**17Python\_04系统管理模块+语法风格及布局+字符串详解**

**一 系统管理模块-shutil模块**

**1.1 复制和移动**

shell utils, 实现文件的复制\剪切\删除等操作

• shutil.copyfileobj(fsrc, fdst[, length])

将类似文件的对象fsrc的内容复制到类似文件的对象fdst。

• shutil.copyfile(src, dst, \*, follow\_symlinks=True)

将名为src的文件的内容(无元数据)复制到名为dst的文件,然后返回dst。

• shutil.copy(src, dst, \*, follow\_symlinks=True)

将文件src复制到文件或目录dst。src和dst应为字符串。如果dst指定目录,则文件将使用src的基本文件名复制到dst中。返回新创建的文件的路径。

• shutil.copy2(src, dst, \*, follow\_symlinks=True)

与copy()相同,但copy2()也尝试保留所有文件元数据。

• shutil.move(src, dst, copy\_function=copy2)

递归地将文件或目录(src)移动到另一个位置(dst),并返回目标。

**通过文件对象拷贝文件(不推荐使用,推荐使用直接拷贝文件)**

>>> import shutil

>>> f1 = open('/etc/shadow','rb')

>>> f2 = open('/tmp/sd','wb')

>>> shutil.copyfileobj(f1,f2)

f1.close()

f2.close()

**直接拷贝文件**

>>> shutil.copy('/etc/shadow','/tmp/shadow')

'/tmp/shadow'

**移动文件**

>>> shutil.move('/tmp/shadow','/var/tmp/shadow')

'/var/tmp/shadow'

**1.2 目录操作**

• shutil.copytree(src, dst, symlinks=False, ignore

=None, copy\_function=copy2, ignore\_danglin

g\_symlinks=False)

递归地复制以src为根的整个目录树,返回目标目录。由dst命名的目标目录不能已经存在。

• shutil.rmtree (path, ignore\_errors=False, onerror=None)

删除整个目录树; 路径必须指向目录(而不是指向目录的符号链接)。

**移动目录**

shutil.copytree('/etc/security','/tmp/anquan')

'/tmp/anquan'

**删除目录**

shutil.rmtree('/tmp/anquan')

**1.3 权限管理**

• shutil.copymode(src, dst, \*, follow\_symlinks=True)

将权限位从src复制到dst。文件内容,所有者和组不受影响。src和dst是以字符串形式给出的路径名称。

• shutil.copystat(src, dst, \*, follow\_symlinks=True)

将权限位,最后访问时间,上次修改时间和标志从src复制到dst。

• shutil.chown(path, user=None, group=None)

更改给定路径的所有者用户和/或组

**改变文件的属主属组**

>>> shutil.chown('/tmp/sd',user='student',group='student')

[root@room9pc01 tmp]# ls -ld sd

-rw-r--r-- 1 student student 1388 11月 4 09:46 sd

**二 系统管理模块-subprocess模块**

**2.1 概述**

• subprocess模块主要用于调用系统命令

• subprocess模块允许你产生新的进程,连接到它们的输入/输出/错误管道,并获得它们的返回代码

• 本模块旨在替换几个较早的模块和功能,如os.system、os.spawn\*

**2.2 run方法**

• subprocess.run方法在python3.5引入。早期版本可以使用subprocess.call方法

**• 直接执行命令会报错**

>>> subprocess.run('ls')

>>> subprocess.run(['ls', '/home'])

>>> subprocess.run('ls /home')

**• 通过shell执行命令**

>>> subprocess.run(['ls', '~'], shell=True)

>>> subprocess.run('ls /home', shell=True)

**• run方法返回值,即shell中的$?**

>>> result = subprocess.run(['ls', '/home'])

>>> result.args

['ls', '/home']

>>> result.returncode

0

**2.3 输出和错误**

**• run方法执行的结果默认打印在屏幕上,也可以通过管道将其存储在标准输出和标准错误中**

>>> result = subprocess.run(['ls','/home'], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

>>> result.stdout

b'alice\nbob\njerry\nlisi\nStudent\ntom\nwangwu\n'

>>> print(result.stdout.**decode()**)

alice

...

wangwu

**执行系统命令,将输出保存到stdout变量中,错误输出保存到变量stderr中**

>>> rsult = subprocess.run('ls -a ~student',shell=True,stdout=subprocess.PIPE,stderr=subprocess.PIPE)

**查看正确输出**

>>> rsult.stdout

b'.\n..\n.adobe\n.bash\_history\n.bash\_logout\....’

**将正确输出转换为正常输出**

print(rsult.stdout.**decode()**)

**查看错误输出,为空**

>>> rsult.stderr

b''

**查看返回码,即shell中的$?**

>>> rsult.returncode

0

**查询一个用户是否存在**

result1 = subprocess.run('id natasha',shell=True,stdout=subprocess.PIPE,stderr=subprocess.PIPE)

>>> result1.stderr

b'id: natasha: no such user\n'

>>> print(result1.stderr.decode())

id: natasha: no such user

**三 语法风格及布局**

**3.1 语法风格-变量赋值**

**• python支持链式多重赋值**

x = y = 10

若y重新赋值y = 20,原x的值不变

>>> x = y =10

>>> y = 20

>>> x

10

>>> alist = blist = [10, 20]

>>> blist[0] = 30

>>> alist

[30, 20]

• 另一种**将多个变量同时赋值**的方法称为**多元赋值**

a, b = 10, 20

>>> a, b = 'xy'

>>> c, d = (10,20) #元组赋值时可以不要小括号

>>> e, f = ['hello', 'world']

a b c d e f

'x' 'y' 10 20 'hello' 'world'

**交换2个变量的值**

**其他语言写法: Python写法:**

>>> a, b = 1, 100 >>> a, b = 1, 100

>>> t = a >>> a, b = b, a

>>> a = b

>>> b = t

>>> a,b,c = 1,2,3

>>> a,b,c = b,c,a

**3.2 语法风格-合法的标识符**

**标识符: 各种各样的名称,如变量 函数 模块 类,统称为标识符**

**合法的标识符需要满足的条件:**

**首字符必须是字母或下划线**

**其他字符是字母 下划线或数字**

**区分大小写**

• python标识符字符串规则和其他大部分用C编写的高级语言相似

• 第一个字符必须是字母或下划线(\_)

• 剩下的字符可以是字母和数字或下划线

• 大小写敏感

**3.3 语法风格-关键字**

**关键字: 为了实现Python的语法,Python保留了一些名字,叫关键字,不能被赋值**

**if while when then for pass 等**

• 和其他的高级语言一样,python也拥有一些被称作关键这字的保留字符

• 任何语言的关键字应该保持相对的稳定,但是因为python是一门不断成长和进化的语言,其关键字偶尔会更新

• 关键字列表和iskeyword()函数都放入了keyword模块以便查阅

**查看关键字**

>>> import keyword

>>> keyword.kwlist

**两种方法判断是否为关键字**

>>> 'pass' in keyword.kwlist

True

keyword.iskeyword('pass')

True

**3.4 语法风格-內建(内置函数名),不需要导入即可使用**

**内建不是关键字,但是内建不建议赋值**

• 除了关键字之外,python还有可以在任何一级代码使用的“内建”的名字集合,这些名字可以由解释器设置或使用

• 虽然built-in不是关键字,但是应该把它当作“系统保留字”

>>> type(len)

<class '**builtin\_function\_or\_method**'> #内建的函数或方法

>>> len('abcd') #使用內建len

4

>>> len = 10 #给len赋值

>>> len('abcd') #再次使用len,此时len已被赋值,会报错

Traceback (most recent call last):

File "<input>", line 1, in <module>

TypeError: 'int' object is not callable

查看Python內建: docs.python.org/zh-cn/3/ -> 标准库参考 -> 内置函数

**3.5 语法风格-模块结构及布局**

• 编写程序时,应该建立一种统一且容易阅读的结构,

并将它应用到每一个文件中去

#!/usr/bin/env python3 #起始行说明解释器

“””模块说明文档 #也可理解为注释

用于帮助时使用

“””

import sys #导入模块

import os

debug = True #全局变量声明

hi = ‘hello world’ #定义全局变量

class FooClass(object): #类定义

'Foo class'

pass

def test(): #函数定义

"test function"

foo = FooClass()

if \_\_name\_\_ == ‘\_\_main\_\_’: #程序主体

test()

案例1:创建文件

1. 编写一个程序,要求用户输入文件名

2. 如果文件已存在,要求用户重新输入

3. 提示用户输入数据,每行数据先写到列表中

4. 将列表数据写入到用户输入的文件名中

import os

def get\_fname():

'返回一个文件名字符串'

while 1:

fname = input('文件名:')

if not os.path.exists(fname):

break

print('文件已存在,请重新输入!')

return fname

def get\_content():

'返回文件内容的字符串列表'

content = []

print('请输入文件内容,单独输入end表示结束.')

while 1:

line = input('(end to quit)>')

if line == 'end':

break

content.append(line)

return content

def wfile(fname, content):

'将content中的内容写入文件fname中'

with open(fname, 'w') as fobj:

fobj.writelines(content)

if \_\_name\_\_ == '\_\_main\_\_':

fname = get\_fname()

content = get\_content()

content = ['%s\n' % line for line in content] #添加换行符

wfile(fname, content)

**四 字符串详解-序列**

**4.1 序列类型操作符**

seq[ind] 获得下标为ind的元素

seq[ind1:ind2] 获得下标为ind1到ind2间的元素集合

seq\*expr 序列重复expr次

seq1 seq2 连接序列seq1和序列seq2

obj in seq 判断obj元素是否包含在seq中

obj not in seq 判断obj元素是否毕业包含在seq中

**4.2 內建函数**

list(iter) 把可迭代对象转换为列表

str(obj) 把obj对象转换成字符串

tuple(iter) 把可迭代对象转换成元组

list('abc')

['a', 'b', 'c']

>>> list(range(10))

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

>>> tuple('abc')

('a', 'b', 'c')

>>> tuple(range(10))

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

>>> str(10)

'10

• len(seq): 返回seq的长度

• enumerate: 接受一个可迭代对象作为参数,返回一个enumerate对象

• reversed(seq): 接受一个序列作为参数,返回一个以逆序访问的迭代器

• sorted(iter): 接受一个可迭代对象作为参数,返回一个有序的列表

>>> from random import randint

>>> nums = [randint(1,100) for i in range(10)]

>>> nums

[66, 14, 19, 5, 12, 16, 40, 94, 44, 94]

>>> reversed(nums) #不直接显示出来是为了节省空间

<list\_reverseiterator object at 0x7f5bbb6fdb50>

>>> list(reversed(nums))

[94, 44, 94, 40, 16, 12, 5, 19, 14, 66]

>>> sorted(nums)

[5, 12, 14, 16, 19, 40, 44, 66, 94, 94]

>>> enumerate(nums) #同时得到下标和值

<enumerate object at 0x7f5bbb6939c0>

>>> list(enumerate(nums))

[(0, 66), (1, 14), (2, 19), (3, 5), (4, 12), (5, 16), (6, 40), (7, 94), (8, 44), (9, 94)]

**遍历nums**

>>> for i,n in enumerate(nums):

... print(i,n)

...

0 66

...

9 94

>>> for data in enumerate(nums): #输出的是元组

... print(data)

...

(0, 66)

...

(9, 94)

**五 字符串详解-字符串**

**5.1 字符串操作符**

• 比较操作符: 字符串大小按ASCII码值大小进行比较

• 切片操作符:[ ]、[ : ]、[ : : ]

• 成员关系操作符:in、not in

>>> py\_str ='Hello World!'

>>> py\_str[::2]

'HloWrd'

>>> py\_str[::-1]

'!dlroW olleH'

>>> s = '中'

>>> s

'中'

>>> s.encode() #将字符(str)转成bytes类型

b'\xe4\xb8\xad'

>>> a = s.encode()

>>> a

b'\xe4\xb8\xad'

>>> a.decode() #将bytes类型转成str类型

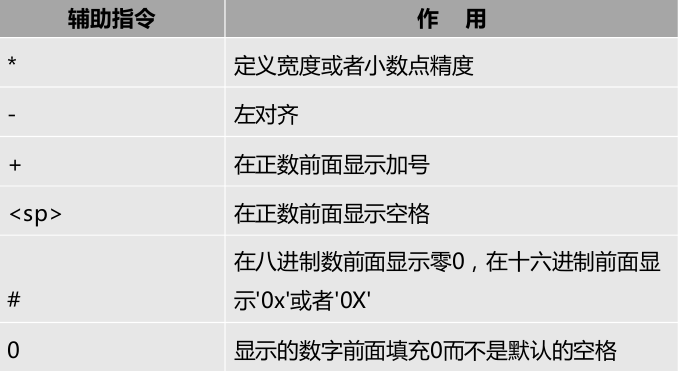
'中'

**5.2 格式化操作符**

字符串可以使用格式化符号来表示特定含义



**%s为最常用**



>>>'%s is %s years old' % ('tom', 20)

'tom is 20 years old'

>>>'%s is %d years old' % ('tom', 20) # %d表示需要用整数进行替换

'tom is 20 years old'

>>>'%d is %s years old' % ('tom', 20) # 错误,tom转不成整数

>>> '%8s%5s' % ('name', 'tom') #数字表示宽度,默认右对齐

' name tom'

>>> '%-8s%-5s' % ('name', 'tom') #数字为负表示左对齐

'name tom '

**其他操作符示例**

>>> '%#o' % 10 #转换为8进制

'0o12'

>>> '%#x' % 10 #转换为16进制

'0xa'

>>> '%f' % (5/3) #转换为浮点数

'1.666667'

>>> '%.2f' % (5/3) #转换为有2位小数的浮点数

'1.67'

>>> '%5.2f' % (5/3) #转换为长度为5,2位小数的浮点数

' 1.67'

>>> '%e' % 10000 #转换为科学计数法

'1.000000e+04'

**5.3 format函数**

• 使用位置参数

– 'my name is {} ,age {}'.format('hoho',18)

• 使用关键字参数

– 'my name is {name},age is

{age}'.format({'name':'bob', 'age':23})

• 填充与格式化

– {:[填充字符][对齐方式 <^>][宽度]}

• 使用索引

– 'name is {0[0]} age is {0[1]}'.format(['bob', 23])

**案例2:创建用户**

1. 编写一个程序,实现创建用户的功能

2. 提示用户输入用户名

3. 随机生成8位密码

4. 创建用户并设置密码

5. 将用户相关信息写入指定文件

**randpass.py**

from random import choice

from string import ascii\_letters,digits

all\_chs = ascii\_letters + digits

#print(all\_chs)

def randpass(n=8):

result = ''

for i in range(n):

ch = choice(all\_chs)

if ch in result:

continue

result += ch

return result

if \_\_name\_\_ == '\_\_main\_\_':

a = randpass()

print(a)

**useradd.py**

import sys

import subprocess

import randpass

def adduser(uname, passwd, fname):

# 判断用户是否已存在

result = subprocess.run(

'id %s &> /dev/null' % uname, shell=True

)

if result.returncode == 0:

print('用户已存在')

# 函数遇到return就结束了,不再向下执行

return

# 创建用户,设置密码

subprocess.run('useradd %s' % uname, shell=True)

subprocess.run(

'echo %s | passwd --stdin %s'

% (passwd, uname), shell=True

)

# 将用户名和密码追加写到文件中

with open(fname, 'a') as fobj:

fobj.write('用户名:%s\n密码:%s\n' % (uname, passwd))

if \_\_name\_\_ == '\_\_main\_\_':

uname = sys.argv[1]

passwd = randpass.randpass()

fname = sys.argv[2]

adduser(uname, passwd, fname)

**5.4 原始字符串操作符**

• 原始字符串操作符是为了对付那些在字符串中出现的特殊字符

• 在原始字符串里,所有的字符都是直接按照字面的意思来使用,没有转义特殊或不能打印的字符

>>> winPath = "c:\windows\temp"

>>> print(winPath)

c:\windows emp #未转义前,\t表示tab

>>> newPath = r"c:\windows\temp"

>>> print(newPath)

c:\windows\temp

**5.5 內建函数**

• string.capitalize():把字符串的第一个字符大写

• string.center(width):返回一个原字符串居中,并使用空格填充至长度width 的新字符串

• string.count(str,beg=0,end=len(string)):返回str在string里面出现的次数,如果beg或者end指定,则返回指定范围内str出现的次数

• string.endswith(obj, beg=0,end=len(string)):检查字符串是否以obj结束,如果beg或者end指定,则检查指定的范围内是否以obj结束,如果是,返回True,否则返回False

• string.islower():如果string中包含至少一个区分大小写的字符,并且所有这些字符都是小写,则返回True,否则返回False

• string.strip():删除string 字符串两端的空白

• string.upper():转换string 中的小写字母为大写

• string.split(str="", num=string.count(str)):以str为分隔符切片string,如果num有指定值,则仅分隔num个子字符串

**没有字符串方法,判断字符串中的所有字符是否为数字**

>>> a = input('data:')

data:>? 123456

>>> for i in a:

... if i not in '0123456789':

... print('False')

... break

... else:

... print('True')

**使用字符串方法,判断字符串中的所有字符是否为数字**

>>> a = '123456'

>>> b = 'abc'

>>> a.isdigit()

True

>>> b.isdigit()

False

使用strip去除空白字符

>>> s1 = ' hello \n'

>>> s1.strip() #去除两端空白字符

'hello'

>>> s1.rstrip() #去除右边空白字符

' hello'

>>> s1.lstrip() #去除左边空白字符

'hello \n'

>>> 'hello world hao 123'.split() #切割字符串

['hello', 'world', 'hao', '123']

>>> 'hello.world.hao.123'.split('.') #使用.号切割字符串

['hello', 'world', 'hao', '123']

>>> a = ['hello', 'world', 'hao', '123']

>>> '-'.join(a) #用-拼接字符串

'hello-world-hao-123'

>>> 'hello world'.replace('l','a') #将l替换为a

'heaao worad'

>>> 'hello'.center(48) #字符串居中

' hello '

>>> 'hello'.center(20,'#')

'#######hello########'

'hello'.ljust(20,'#') #左对齐

'hello###############'

'hello'.rjust(20,'#') #右对齐

'###############hello'

>>> 'hello'.upper() #转大写

'HELLO'

>>> 'HELLO'.lower() #转小写

'hello'

>>> 'hao123'.islower() #字母都是小写的吗?

True

>>> 'hao123'.isdigit() #所有字符都是数字吗?

False

完整的字符串方法参见:

https://docs.python.org/zh-cn/3.6/library/stdtypes.html#text-sequence-type-str