中蟒入门

Release 0.00

2004年2月11日

glace-at-chinesepython.org

中蟒说明文件

Python 自己有一套很不错的说明文件系统. 不过用的是LaTex. 虽然好用, 但是略嫌复杂了些. 我本来打算用DocBook, 搞半天才弄明白那个xml.apache.org 上面的东东. 最后想到既然中蟒都支援XML, 何必外求, 不如自己整一套当做是写中蟒的练习. 结果一拖再拖, 说明文件也没写成. 现在后悔已晚, 只好急就章地凑合了一些, 多多包涵. 慢慢会多些文件的了, 我保证. 不然中蟒作为教育工具的第一步就无法达成了.

Python 的说明文件其实也可以拿来用, 只要把其中英文程式范例部份译成中文就行了.

暂时除了本网页上的资料,中蟒大杂院中也有一些文献可用, http://www.chinesepython.org/cgi_bin/moingb.cgi

目录

1 启动中蟒: 在视窗和linux 环境中使用中蟒即译器.

视窗9x 系统:

视窗系统下, 中蟒的安装非常容易. 只要把下载的压缩档解压到某个目录就可以了.

为了让执行时更方便, 可以把所有.py 和.pyc 档和中蟒即译器关联起来.

执行档cpython.exe 和cpythonw.exe 是一样的. 不过cpythonw.exe 在运行时不会打开DOS 窗口.

另外在autoexec.bat 中加入相关路径会更方便些. 如"set path=

启动中蟒:

```
c:\> c:\chinesepython\python.exe -g/-b c:\chinesepython\tools\pyshell\pyshell.py
```

记住, 如果你的视窗系统是GB 编码的话请用-g 选项, 如果是BIG5 的话请用-b 选项. 如果是NT, XP 等什么的是用unicode 的话, 我没试过, 但听说能用.

pyshell.py 是一个中蟒程式, 它便用wxWindow 库来打开一个视窗画面来运行中蟒即译器.这是因为在dos 画面下虽可显示中文, 但却无法输入中文字. 所以如果想要用互动功能的话就必须用pyshell.py 程式. 该程式很简单, 可以做为初写图像介面的示范.

(另外也可以用

```
chinesepython\tools\idle.py
```

,

这个是用Tk 来写作的视窗环境即译器, 比起pyshell.py 多很多功能. 但因为Tk 是用统一码(unicode) 所以有些时候会有些怪码.)

因pyshell.py 的原因, 中蟒安装时已有了wxPython 库, 这是个很好用而且速度很快的图像介面函数库, 可以代替Tk.

进入互动环境后, 就可以开始发出指令了. 退出方法可以直接关闭中蟒视窗.

互动环境下输入的内容在视窗关闭后便会消失, 如要写程式的话就必须用文字编辑器来编写. 最简单的notepad.exe 会是不错的选择. 记住中蟒程式是纯文字形式的, 因此不是以纯文字格式存档的程式将无法运行.

linux:

在linux 环境下安装中蟒必须经过编译的工作. 因为大部份linux 环境都有gcc 编译器, 重新编译一次也是很容易的事.

下载中蟒源码档并进行编译:

```
$ tar xvfz chinesepython-0-3.tgz
$ cd chinesepython-0-3
$ ./configure --prefix=$HOME/share
$ make
$ make install
```

上面假设你并没有超级用户的权限, 因此中蟒会安装到你的家目录中share 文件夹下. 你需要在你的$PATH\#\emptyset$HOME/share/bin 否则会找不到执行档.

由于中蟒需要显示中文字体, 你必须有一个可以显示中文字的终端, 如rxvt.

chinesepython/Tools/pyshell/pyshell.py 要用到wxPython 库. 在linux 中没有dos 窗口中文输入的问题, 所以没有包括这个库. 如果想用wxPython 的用户必须自行安装http://www.wxpython.org. 我试过, 安装后pyshell.py 可以正常运作, 只是中文字体上要自己设定一下.

启动中蟒:

```
$ cpython [-g/-b]
.....
>>>
>>> <<Ctrl-d>>
$
```

根据不同的locale 设定, 你需要告诉中蟒用哪种编码来处理程式码. -g 代表GB, -b 代表BIG5. 如果不指定编码, 中蟒会侦查LANG 环境变数, zh_TW, zh_HK 是大五码, zh_CN 是国标码. 如果都找不到中蟒会以大五码来起动.

退出的方法是键入Ctrl-D (档案终结字符).

执行程式: 要执行中蟒程式, 只要起动时给出一个程式名就可以了.

```
cpython [-b/-g] 程式名.py
```

在启动中蟒时有许多的选项, 部份列于下: (由于还没有全部译好, 所以只列出重要的, 原文可以用'-h' 选项来看)

```
 -V : 印出版本资讯后退出
 -c : 执行字串 <br>
如: cpython -g -c "写 12,; 写 14+5" <br>
得: "12 19"
 -t : 检查程式码中有没有混用的跳格和空格符号
 -h : 印出选项说明并退出
 -O : 编译至 .pyc 档时使用最佳代码
 -S : 启动时不载入 site.py 设定档. 这样会快一些. site.py 档案中有一些有关安装, 操作系统, 中蟒即译器的额外资讯. 最重要的是会加入第三方扩展模组的搜寻路径, 例如在 site-packages 中的模组.
 -i : 执行档案后并不即时退出, 而是进入互动环境. 除错时可用.
 任何跟在命令行后面不属于选项的会直接交给中蟒程式, 它们会变成调用程式的参数.
```

中蟒程式的执行:

中蟒是利用即译的方式来执行的. 不过为了运行时的速度要快一些, 中蟒会先把程式源码(.py 档名) 编译成半成品(.pyc 档名) 格式. .pyc 是.py 经过了程式码的语法检查和句意分析等步骤, 其储存内容可以直接运算. 因此它们的分别只是在开始载入的时候.pyc 会较快些.

在命令行执行的程式不会保留.pyc 档. 但以"载入模组名" 方式编译的程式会保留.pyc 档以便下次载入时会更快一些. 中蟒在载入前会先检查一下.py 和.pyc 档是否一样, 如不一样则会重新编译一次.

一般安装在系统中的.py 最好先编译一次, 这样那些常用的模组就可以较快速的载入. 这在*inx 安装时已经做好了. 在视窗9x 系统下, 因为没所谓用户权限的说法, 因此没有关系.

2 基本操作: 基本功能和内建数据的使用方法

当计算机: 进入中蟒互动环境后, 你可以当它是一部现成的计算机来用:

```
>>> 12 + 3
15
>>> 15 * 2
30
>>> 21%2  #(取余数)
1
>>> 12/13 #(因为答案要取整数值, 0.923... 会变成 0)
0
>>> 12.0 / 13 #(这样就没问题了, 因为中蟒明白你现在要小数
0.92307692307692313
>>> 12 + 3 * 3 #(先乘除, 后加减)
21
>>> 2 ** 4
16
>>> 2 ** 64 #(超出了最大整数上限)
OverflowError: integer exponentiation
>>> 2L ** 64 #(用了 2L, 中蟒知道你要用大整数, 就没问题了)
18446744073709551616L

变量: 你可以在中蟒中定义一些变量, 并为给它们指定值
>>> 甲 = 12
>>> 乙 = 3
>>> 甲 + 乙
15
>>> 甲 + 乙**乙
39
>>> 甲 = 0 #(可以重新指定值)
>>> 乙 = 甲 + 1
>>> 写 乙
1
>>> 删除 甲 # 从变数表中除去 '甲', 共空出记忆体

字串: 字串是指一串的文字符. 如 "This  is a test", 或 "这个也算 1 个"

 中蟒内建了许多字串的操作. 其中特别针对中文做了几个常用到的函数. 包括:

\begin{verbatim}
>>> 甲 = "这是this字串"
>>> 甲.中英文字数()
8
>>> 甲.中文字数()
4
>>> 甲.英文字数()
4
>>> 甲.拆字()
[ '这', '是', 't', 'h', 'i', 's', '字', '串' ]
>>> 甲.字串编码() #传回该字串的编码
'国标'
>>> 乙 = 甲.强设编码('大五') # 不做变换, 只改变编码代号
>>> 丙 = 甲.国标变大五() # 做编码变换
>>> 甲.十六进() # 以十六进位代表字串内容
'\\xb3\\x6f\\xac\\x4f\\x74\\x68\\x69\\x73\\xa6\\x72\\xa6\\xea'
>>> 长度(甲) # 记住中文字是占两个位元的
12
>>> 甲[0:2] # 抽出'甲'中的第 0 位到第 2 个子字串
'这'
>>> 写 '这也' + 甲[2:]  # 产生一个新字串
'这也是this字串'
```

有几件事要留心:

抽取子字串时, 甲[始, 终] 里'始' 是从'0 位' 开始算的, 而'终' 则是指到第几'个'. 像上面字串中首字'这' 的第位元是'甲' 字串中的第'0' 位, 但算是第'1' 个字元. 如果不给出'始', '终' 则暗示从字串的头, 尾开始数.

如果'始', '终' 是负数的话, 则表示从字串的右面开始数起. '甲[-1]' 其实就是指最尾的字元

字串在中蟒属于不可变型态. 你不能做"甲[0] = 't'" 这样的操作. 你只能利用字串中的值来产生新的字串, 像" 甲= 't' + 甲[1:] " 就没问题了.

序列: 你可以把序列想像成是阵列. 分别是你可以用任何型态的值放到序列中. 同样你可以抽取子序列. 和字串不同的是序列是可变的.

```
>>> 甲 = [ 'a', 12.5, 3, '你好' ]
>>> 长度(甲)
4
>>> 甲[0] = 'wah'
>>> 写 甲
[ 'wah', 12.5, 3, '你好' ]
>>> 甲[2] = [ 3, 4, 5 ]
>>> 写 甲
[ 'wah', 12.5, [ 3, 4, 5], '你好' ]
>>> 删除 甲[0], 甲[1], 甲[2]  # 或者也可以用 "删除 甲[0:3]"
>>> 写 甲
[ '你好' ]
>>> 甲[:0] = [ 12 ] # 插入一个序列
>>> 写 甲
[ 12, '你好' ]
```

字典: 字典其实也可以想像成一个阵列, 不过除了阵列中的元素可以是任何值外, 连阵列的索引也可以是任意字串或数字. 因为用起来很像字典的用法, 所以就叫字典型态了. 学术名字大概叫"关键字索引式阵列".

```
>>> 甲 = { '我': 12, '你': 13, '他': 18 }
>>> 写 甲 # 注意字典中各条的先后次序是不定的 !
{ '你' : 13, '我': 12, '他': 18 }
>>> 甲['我']
12
>>> 甲['我们'] = 22  # 可以随时加入新条
>>> 甲.索引() # 取出字典中的所有条目
[ '我们', '我', '你', '他' ]
>>> 甲.值() # 取出字典中所有的值
[ 22, 12 ,13, 18 ]
```

另外一个学名是'拼揍表'. 字典类是无法直接做子项抽取的, 必须先取出索引值. 意思就是如果'甲' 是一个字典类, '甲[1]' 这样句法是错的, 要用'甲.索引()[1]' 才行.

输入输出: 写变量: 把变量显示到萤幕上, 另起一行

```
写 变量1, 变量2, 变量3: 依之写出变量组

写 变量1, 变量2, : 如果最后加上 ',', 则不另起新一行

读入(指示字串): 读取用户的键盘输入

文件 = 打开(档案名, 'r'): 打开文件档以备读取资料
文章 = 文件.读(): 将文件内容全部读入, 资料为字串型态
文章 = 文件.读一行(): 依次读取文件中的一行, 资料为字串型态
文章 = 文件.读多行(): 全部读入, 依 '回车' 键分成多行, 资料为序列型态
文件  关闭(): 关闭文件.

文件 = 打开(档案名, 'w'): 打开文件档以备写出资料
文件.写(字串): 把字串内容写到档案中去
文件.关闭(): 关闭文件. 注意文件书写完, 很多时只是写到暂存区中,
有时要关闭文件才能把资料真正存到档案中.
```

熟悉你的周围: 中蟒提供了一些功能让你检视整个中蟒即译器的状态. 例如定义了哪些变量, 载入了什么模组, 内建函数的用法等. 这些功能被唤做'自省' 功能. 可用的主要有

```
内容(): 目前的变量表里都有什么
内容(变量): 在 '变量' 中都定义了些什么
总内容: 和 '内容' 是一样的不过同时也显示英文项. 因为中蟒尚未完全翻译完,
所有有些功能必须参考英文版.
共用变数(): 整个程式中的全局变量表
私有变数(): 目前执行域的变量表
代号(变量): 中蟒为 '变量' 指派的代号, 如两个变量代号一样则两变量是完全相同的.
(在记忆体中占相同的地址)
```

3 流程控制: 判别式和回圈

判别式: 在程式的运行中, 很多时需要根据不同的情况做相应的运算. 为了处理这些不同的情况, 我们需要把每个情况下应做的操作全部写好. 这些称为程式中的分枝. 中蟒提供了判别式"如.. 不然.. 否则" 来让程式决定该执行哪一条分枝.

例子:

```
答 = 整数(输入("请告诉我你的年纪: "))
如 答 == 0:
   写 "别开玩笑了, 你刚出生吗 ?"
不然 答 < 0:
   写 "哇! 妖怪!"
不然 答 > 200:
   写 "哇! 妖怪!"
   写 "不! 是老妖怪!"
不然 答 < 40:
   写 "嘿! 小伙子"
否则:
   相差 = 答 - 1
   写 "你好, 你的年纪比中蟒大", 相差 , '岁.'
```

"如" 后面要跟着一个判别项, 判别项可以是任何能化简的表达式. 如该表达式为真的话则执行冒号后面的指令.

冒号":" 中蟒用来区分判别式的终了, 因此不要忘了加.

注意执行多少行指令是由程式码的缩排来决定的. 也就是说:

```
如 2 < 1:
  写 "2<1",
写 "当然了"
```

和

```
如 2 < 1:
  写 "2<1",
  写 "当然了"
```

这两个程式段的结果是不同的. 第二个例子中'写"当然了"' 是属于'如' 的码区中, 所以'如' 判别失败后该指令不会被执行.

'如' 判别失败后, 程式会继续判别之后的'不然' 语句. 如有匹配的话则会执行其下的指令. 当所有判别都失败后程式的流程会分枝到'否则' 之下.

'不然' 和'否则' 判别式都是可有可无的. 但要注意是判别式中随便一项为真, 则程式执行完其下的指令后会直接跳到整个判别区之下继续运行下面的程式码. 就是说

```
甲 = 20
如 甲 > 1:
  写 '甲大于 1'
不然 甲 > 2:
  写 '甲大于 2'
不然 甲 > 3:
  写 '甲大于 3'

写 '完成'
```

这个程式写完'甲大于1' 后就直接跳去写'完成' 了.

判别式容许嵌套形式, 就是说你可以在'如' 之下的码段中用第二个, 第三个'如' 语句, 只要你的程式书写依照合理的缩排.

回圈式: 程式有时需要重复执行同一运算许多次. 比如计算从1 加到100 的和, 就要算100 次加法. 这个情况可利用回圈运算来完成. 中蟒有两类回圈: '取.. 自..' 和'只要..'

```
和 = 0
循环子 = 0
只要 循环子 < 100:
    和 = 和 + 循环子
    循环子 = 循环子 + 1
写 和

和 = 0
取 循环子 自 范围(0, 100, 1):
    和 = 和 + 循环子
写 和
```

上面示范了两种方法. 留意上面只是算从0 加到99 的值而不是1 到100.

第一种结合了判别方式, 程式员需要在回圈中调整"循环子" 的值. 如果程式中漏了"循环子+ 1" 这句, 那程式会一直不停地运算下去.

第二种则给出循环范围是从0 到100 每次递增1. 范围是中蟒的内建函数, 它可以产生一个数列.

两种方法都各有长处. '只要' 回圈中循环子的值可以随便更改, 因此可以任意决定何时跳出回圈. '取..' 回圈则可以取值自任何序列, 对遍历序列很有用, 如:

```
取 算子 自 [0, 25, 3, 19, '完']:
    写 算子,
```

跳出回圈: 中蟒提供了'下一个' 和'中断' 两个指令. '下一个' 更新目前回圈的循环子值再重新执行, '中断' 则干脆跳离回圈.

回圈也是可以嵌套的.

4 定义函数: 结构化编程概念

中蟒有'定义' 和'函数' 两个关键字来定义函数. 这两个字是通用的.

很多人都爱把常用的指令组用一个代号(函数名)表示, 当程式需要该项功能时便只需调用该函数. 多用函数可以养成编写结构化程式的习惯.

例子:

```
定义 问电话(人名, 关系 = '朋友'):
     """这里是说明文件, 随便你打什么. 不写也行
如果写了的话会成为 "问电话.__说明___" """
     问句 = "请问你中蟒的" + 关系 + 人名 + "家的电话是多少号 ? ")
     答 = 输入(问句)
     传回 人名 + 关系 + 答

电话1 = 问电话('青竹蛇')
电话2 = 问电话('大蟒', '爸爸')
电话本 = [ 电话1, 电话2 ]

取 名字 自 [ '腹蛇', '响尾蛇', '大懒蛇']:
    电话本.附加(问电话(名字))
```

(记住要加冒号)

每个函数在调用时可以用不同的参数, 像上面那样. 不用参数也行, 不过函数的弹性也较低.

中蟒允许预设参数. 像上面的'关系' 值, 如果调用时不给出该参数值则会使用预设的值.

有时候函数需要用任意数量的参数, 简单的做法是把所有参数包在一个元组或序列中传递.

嵌套函数也可以, 不过通常情况下并不需要这样做.

不过要记住, 因为中蟒是即译语言, 用太多函数会造成系统的负担, 这是因为每次调用函数都要经过一输寻找, 运行域转换等的工作. 其实就算是编译语言, 函数最好也不要用的太滥.

5 类别/对象: 用概念为主的编程法代替以功能为主的编程

这里必须要指出, '概念' 其实就是面向对象中的对象, 也就是物件导向中的物件. 中蟒有两个互通的保留字"概念" 和"类别", 它们是一样的.

为什么要另外用一个新名字? 我认为无论是面向对象还是物件导向的说法都会令初学者摸不着头脑. 好像我当初听了半天都不明白, 于是以为是很深奥的东西. 现在有机会重新弄一套程式语言了, 我研究了一下, 觉得'概念' 的讲法比较浅白. 当然这个名字会太笼统了, 因此保留了'类别' 这个保留字, 如果你丝毫不觉得'概念' 是个好主意的话, 当它不存在好了.

这是一个不算深, 但很烦的题目. 我还没想好该怎样表达. 因此暂时只好用一些例子, 希望你能掌握到其中

的要点.

```
#定义一个概念
概念 印刷员:

    #定义这个概念的初始动作, '__初始__'是中蟒的特别函数.
    #而 '自己' 则是概念本身, 中蟒规定概念中的函数 (称为方法)
    #的第一个参数一定是概念本身. 至于叫什么名字可以随便改,
    #一般叫做 '自己'. ('我' 也不错)
    定义 __初始__(自己, 名字):
自己.名字 = 名字

    #定义本概念的某个功能
    定义 印刷(自己, 稿件):
写 "由" + 自己.名字 + "印"
写 "<html>" + 稿件 + "</html>;"

    #定义另一个功能
    定义 检查(自己, 稿件):
写 "稿件长度为:", 长度(稿件)

#开始操作

小王 = 印刷员('王小明')
小李 = 印刷员('李四')

小王.印刷('这是一份广告')
写 '完成'
小李.检查('论文一篇')
```

执行结果:

```
由王小明印
<html>;这是一份广告</html>;
完成
稿件长度为: 8
```

可以看到所谓的概念, 其实像是一整套完整的, 自给自足的小程式. 它包括了自己的变数(称为属性), 自己的函数(称为方法).

用概念的方法来为程式进行分工往往有很好的成效. 尤其是所写的功能可以很容易的重用. 这是因为在分析的过程中, 互相牵连的部份已经尽量分开了.

在上面的方法定义中, 我们并没有传回任何值. 一个更好的做法是传回概念本身:

```
...
    #定义另一个功能
    定义 检查(自己, 稿件):
写 "稿件长度为:", 长度(稿件)
传回 自己
...
```

我们可以这样调用: 小王.检查(稿).印刷(稿)

或者我们传回稿件:

```
...
    #定义另一个功能
    定义 检查(自己, 稿件):
写 "稿件长度为:", 长度(稿件)
传回 稿件
...
```

这样我们又可以用

```
小王.印刷(小王.检查(稿))
```

一个是以工作人员的动作为主, 一个是以稿件的流程为主. 这样写传回值, 实际操作时会带来很大方便

如果按上面的写法再写多几个概念出来, 写程式就可以更加接近日常的操作:

```
#假设已定义好概念

小王 = 印刷员('王小明')
小张 = 外勤('张三')
段总 = 编辑('段正淳')
老朱 = 会计('朱子柳')

#实际操作

作者 = '老粗'
老朱.开稿费(小张,500).拿稿件(作者).交编辑(段总).正稿().付印(小王).印刷()
```

哈, 好玩吗?

6 写作模组: 让你的编程工具箱功能倍增

模组其实就是存成.py 档的函数定义或概念定义. 中蟒鼓励重用你已写好的代码, 模组的设计使这项工作变的很容易.

假设你写了这样两个函数并把它存成"费氏.py"

```
定义 数项(上限):
    """ 本函数计算不大于 '上限' 值的费氏级数项, 传回该项值. """
    甲, 乙 = 0, 1
    只要 乙 < 上限:
甲, 乙 = 乙, 甲 + 乙
    传回 乙

定义 数列(上限):
    """ 本函数计算不大于 '上限' 值的费氏级数列, 传回该级数列 """
    甲, 乙 = 0, 1
    答 = []
    只要 乙 < 上限:
答.附加(乙)
  甲, 乙 = 乙, 甲 + 乙
    传回 答
```

你可以在别的程式中调用这两个函数了:

```
>>> 载入 费氏
>>> 内容(费氏)
['__档案__', '__名称__', '__说明__', '__内建__', 数项, 数列 ]
>>> 费氏.数项(100)
89
>>> 费氏.数列(100)
1 1 2 3 5 8 13 21 34 55 89
>>> 费氏.数列.__说明__
'本函数计算不大于 '上限' 值的费氏级数, 传回该级数列'
>>> 费氏.__档案__
'费氏.py'
>>> 甲 = 费氏.数项
>>> 写 "黄金比例接近于: " , (浮点数)(甲(60)) / 甲(100)
黄金比例接近于: 0.6179775280898876
```

"载入" 是中蟒的保留字, 用来读取模组档的内容. 模组档其实和普通中蟒程式档没分别. 中蟒会把该模组档中的函数, 变量等指派到模组名下. 用的时候把该模组当成是一个概念就行了, 它也有自己的成员函数, 属性等. 中蟒这样做, 大大简化了扩展的步骤, 再也不用为占用了相同的函数名称而头痛了.

除了"载入" 外, 还有别的载入方法:

```
>>> 载入 费氏 名 无名氏
>>> 无名氏.数项(100)
89

>>> 从 费氏 载入 数项
>>> 数项.(100)
89
```

注意! 用"从模组载入名称" 的方法, 函数会被加到目前的名称空间中. 也就是说, 你不会看到'费氏' 这个名字而只会算到'数项'. 另外, "从模组载入*" 的意思是说载入该模组中所有的东西. 可以想像成把模组中的程式码搬到目前的码段中再执行一遍(事实上有些分别, 但无妨).

7 保留字: 所有中蟒关键字表

写，删除，定义/函数， 忽略， 中断，

下一个，传回，示警， 载入...名...，从...载入...，

共用，执行，断言，如..不然..否则..， 只要..否则..，

取..自..然后..， 试..失败..否则..然后..，
概念/类别，来自/不来自，是/不是

写 ..:
将 .. 输出到萤幕．中蟒会自动加上断行符号，如果你不想中蟒自动为你断行的话可以在最后加一个逗号 ’,’.

删除 ..:
在变数表中删除记项，并释放出所占的记忆体．

定义/函数 :
告诉中蟒接下来会定义一个函数功能．在程式中，在调用函数前必须先定义好．因为中蟒是即译执行的关系，因此编程时上下文必须顺次序．

忽略 :
什么也不做，多数用在捕捉异常态后回复原执行态

中断 :
跳出回圈用

下一个 :
在回圈中直接跳到下个循环

传回 :
指定函数的传回值

示警 :
向中蟒即译器报告一个系统异常

载入 ... 名 ... :
载入模组，并可以为该模组指定新的参考名称

从 ... 载入 ...:
载入模组中的部份功能至目前的运行空间

共用 :
宣示变量为全局变量．在程式的任何地方都可以读取全局变量

执行 :
执行一段程式码．可以是档案或是内装程式码的文字串

断言 :
检查断言条件是否符合，不合的话会产生错误报告．

如 .. 不然 .. 否则 .. :
判别式流程控制．

只要 .. 否则 .. :
判别回圈型．只要条件为真会一直重覆直到条件不符．’否则’ 项很少用．

取 .. 自 .. 然后 .. :
范围回圈型．循环子依次取自一个序列中的元素．
一般用 ’范围()’ 函数来产生一个整数序列做为循环子用．
注意! 虽然在执行可以更改循环子或范围序列但这样会另程式很难看得懂．

试 .. 失败 .. 否则 .. 然后 .. :
容错执行方式．如果 ’试’ 下的码段出错会依错误情况跳到 ’失败’ 下的码段．
’失败’下的码段会尝试解决错误或’忽略’错误．如没有适当的错误项则跳到 ’否则’ 下的码段．

概念/类别 :

8 内建函数参考

内建函数参考: 请参阅¡a href=”../ref/ref.html”¿参考手册¡/a¿

A 编程范例

此页收集了一些有用的中蟒程式范例. 以简单, 有趣, 精巧为主. 主要作为学习参考用.

A.1 为中蟒加油

```
#!/usr/local/bin/cpython
回答 = 读入('你认为中文程式语言有存在价值吗 ? (有/没有)')

如 回答 == '有':
写 '好吧, 让我们一起努力!'
不然 回答 == '没有':
写 '好吧,中文并没有作为程式语言的价值.'
否则:
写 '请认真考虑后再回答.'
```

这个程式示范了如何取得用户输入和程式的流程控制.

同时它也带出了大五和国标编码下的通用程式应如何写作. 如果你是在用linux **的话, 利用像**rxvt **加**xcin **或**chinput **等的程式是可以运行两种不同编码的终端. 不然无法比较分别.**

在把信息统一转成了内部编码后再调用字串的”**调整编码**()” **方法把字串换成目前即译器起动时的编码. 这样就可以做到无论在何种编码下运作都可以得到相同的答案了.**

```
另外是 "#--BIG5-" 和 "#--GBK--" 的暗号. 它们告诉中蟒程式档的编码方式.
```

大多数情况下大五码(或国标码)的程式如果利用了连三引号和暗号, 写的程式/模组应可自动在不同的编码下行无误. 不必另外转码. 不过有时候也会有异常, 如果确定是中蟒的bug **可以通知开发小组. 这些异常一般可以用**’**强设编码**()’ **的方式解决.** –¿

这里示范了判别式分流, ”**如**– **不然**– **否则**” **等于是英文的**”if – else if – else” **的用法.**

要特别留意程式中码区的分段方法. 跟c, perl **很不一样的是你看不到有**””, ”” **这样的符号. 取而代之的是一致的缩排方法.**

A.2 中蟒网站的繁简互换网关程式

因为中蟒内建了对大五和国标码的支持. 所以这变成一件十分容易的事.

```
#!/.../cpython -bS

载入 os
写 "Content-type: text/html\n\n"
文件 = os.getenv('PATH_TRANSLATED')
内文 =  打开(文件, "r").读入(20000).大五变国标()
写 内文.原始码()
```

只有6 **行而已**!!

第一行是告诉伺服器到哪里去找中蟒即译器. 这是unix **上常用的方法**. -bS **选项指令即译器载入时用大五作为目前编码**(b **是**BIG5 **的意思**), S **则代表不必载入**site.py **这个模组. 这样可以快一些**.

第二行: **载入**os **模组. 因为我们需要问**'PATH_TRANSLATED' **这个环境变量, 而查问环境变量的功能在**os **模组中. 目前该模组还没有中译本, 因此用英文原版就行**.

第三行: **写出**HTTP **通讯协定的标头**.

第四行: **取得要作繁简互换的文件名称. 如果你给出的**URL **为**

```
http://chinesepython.sourceforge.net/cgi-bin/cgb/mypath/file.hthml
```

则环境变量会设成"**本机路径**/mypath/file.html", **你可以用该路径找到该档的位置**.

第五行: **望文生意可也. 打开刚才的文件, 读入最多**20000 **个位元(大约**1 **万个字了, 普通网页应该够用有余了. 然后把读进来的内容(应为大五码) 转成国标. 转换结果放到**"**内文**" **这个变量中**.

第六行: **印出**"**内文**" **的原始码(国标码). 注意: 因为我们启动中蟒即译器时是用了大五码作目前中文编码, 因此如果直接写出**"**内文**" **的话, 中蟒会尝试把国标码的内文以大五方式显示, 所以要叫它直接印出**"**内文**" **的原始码**.

A.3 基因字典模组

汉字基因字典

汉字基因字典是朱邦复先生编修的字典. 字典内容多取自康熙字典, 但加入了汉字基因的概念: 每个单字以其基因字首字身加以解释. 为学习"**汉字**" **的好材料. 本模组作为示范用途, 仅供查询单字, 并不涉及朱老汉字基因的诸种妙用**.

```
#--BIG5-- (此行为暗号, 向中蟒表明本文件用大五编码)
从 sys 载入 modules
从 os.path 载入 dirname, join
基因字典 = {}
路 = dirname(modules[__名称__].__档案__)
取 文件 自 [ 'A1', 'B1', 'C1', 'D1', 'E1', 'F1', 'G1', 'H1', 'I1', 'J1',
    'K1', 'L1', 'M1', 'N1', 'O1', 'P1', 'Q1', 'R1', 'S1', 'T1',
    'U1', 'V1', 'W1', 'X1', 'Y1' ]:
全文 = 打开(join(路,文件+'.TXT')).读入().强设编码('大五')
全文 = 全文.替换('\r\n','\n')
条列 = 全文.分割('\n@')
取 条 自 条列:
目 = 条[2:4]
基因字典[目] = 条
写 """共得 %i 条""".强设编码("大五")%(长度(基因字典))
删除 全文, 目, 条, 条列, dirname, join, modules, 文件

定义 查(字):
如 长度(字) != 2:
写 """只能查询单字""".强设编码("大五")
传回 空
如 字.字串编码() == '国标':
字 = 字.国标变大五()
如 基因字典.有(字):
传回 基因字典[字]
否则:
传回 """字典中无此字\n""".强设编码("大五")

定义 问(字):
写 查(字)
```

基因字典的内文如下:

```
@【日】      日  囚     Ryh
象形－甲骨文
  太阳。
     ：  时日：太阳出没一次为一日，分为二十四时。
落日：下落中的太阳。
          日晖：日光影响下，人的感受。
日光：日发出强烈光线，人得以见。
日圭：古代以日影计时，其计时器名日圭、日规或日晷。
日历：记载时间之历表。
日子：指定词。指生活所经历的时间。
日记：逐日记录之文稿。
     ：  明日：主观立场所在的下一天。
昨日：主观立场所在的上一天。
今日：主观立场所在的当天。
前日：主观立场所在的上两天。
                    日蚀：月绕行至日及地球之间，遮没了太阳，谓之日蚀。
日落：日行至西天，时已晚，落于地平线下。
日出：日由东方升起，是一日之始。
@【昌】       日日  亻尢  Chang
字首［日］：  太阳。
字身［曰］：  张口说。
会意－甲骨文
  说太阳之光，明亮，光大，美好。
组合字：????秧倡唱菖娼??阊猖??鲳??
     ：  昌言：人性以光大为美，是为美好之言。
     ：  昌盛：光大的程度高盛。
昌隆：光大的程度兴隆。
:
:
:
:
:
:
:
```

该字典内容公开. 可到http://www.cbflabs.com 下载. 如果你下载中蟒则已包括该字典.

一般来说中蟒写的程式都很容易明白, 基本上不要解释. 但因为这个字典在大五及国标环境下都可运作, 然而两种编码同时操作很容易把人弄糊涂, 所以要特别说明一下:

基因字典本身是大五码的, 所以读进来的字都是大五编码的字. (注意字串的编码记号是和即译器的目前编码一致的, 因此和文本实际的编码不一定吻合)

即译器运行在国标编码下的话, 用户输入的查询字应为国标码, 所以要先检查是否需要转换.

在显示文字上, 因为中蟒在写字串时会检查目前编码和字串的编码作自动转换, 所以不必操心. 那些使户讯息如"只能查询单字" 什么的就要处理一下. 用'强设编码("大五")' 这个函数为每个要输出的讯息设好编即成.

三引号的用法在新版2.1.3-0.4 中已取消. 那些使户讯息如"只能查询单字" 什么的就要处理一下. 有两个方法, 第一个是用强设编码的方法指定该字串的编码, 另一个方法就是用三个引号把它们包起来, 三引号在python 中经常用来作说明文件用, 在中蟒的情况下被三引号包住的字串会自动做编码转换. 这样就省去了许多手续.

注意! 普通字串中一般会设为中蟒启动时的编码设定, 这是因为要方便处理二进位的资料流. 不问三七二十一都转换编码会弄的一团糟.

A.4 中蟒内建函数说明文件产生器

参考手册自动产生器

```
定义 做文件(名字, 实体, 档名):
  内文 = "<h3>%s的操作</h3>\n"%(名字)
  表 = 内容(实体)
  如 名字 == '"__内建__"':
     表 = [ 甲 取 甲 自 表 如 甲[-4:] != '异常' ]
  内文 += '''<table width=95% align=center bgcolor=#ddeeff cellspacing=5
cellpadding=5 cols=2>'''
  取 条 自 表:
     内文 += "<tr><td width=100 valign=top>%s</td>\n"%(条)
     说明 = '实体.' + 条 + '.__说明__'
     试:
       内文 += "<td>%s</td></tr>\n"%(推算(说明).替换('\n','<br>\n'))
     失败:
 忽略
  内文 += "</table><br><br>"
  文件 = 打开(档名+'.ht', 'w')
  文件.输出(内文)
  文件.关闭()
  写 "<a href=%s.html>%s</a><br>\n"%(档名, 名字)

写 "<h3>参考手册(自动产生)</h3>"
做文件('"__内建__"', __内建__, 'builtins')
做文件('字串', '字串', 'string')
做文件('序列', ['序列'], 'list')
做文件('字典', {'字典':'字典'}, 'dict')
做文件('档案', 打开('/dev/null','w'), 'file')
载入 系统
做文件('"系统" 模组', 系统, 'sys')
载入 异常
做文件('"异常"', 异常, 'exception')
```

这个程式可以看出中蟒所拥有强大的自省功能和线上说明系统.

B 中英保留字对照表

假如你已很熟Python 的用法, 这里只记录中英对照版本, 可以作为快速入门.

```
写 ..(print)
删除 ..(del)
定义/函数 (def)
忽略 (pass)
中断 (break)
下一个 (continue)
传回 (return)
示警 (raise)
载入 ... 名 ... (import ... as ...)
从 ... 载入 ...
共用 (global)
执行 (exec)
断言 (assert)
如 .. 不然 .. 否则 .. (if .. elif .. else ..)
只要 .. 否则 .. (while .. else ..)
取 .. 自 .. 然后 .. (for .. in .. else ..)
试 .. 失败 .. 否则 .. 然后 .. (try .. except .. else .. finally .. )
概念/类别 (class)
来自/不来自 (in / not in)
是/不是 (is / is not)
或 or
且 and
不是 not
```

-
-
-