nsd1906_py01_day03

文件对象

操作文件的步骤

- 打开：在磁盘中找到文件的存储位置
- 读写：不管是文字还是声音还是图像，最终在磁盘上都是以2进制的0、1表示，读写是读写0、1的组合
- 关闭

```
# 打开本文件
(nsd1906) [root@room8pc16 day03]# cp /etc/passwd /tmp/mima
# open默认以读的方式打开，文件不存在则报错
>>> f = open('/tmp/mima.txt')
>>> f = open('/tmp/mima')
# 将全部内容读取出来，赋值给data
>>> data = f.read()
>>> print(data)
>>> data = f.read()  # 继续向后读，因为全部内容已读完，所以剩下的是空
>>> data
''
>>> f.close()   # 关闭文件
>>> f = open('/tmp/mima')   # 重新打开
>>> f.read(1)   # 读取1字节
'r'
>>> f.read(3)   # 继续读3个字节
'oot'
>>> f.readline()  # 从文件指针开始，读一行
':x:0:0:root:/root:/bin/bash\n'
>>> f.readlines()  # 将剩余部分读到列表中，每一行是列表的一项
>>> f.close()

# ***重要：文本文件一般采用的方式是for循环遍历***
>>> f = open('/tmp/mima')
>>> for line in f:
...   print(line, end='')
>>> f.close()

# 读取非文本文件
(nsd1906) [root@room8pc16 day03]# cp /bin/ls /tmp/ls
>>> f = open('/tmp/ls', 'rb')  # r是读，b是bytes
>>> f.read(10)
b'\x7fELF\x02\x01\x01\x00\x00\x00'
>>> f.close()

# 写入文本文件。注意：以w方式打开文件，会将文件清空
>>> f = open('/tmp/mima', 'w')
>>> f.write('hello world!\n')   # 写入了13字节
```

```
13
>>> f.flush()   # 立即将内存数据同步至磁盘
>>> f.writelines(['2nd line.', 'new aaa\n', '3rd line.\n'])
>>> f.close()


# 写入非文本文件
>>> f = open('/tmp/aaaa', 'wb')
>>> hi = '你好\n'
>>> hi.encode()
b'\xe4\xbd\xa0\xe5\xa5\xbd\n'
# 共写入七个字节，在utf8编码中一个汉字占3字节，\n占1字节
>>> f.write(hi.encode())
7
>>> f.close()
```

with语句

通过with语句打开文件，当with语句结束时，文件自动关闭

```
>>> with open('/tmp/mima') as f:
...   f.readline()
...
'hello world!\n'
>>> f.readline()   # 报错，因为文件已经关闭了
```

seek移动文件指针

seek可以在不关闭文件的情况下，移动指针。它有两个参数，第二个参数是相对位置，0表示开头，1表示当前指针位置，2表示结尾；第一个参数是偏移量。

```
>>> f = open('/tmp/mima')
>>> f.tell()   # 总是显示文件指针距离开头多少字节
0
>>> f.seek(6, 0)   # 从开头向右移动6字节
6
>>> f.read(5)
'world'
>>> f.close()
```

函数

- 函数就是一组代码的集合
- 函数定义的时候，它里面的代码不会执行
- 函数使用一对()进行调用，调用时，它里面的代码执行一遍
- 函数定义的语法

```
def 函数名(参数):
    代码组
```

例：

```
>>> def pstar():
...   print('*' * 30)
...
>>> pstar()  # 调用函数
******************************
>>> a = pstar()
******************************
>>> a
>>> print(a)
None
```

函数返回值

函数如果需要有返回值，则使用return进行返回；没有明确的return语句，默认返回None

```
>>> def add():
...   a = 10 + 5
...   return a
...
>>> n = add()
>>> print(n)
15
```

参数

- 形式参数：形参。在定义函数时，函数名称后面括号中的变量
- 实际参数：实参。调用函数时，传递给函数的参数。

```
>>> def add(x, y):
...   return x + y
...
>>> add(5, 5)
10
```

位置参数

python将命令行上的位置参数，保存到了sys模块的argv列表中。注意，位置参数接收的全部是字符串类型

```
# vim position.py
import sys

print(sys.argv)
(nsd1906) [root@room8pc16 day03]# python position.py
['position.py']

(nsd1906) [root@room8pc16 day03]# python position.py hao 123
['position.py', 'hao', '123']
```

默认参数

具有默认值的参数

```
>>> def pstar():
...     print('*' * 30)
...
>>> pstar()   # 函数只能打印30个*
******************************

>>> def pstar(n):    # 传参，n的值就是*号的个数
...     print('*' * n)
...
>>> pstar(40)
****************************************
>>> pstar(20)
********************

>>> def pstar(n=30):  # 提供默认参数
...     print('*' * n)
...
>>> pstar()   # 不传参，n使用默认的30
******************************
>>> pstar(15)  # 传参，n使用传递进来的15
***************
```

模块

- 当代码量增加时，可以考虑把代码放到不同的文件里
- 每个以.py作为结尾的文件被称作模块文件
- 文件是代码的物理组织形式
- 模块是代码的逻辑组织形式。模块文件，将扩展名.py去掉就是模块名

模块导入方法

```
# 直接导入， 常用
>>> import time
>>> time.ctime()
'Sat Nov  2 15:45:46 2019'

# 从模块中导入一部分功能，常用
>>> from random import randint, choice
>>> randint(1, 100)
9
>>> choice('abcde')
'e'

# 一行导入多个模块，不常用，因为可读性差
>>> import os, sys
```

```
# 导入模块的同时，为模块创建别名，不常用
>>> import getpass as gp
>>> p = gp.getpass()
Password:
```

自定义模块

```
# vim star.py
hi = 'hello world!'

def pstar(n=30):
    print('*' * n)

# vim call_star.py
import star

print('this is call_star')
star.pstar()
print(star.hi)

# python call_star.py
```

模块的特性

- 导入模块时，模块内的代码将会执行一遍。
- 模块中的代码，有时希望它在作为一个脚本文件直接运行时执行；希望它被当成模块导入时，不要执行，这个时候可以使用__name__属性
- __name__是一个特殊的变量，它的值有两个
    - 当程序文件直接运行时，值是__main__
    - 当程序文件作为模块导入时，值是模块名

```
(nsd1906) [root@room8pc16 day03]# cat foo.py
print(__name__)
(nsd1906) [root@room8pc16 day03]# cat bar.py
import foo
(nsd1906) [root@room8pc16 day03]# python foo.py
__main__
(nsd1906) [root@room8pc16 day03]# python bar.py
foo
```

练习：生成随机字符串

1. 决定从哪些字符中随机选取
2. 根据要求选n次
3. 把这n个字符拼接起来