本文适合有经验的程序员尽快进入Python世界.特别地,如果你掌握Java和Javascript,不用1小时你就可以用Python快速流畅地写有用的Python程序.

-

为什么使用Python?

假设我们有这么一项任务:简单测试局域网中的电脑是否连通.这些电脑的ip范围从192.168.0.101到192.168.0.200.

思路:用shell编程.(Linux通常是bash而Windows是批处理脚本).例如,在Windows上用ping ip 的命令依次测试各个机器并得到控制台输出.由于ping通的时候控制台文本通常是"Reply from ... " 而不通的时候文本是"time out ... ",所以,在结果中进行字符串查找,即可知道该机器是否连通.

实现:Java代码如下:

```
String cmd="cmd.exe ping ";
String ipprefix="192.168.10.";
int begin=101;
int end=200;
Process p=null;
for(int i=begin;i<end;i++){
p= Runtime.getRuntime().exec(cmd+i);
String line = null;
BufferedReader reader = new BufferedReader(new
InputStreamReader(p.getInputStream()));
while((line = reader.readLine()) != null)
{
//Handling line , may logs it.
}
reader.close();
p.destroy();
}
```

这段代码运行得很好,问题是为了运行这段代码,你还需要做一些额外的工作.这些额外的工作包括:

编写一个类文件

编写一个main方法

将之编译成字节代码

由于字节代码不能直接运行,你需要再写个小小的bat或者bash脚本来运行.

当然,用C/C++同样能完成这项工作.但C/C++不是跨平台语言.在这个足够简单的例子中也许看不出C/C++和Java实现的区别,但在一些更为复杂的场景,比如要将连通与否的信息记录到网络数据库.由于Linux和Windows的网络接口实现方式不同,你不得不写两个函数的版本.用Java就没有这样的顾虑.

同样的工作用Python实现如下:

```
import subprocess
cmd="cmd.exe"
begin=101
```

```
end=200
while begin<end:
p=subprocess.Popen(cmd,shell=True,stdout=subprocess.PIPE,
stdin=subprocess.PIPE,
stderr=subprocess.PIPE)
p.stdin.write("ping 192.168.1."+str(begin)+"\n")
p.stdin.close()
p.wait()
print "execution result: %s"%p.stdout.read()
```

对比 Java,Python 的实现更为简洁,你编写的时间更快.你不需要写 main 函数,并且这个程序保存之后可以直接运行.另外,和 Java 一样,Python 也是跨平台的.有经验的 C/Java 程序员可能会争论说用 C/Java 写会比 Python 写得快.这个观点见仁见智.我的想法是当你同时掌握 Java 和 Python 之后,你会发现用 Python 写这类程序的速度会比 Java 快上许多.例如操作本地文件时你仅需要一行代码而不需要 Java 的许多流包装类.各种语言有其天然的适合的应用范围.用 Python 处理一些简短程序类似与操作系统的交互编程工作最省时省力.

Python 应用场合

足够简单的任务,例如一些 shell 编程.如果你喜欢用 Python 设计大型商业网站或者设计复杂的游戏,悉听尊便.

---

2 快速入门

2.1 Hello world

安装完 Python 之后(我本机的版本是2.5.4),打开 IDLE(Python GUI) , 该程序是 Python 语言解释器,你写的语句能够立即运行.我们写下一句著名的程序语句:

print ("Hello,world!")或者 print ('hello,world!') //必须有括号,不然是语法错误

并按回车.你就能看到这句被 K&R 引入到程序世界的名言.

在解释器中选择"File"--"New Window" 或快捷键 Ctrl+N , 打开一个新的编辑器.写下如下语句:

```
print "Hello,world!"
raw_input("Press enter key to close this window");
```

保存为 a.py 文件.按 F5,你就可以看到程序的运行结果了.这是 Python 的第二种运行方式.

找到你保存的 a.py 文件,双击.也可以看到程序结果.Python 的程序能够直接运行,对比 Java,这是一个优势.

2.2 国际化支持

我们换一种方式来问候世界.新建一个编辑器并写如下代码:

#coding=UTF-8　　//必须告诉 eclipse 你的编码方式，而不是默认的 ASCII 码方式，或者用 eclipse 直接指定编码，不用再手动指定。

print ('中国人')

print "欢迎来到奥运中国!"

raw_input("Press enter key to close this window");

在你保存代码的时候,Python 会提示你是否改变文件的字符集,结果如下:

-*- coding: cp936 -*-

print "欢迎来到奥运中国!"

raw_input("Press enter key to close this window");

将该字符集改为我们更熟悉的形式:

-*- coding: GBK -*-

print "欢迎来到奥运中国!" # 使用中文的例子

raw_input("Press enter key to close this window");

程序一样运行良好.

2.3 方便易用的计算器

用微软附带的计算器来计数实在太麻烦了.打开 Python 解释器,直接进行计算:

a=100.0

b=201.1

c=2343

print ((a+b+c)/c)　　//在外面一定还要加一层括号，否则会出现 TypeError: unsupported operand type(s) for /: 'NoneType' and 'int'的错误提示

2.4 字符串,ASCII 和 UNICODE

可以如下打印出预定义输出格式的字符串:

print """

Usage: thingy [OPTIONS]

-h Display this usage message

-H hostname Hostname to connect to

"""

字符串是怎么访问的?请看这个例子:

word="abcdefg"

a=word[2]

print ("a is: "+a)

b=word[1:3]

print ("b is: "+b) # index 1 and 2 elements of word.

c=word[:2]

print ("c is: "+c) # index 0 and 1 elements of word.

d=word[0:]

```
print ("d is: "+d) # All elements of word.
e=word[:2]+word[2:]
print ("e is: "+e) # All elements of word.
f=word[-1]
print ("f is: "+f) # The last elements of word.
g=word[-4:-2]
print ("g is: "+g) # index 3 and 4 elements of word.
h=word[-2:]
print ("h is: "+h) # The last two elements.
i=word[:-2]
print ("i is: "+i) # Everything except the last two characters
l=len(word)
print ("Length of word is: "+ str(l))
```

请注意 ASCII 和 UNICODE 字符串的区别:

```
print ("Input your Chinese name:")
s=input("Press enter to be continued");
print ("Your name is   : " +s)
l=len(s)
print ("Length of your Chinese name in asc codes is:"+str(l))
a=(s,'GBK');
l=len(a)
print ("I'm sorry we should use unicode char!Characters number of your Chinese \
name in unicode is:"+str(l))
```

2.5 使用 List

类似 Java 里的 List,这是一种方便易用的数据类型:

```
word=['a','b','c','d','e','f','g']
a=word[2]
print ("a is: "+a)
b=word[1:3]
print ("b is: ")
print (b) # index 1 and 2 elements of word.
c=word[:2]
print ("c is: ")
print (c) # index 0 and 1 elements of word.
d=word[0:]
print ("d is: ")
print (d) # All elements of word.
e=word[:2]+word[2:]
print ("e is: ")
print (e) # All elements of word.
f=word[-1]
```

```
print ("f is: ")
print (f) # The last elements of word.
g=word[-4:-2]
print ("g is: ")
print (g) # index 3 and 4 elements of word.
h=word[-2:]
print ("h is: ")
print (h) # The last two elements.
i=word[:-2]
print ("i is: ")
print (i) # Everything except the last two characters
l=len(word)
print ("Length of word is: "+ str(l))
print ("Adds new element")
word.append('h')
print (word)
```

2.6 条件和循环语句

```
# Multi-way decision

x=int(input("Please enter an integer:"))
if (x<0):
    x=0
    print ("Negative changed to zero")
elif x==0:
    print ("Zero")
else:
    print ("More")

# Loops List
a = ['cat', 'window', 'defenestrate']
for x in a:
    print (x,len(x))
```

2.7 如何定义函数

```
# Define and invoke function.
def sum(a,b):
    return (a+b)      //缩进很严谨
func = sum
r = func(5,6)
print (r)
```

```
# Defines function with default argument
def add(a,b=2):
    return a+b
r=add(1)
print (r)
r=add(1,5)
print (r)
```

并且,介绍一个方便好用的函数:

```
# The range() function
a =range(5,10)
print (a)
a = range(-2,-7)
print (a)
a = range(-7,-2)
print (a)
a = range(-2,-11,-3) # The 3rd parameter stands for step
print (a)
```

2.8 文件 I/O

```
spath="F:/baa.txt"
# Opens file for writing.
#Creates this file doesn't exist.
f=open(spath,"w")
f.write("First line 1.\n")
f.writelines("First line 2.")
f.close()
f=open(spath,"r") # Opens file for reading
for line in f:
    print (line)
f.close()
```

2.9 异常处理

```
s=input("Input your age:")
if s =="":
    raise Exception("Input must no be empty.")
try:
    i=int(s)
except ValueError:
    print ("Could not convert data to an integer.")
```

```
except:
    print ("Unknown exception!")
else: # It is useful for code that must be executed if the try clause does not raise an exception
    print ("You are %d" % i," years old")
finally: # Clean up action
    print ("Goodbye!")
```

2.10 类和继承

python 中 类的私有对象要用 self 来做前缀引用，如上例中的 self.name 等等。而一个类的公用变量这用类名做前缀来引用。比如上例中的 SchoolMember.count。类的公用变量，跟 Java 中的 static 变量一样，被类的所有实例共享。
另外 python 中类的构造函数 默认名字为 __init()__，左右都是双下横线，类实例化是解释器自动首先调用。同 C++一样，Python 有一个析构函数__del()__，实例被 del 时，自动调用该析构函数。
如果要求打印不换行，则需要 print(your_str,end='')

```
class Base:
    def __init__(self):
        self.data = []
#这里定义 add 函数必须进行缩进，不然会提示：AttributeError: 'Child' object has no attribute 'add'
#python 就是用缩进代替 java 的{}，也就是以缩进来判断方法的结束。
    def add(self, x):
        self.data.append(x)
    def addtwice(self, x):
        self.add(x)
        self.add(x)
# Child extends Base
class Child(Base):
    def plus(self,a,b):
        return a+b
oChild =Child()
oChild.add("str1")
print (oChild.data)
print (oChild.plus(2,3))
```

#另一个实例

```
class SchoolMember:
    '''Represents any school member.'''
    count = 0
    def __init__(self,name,age):
        self.name = name
        self.age = age
        SchoolMember.count += 1
        print('(Initialized SchoolMember: %s)' % self.name)

    def tell(self):
        '''Tell my details.'''
        print('''Name:"%s" Age:"%s"''' %(self.name,self.age),end=' '),

class Teacher(SchoolMember):
    '''Represents a   teacher.'''
    def __init__(self,name,age,salary):
        SchoolMember.__init__(self,name,age)
        self.salary = salary
        print('(Initialized Teacher: %s)' %self.name)

    def tell(self):
        SchoolMember.tell(self)
        print(',Salary: "%d"' %self.salary)

class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self,name,age,marks):
        SchoolMember.__init__(self,name,age)
        self.marks = marks
        print('(Initialized Student: %s)' % self.name)

    def tell(self):
        SchoolMember.tell(self)
        print('Marks: "%d"' % self.marks)

t = Teacher('Mr',40,30000)
s = Student('Qi Jiang',23,83)

print()
print(SchoolMember.count)
members = [t,s]
for member in members:
    member.tell()
```

2.11 包机制

每一个.py 文件称为一个 module,module 之间可以互相导入.

请参看以下例子:

```
# a.py
def add_func(a,b):
    return a+b
# b.py

from a import add_func # Also can be : import a
print ("Import add_func from module a")
print ("Result of 1 plus 2 is: ")
# If using "import a" , then here should be "a.add_func"
print (add_func(1,2))
```

module 可以定义在包里面.Python 定义包的方式稍微有点古怪,假设我们有一个 parent 文件夹,该文件夹有一个 child 子文件夹.child 中有一个 module a.py .如何让 Python 知道这个文件层次结构?很简单,每个目录都放一个名为 _init_.py 的文件.该文件内容可以为空.这个层次结构如下所示: parent

```
--__init_.py
--child
-- __init_.py
--a.py
b.py
```

那么 Python 如何找到我们定义的 module?在标准包 sys 中,path 属性记录了 Python 的包路径.你可以将之打印出来:

```
import sys
print (sys.path)
```

通常我们可以将 module 的包路径放到环境变量 PYTHONPATH 中,该环境变量会自动添加到 sys.path 属性.另一种方便的方法是编程中直接指定我们的 module 路径到 sys.path 中:

```
import sys
print(sys.path)
sys.path.append('XXXX')
from a import add_func

print (sys.path)
print ("Import add_func from module a")
```

```
print ("Result of 1 plus 2 is: ")
print (add_func(1,2))
```

总结你会发现这个教程相当的简单.许多 Python 特性在代码中以隐含方式提出,这些特性包括:Python 不需要显式声明数据类型,关键字说明,字符串函数的解释等等.我认为一个熟练的程序员应该对这些概念相当了解,这样在你挤出宝贵的一小时阅读这篇短短的教程之后,你能够通过已有知识的迁移类比尽快熟悉 Python,然后尽快能用它开始编程.

当然,1小时学会 Python 颇有哗众取宠之嫌.确切的说,编程语言包括语法和标准库.语法相当于武术招式,而标准库应用实践经验则类似于内功,需要长期锻炼.Python 学习了 Java 的长处,提供了大量极方便易用的标准库供程序员"拿来主义".(这也是 Python 成功的原因),在开篇我们看到了 Python 如何调用 Windows cmd 的例子.