内容了。所以，要创建一个基于 TCP 连接的 Socket，可以这样做：

```
# 导入 socket 库:
import socket
```

```
# 创建一个 socket:
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 建立连接:
s.connect(('www.sina.com.cn', 80))
```

创建 Socket 时，AF_INET 指定使用 IPv4 协议，如果要用更先进的 IPv6，就指定为 AF_INET6。SOCK_STREAM 指定使用面向流的 TCP 协议，这样，一个 Socket 对象就创建成功，但是还没有建立连接。客户端要主动发起 TCP 连接，必须知道服务器的 IP 地址和端口号。新浪网站的 IP 地址可以用域名 www.sina.com.cn 自动转换到 IP 地址，但是怎么知道新浪服务器的端口号呢？

答案是作为服务器，提供什么样的服务，端口号就必须固定下来。由于我们想要访问网页，因此新浪提供网页服务的服务器必须把端口号固定在 80 端口，因为 80 端口是 Web 服务的标准端口。其他服务都有对应的标准端口号，例如 SMTP 服务是 25 端口，FTP 服务是 21 端口，等等。端口号小于 1024 的是 Internet 标准服务的端口，端口号大于 1024 的，可以任意使用。

因此，我们连接新浪服务器的代码如下：

```
s.connect(('www.sina.com.cn', 80))
```

注意参数是一个 tuple，包含地址和端口号。

建立 TCP 连接后，我们就可以向新浪服务器发送请求，要求返回首页的内容：

```
# 发送数据:
s.send('GET / HTTP/1.1\r\nHost: www.sina.com.cn\r\nConnection: close\r\n\r\n')
```

TCP 连接创建的是双向通道，双方都可以同时给对方发数据。但是谁先发谁后发，怎么协调，要根据具体的协议来决定。例如，HTTP 协议规定客户端必须先发请求给服务器，服务器收到后才发数据给客户端。

发送的文本格式必须符合 HTTP 标准，如果格式没问题，接下来就可以接收新浪服务器返回的数据了：

```
# 接收数据:
buffer = []
while True:
    # 每次最多接收 1k 字节:
    d = s.recv(1024)
    if d:
        buffer.append(d)
    else:
        break
data = ''.join(buffer)
```

接收数据时，调用 recv(max)方法，一次最多接收指定的字节数，因此，在一个 while 循环中反复接收，直到 recv()返回空数据，表示接收完毕，退出循环。

当我们接收完数据后，调用 close()方法关闭 Socket，这样，一次完整的网络通信就结束了：

```
# 关闭连接:
s.close()
```

接收到的数据包括 HTTP 头和网页本身，我们只需要把 HTTP 头和网页分离一下，把 HTTP 头打印出来，网页内容保存到文件：

```
header, html = data.split('\r\n\r\n', 1)
```

```
print(header)
# 把接收的数据写入文件:
with open('sina.html', 'wb') as f:
    f.write(html)
```

现在，只需要在浏览器中打开这个 sina.html 文件，就可以看到新浪的首页了。

（2）服务器

和客户端编程相比，服务器编程就要复杂一些。

服务器进程首先要绑定一个端口并监听来自其他客户端的连接。如果某个客户端连接过来了，服务器就与该客户端建立 Socket 连接，随后的通信就靠这个 Socket 连接了。

所以，服务器会打开固定端口（比如 80）监听，每来一个客户端连接，就创建该 Socket 连接。由于服务器会有大量来自客户端的连接，所以，服务器要能够区分一个 Socket 连接是和哪个客户端绑定的。一个 Socket 依赖 4 项：服务器地址、服务器端口、客户端地址、客户端端口来唯一确定一个 Socket。

但是服务器还需要同时响应多个客户端的请求，所以，每个连接都需要一个新的进程或者新的线程来处理，否则，服务器一次就只能服务一个客户端了。

我们来编写一个简单的服务器程序，它接收客户端连接，把客户端发过来的字符串加上 Hello 再发回去。

首先，创建一个基于 IPv4 和 TCP 协议的 Socket：

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

然后，我们要绑定监听的地址和端口。服务器可能有多块网卡，可以绑定到某一块网卡的 IP 地址上，也可以用 0.0.0.0 绑定到所有的网络地址，还可以用 127.0.0.1 绑定到本机地址。127.0.0.1 是一个特殊的 IP 地址，表示本机地址，如果绑定到这个地址，客户端必须同时在本机运行才能连接，也就是说，外部的计算机无法连接进来。

端口号需要预先指定。因为我们写的这个服务不是标准服务，所以用 9999 这个端口号。请注意，小于 1024 的端口号必须要有管理员权限才能绑定：

```
# 监听端口:
s.bind(('127.0.0.1', 9999))
```

紧接着，调用 listen()方法开始监听端口，传入的参数指定等待连接的最大数量：

```
s.listen(5)
print 'Waiting for connection...'
```

接下来，服务器程序通过一个永久循环来接受来自客户端的连接，accept()会等待并返回一个客户端的连接:

```
while True:
    # 接受一个新连接:
    sock, addr = s.accept()
    # 创建新线程来处理 TCP 连接:
    t = threading.Thread(target=tcplink, args=(sock, addr))
    t.start()
```

每个连接都必须创建新线程（或进程）来处理，否则，单线程在处理连接的过程中，无法接受其他客户端的连接：

```
def tcplink(sock, addr):
    print 'Accept new connection from %s:%s...' % addr
    sock.send('Welcome!')
    while True:
```

```
            data = sock.recv(1024)
            time.sleep(1)
            if data == 'exit' or not data:
                break
            sock.send('Hello, %s!' % data)
        sock.close()
        print 'Connection from %s:%s closed.' % addr
```

连接建立后，服务器首先发一条欢迎消息，然后等待客户端数据，并加上 Hello 再发送给客户端。如果客户端发送了 exit 字符串，就直接关闭连接。

要测试这个服务器程序，我们还需要编写一个客户端程序：

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 建立连接:
s.connect(('127.0.0.1', 9999))
# 接收欢迎消息:
print s.recv(1024)
for data in ['Michael', 'Tracy', 'Sarah']:
    # 发送数据:
    s.send(data)
    print s.recv(1024)
s.send('exit')
s.close()
```

我们需要打开两个命令行窗口，一个运行服务器程序，另一个运行客户端程序，就可以看到效果了：

2）UDP 开发

TCP 是建立可靠连接，并且通信双方都可以以流的形式发送数据。相对 TCP，UDP 则是面向无连接的协议。使用 UDP 协议时，不需要建立连接，只需要知道对方的 IP 地址和端口号，就可以直接发数据包。但是，能不能到达就不知道了。虽然用 UDP 传输数据不可靠，但它的优点是和 TCP 比，速度快，对于不要求可靠到达的数据，就可以使用 UDP 协议。

和 TCP 类似，使用 UDP 的通信双方也分为客户端和服务器。服务器首先需要绑定端口：

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# 绑定端口:
s.bind(('127.0.0.1', 9999))
```

创建 Socket 时，SOCK_DGRAM 指定了这个 Socket 的类型是 UDP。绑定端口和 TCP 一样，但是不需要调用 listen()方法，而是直接接收来自任何客户端的数据：

```
Print('Bind UDP on 9999...')
while True:
    # 接收数据:
    data, addr = s.recvfrom(1024)
    print('Received from %s:%s.' % addr)
    s.sendto('Hello, %s!' % data, addr)
```

recvfrom()方法返回数据和客户端的地址与端口，这样，服务器收到数据后，直接调用 sendto()就可以把数据用 UDP 发给客户端。

注意这里省掉了多线程，因为这个例子很简单。

客户端使用 UDP 时，首先仍然创建基于 UDP 的 Socket，然后，不需要调用 connect()，直接通过 sendto()给服务器发数据：

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
for data in ['Michael', 'Tracy', 'Sarah']:
    # 发送数据:
    s.sendto(data, ('127.0.0.1', 9999))
    # 接收数据:
    print(s.recv(1024))
s.close()
```

从服务器接收数据仍然调用 recv()方法。

3）邮件操作

（1）发送邮件

简单邮件传输协议(SMTP)是一种协议，用于在邮件服务器之间发送电子邮件和路由电子邮件。python 提供 smtplib 模块，该模块定义了一个 SMTP 客户端会话对象，可用于使用 SMTP 或 ESMTP 侦听器守护程序向任何互联网机器发送邮件。

```
import smtplib
smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
Python
```

这里是上面语法的参数细节 -

host - 这是运行 SMTP 服务器的主机。可以指定主机的 IP 地址或类似 yiibai.com 的域名。这是一个可选参数。

port - 如果提供主机参数，则需要指定 SMTP 服务器正在侦听的端口。通常这个端口默认值是：25。

local_hostname - 如果 SMTP 服务器在本地计算机上运行，那么可以只指定 localhost 选项。

SMTP 对象有一个 sendmail 的实例方法，该方法通常用于执行邮件发送的工作。它需要三个参数 -

sender - 具有发件人地址的字符串。

receivers - 字符串列表，每个收件人一个。

message - 作为格式如在各种 RFC 中指定的字符串。

1. 示例：使用 Python 发送纯文本电子邮件

以下是使用 Python 脚本发送一封电子邮件的简单方法

```
#!/usr/bin/python3

import smtplib

sender = 'from@fromdomain.com'

receivers = ['to@todomain.com']

message = """From: From Person <from@fromdomain.com>

To: To Person <to@todomain.com>
```

```
Subject: SMTP e-mail test

This is a test e-mail message.
"""

try:
   smtpObj = smtplib.SMTP('localhost')
   smtpObj.sendmail(sender, receivers, message)
   print "Successfully sent email"
except SMTPException:
   print "Error: unable to send email"
```

在这里，已经发送了一封基本的电子邮件，使用三重引号，请注意正确格式化标题。一封电子邮件需要一个 From，To 和一个 Subject 标题，与电子邮件的正文与空白行分开。

要发送邮件，使用 smtpObj 连接到本地机器上的 SMTP 服务器。 然后使用 sendmail 方法以及消息，从地址和目标地址作为参数(即使来自和地址在电子邮件本身内，这些并不总是用于路由邮件)。

如果没有在本地计算机上运行 SMTP 服务器，则可以使用 smtplib 客户端与远程 SMTP 服务器进行通信。除非您使用网络邮件服务(如 gmail 或 Yahoo! Mail)，否则您的电子邮件提供商必须向您提供可以提供的邮件服务器详细信息。以腾讯 QQ 邮箱为例，具体如下：

```
mail = smtplib.SMTP('smtp.qq.com', 587) # 端口 465 或 587
```

2. 示例：使用 Python 发送 HTML 电子邮件

当使用 Python 发送邮件信息时，所有内容都被视为简单文本。 即使在短信中包含 HTML 标签，它也将显示为简单的文本，HTML 标签将不会根据 HTML 语法进行格式化。但是，Python 提供了将 HTML 消息作为 HTML 消息发送的选项。发送电子邮件时，可以指定一个 Mime 版本，内容类型和发送 HTML 电子邮件的字符集。

以下是将 HTML 内容作为电子邮件发送的示例 -

```
#!/usr/bin/python3

import smtplib

message = """From: From Person <from@fromdomain.com>

To: To Person <to@todomain.com>

MIME-Version: 1.0

Content-type: text/html

Subject: SMTP HTML e-mail test

This is an e-mail message to be sent in HTML format

<b>This is HTML message.</b>

<h1>This is headline.</h1>

"""
```

```
try:
   smtpObj = smtplib.SMTP('localhost')
   smtpObj.sendmail(sender, receivers, message)
   print "Successfully sent email"
except SMTPException:
   print "Error: unable to send email"
```

3. 示例：发送附件作为电子邮件

要发送具有混合内容的电子邮件，需要将 Content-type 标题设置为 multipart / mixed。 然后，可以在边界内指定文本和附件部分。一个边界以两个连字符开始，后跟一个唯一的编号，不能出现在电子邮件的消息部分。 表示电子邮件最终部分的最后一个边界也必须以两个连字符结尾。所附的文件应该用包(“m”)功能编码，以便在传输之前具有基本的 64 编码。

（2）接收邮件

主要通过 POP3 下载邮件。

1. 接收邮件

POP3 协议本身很简单，以下面的代码为例，我们来获取最新的一封邮件内容：

```
import poplib

# 输入邮件地址, 口令和 POP3 服务器地址:
email = raw_input('Email: ')
password = raw_input('Password: ')
pop3_server = raw_input('POP3 server: ')

# 连接到 POP3 服务器:
server = poplib.POP3(pop3_server)
# 可以打开或关闭调试信息:
# server.set_debuglevel(1)
# 可选:打印 POP3 服务器的欢迎文字:
print(server.getwelcome())
# 身份认证:
server.user(email)
server.pass_(password)
# stat()返回邮件数量和占用空间:
print('Messages: %s. Size: %s' % server.stat())
# list()返回所有邮件的编号:
resp, mails, octets = server.list()
# 可以查看返回的列表类似['1 82923', '2 2184', ...]
print(mails)
# 获取最新一封邮件, 注意索引号从 1 开始:
index = len(mails)
resp, lines, octets = server.retr(index)
# lines 存储了邮件的原始文本的每一行,
```

```
# 可以获得整个邮件的原始文本:
msg_content = '\r\n'.join(lines)
# 稍后解析出邮件:
msg = Parser().parsestr(msg_content)
# 可以根据邮件索引号直接从服务器删除邮件:
# server.dele(index)
# 关闭连接:
server.quit()
```

用 POP3 获取邮件其实很简单，要获取所有邮件，只需要循环使用 retr()把每一封邮件内容拿到即可。真正麻烦的是把邮件的原始内容解析为可以阅读的邮件对象。

2. 解析邮件

解析邮件的过程和上一节构造邮件正好相反，因此，先导入必要的模块：

```
import email
from email.parser import Parser
from email.header import decode_header
from email.utils import parseaddr
```

只需要一行代码就可以把邮件内容解析为 Message 对象：

```
msg = Parser().parsestr(msg_content)
```

但是这个 Message 对象本身可能是一个 MIMEMultipart 对象，即包含嵌套的其他 MIMEBase 对象，嵌套可能还不止一层。

所以我们要递归地打印出 Message 对象的层次结构：

```
# indent 用于缩进显示:
def print_info(msg, indent=0):
    if indent == 0:
        # 邮件的 From, To, Subject 存在于根对象上:
        for header in ['From', 'To', 'Subject']:
            value = msg.get(header, '')
            if value:
                if header=='Subject':
                    # 需要解码 Subject 字符串:
                    value = decode_str(value)
                else:
                    # 需要解码 Email 地址:
                    hdr, addr = parseaddr(value)
                    name = decode_str(hdr)
                    value = u'%s <%s>' % (name, addr)
            print('%s%s: %s' % ('  ' * indent, header, value))
    if (msg.is_multipart()):
        # 如果邮件对象是一个 MIMEMultipart,
        # get_payload()返回 list，包含所有的子对象:
        parts = msg.get_payload()
        for n, part in enumerate(parts):
            print('%spart %s' % ('  ' * indent, n))
            print('%s--------------------' % ('  ' * indent))
```

```
            # 递归打印每一个子对象:
            print_info(part, indent + 1)
    else:
        # 邮件对象不是一个 MIMEMultipart,
        # 就根据 content_type 判断:
        content_type = msg.get_content_type()
        if content_type=='text/plain' or content_type=='text/html':
            # 纯文本或 HTML 内容:
            content = msg.get_payload(decode=True)
            # 要检测文本编码:
            charset = guess_charset(msg)
            if charset:
                content = content.decode(charset)
            print('%sText: %s' % ('  ' * indent, content + '...'))
        else:
            # 不是文本,作为附件处理:
            print('%sAttachment: %s' % ('  ' * indent, content_type))
```

邮件的 Subject 或者 Email 中包含的名字都是经过编码后的 str，要正常显示，就必须 decode：

```
def decode_str(s):
    value, charset = decode_header(s)[0]
    if charset:
        value = value.decode(charset)
    return value
```

decode_header()返回一个 list，因为像 Cc、Bcc 这样的字段可能包含多个邮件地址，所以解析出来的会有多个元素。上面的代码我们偷了个懒，只取了第一个元素。

文本邮件的内容也是 str，还需要检测编码，否则，非 UTF-8 编码的邮件都无法正常显示：

```
def guess_charset(msg):
    # 先从 msg 对象获取编码:
    charset = msg.get_charset()
    if charset is None:
        # 如果获取不到，再从 Content-Type 字段获取:
        content_type = msg.get('Content-Type', '').lower()
        pos = content_type.find('charset=')
        if pos >= 0:
            charset = content_type[pos + 8:].strip()
    return charset
```

把上面的代码整理好，我们就可以来试试收取一封邮件。先往自己的邮箱发一封邮件，然后用浏览器登录邮箱，看看邮件收到没，如果收到了，我们就来用 Python 程序把它收到本地：

运行程序，结果如下：

```
+OK Welcome to coremail Mail Pop3 Server (163coms[...])
Messages: 126. Size: 27228317

From: Test <xxxxxx@qq.com>
To: Python 爱好者  <xxxxxx@163.com>
Subject: 用 POP3 收取邮件
part 0
--------------------
  part 0
  --------------------
    Text: Python 可以使用 POP3 收取邮件.....
  part 1
  --------------------
    Text: Python 可以<a href="...">使用 POP3</a>收取邮件.....
part 1
--------------------
  Attachment: application/octet-stream
```

我们从打印的结构可以看出，这封邮件是一个 MIMEMultipart，它包含两部分：第一部分又是一个 MIMEMultipart，第二部分是一个附件。而内嵌的 MIMEMultipart 是一个 alternative 类型，它包含一个纯文本格式的 MIMEText 和一个 HTML 格式的 MIMEText。

3. 小结

用 Python 的 poplib 模块收取邮件分两步：第一步是用 POP3 协议把邮件获取到本地，第二步是用 email 模块把原始邮件解析为 Message 对象，然后，用适当的形式把邮件内容展示给用户即可。

3.4 数据库操作

1）SQLite 简介

SQLite 是一个包含在 C 库中的轻量级数据库，Python 内置了 SQLite3。它并不需要独立的维护进程，并且允许使用非标准变体(nonstandard variant)的 SQL 查询语句来访问数据

库。一些应用可是使用 SQLite 保存内部数据。它也可以在构建应用原型的时候使用，以便于以后转移到更大型的数据库，比如 PostgreSQL 或者 Oracle。

2）创建并打开数据库

为了使用这个模块，必须先创建一个连接(Connection)对象来代表数据库。在以下的例子中，数据将会被保存在 example.db 文件中：

```
import sqlite3
conn = sqlite3.connect('example.db')
```

如果指定的数据库存在，就会直接打开这个数据库，否则将新建再打开。也可以提供专用名 :memory: 来在内存中建立数据库：

```
conn = sqlite3.connect(":memory:")
```

3）数据库连接对象

一旦拥有了连接(Connection)对象，就可以创建游标(Cursor)对象并调用他的 execute()方法来执行 SQL 语句：

复制代码代码如下:

```
c = conn.cursor()
 # Create table
 c.execute('''CREATE TABLE stocks (date text, trans text, symbol text, qty real, price real)''')
 # Insert a row of data
 c.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")
 # Save (commit) the changes
 conn.commit()
 # We can also close the connection if we are done with it.
 conn.close()
```

保存后的数据是持久的，并且可以在以后的访问中可用。

4）增删改查

1.建(create)表

c.execute("create table catalog (id integer primary key,pid integer,name varchar(10) UNIQUE,nickname text NULL)")

上面语句创建了一个叫 catalog 的表，它有一个主键 id，一个 pid，和一个 name，name 是不可以重复的，以及一个 nickname 默认为 NULL。

2.删除表(DROP)，清空表(TRUNCATE)

c.execute("drop table catalog")

上面语句将 catalog 表删除。

另外 SQLite 中没有清空表的操作，使用如下方式替代:

c.execute("delete from catalog")

3.插入(insert)数据，更改(uptate)数据

通常 SQL 语句中会用到 python 变量作为值(value)。不建议直接使用 python 的字符串运算来构造查询语句，因为这样是不安全的，会使你的程序容易受到 SQL 注入攻击。

可以使用 DB-API 提供的参数代换。在想使用值(value)的地方放置一个'?'作为占位符，然后提供一个由值(value)组成的元组作为游标(cursor)中 execute()方法的第二个参数。（其他的数据库模块可能使用别的占位符，比如 '%s' 或者 ':1'）

复制代码代码如下:

```
# Larger example that inserts many records at a time
purchases = [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
('2006-04-05', 'BUY', 'MSFT', 1000, 72.00),
('2006-04-06', 'SELL', 'IBM', 500, 53.00)]
c.executemany('INSERT INTO stocks VALUES (?,?,?,?,?)', purchases)
c.execute("UPDATE catalog SET trans='SELL' WHERE symbol = 'IBM'")
```

4.查询(select)数据

正如前面所说，提倡使用元组进行操作。

```
t = ('RHAT',)
c.execute('SELECT * FROM stocks WHERE symbol=?', t)
print c.fetchone()
```

5.删除(delete)数据

```
t=('RHAT')
c.execute("DELETE * FROM stocks WHERE symbol=?", t)
```

3.5 程序练习

1）统计文件中各单词（区分大小写）的出现频率
2）实时股价：可以查询股票当前价格。用户可以设定数据刷新频率，程序会用绿色和红色的箭头表示股价走势。
3）邮件检查工具（POP3/IMAP）——用户输入一些账号信息，包括服务器、ip、协议类型（POP3或者 IMAP），应用每隔一段时间就会检查下该账号下的邮箱。
4）学习计划：根据课堂内容、自定制学习目标，确定学习计划，以电子邮件和短信的形式提醒完成计划。
备注：1 必做，2/3/4 任选一个。

4 参考资料

1 征服 python：语言基础与典型应用 孙广磊 人民邮电出版社
2 Python cookbook 第三版以后 人民邮电出版社
3 廖雪峰的官方网站：https://www.liaoxuefeng.com/
4 菜鸟教程 http://www.runoob.com/python3/python3-socket.html