**Python1-3**

**for循环语法结构**

**• python中的for接受可迭代对象(例如序列或迭代器)作为其参数,每次迭代其中一个元素**

**for iter\_var in iterable:**

**suite\_to\_repeat**

**• 与while循环一样,支持break、continue、else语句**

**• 一般情况下,循环次数未知采用while循环,循环次数已知,采用for循环**

**列:遍历循环**

py\_str=**'python'**alist=[**'张三'**,**'李四'**,**'王二麻'**]  
atuple = (**'赵四'**,**'孙傻'**)  
adict = {**'马五'**:**'南京'**,**'程六'**:**'北京'**}  
**for** i **in** py\_str:  
 print(i)  
**for** name **in** alist:  
 print(name)  
**for** name **in** atuple:  
 print(name)  
**for** key **in** adict:  
 print(key,adict[key])

**range函数**

**• for循环常与range函数一起使用**

**• range函数提供循环条件**

**• range函数的完整语法为:range(start, end, step =1)**

**>>> list(range(10)) 用list函数把range(10)转换成列表**

**[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**

**>>> list(range(6,11)) 不包含最后11**

**[6, 7, 8, 9, 10]**

**>>> list(range(1,11,2)) 最后的2表示布长值 起始为1**

**[1, 3, 5, 7, 9]**

**>>> list(range(10,0,-1)) 布长值是-1 从10开始反向排序**

**[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]**

**斐波那契数列就是某一个数,总是前两个数之和,比如0,1,1,2,3,5,8**

**思路 ：添加最后两位的和**

fib = [0, 1]  
**for** i **in** range(8):  
 fib.append(fib[-1] + fib[-2])  
print(fib)

**九九乘法表**

**for** i **in** range(3): 【0，1，2】外层循环控制打印哪一行  
 **for** j **in** range(i + 1): 内层循环控制打印几次  
 print(**'hello'**,end=**' '**) 取消回车 多个hello打印在一行  
 print() 每一行需要打印回车 否则就是一行

**for** i **in** range(1,10): 取值1---9 打印年9行  
 **for** j **in** range(1, i + 1): 取值1--9 [1] [1,2]...  
 print(**'%sX%s= %s'** % (j, i, i\*j ),end=**' '**)

%s是将数值转换字符输出 注意格式 顺序替换  
 print()

**列表解析**

**• 它是一个非常有用、简单、而且灵活的工具,可以用**

**来动态地创建列表**

**• 语法:**

**[expr for iter\_var in iterable]**

**• 这个语句的核心是for循环,它迭代iterable对象的所有条目**

**• expr应用于序列的每个成员,最后的结果值是该表达式产生的列表**

**快速生成列表**

**>>> [10+5 for i in range(10)]**

**[15, 15, 15, 15, 15, 15, 15, 15, 15, 15]**

**>>> [10+5 for i in range(1,11,2)]**

**[15, 15, 15, 15, 15]**

**>>> [10+5 for i in range(1,11)]**

**[15, 15, 15, 15, 15, 15, 15, 15, 15, 15]**

**>>> ['192.168.1.'+ str(i) for i in range(1,255)]**

**open及file内建函数**

**• 作为打开文件之门的“钥匙”,内建函数open()以及**

**file()提供了初始化输入/输出(I/O)操作的通用接口**

**• 成功打开文件后时候会返回一个文件对象,否则引发一个错误**

**• open()方法和file()方法可以完全相互替换**

**• 基本语法:**

**file\_object = open(file\_name, access\_mode='r', buffering=-1)**

**文件操作的基本步骤 打开 读写 关闭**

**>>> f = open('/tmp/zhuji') 打开文件没有指定打开方式默认以r打开 文件不存在则报错**

**>>> data = c.read() 默认将全部内容读入内存**

**>>> c.close 关闭文件**

**>>> print(data)**

**>>> data 可以查看到原始数据 也就是\n没有转换成换行**

**>>> f = open('/tmp/zhuji') 打开ia文件没有关闭**

**>>> f .read(10) 读取十个字节**

**'127.0.0.1 '**

**>>> f .read(10) 文件没有关闭则继续向后读取**

**' localhos'**

**>>> f .readline() 继续（文件没有关闭）读取一行**

**>>> f .readlines() 读取所有的行放在列拜哦中 每一行是列表的一项**

**文件中的数据读取完毕以后就没有了**

**>>> f = open('/bin/ls','rb') 打开文本文件必须明确指定是b（bytes）(字节)**

**>>> f.read(10)**

**b'\x7fELF\x02\x01\x01\x00\x00\x00' 前面的b字符表示bytes**

**>>> f = open('/tmp/zhuji','w') 以w打开文件 文件不存在则创建 存在则清空**

**>>> f.write('hello world\n')**

**12 表示一共写了12个字节**

**>>> f.flush() 立即将数据写入磁盘否则存在缓冲区 默认4K自动写入磁盘**

**>>> f.writelines(['2nd line.\n','3rd line.\n']) 写的是列表的值**

**>>> f.write(100) 只能将字符写入文件 不能是其它数据**

**>>>f.close() 关闭文件也会将数据写入磁盘**

**>>> f = open('/tmp/zhuji')**

**>>> f.write(asda) 报错 以r打开不能写入**

**>>> f = open ('/tmp/zhuji','a') 以追加的方式打开可以**

**>>> f.write('new line.\n') 写入行**

**10**

|  |  |
| --- | --- |
| **文件模式** | **操作** |
| **r** | **以读方式打开(文件不存在则报错)** |
| **w** | **以写方式打开(文件存在则清空,不存在则创建)** |
| **a** | **以追加模式打开(必要时创建新文件)** |
| **r+** | **以读写模式打开(参见r) 写入数据会将源文件覆盖** |
| **w+** | **以读写模式打开(参见w)** |
| **a+** | **以读写模式打开(参见a)** |
| **b** | **以二进制模式打开** |

**• seek(offset[, whence]):移动文件指针到不同的位**

**– offset是相对于某个位置的偏移量**

**– whence的值,0表示文件开头,1表示当前位置,2表示文件的结尾**

**• tell():返回当前文件指针的位置**

**>>> f = open('/tmp/zhuji','r+') 以读写方式打开**

**>>>f.write('abcde') abcde会将在源文件开头的前五个字符覆盖**

**>>> f.tell() 获取指针的位置**

**>>> f.seek(3,0) 从当前位置向前移动3个位置**

**>>> f.seek(0,2) 指针移动到末尾**

**>>> f.seek(0,0) 指针移动到开头**

**readline方法**

**• 读取打开文件的一行(读取下个行结束符之前的所有字节)**

**• 然后整行,包括行结束符,作为字符串返回**

**• 它也有一个可选的size参数,默认为-1,代表读至行结束符**

**• 如果提供了该参数,那么在超过size个字节后会返回不完整的行**

**readlines方法**

**• readlines()方法读取所有(剩余的)行然后把它们作为一个字符串列表返回**

**文件迭代**

**• 如果需要逐行处理文件,可以结合for循环迭代文件**

**• 迭代文件的方法与处理其他序列类型的数据类似**

**>>> fobj = open('star.py')**

**>>> for eachLine in fobj:**

**... print(eachLine, end= '')**

**with** open(**'/tmp/zhuji'**) **as** f :  
 **for** line **in** f:  
 print(line,end=**''**) 文件每行\n结束 print默认\n 所以end取消一个

**write方法**

**• write()内建方法功能与read()和readline()相反。它把含有文本数据或二进制数据块的字符串写入到文件中去**

**• 写入文件时,不会自动添加行结束标志,需要程序员手工输入**

**>>> fobj.write('Hello World!\n')**

**13**

**writelines方法**

**• 和readlines()一样,writelines()方法是针对列表的操**

**• 它接受一个字符串列表作为参数,将它们写入文件**

**• 行结束符并不会被自动加入,所以如果需要的话,必须在调用writelines()前给每行结尾加上行结束符**

**>>> fobj.writelines(['Hello World!\n', 'python programing\n'])**

**with子句**

**• with语句是用来简化代码的**

**• 在将打开文件的操作放在with语句中,代码块结束后,文件将自动关闭**

**>>> with open('foo.py') as f:**

**... data = f.readlines()**

**...**

**>>> f.closed**

**True**

**with** open(**'/tmp/zhuji'**) **as** f : # f = open(‘/tmp/zhuji’)  
 line = f.readline()  
print(line) 文件第一行  
f.readline() 出错 文件已经关闭

**标准文件**

**• 程序一执行,就可以访问三个标准文件**

**– 标准输入:一般是键盘,使用sys.stdin**

**– 标准输出:一般是显示器缓冲输出,使用sys.stdout**

**– 标准错误:一般是显示器的非缓冲输出,使用sys.stderr**

**模拟cp操作 练习**

**with** open(**'/bin/ls'**,**'rb'**) **as** f:  
 **with** open(**'/root/ls'**,**'wb'**) **as** c:  
 **while True**:  
 c.write(f.read())   
 **break**

**模拟cp操作 使用**

**with** open(**'/bin/ls'**,**'rb'**) **as** f:  
 **with** open(**'/root/ls'**,**'wb'**) **as** c:  
 **while True**:  
 data = f.read(4096)  
 **if not** data:  
 **break** c.write(data)

**函数基本概念**

**• 函数是对程序逻辑进行结构化或过程化的一种编程方法**

**• 将整块代码巧妙地隔离成易于管理的小块**

**• 把重复代码放到函数中而不是进行大量的拷贝,这样既能节省空间,也有助于保持一致性**

**• 通常函数都是用于实现某一种功能**

**函数是用def语句来创建的,语法如下:**

**def funcOon\_name(arguments):**

**"funcOon\_documentaOon\_string"**

**funcOon\_body\_suite**

**• 标题行由def关键字,函数的名字,以及参数的集合(如果有的话)组成**

**• def子句的剩余部分包括了一个虽然可选但是强烈推荐的文档字串,和必需的函数体**

**def** gen\_fib():  
 fib = [0,1]  
 **for** i **in** range(8):  
 fib.append(fib[-1]+fib[-2])  
 print(fib) 一般没有直接输出 这里只是示例  
gen\_fib() 调用函数 将函数内的代码运行一遍  
gen\_fib()

**• 多数情况下,函数并不直接输出数据,而是向调用者返回值**

**• 函数的返回值使用return关键字**

**• 没有return的话,函数默认返回None**

**def** gen\_fib():  
 fib = [0,1]  
 **for** i **in** range(8):  
 fib.append(fib[-1]+fib[-2])  
 **return** fib  
alist = gen\_fib()  
print(alist)  
blist = [i\*2 **for** i **in** alist]  
print(blist)

**• 形式参数**

**– 函数定义时,紧跟在函数名后(圆括号内)的参数被称为形式参数,简称形参。由于它不是实际存在变量,所以又称虚拟变量**

**• 实际参数**

**– 在主调函数中调用一个函数时,函数名后面括弧中的参数(可以是一个表达式)称为“实际参数”,简称实参**

**def** gen\_fib(n):  
 fib = [0,1]  
 **for** i **in** range(n-2):  
 fib.append(fib[-1]+fib[-2])  
 **return** fib  
a = gen\_fib(10)  
print(a)  
b = int(input(**'lenth:'**))  
c = gen\_fib(b)  
print (c)

**位置参数**

**• 与shell脚本类似,程序名以及参数都以位置参数的方式传递给python程序**

**• 使用sys模块的argv列表接收**

**Sys.argv是sys模块中名为argv的列表 用于存储位置参数**

**import** sys  
print(sys.argv)

**[root@room9pc01 ~]# python3 /root/devops/hello.py qwe**

**['/root/devops/hello.py', 'qwe']**

**默认参数**

**• 默认参数就是声明了默认值的参数**

**• 因为给参数赋予了默认值,所以在函数调用时,不向该参数传入值也是允许的**

**def** pstar(n=10):  
 print(**'\*'**\*n)   
pstar() 不给参数默认是10个 10在这里就是默认参数  
pstar(50)

**模块基本概念**

**• 模块是从逻辑上组织python代码的形式**

**• 当代码量变得相当大的时候,最好把代码分成一些有组织的代码段,前提是保证它们的彼此交互**

**• 这些代码片段相互间有一定的联系,可能是一个包含数据成员和方法的类,也可能是一组相关但彼此独立的操作函数**

**创建模块**

**• 模块物理层面上组织模块的方法是文件,每一个以.py作为结尾的python文件都是一个模块**

**• 模块名称切记不要与系统中已存在的模块重名**

**• 模块文件名字去掉后面的扩展名(.py)即为模块名导入模块(import)**

**• 使用import导入模块**

**• 模块被导入后,程序会自动生成pyc的字节码文件以提升性能**

**• 模块属性通过“模块名.属性”的方法调用**

**• 如果仅需要模块中的某些属性,也可以单独导入**

**模块加载(load)**

**• 一个模块只被加载一次,无论它被导入多少次**

**• 只加载一次可以阻止多重导入时代码被多次执行**

**• 如果两个文件相互导入,防止了无限的相互加载**

**• 模块加载时,顶层代码会自动执行,所以只将函数放入模块的顶层是良好的编程习惯**

**• 创建randpass.py脚本,要求如下:**

**1. 编写一个能生成8位随机密码的程序**

**2. 使用random的choice函数随机取出字符**

**3. 改进程序,用户可以自己决定生成多少位的密码**

**import** random  
**import** string  
b = string.ascii\_letters + string.digits  
x = input(**'你的用户名:'**)  
p = random.choices(b)  
n = input(**'密码位数:'**)  
**if** n:  
 n = int(n)  
**def** password(n=8):  
 **for** i **in** range(n):  
 p = random.choices(b)  
 print(p,end=**''**)  
 **return** n  
**if not** n :  
 password()  
**else**:  
 password(n)