**一.函数**

**1.调用函数**

Python内置了很多有用的函数，我们可以直接调用。以在交互式命令行通过help(函数)查看某函数的帮助信息。可以直接从Python的官方网站查看文档：http://docs.python.org/3/library/

调用函数的时候，如果传入的参数数量不对，会报TypeError的错误；如果传入的参数数量是对的，但参数类型不能被函数所接受，也会报TypeError的错误

**2.定义函数**

在Python中，定义一个函数要使用def语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用return语句返回。请注意，函数体内部的语句在执行时，一旦执行到return时，函数就执行完毕，并将结果返回。因此，函数内部通过条件判断和循环可以实现非常复杂的逻辑。如果没有return语句，函数执行完毕后也会返回结果，只是结果为None。return None可以简写为return。

在Python交互环境中定义函数时，注意Python会出现...的提示。函数定义结束后需要按两次回车重新回到>>>提示符下。

引用函数：from 文件名 import 函数名。

空函数：def 函数名():pass。pass语句什么都不做，那有什么用？实际上pass可以用来作为占位符，比如现在还没想好怎么写函数的代码，就可以先放一个pass，让代码能运行起来。

参数检查：调用函数时，如果参数个数不对，Python解释器会自动检查出来，并抛出TypeError；但是如果参数类型不对，Python解释器就无法帮我们检查。

返回多个值：python函数可以返回多个值，返回值是一个tuple！但是，在语法上，返回一个tuple可以省略括号，而多个变量可以同时接收一个tuple，按位置赋给对应的值，所以，Python的函数返回多值其实就是返回一个tuple，但写起来更方便。

**3.函数的参数**

定义函数的时候，我们把参数的名字和位置确定下来，函数的接口定义就完成了。对于函数的调用者来说，只需要知道如何传递正确的参数，以及函数将返回什么样的值就够了，函数内部的复杂逻辑被封装起来，调用者无需了解。Python的函数定义非常简单，但灵活度却非常大。除了正常定义的必选参数外，还可以使用默认参数、可变参数和关键字参数，使得函数定义出来的接口，不但能处理复杂的参数，还可以简化调用者的代码。

**位置参数：**调用函数时，需要依次传入的参数。

**默认参数：**def 函数名(x=??):，默认参数可以简化函数的调用。

注意：一是必选参数在前，默认参数在后，否则Python的解释器会报错。

二是如何设置默认参数。

当函数有多个参数时，把变化大的参数放前面，变化小的参数放后面。变化小的参数就可以作为默认参数。使用默认参数有什么好处？最大的好处是能降低调用函数的难度。

定义默认参数要牢记一点：默认参数必须指向不变对象！因为不变对象一旦创建，对象内部的数据就不能修改，这样就减少了由于修改数据导致的错误。此外，由于对象不变，多任务环境下同时读取对象不需要加锁，同时读一点问题都没有。我们在编写程序时，如果可以设计一个不变对象，那就尽量设计成不变对象。

**可变参数：**可变参数就是传入的参数个数是可变的，可以是1个、2个到任意个，还可以是0个。可变参数允许你传入0个或任意个参数，这些可变参数在函数调用时自动组装为一个tuple。

def 函数名(\*[]): 带\*参数接受的是一个tuple;如果参数本身就是tuple，则调用时可用函数(\*tuple)；表示把tuple所有元素作为可变参数传进去。这种写法相当有用，而且很常见。

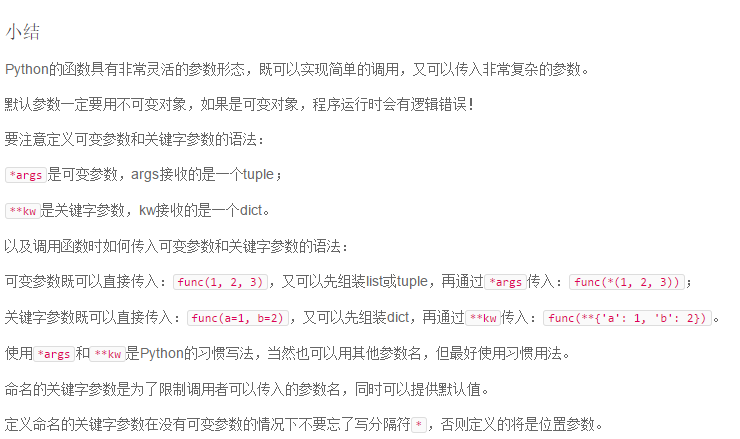
**关键字参数：**关键字参数允许你传入0个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个dict。



**命名关键字参数：**和关键字参数\*\*参数名不同，命名关键字参数需要一个特殊分隔符\*，\*后面的参数被视为命名关键字参数。如果函数定义中已经有了一个可变参数，后面跟着的命名关键字参数就不再需要一个特殊分隔符\*了；使用命名关键字参数时，要特别注意，如果没有可变参数，就必须加一个\*作为特殊分隔符。如果缺少\*，Python解释器将无法识别位置参数和命名关键字参数。

**参数组合：**



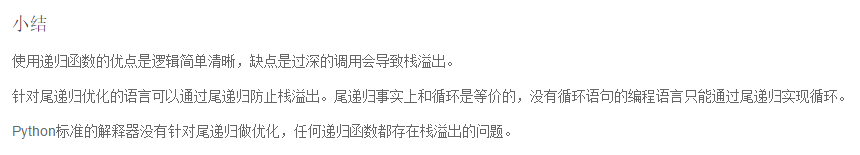


**4.递归函数**

一个函数在内部调用自身本身，这个函数就是递归函数。递归函数的优点是定义简单，逻辑清晰。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。使用递归函数需要注意防止栈溢出。在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。



解决递归调用栈溢出的方法是通过尾递归优化，事实上尾递归和循环的效果是一样的，所以，把循环看成是一种特殊的尾递归函数也是可以的。尾递归是指，在函数返回的时候，调用自身本身，并且，return语句不能包含表达式。这样，编译器或者解释器就可以把尾递归做优化，使递归本身无论调用多少次，都只占用一个栈帧，不会出现栈溢出的情况。尾递归调用时，如果做了优化，栈不会增长，因此，无论多少次调用也不会导致栈溢出。遗憾的是，大多数编程语言没有针对尾递归做优化，Python解释器也没有做优化，所以，即使把上面的fact(n)函数改成尾递归方式，也会导致栈溢出。



**二.高级特性**

**1.切片**

list[0:3]表示从索引0开始检索到索引3为止(不包含3)。如果从索引0开始时，0还可以省略不写。Python也支持倒数切片，倒数的第一个元素的索引是1。



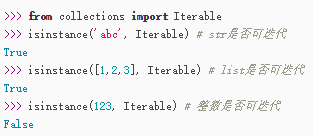
**2.迭代**

如果给定一个list或tuple，我们可以通过for循环来遍历这个list或tuple，这种遍历我们称为迭代（Iteration）。在Python中，迭代是通过for ... in来完成的。

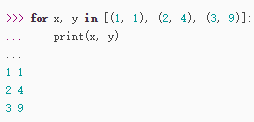
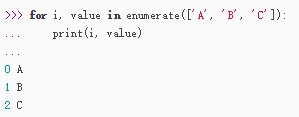
默认情况下，dict迭代的是key。如果要迭代value，可以用for value in d.values()，如果要同时迭代key和value，可以用for k, v in d.items()。

Python的for循环不仅可以用在list或tuple上，还可以作用在其他可迭代对象上。字符串也是可迭代对象。所以，当我们使用for循环时，只要作用于一个可迭代对象，for循环就可以正常运行，而我们不太关心该对象究竟是list还是其他数据类型。任何可迭代对象都可以作用于for循环，包括我们自定义的数据类型，只要符合迭代条件，就可以使用for循环。

通过collections模块的Iterable类型判断一个对象是否是可迭代对象。



Python内置的enumerate函数可以把一个list变成索引-元素对，这样就可以在for循环中同时迭代索引和元素本身。在for循环里，同时引用了两个变量，在Python里是很常见的，比如下面的两种代码形式：



**3.列表生成式**

列表生成式即List Comprehensions，是Python内置的非常简单却强大的可以用来创建list的生成式。写列表生成式时，把要生成的元素x \* x放到前面，后面跟for循环，就可以把list创建出来。for循环后面还可以加上if判断，用来指定筛选条件。



还可以使用两层循环，可以生成全排列。for循环其实可以同时使用两个甚至多个变量，比如dict的items()可以同时迭代key和value

isinstance函数可以判断一个变量是不是字符串。lower()函数可以使字符串变成小写。

**4.生成器**

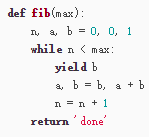
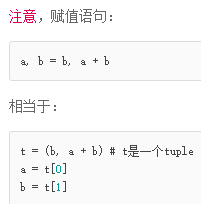
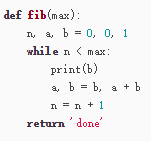
在Python中，一边循环一边计算的机制，称为生成器：generator。

要创建一个generator，有很多种方法。第一种方法很简单，只要把一个列表生成式的[]改成()，就创建了一个generator。区别仅在于最外层的[]和()，[]是一个list，而()是一个generator。

如果要一个一个打印出来，可以通过next()函数获得generator的下一个返回值。generator保存的是算法，每次调用next(g)，就计算出g的下一个元素的值，直到计算到最后一个元素，没有更多的元素时，抛出StopIteration的错误。

不断调用next(g)实在是太变态了，正确的方法是使用for循环，因为generator也是可迭代对象。所以创建了一个generator后，基本上永远不会调用next()，而是通过for循环来迭代它，并且不需要关心StopIteration的错误。

generator非常强大。如果推算的算法比较复杂，用类似列表生成式的for循环无法实现的时候，还可以用函数来实现。



仔细观察，可以看出，fib函数实际上是定义了斐波拉契数列的推算规则，可以从第一个元素开始，推算出后续任意的元素，这种逻辑其实非常类似generator。也就是说，上面的函数和generator仅一步之遥。要把fib函数变成generator，只需要把print(b)改为yield b就可以了。这就是定义generator的另一种方法。如果一个函数定义中包含yield关键字，那么这个函数就不再是一个普通函数，而是一个generator。

最难理解的就是generator和函数的执行流程不一样。函数是顺序执行，遇到return语句或者最后一行函数语句就返回。而变成generator的函数，在每次调用next()的时候执行，遇到yield语句返回，再次执行时从上次返回的yield语句处继续执行。





yield只能是在function内部。

**5.迭代器**

可以直接作用于for循环的数据类型有以下几种：一类是集合数据类型，如list、tuple、dict、set、str等；一类是generator，包括生成器和带yield的generator function。这些可以直接作用于for循环的对象统称为可迭代对象：Iterable。可以使用isinstance()判断一个对象是否是Iterable对象。

而生成器不但可以作用于for循环，还可以被next()函数不断调用并返回下一个值，直到最后抛出StopIteration错误表示无法继续返回下一个值了。可以被next()函数调用并不断返回下一个值的对象称为迭代器：Iterator。可以使用isinstance()判断一个对象是否是Iterator对象。

生成器都是Iterator对象，但list、dict、str虽然是Iterable，却不是Iterator。把list、dict、str等Iterable变成Iterator可以使用iter()函数。

为什么list、dict、str等数据类型不是Iterator？因为Python的Iterator对象表示的是一个数据流，Iterator对象可以被next()函数调用并不断返回下一个数据，直到没有数据时抛出StopIteration错误。可以把这个数据流看做是一个有序序列，但我们却不能提前知道序列的长度，只能不断通过next()函数实现按需计算下一个数据，所以Iterator的计算是惰性的，只有在需要返回下一个数据时它才会计算。Iterator甚至可以表示一个无限大的数据流，例如全体自然数。而使用list是永远不可能存储全体自然数的。

