* # -\*- coding: UTF-8 -\*-
* Python运算优先级（高→低）：括号、指数、乘、除、加、减
* **格式化字符串**时，Python使用一个字符串作为模板。模板中有格式符，这些格式符为真实值预留位置，并说明真实数值应该呈现的格式。Python用一个tuple将多个值传递给模板，每个值对应一个格式符。这些值可以是数字、字符串、布尔值、变量等，同时也可以直接调用有对应数量返回值的函数→ex24.
* “a” \* 10 == aaaaaaaaaa
* **print “abcde”,**

**print “edcba” 输出在一行，且两个字符串之间有一个空格，即输出：abcde edcba**

* 犯的错：为了美观，tuple多个值进行了分行显示，但是每行后面按忘了加逗号
* 字符串中输入\n即可在显示中换行；也可以用三引号””” “””或者’’’ ‘’’来实现
* input与raw\_input
* 输入为纯数字时: input返回的是数值类型，如int,float；raw\_inpout返回的是字符串类型，string类型，因此若需要数字，需要**int（raw-input（））**
* 输入字符串时，input需要输入’ ’或者” ”，raw\_input直接输入字符即可
* help(math.pow)，查看math模块中pow函数的帮助。pow（x，y）即求xy，其等于x\*\*y
* open()的参数需是字符串类型（字符串或者**代表字符串**的变量）
* pydoc是python自带的一个文档生成工具，使用pydoc可以查看类和方法结构，主要用于从python模块中自动生成文档。在命令行中，python –m pydoc x。x有以下四种：
* 文件名。可以直接查看文件（要先进入该文件所在目录；或者键入路径）
* -k <keyword>。通过关键词查找
* -p <port>。启用本机上给定端号的本地服务
* -w <name>。生成html说明文档
* -b
* -g
* 模块（可以是python自带的，也可以是**自己写的py文件**）

1、创建模块：创建一个扩展名为.py的文件就相当于创建了一个模块，**文件名即为模块名**，文件中的内容即为模块所包含的内容。

　　2.导入模块：**import** 模块名

　　3.使用模块：模块名.变量名（可为普通变量或函数或类等）

　　4.导入模块中的某个元素：**from** 模块名 **import**变量。（可多级导入）

　　5.导入模块中的所有元素：**from** 模块名 **import** **\***。

**注：2和4、5的区别：使用2方式时，引用模块中的元素需要使用“模块名.变量名”的形式，而4、5可直接使用变量名。同时，2是导入的“模块”，可以使用help（模块名）获得帮助，而5是导入的模块中的具体的东西，不可以用help（模块名）**

* 文件操作 <https://www.cnblogs.com/heyongqi/p/5565651.html>

<http://www.runoob.com/python/file-methods.html>（有其他知识）

* Write（命令有行参数时，可以用加号连接）：target.write(line1 + "\n" + line2 + "\n" + line3 + "\n")
* 函数有多种参数传递方式：数字、字符串、变量、数学表达式、数学表达式与变量合起来用（**用 + 连接**）
* 函数定义：以def 开头，函数名由字符、下划线和数字（不能开头）组成，函数名紧跟着括号，括号里可以有0个、1个、2个及以上（逗号隔开）参数，参数名称不能重复（不同函数中参数名可以相同），紧跟函数定义的代码使用4个空格（一个tab）缩进，函数结束的位置取消了缩进。
* 函数运行(run) / 调用(call) /使用(use)：使用函数名，函数名紧跟着（，括号中参数与定义中对应，以）结尾。
* **习题22**

|  |  |
| --- | --- |
| print（） | 打印（输出）。**打印中文时**，需在前面加u：  print u‘我是派森  如果是从excel等office系列软件或者从windows系统里导入内容，请写出 temp\_str.decode('gb2312')  再有一些从网上爬下的数据可能要写成 temp\_str.encode('utf-8')’  在window下，系统编码方式是gb2321，可先解码在编码：  **print ’下载完成’.decode("utf-8").encode("gb2312")** |
| # | octothorpe/pound character 注释符号 |
| -\*- coding： utf-8 -\*- | 编码格式声明 |
| + - / \* % < > <= >= | 数学运算符 |
| 变量 | 指代数字、字符、其他变量等等的名字 |
| = | 赋值符号，将=左边的值（量）给右边的变量 |
| \_ | underscore 下划线 |
| 浮点数（float）和整数（int） | —— |
| round（） | 将浮点数四舍五入 |
| %s %r %d %f | 格式控制符，分别显示给用户的输出（用于显示），原始数据（用于调试，会使转义字符失灵），数字  **在%与字母之间可以加入控制有效位数的值。**例如，  %3d 表示输出3位整型数,不够3位右对齐（即补空格）  %03d表示输出3位整型数,不够3位右对齐（前面补0）  %9.2f 表示输出场宽为9的浮点数,其中小数位为2,整数位为6,**小数点占一位**,不够9位右对齐。 （注意：小数点前的数字必须大于小数点后的数字。**小数点**前的数值规定了打印的数字的**总宽度**。如果忽略了（如：%**.**2f），则意味着总宽度无限制。） |
| “ ” ‘ ’ | 双、单引号，用来标识字符（串） |
| \ | 反斜杠， |
| **+** | **可以用于字符串的连接** |
| formatter = ‘%r %r %r %r’ | 连续几个格式控制符的格式 |
| ‘‘’ ‘’‘、“”“ “”” | 三引号，可以加入任意多的行而不用加入换行符 |
| \\ | 转义字符，\ |
| \’ \” | 转义字符，单引号、双引号 |
| \a | 转义字符，响铃符 |
| \n \r | 转义字符，换行符、回车符 |
| raw\_input() | 输入，返回值是**字符；命令行参数（包括输入的数字）也是字符，需要先用int（）转换** |
| input() | 会把输入的东西当做python的代码来处理，这样会有安全问题 |
| print “”,  “” | 逗号可以使上下两行内容在一行 |
| print “... %r...” % (  ...) | 为了使每行不超过80个字符，可以用这种形式，但要使% ( 在一行 |
| pydoc | 文档生成工具，使用pydoc可以查看类和方法结构 |
| from sys import argv | 从sys模块将argv参数变量导入脚本，需在执行命令时输入与解包时数量对应的变零名 |
| help(math.pow) | 查看math模块中pow函数的帮助 |
| fileobject.open(file, op) | op无时，默认为‘r’模式打开；以‘w’模式打开时，若文件不存在则会被创建 |
| fileobject.read() | 返回文件中剩余的文本组成的多行字符串，若打开文件时调用则返回文件中的所有内容（即使用read()之前没有使用readline()） |
| fileobject.write() | 向文件中写内容，写入的内容不会自动换行，但可使用换行符“\n”；多行字符串可以用 + 连接 |
| fileobject.close() | 对文件操作后要关闭文件；但若对文件执行了.read()方法后，文件就已经关闭，不必再进行.close() |
| fileobject.truncate() | 清空文件内容，但若用‘w‘模式打开文件，则会自动清除文件，不必再用该命令；若不清除内容进行添加，用‘a’模式开 |
| from os.path import exists | 从os.path模块中导入exists变量方法至脚本 |
| exists(filename) | filename变量代表的**文件**若存在，则返回True，反之返回False |
| len() | 以数的形式返回你传递的字符串长度; 计算列表的元素个数； 计算字典的总长度(即键值对总数)；计算元组元素个数：t = ('G','o','o','d')，len(t) →4 |
| def | 新建函数 |
| def print\_two(\*args): | \*args表示把所有参数都接收进名叫args中的列表中去（类似于argv的功能），之后解包 |
| 函数和变量的命名 | 以字母、数字与下划线组成，不能以数字开头 |
| fileobject.seek(0) | 返回到文件0字节处（即开头），seek的处理对象是字节。详细用法见上面链接 |
| fileobject.readline( x ) | readline()里的代码会扫描文件的每一个字节，直到找到一个\n为止停止读取文件，并且返回此前文件的内容。文件fileobject会记录每次调用readline()后读取的位置，在下一次执行该命令时，会从上一次的\n之后开始扫描。x表示扫描的字节数，空则表示一直扫描直到遇到\n |
| return | 定义返回值（表达式的值、数字、字符、变量等） |
| Start-Process –FilePath 网址 | 在powershell中用默认浏览器打开指定网址的网页 |
| Start-Process -FilePath chrome –ArgumentList 网址 | 在powershell中用chrome浏览器（可以指定其他的）打开指定网址的网页 |
| str.split(“x”, num)[n] | 将一个字符串分裂成多个字符串组成的**列表**。x表示分隔符（x不能为空，但“x”都没有为空格；当使用默认空格（ ）作为分隔符时，对于中间为空的项会自动忽略），num表示仅分隔 num 个子字符串（后面的所有为一个列表元素）,n表示取第n（n从0开始）个分片。若字符串中没有分隔符，则把整个字符串作为列表的一个元素  e.g.1. u1, u2, u3 =string.split('.',2) 表示将string代表的字符串分隔两次并分给三个变量  e.g.2 str = "Line1-abcdef \nLine2-abc \nLine4-abcd";  print str.split( ); #默认空格  print str.split(' ', 1 ); #‘ ’  ['Line1-abcdef', 'Line2-abc', 'Line4-abcd']  ['Line1-abcdef', '\nLine2-abc \nLine4-abcd'] |
| list.sort()  sorted() | 排序操作，默认升序。list表示一个列表容器。sort()是list的方法，对已经存在的列表进行操作，列表会被改变；sorted()是全局方法，返回的是一个新的 list，而不是在原来的基础上进行的操作，原列表不会被改变。详细与例：  http://www.runoob.com/python/python-func-sorted.html |
| lambda基础 | lambda语句中，冒号前是参数，可以有多个，用逗号隔开，冒号右边的返回值。lambda语句构建的其实是一个函数对象：  g = lambda x : x\*\*2  print g(4) 结果：16 |
| list.pop(n) | 删除列表中的第n+1个元素，n=-1时表示最后一个元素 |
| list.remove() | 递待删除元素，如果多个元素一样，默认删除第一个 |
| range(start, stop[, step]) | 创建一个**整数列表**。计数从 start 开始，默认是从 0 开始，例如range（5）等价于range（0， 5）; 计数到 stop 结束，但**不包括 stop**，例如：range（0， 5） 是[0, 1, 2, 3, 4]没有5，range（0）是[]；步长step，默认为1，例如：range（0， 5） 等价于 range(0, 5, 1)。**range（1,0）是[]** |
| list.append() | 在**列表尾部**追加元素。参数也可以是另一个列表名/列表，加入的是**一个列表** |
| list.count() | 统计某个元素在列表中的出现次数 |
| list.extend() | 在列表末尾一次性追加**另一个序列中**的**多个值**(用新列表扩展原来的列表) |
| list.index() | 从列表中找出某个**值/列表**第一个匹配项的索引位置 |
| list.insert(index,obj) | 将对象插入列表,第一个参数可以是位置 |
| list.reverse() | 反向列表中的元素，不创建反转列表的拷贝 |
| 读取列表中指定位置的元素 | 列表/代表列表的变量名[n]，n为基数，从0开始 |
| 查看默认编码值 | import sys  sys.getdefaultencoding() |

* 小插曲：**Git的基本使用（下面代码若非特殊说明，前面均有git）**

|  |  |
| --- | --- |
| init | 初始化仓库，创建一个.git文件 |
| status | 查看变化 |
| add -A | 将文件加入stage，使其成为追踪的文件 |
| commit –m “提交信息” | 将文件提交到repository中，“提交信息”就是记录的提示信息，即一条commit |
| log | 查看当前版本及之前的commit记录 |
| diff | 查看文件内容发生了那些变化；红色（前面有-）表示删除，绿色（前面有加号+）表示添加 |
| checkout -- | 撤销将文件放入repository中 |
| reset –hard 版本号前7为 | 回退到指定版本号的版本，该版本**之后**的修改都被删除。同时也是通过这个命令回到最新版本。需要reflog配合 |
| reflog | head的变更记录 |
| clean -xf | 删除所有没有被追踪的文件 |
| config --global core.quotepath false | 让git显示中文 |
| config -global user.name “用户名” | 本地配置用户名 |
| config –global user.email “邮箱地址” | 本地配置邮箱 |
| ssh-keygen -t rsa -C "邮箱地址" | 生成ssh key；直接三次回车；**这条命令前不加git** |
| ssh -T git@github.com | 测试一下有有没有连接成功 |
| ssh -T git@git.oschina.net | 对于 oschina 的 “码云” |
| ssh -T git@git.coding.net | 对于 coding 的 “码市” |
| git remote add origin 你复制的地址（一般是SSH地址） | 设置origin，与远程仓库建立联系。若在创造repository的时候，加入了README.md或者LICENSE，那么需要先运行git pull origin master |
| git push -u origin master | 指定origin为默认主机，以后push默认上传到origin上。将本地仓库上传至Github的仓库并进行关联。**初次关联需要-u**，之后就不用再加 |
| push | 将当前分支增加的commit提交到远程仓库 |
| pull | 当本地版本低于远程仓库版本的时候，获取远程仓库的commit |

* ex25.py中有较多笔记
* def break\_words(stuff):

"""This function will break up words for us."""

words = stuff.split(' ')

return words #将句子分散成词

#首行引号中的语句是 文档注释，在使用帮助命令时会显示，用以向用户说明。例如，python中运行help(ex25.break\_words)（用import ex25）或help(break\_words)（用from ex25 import \*），会显示

>>> help(break\_words)

Help on function break\_words in module ex25:

break\_words(stuff)

This function will break up words for us.

* if语句后要有冒号: ，下面的行要缩进4个空格（一个tab），若判断为True，则其紧挨着的所有的缩进行语句都会被执行。这些语句为一个if“代码块”/“语句块”
* if语句两种形式 if（条件）：代码块

if 条件：

代码块

* 在python中，只有一行以冒号结尾，它接下来的内容就应该有缩进。
* 如果多个elif块都是True，Python**只会运行**它遇到的是True的第一个块。
* if判断条件还可以简写成if X:

print()

只要x是**非零数值、非空字符串、非空lis**t等，就判断为**True**，否则为False。

* 注意if、elif（多个判断条件时使用）、else语句后都有冒号 :
* **短路逻辑：**任何以结果为False开头的and语句会被直接处理成False，不会继续检查后面的语句；任何包含True的or语句，只要处理到True，就不会继续往下计算。elif同样具有短路逻辑的性质，**if没有**，程序会检测所有的if条件。例如：

a=0

b=1

if ( a > 0 ) and ( b / a > 2 ):

print "yes"

else :

print "no"

此时，该程序能正常运行，因为if判断句是and语句，其第一个条件（a>0）是False，因此就直接返回False，不会继续计算。但若改成，if ( a > 0 ) or ( b / a > 2 ): 就会进一步计算b / a > 2，此时0做分母，程序就会报错。

* 某个数处于某个值域中，可用：1 < x <10 或者 x in range(1, 10)
* 当列表里既有数字又有字符时，循环打印要用%r，此时输出与与列表里的形式一样
* Ctrl + C可以强制停止运行中的程序
* 在**函数内部**可用**global 变量名**来把变量声明为全局变量，这样就可以正常打印
* def xh(k):

global i

while i < k: #

print "At the top i is ", i

numbers.append(i)

i = i + 1

print "Number now: ", numbers

print "At the bottom i is %d" % i

#若不加global i，则由于是在函数内部，且对i进行了<操作，i会被作为局部变量处理，而此时i这个“局部变量”显然是没有被定义的。

#若不用global i，也可以用for i in range（k），此时由于i在for开始时即被定义。

#可以看出，while中的i必须是已经定义过的变量，而for中的i是在for执行时随即定义的