## [Python](#目录)

sourceforge 开源软件开发者进行开发管理的集中式场所；在这里下载的Python和Pythonwin；

Pythonwin下载地址：<http://sourceforge.net/projects/pywin32/files/pywin32/>

### 目录

4，Python对象 P85

5，数字 P115

6，字符串、列表和元组 P152

7，映射和集合类型 P250

8，条件和循环 P285

9，文件和输入输出 P317

10，错误和异常 P345

11，函数和函数式编程 P388

12，模块 P450

13，面向对象编程 P472

14，执行环境 P582

15，正则表达式 P621

16，网络编程 P655

17，网络客户端编程 P685

18，多线程编程 P721

19，图形用户界面编程 P748

20，Web编程 P781

21，数据库编程 P841

22，扩展Python P880

23，其它话题 P901

### 概念：

1，Python内存管理由Python解释器负责。

2，Python是一种解释型语言，没有编译环节，其源文件为.py，解释器加载它会将其编译成字节码（.pyc，.pyo文件），然后机器码。

3，运行Python;

启动解释器、运行Python脚本、集成开发环境。

PythonWin是windows下的集成开发环境，需下载，windows下建议用这个。

IDLE 是下载Python时自带的集成开发环境。

交互运行：

运行脚本：

#!/usr/local/bin/python XX.py 环境变量

开发环境：

PythonWin针对MFC，包含Win API和COM，可以开发Windows 应用程序；

4，帮助文档

本地手册；

在线文档；

### 语法：

(\_)在解释器中代表最后一个表达式的值，也不要作变量的开始。

\_xxx 不用from module import \* 导入

\_\_\_xxx\_\_\_系统定义

\_\_\_xxx 类中的私有变量

**不同版本，语法不一样**，比如print “%s is number %d” % (“Python”, 1)在343版本中不行，因为print是一个函数了，改成：print("%s is number %d" % ("Python", 1))

print >> sys.stderr ‘..’ print >> logfile,’..’ 输出重定向

用户输入：var = input(“tip:”) int(var) 解析…

注释：# 单行； ‘’’ 多行注释 ‘’’ ；

运算符：+ - \* /（float除法四舍五入） //（整除） % \*\*（幂）

< > == <= >= !=

and 、or、 not ； 就是C++的 &&、||、!；

支持n \*= 10，不支持++，--。

is， is not 变量是否指向同一个对象；

位运算：~ & | ^ << >>

pass就是 执行空语句；

### 变量

动态类型，自动识别；无须事先声明；变量名会被回收（所有对象引用计数）；

赋值：（x,y,z） = (1,2,’a string’)

交换：z,x,y = y,z,x

del obj1，obj2。。。删除对象一个引用。

type(变量) 返回变量类型 class ‘int’

浮点型返回false；None；

### 类型：

数字：int、long(巨长)、bool(True)、float、complex；

字符串：’’；

列表[]和元组()：数组，切片；

字典{}：map；

### 缩进区分不同的代码块

反斜杠：\ 继续上一行，含括号和三引号不用。

### if语法

if x < .0 and (a >= 10): 括号可要可不要

..

if ..:

else: ..

三目运算 X if C else Y

if ..: elif ..: else:

### while语法

while .. :

### for语法

for iter in iterable: …

range(n) 能生成 0 ~ n-1的数字；

for i,ch in enumerate(foo): .. i索引，ch元素

列表解析：

[expr for iter\_var in iterable if cond\_expr]

sqdEvens = [x \*\* 2 for x in range(8) if not x % 2]；

迭代器对象有一个next()方法，迭代完后触发StopIteration异常，for内部处理了；

i = iter(iterable) 使用 iter(func, sentinel) 重复调用func，返回值=sentinel的iter；

i.next()

让类具有iter功能 P537；

### 文件语法

handle = open(file\_name, access\_mode = 'r',buffering=-1) ‘w’ ‘a’追加 ‘+’读写 ‘b’二进制访问，可自由组合；

工厂 ：file()，等价open()；

read() 读取字节；

readline()

readlines()

write、readline、readlines；

读会有行结束符，写不会自动加；

seek() 、 tell() 告诉你文件指针在哪里；

遍历 行for eachLine in f:

close()、flush()

truncate() 截取文件；

文件属性

file.closed encoding mode name文件名

stdin,stdout,stderr文件句柄 sys模块

命令行sys.argv 参数列表 len(sys.argv) 个数；

os模块 和 os.path模块 提供与操作系统无关的统一文件访问

linesep，sep，pathsep，curdir，pardir；

如：fobj.write('%s%s' % (aLine, os.linesep)

P330 os模块

mkfifo/mknod 创建fifo/文件节点

remove/unlink 删除文件

rename/renames 重命名

\*stat 文件信息

symlink 创建符号链接

utime 更新时间戳

tmpfile 创建并打开w+b临时文件

walk

chdir/fchdir 改变当前目录

chroot 改变当前进程根目录

listdir 列出指定目录文件

getcwd/getcwdu 返回当前目录

mkdir/makedirs 创建目录/多层目录

rmdir/removedirs

access 检验权限

chmod 改变权限

chown/lchown 改变owner和group ID

umask 设置默认权限

open，read，write

dup/dup2 复制文件描述符/复制到另一个

makedev，major，minor

P331 os.path模块

basename ，dirname，join，split，splitdrive，getatime，exists，isdir…

### 异常语法

try:

try\_suite #监控这里的异常

**发生异常之后的语句不会执行！解释器搜索处理器，找不到会一直向上抛；**

except Exception[, reason]:

except\_suite #异常处理代码

except (Exc1[, Exc2[, ... ExcN]])[, reason]:

except\_suite #多种类型一种处理

捕获所有异常

try:

:

except BaseException, e:

不要把大片代码放入try-except，异常的作用是提供一个可以提示错误或处理错误的机制；

内建异常 P375

ValueError，TypeError，IOError…

finally: 无论如何都会执行；

明确触发异常

raise [SomeException [, args [, traceback]]]

raise [异常名字，字符串,类或实例[, args [, traceback]]]

args：单一的字符串用来指示错误的原因.如果传的是元组,通常的组成是一个错误字符串,一个

错误编号,可能还有一个错误的地址,比如文件,等等；

### 函数语法

def addMe2Me(x=True):

…

return (x + x) 没有return返回None

cmp() 比较两对象

str()、repr() 以字符串形式返回对象的内容、类型、数值等，str()可读性更好，repr则是：

obj == eval(repr(obj)) 可以重新获得对象。

数值工厂函数：int(obj, base=10) complex(…)…

功能函数：abs()， divmod()商余，round()四舍五入，

Int直接截取小数部分；floor得到最接近原数但小于原数；round最接近原数；

进制转换：hex oct

ASCII转换：ord(‘a’) -> 97 chr(65) -> ‘A’ unichr返回unicode码值

高级科学计算包：

http://numeric.scipy.org/

http://scipy.org/

序列相关的内建函数

sorted()，zip() 返回排序的列表；

reversed() ，enumerate() 返回迭代器；

内嵌函数

内部def，另一个方法是lambda语句；

装饰器

接受函数对象的 函数；

@decorator(dec\_opt\_args)

def func2Bdecorated(func\_opt\_args):

作用：

加入通用功能，如 1）日志； 2）性能测试；3）事物；

使用：

def tsfunc(func): #装饰器函数

def wrappedFunc(): #内嵌函数

print '[%s] %s() called' % (ctime(), func.\_\_name\_\_)

return func() #内嵌函数 = print + 原函数

return wrappedFunc #返回包装了的函数

@tsfunc #装饰器

def foo():

pass

foo()

sleep(4)

for i in range(2):

sleep(1)

foo()

参数

任意数量的位置参数

def func(arg1, arg2, \*args):

func("hello", "Tuple, values is:", 2, 3, 3, 4) #自动被放入元组中提供给函数使用

func("hello", "Tuple, values is:", \*(2, 3, 3, 4)) 传入元组

任意数量的关键字参数

def func(arg1, arg2, \*\*kwargs):

func("hello", "Dict, values is:", x=2, y=3, z=3) #自动被放入字典中提供给函数使用

func("hello", "Dict., values is:", s=3, \*\*{'x':2, 'y':3, 'z':3,})

可变长度参数

lambda

lambda [arg1[, arg2, ... argN]]: expression

>>> a = lambda x, y=2: x + y

>>> a(3)

生成器 P443

### 类语法

#### 定义

class FooClass(object):

version = 0.1 静态变量

def \_\_init\_\_(**self**, nm='John Doe'): #构造，FooClass(‘myn’)；

self.name = nm #name object的

print 'Created a class instance for', nm

def showname(self): #有self的是类方法，没有的是static方法，self换成cls就是类方法

print 'Your name is', self.name

print 'My name is', self.\_\_class\_\_.\_\_name\_\_

def showver(self):

print self.version

def addMe2Me(self, x):

return x + x

创建类实例，调用方法

foo1 = FooClass()

foo1.showname()

#### 子类

class SubClassName (ParentClass1[, ParentClass2, ...]):

def \_\_init\_\_(self, nm, ph, id, em):

父类.\_\_init\_\_(self, nm, ph) 调父类构造

#### 查看类属性

dir(类名) 或者 类.\_\_dict\_\_

特殊类属性（可以通过类.XX访问）

C.\_\_name\_\_ 类Ｃ的名字（字符串）

C.\_\_doc\_\_ 类Ｃ的文档字符串

C.\_\_bases\_\_ 类Ｃ的所有父类构成的元组

C.\_\_dict\_\_ 类Ｃ的属性

C.\_\_module\_\_ 类Ｃ定义所在的模块（1.5 版本新增）

C.\_\_class\_\_ 实例Ｃ对应的类（仅新式类中）

#### 调用父类方法

super(type[,obj]).方法 返回给定type的父类

super(CMyClass, self).\_\_init\_\_()

issubclass, isinstance, \*attr系列

obj.attr 其实就是调用 \*attr(obj, ‘attr’ …) 系列函数；

#### 可重载的方法

P525；

C.\_\_init\_\_(self[, arg1, ...]) 构造器（带一些可选的参数）

C.\_\_new\_\_(self[, arg1, ...])a 构造器（带一些可选的参数）；通常用在设置不变数据类

型的子类。

C.\_\_del\_\_(self) 解构器

C.\_\_str\_\_(self) 可打印的字符输出；内建str()及print 语句

C.\_\_repr\_\_(self) 运行时的字符串输出；内建repr() 和‘‘ 操作符

C.\_\_unicode\_\_(self)b Unicode 字符串输出；内建unicode()

操作符 等等等，有几十个；

P533 重载例子；

#### 包装

对已存在的对象(类型,代码,对象..)的一些定制，这样可以**新建，修改或删除原有类的一些功能**；

覆盖\_\_getattr\_\_()方法，包含getattr()内建函数的调用（返回属性或方法），对于切片这种操作，不是以属性存在的报错 P548；

例子 P550；

getattr(self, '\_%s\_\_%stime' % \

(self.\_\_class\_\_.\_\_name\_\_, t\_type[0])) # \_\_atime 会改名

P553，将字符串转大写写入文件，包装write方法 包装对象.write()，其它还是file的默认行为；

描述符

### 模块语法

搜索路径：编译或安装时指定；sys.path 里；sys.path.append(‘..’)；

导入模块 import 模块名

顺序：标准库，三方库，自定义；

导入指定属性：from module import name1[, name2[,... nameN]] [as 别名]

访问 module.func() module.var

布局：

1）起始行 总说明

2）模块文档 模块字符串

3）模块导入 import

4）变量定义

5）类定义

6）函数定义

7）主程序 if \_\_name\_\_ == ‘\_\_main\_\_’:

#### 相关模块：

Debugger：pdb 调试

Logger：logging 日志

Profilers：profile，hotshot，cProfile 性能测试

decimal 十进制浮点运算类 Decimal

array 高效数值数组（字符，整数，浮点数等等）

math/cmath 标准C库数学运算函数。常规数学运算在match模块，复数运算在cmath模块

random 多种伪随机数生成器

randrange() 随机返回range([start,] stop [,step])结果的一项；

uniform() 返回浮点数，不包括上限；

random 0.0 ~ 1.0之间；

choice随机返回给定序列一个元素；

### 包语法

有层次的文件目录结构；

Phone/

\_\_init\_\_.py

common\_util.py

Voicedta/

\_\_init\_\_.py

Pots.py

Isdn.py

Fax/

\_\_init\_\_.py

G3.py

Mobile/

\_\_init\_\_.py

Analog.py

igital.py

Pager/

\_\_init\_\_.py

Numeric.py

导入

import Phone.Mobile.Analog

Phone.Mobile.Analog.dial()

from Phone import Mobile

Mobile.Analog.dial('555-1212')

### Python对象

特性：身份id()、类型type()、值；

类型type(42) <type ‘int’> type(type(42)) <type ‘type’>

None对象 False

### 序列

#### 概念

成员关系操作符：

in ，not in 操作符；

Seq[i] 下标为i的元素

Seq[ i1 : i2: step] seq[i1] ~ seq[i2-1]的集合 step步长默认1，负数是倒过来，None是最后一个

Seq \* expr 序列重复expr次

Seq1 + seq2 连接序列

i = iter(myTuple) 获取序列迭代器

>>> i.next() 第一个元素

#### 序列内建函数

转换：list() str() tuple()

辅助：enumerate() 返回索引，值的pair， len，max，min

Reversed，sorted，sum，zip返回pair列表

#### 字符串

‘http://’#protocol ‘localhost’ #hostname … 字符可以自动拼接，可以加注释了。

‘Hello’+u’’ + u’!’ -> u’Hello!’ 自动转成Unicode。

重复操作符：\*

‘Ni’ \* 3 -> ‘Ni!Ni!Ni!’

格式控制：

同C。

辅助指令：\* 精度；- 左对齐；+ 加号；<sp> 空格；

# 八进制0，十六进制0x；0 填0；m.n 总宽.小数位

原始字符串：

所有的字符都是按照字面意思使用，没有转义或不能打印字符，这对regex的创建等工作变得简单。

r’\n’ -> ‘\\n’

内建函数 P186

#### unicode

u’…’；

unicode() 和 unichar() 转换；

使用：

1. 一定要加前缀u
2. 不要用string模块，str()等函数，用unicode版本；
3. 直到写文件或数据库或网络时才调用encode()编码，读的时候decode()解码；

#### 列表

aList = [123, 'abc', 4.56, ['inner', 'list'], 7-9j]

list('foo') -> ['f', 'o', 'o']

aList[2] = 'float replacer'

del aList[1]

aList.remove(123)

1ist1 < list2

切片访问

+ ,\*, in,not in,

#### 元组

和列表相似，用()，区别：不可变类型，所以可以用作字典的key；

并不是绝对不可变：

t = ('third', 'fourth')

t = t + ('fifth', 'sixth')

t = (['xyz', 123], 23, -103.4)

t[0][1] = ['abc', 'def']

return 'abc', -4.24e93

x, y = 1, 2 等，默认都是元组

一个元素的元组：(‘xyz’,) 有个逗号

#### 浅拷贝和深拷贝

序列类型对象的浅拷贝是默认的，1）完全切片[:] 2）工厂函数list(),dict() 3）copy模块的copy函数 都会实施；

person = ['name', ['savings', 100.00]]

hubby = person[:] # slice copy

wifey = list(person) # fac func copy

hubby[0] = 'joe'

wifey[0] = 'jane'

>>> hubby, wifey

(['joe', ['savings', 100.0]], ['jane', ['savings', 100.0]])

>>> hubby[1][1] = 50.00

>>> hubby, wifey

(['joe', ['savings', 50.0]], ['jane', ['savings', 50.0]])

字符串是不可变对象，序列是浅拷贝；

>>> import copy

>>> wifey = copy.deepcopy(person)

这样再修改就不会了；

#### 字典

dict2 = {'name': 'earth', 'port': 80}

>>> ddict = {}.fromkeys(('x', 'y'), -1)

>>> ddict

{'y': -1, 'x': -1}

>>> edict = {}.fromkeys(('foo', 'bar'))

>>> edict

{'foo': None, 'bar': None}

>>>> for key in dict2:

... print 'key=%s, value=%s' % (key, dict2[key])

del dict2['name'] # 删除键为“name”的条目

dict2.clear() # 删除dict2 中所有的条目

del dict2 # 删除整个dict2 字典

dict2.pop('name') # 删除并返回键为“name”的条目

'name' in dict2

比较顺序

长度， key， 值；

#### 集合

set，frozenset

s = set('cheeseshop') 唯一创建方法；

联合 |，交集 &， 差集 s-t 只属于s， 对称差分^ ；

性能

while i < len(s) : 计算一次

ls = os.line 取别名

### guidata

python的图形用户界面库，基于QT库；

安装请看帮助文档，import guidata；

#### demo

import guidata

\_app = guidata.qapplication() # not required if a QApplication has already been created

import guidata.dataset.datatypes as dt

import guidata.dataset.dataitems as di

class Processing(dt.DataSet):

"""Example"""

a = di.FloatItem("Parameter #1", default=2.3)

b = di.IntItem("Parameter #2", min=0, max=10, default=5)

type = di.ChoiceItem("Processing algorithm",

("type 1", "type 2", "type 3"))

param = Processing()

param.edit()

param.a, param.b... 直接使用用户输入，非常方便

下面有更多的demo：

from guidata import tests

tests.run()

### Tkinter(3.7)

Tcl/Tk的封装层，\_tkinter线程安全；

Lib/tkinter；

import tkinter / from tkinter import \*

#### 公共模块

tkinter.filedialog

#### hello,world

import tkinter as tk

class App(tk.Frame):

def \_\_init\_\_(self, master=None):

super().\_\_init\_\_(master)

self.master.geometry('250x150')

self.pack()

self.CreateWidgets()

def CreateWidgets(self):

self.label = tk.Label(self, text="hello,world!")

self.label.pack(fill='y', expand=1)

top = tk.Tk()

app = App(master=top)

app.mainloop()

#### Tk&Tkinter映射

button .panel.fred =====> fred = Button(panel)

button .fred -fg red =====> fred = Button(panel, fg="red")

.fred configure -fg red =====> fred["fg"] = red

OR ==> fred.config(fg="red")

.fred invoke =====> fred.invoke()

pack .fred -side left =====> fred.pack(side="left")

查看widget配置项

import tkinter as tk

class App(tk.Frame):

def \_\_init\_\_(self, master=None):

super().\_\_init\_\_(master)

self.pack()

self.fred = tk.Scale(self)

print(self.fred.config())

app = App()

打印的key就是配置，value分别是（配置名，db中的配置名，db中的配置类，默认值，当前值）

#### Packer

界面管理器，管理布局、尺寸等；

widgets只有pack了才会显示，比如fred.pack(side="left")；

pack选项：26.1.6.3；

anchor 位置、expand[0,1]、fill[‘x’,’y’,’both’,’none’]、ipadx ipady、padx pady、side[left,right,top,bottom]

#### 数据交换

widgets配置项：variable, textvariable, onvalue, offvalue, and value，用控件的set/get就可以了，但是it is not possible to hand over an arbitrary Python variable to a widget through a variable or textvariable option

Python需要一个变量如：self.contents = StringVar()

StringVar, IntVar, DoubleVar, and BooleanVar，

然后去绑定事件：self.entrythingy.bind('<Key-Return>',

self.print\_contents)

self.contents.get() / set(value)

#### wm

window manager，你可以控制titles, placement, icon bitmaps, and the like

所有的Toplevel widgets都继承了Wm类，比如：

myapp = App()

myapp.master.title("My Do-Nothing Application")

myapp.master.maxsize(1000, 400)

26.1.6.6；

Label(top, text=’’, font = ‘Helvetica -12 boild’)

top.geometry(‘250x150’)

#### Tk选项数据类型

26.1.6.6；

#### 事件

def bind(self, 绑定事件类型, Python函数带Event参数, add=''/’+’事件列表):

def turn\_red(self, event):

event.widget["activeforeground"] = "red"

self.button.bind("<Enter>", self.turn\_red)

| **Tk** | **Tkinter Event Field** | **Tk** | **Tkinter Event Field** |
| --- | --- | --- | --- |
| %f | focus | %A | char |
| %h | height | %E | send\_event |
| %k | keycode | %K | keysym |
| %s | state | %N | keysym\_num |
| %t | time | %T | type |
| %w | width | %W | widget |
| %x | x | %X | x\_root |
| %y | y | %Y | y\_root |

#### 索引

Entry、Text、Menu等，

字符0，active/last/none字如其意，@6是menu的y pixel；

#### Images

26.1.6.9；