nsd1903_python1_day03

文件对象

操作步骤：

1. 打开
2. 读写
3. 关闭

读取文本文件

```
(nsd1903) [root@room8pc16 day02]# cp /etc/passwd /tmp/
>>> f = open('/tmp/passwd')   # 打开文件
>>> data = f.read()   # 默认读取全部内容
>>> data
>>> print(data)
>>> data = f.read()
>>> data
''
>>> f.close()   # 关闭文件

>>> f = open('/tmp/passwd')
>>> f.read(4)   # 读取4字节
'root'
>>> f.read(3)
':x:'
>>> f.readline()   # 读取一行
'0:0:root:/root:/bin/bash\n'
>>> f.readline()
'bin:x:1:1:bin:/bin:/sbin/nologin\n'
>>> f.readlines()   # 将所有行读出来，放到列表中，每行是列表的一项
>>> f.close()

# 读文本文件使用最多的方式是for循环
>>> f = open('/tmp/passwd')
>>> for line in f:
...   print(line, end='')
>>> f.close()
```

以**bytes**方式读取文件

1个16进制数，正好对应4位2进制：

```
0b0000 <-> 0x0
0b1111 <-> 0xF
```

读取任何文件都可以用bytes方式打开。读取文件内容时，如果是文本内容，将会以字符的形式显示，如果不能转成字符，将会直接显示16进制数。

```
>>> f = open('/tmp/passwd', 'rb')
>>> f.read(4)
b'root'
>>> f.close()

>>> f = open('/bin/ls')
>>> f.read(10)   # 读取时，python试图将内容转换成字符，但是失败了，报错
>>> f.close()

>>> f = open('/bin/ls', 'rb')
>>> f.read(10)
b'\x7fELF\x02\x01\x01\x00\x00\x00'
>>> f.close()
```

写文本文件

```
>>> f = open('/tmp/passwd', 'w')   # 创建或清空文件
>>> f.write('hello world!\n')   # \n表示换行
13    # 表示共写入了13字节
>>> f.flush()  # 默认数据写到缓存，不会立即同步至磁盘，flush()立即写入磁盘
>>> f.writelines(['2nd line.\n', '3rd line.\n'])
>>> f.close()  # 关闭文件，数据也会同步到磁盘
```

with语句

使用with打开文件，with语句结束后，文件自动关闭。

```
>>> with open('/tmp/passwd') as f:
...   for line in f:
...     print(line, end='')
```

移动文件指针

```
>>> f = open('/tmp/passwd', 'rb')
>>> f.tell()   # 返回文件指针位置，从文件开头到文件指针间的偏移量
0
>>> f.read(5)
b'hello'
>>> f.tell()
5
>>> f.seek(0, 0)   # 移动指针到文件开头
# seek第二个参数是相对位置，0表示开头，1表示当前位置，2表示结尾；第一个参数是偏移量
>>> f.seek(-6, 2)   # 移动指针到文件结尾前的第6个位置
>>> f.close()
```

练习：拷贝文件

```
# 初步实现
f1 = open('/bin/ls', 'rb')
f2 = open('/tmp/ls', 'wb')

data = f1.read()
f2.write(data)

f1.close()
f2.close()
```

以上代码存在的问题

- 尽量使用变量，不要直接使用'/bin/ls'这样的直接量
- 变量名应该有意义，f1和f2这样的名称没有意义
- 读取数据时，一次将全部内容读入，有可能数据量太大

函数

给一段代码起个名。假如有一个功能，需要10行代码，而这个功能需要在5个地方重复使用。可以将这些功能代码封装到函数中，以后需要用到这个功能，就调用函数。

函数定义时，代码不会执行。调用函数时，函数内的代码才会执行。

函数的返回值

- 函数执行的结果，可以使用return进行返回
- 如果函数没有return语句，默认返回None
- 当函数遇到return语句时，函数将会返回结果，停止执行

函数的参数

- 函数需要处理的数据，一般是通过参数进行传递的
- 定义函数时，参数值不确定，用一个名称进行占位，称作形式参数、形参
- 函数调用时，将具体的数据传递给函数，这个具体的数据是实际使用的参数，称作实际参数、实参
- 形参赋值给实参时，认为是变量赋值即可
- 函数内部的参数、变量都是局部变量，只在函数内可用。

位置参数

- 在python中，位置参数保存在sys模块的argv列表中
- 位置参数都以字符形式传递

```
(nsd1903) [root@room8pc16 day03]# cat pos.py
import sys

print(sys.argv)
(nsd1903) [root@room8pc16 day03]# python pos.py hao 123
['pos.py', 'hao', '123']
```

默认参数

为函数提供了默认值的参数。

- 调用函数时，传参，形参的值是传递的实参值
- 调用函数时，不传参，形参的值是默认值

```
>>> def pstar(n=30):
...   print('*' * n)
...
>>> pstar(20)
********************
>>> pstar(50)
**************************************************
>>> pstar()
******************************
```

函数练习1：

- 编写一个函数，函数接受一个字符串
- 函数将字符串左边的空格删除
- 返回去除左端空格的子串

```
s1 = '    hello world'
```

函数练习2：

- 编写一个函数，函数接受一个字符串
- 函数将字符串中的数字取出
- 返回字符串中的数字

```
s1 = 'a12bcd89xf2340ll'
```

模块

- 一个以.py作为结尾的python程序文件就是一个模块
- 模块的命名要求
    - 首字符只能是字母或下划线
    - 其他字符可以是字母、数字、下划线
    - 区分大小写
- 模块名是python程序文件名去除.py后前面的部分

```
# vim star.py
"""星星模块

该模块包含了一个全局变和一个函数
"""

hi = 'Hello World'

def pstar(n=30):
```

```
    "默认打印30个星号"
    print('*' * n)

# python3
>>> import star
>>> help(star)
>>> star.hi
'Hello World'
>>> star.pstar()
******************************
```

导入模块的方法

```
# 直接导入
>>> import time
>>> time.strftime('%Y-%m-%d')
'2019-08-02'

# 导入多个模块，不推荐
>>> import sys, os, datetime

# 只导入模块中的某些功能
>>> from random import randint, choice
>>> randint(1, 100)
81
>>> choice('+-*/')
'/'

# 导入模块时，为它起别名
>>> import getpass as gp
>>> gp.getpass()
```

模块的特性

- 每个模块文件都有一个特殊的变量叫__name__
- 它的值可以是__main__，也可以是模块名
    - 当模块文件直接运行时，是__main__
    - 当模块间接运行（被import导入）时是模块名

```
(nsd1903) [root@room8pc16 day03]# cat foo.py
print(__name__)
(nsd1903) [root@room8pc16 day03]# cat bar.py
import foo
(nsd1903) [root@room8pc16 day03]# python foo.py
__main__
(nsd1903) [root@room8pc16 day03]# python bar.py
foo
```