nsd1912-py01-day03

python官方手册：https://docs.python.org/zh-cn/3/library/index.html

文件

- 无论是什么类型的文件，最终都是以二进制01的方式存储
- 文件的操作步骤：打开、读写、关闭

操作文本文件

字符编码

- 不同国家有不同的编码方案。如美国的ASCII码，西欧的ISO-8859-1也叫lartin1，中国的GBK/GB18030/GB2312
- 为了实现字符编码的统一，国际标准化组织ISO发布了万国码Unicode。其中utf8是Unicode的一种方案。
- 字符是str，字节是bytes。一个英文字符是字符，一个汉字字符也是一个字符；但是在utf8中，一个英文字符占1个字节，一个汉字字符通常需要占3个字节
- 在python中，bytes类型数据表示时，如果一个字节正好可以表示成一个字符，就以字符的方式显示；如果一个字节不能表示成字符，就以16进制方式显示

```
>>> s1 = 'a'
>>> s2 = '中'
>>> type(s1)
<class 'str'>
>>> type(s2)
<class 'str'>
>>> s1.encode()  # 将s1转成bytes类型
b'a'    # 前面的b表示bytes类型
>>> s2.encode()
b'\xe4\xb8\xad'   # \x表示后面的e4为16进制

>>> b1 = s2.encode()
>>> b1
b'\xe4\xb8\xad'
>>> b1.decode()   # 将bytes类型转为str类型
'中'
```

读取文本文件

```
[root@localhost day03]# cp /etc/passwd /tmp/
# 默认以'r'的方式打开文件，文件不存在则报错。
>>> f = open('/tmp/mima')    # 报错
>>> f = open('/tmp/passwd')  # 打开文件，得到一个文件对象
>>> data = f.read()  # 默认读取全部数据
>>> print(data)
>>> data = f.read()  # 继续向后读取
>>> data
''
>>> f.close()    # 关闭文件

>>> f = open('/tmp/passwd', 'r')
>>> f.readline()  # 读一行
'root:x:0:0:root:/root:/bin/bash\n'
>>> f.readline()  # 继续读一行
'bin:x:1:1:bin:/bin:/sbin/nologin\n'
>>> f.readlines() # 继续读后续内容，每一行都放到列表中
>>> f.close()

################重要###############
>>> f = open('/tmp/passwd')
>>> for line in f:
...   print(line, end='')
>>> f.close()
```

写入文本文件

```
# 以'w'方式写入文件时，文件不存在则创建；文件存在则清空
>>> f = open('/tmp/passwd', 'w')
[root@localhost day03]# cat /tmp/passwd  # 空的
>>> f.write('Hello World!\n')
13   # 13表示写入了13字节
# 数据先会写入到缓存，等缓存中的数据达到一定量时才会同步至硬盘，关闭文件时，也会写入硬盘。
[root@localhost day03]# cat /tmp/passwd  # 空的
>>> f.flush()   # 立即同步数据到硬盘
[root@localhost day03]# cat /tmp/passwd
Hello World!
>>> f.writelines(['How are you?\n', '吃了吗？\n'])
>>> f.close()
[root@localhost day03]# cat /tmp/passwd
Hello World!
How are you?
吃了吗？

# 以a的方式打开文件，可以将数据追加到结尾
>>> f = open('/tmp/passwd', 'a')
>>> f.write('my test\n')
8
>>> f.close()
[root@localhost day03]# cat /tmp/passwd
Hello World!
How are you?
吃了吗？
my test
```

操作非文本文件

- 无论是什么文件都以二进制方式存储
- 操作非文本文件的方法，也适用于文本文件

```
[root@localhost ~]# wget http://pic1.win4000.com/wallpaper/4/579861684f4e7.jpg -O
girl.jpg
>>> f = open('/root/girl.jpg', 'rb')  # b表示bytes
>>> f.read(100)   # 读取100字节
b'\xff\xd8\xff\xe1\x17%Exif\x00\x00MM\x00*\x00\x00\x00\x08\x00\x0c\x01\x00\x00\x03\x00\x
00\x00\x01\x057\x00\x00\x01\x01\x00\x03\x00\x00\x00\x01\x07\xd0\x00\x00\x01\x02\x00\x03\
x00\x00\x00\x03\x00\x00\x00\x9e\x01\x06\x00\x03\x00\x00\x00\x01\x00\x02\x00\x00\x01\x12\
x00\x03\x00\x00\x00\x01\x00\x01\x00\x00\x01\x15\x00\x03\x00\x00\x00\x01\x00\x03\x00\x00\
x01\x1a\x00\x05\x00\x00'
>>> f.close()

>>> f = open('/root/girl.jpg', 'rb')
>>> data = f.read(1000)
>>> f.close()
>>> f = open('/tmp/myimg.jpg', 'wb')
>>> f.write(data)
1000
>>> f.close()
```

with语句

- 使用with打开文件，with语句结束，文件自动关闭

```
>>> with open('/tmp/passwd') as f:
...     f.readline()
...
'Hello World!\n'
>>> f.readline()   # 报错，因为文件已经关闭
```

seek语句（了解）

- 用于移动文件指针
- seek函数的第二个参数，0表示开头，1表示指针当前位置，2表示结尾；第一个参数是相对于第二个参数的偏移量

```
>>> f = open('/tmp/passwd', 'rb')
>>> f.seek(5, 0)  # 从开头向右移动 5字节
5
>>> f.read(3)
b' Wo'
>>> f.seek(3, 1)  # 从当前位置向右移动 3字节
11
>>> f.read(1)
b'!'
>>> f.seek(0, 2)  # 移动指针到结尾
47
>>> f.read()
b''
>>> f.close()
```

函数

- 函数就是一组代码。方便实现代码重用
- 函数声明语法结构

```
def 函数名():
    代码块
```

- 函数声明时，它里面的代码块不执行
- 调用函数时，函数体的代码块执行

返回值

- 函数的执行结果通过关键字return返回
- 没有return语句，默认返回None

```
>>> def add():
...     a = 10 + 5
...
>>> x = add()
>>> print(x)
None
>>> print(a)   # 报错，没有a这个变量

>>> def add():
...     a = 10 + 5
...     b = a + 6
...     return 'Hello World'
...
>>> x = add()
>>> print(x)
Hello World

>>> def add():
...      a = 10 + 5
...      return a
...
>>> x = add()
>>> print(x)
15
>>> x + 8
23

>>> def add():
...     a = 10 + 5
...     print(a)
...
>>> x = add()
15
>>> x + 8
```

参数

- 可以简单的把参数理解为变量
- 在函数内部需要使用一些数据，这些数据不是固定的，不能提前预知，就使用参数的方式传递
- 在定义函数时，写在函数后面()中的名字，只是形式上占个位置，所以称作形式参数、形参
- 在调用函数时，将具体的数据传给函数，这个时候是实际使用的参数，所以称作实际参数、实参
- 相当于定义函数时的参数是变量名，调用函数时传递的是变量值

默认参数

- 声明函数时，为参数指定默认值

```
>>> def pstar():
...     print('*' * 30)
...
>>> pstar()
******************************
>>> pstar(50)   # 报错，因为函数定义时，没有参数

>>> def pstar(n):
...     print('*' * n)
...
>>> pstar(30)
******************************
>>> pstar(50)
**************************************************
>>> pstar()   # 报错，因为函数需要参数，但是没有给定

>>> def pstar(n=30):  # 默认参数
...     print('*' * n)
...
>>> pstar(20)
********************
>>> pstar()
******************************
```

位置参数

- 相当于shell里的位置参数（$1 / $2等)
- 在python中，位置参数保存在sys模块的argv列表中
- sys.argv[0]是列表的第一项，sys.argv[1]是列表的第二项... ...

```
[root@localhost day03]# cat position.py
import sys

print(sys.argv)
[root@localhost day03]# python3 position.py
['position.py']
[root@localhost day03]# python3 position.py hao 123
['position.py', 'hao', '123']
```

模块

- 文件是python在物理上组织代码的形式
- 模块是python在逻辑上组织代码的形式
- 一个以.py结尾的python文件就是一个模块。模块名是文件名去除.py后的部分
- 模块命名约定与变量一致

```
[root@localhost day03]# cat star.py
hi = 'Hello World'

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

[root@localhost day03]# cat mytest.py
import star

print(star.hi)
star.pstar()

[root@localhost day03]# python3 mytest.py
Hello World
******************************
```

- 导入模块的方法

```
# 常用的方法
>>> import time
>>> time.ctime()
'Tue May 19 16:35:33 2020'
>>> from random import randint, choice
>>> randint(1, 100)
99
>>> choice('abcdefg')
'f'

# 不常用的方法
>>> import os, datetime    # 同时导入多个模块
>>> import getpass as gp   # 导入模块时，起别名
```

- 导入模块叫作import。导入模块时，模块中的代码将会执行一遍，这个过程叫load加载。不管import多少次，只加载一次。

模块的特殊属性

- 每个模块都有一个名为__name__的特殊属性
- __name__是一个变量。它的值有两个：
  - 如果模块是直接运行的，它的值是__main__
  - 如果模块是被导入的，它的值是模块名

```
[root@localhost day03]# cat mymod.py
print(__name__)
[root@localhost day03]# cat aaaa.py
import mymod
[root@localhost day03]# python3 mymod.py
__main__
[root@localhost day03]# python3 aaaa.py
mymod
```

- 可以根据__name__的值，判断模块是直接运行，还是被导入。