

文件操作

- 目标
 - 文件操作的作用
 - 文件的基本操作
 - 打开
 - 读写
 - 关闭
 - 文件备份
 - 文件和文件夹的操作

1、文件操作的作用

思考：什么是文件？



思考：文件操作包含什么？

答：打开、关闭、读、写、复制....

思考：文件操作的作用是什么？

答：读取内容、写入内容、备份内容.....

总结：文件操作的作用就是==把一些内容(数据)存储存放起来，可以让程序下一次执行的时候直接使用，而不必重新制作一份，省时省力==。

2、文件的基本操作

2.1 文件操作步骤

- 1. 打开文件
- 2. 读写等操作
- 3. 关闭文件

注意：可以只打开和关闭文件，不进行任何读写操作。

2.1.1 打开

在python，使用open函数，可以打开一个已经存在的文件，或者创建一个新文件，语法如下：

```
open(name, mode)
```

name：是要打开的目标文件名的字符串(可以包含文件所在的具体路径)。

mode：设置打开文件的模式(访问模式)：只读、写入、追加等。

2.1.1.1 打开文件模式

模式	描述
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。
r+	打开一个文件用于读写。文件指针将会放在文件的开头。
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。
w	打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
w+	打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
a+	打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会追加模式。如果该文件不存在，创建新文件用于读写。
ab+	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

2.1.1.2 快速体验

```
f = open('test.txt', 'w')
```

注意：此时的 `f` 是 `open` 函数的文件对象。

2.1.2 文件对象方法

2.1.2.1 写

- 语法

```
对象对象.write('内容')
```

- 体验

```
# 1. 打开文件
f = open('test.txt', 'w')

# 2. 文件写入
f.write('hello world')

# 3. 关闭文件
f.close()
```

注意：

1. `w` 和 `a` 模式：如果文件不存在则创建该文件；如果文件存在，`w` 模式先清空再写入，`a` 模式直接末尾追加。
2. `r` 模式：如果文件不存在则报错。

2.1.2.2 读

- `read()`

```
文件对象.read(num)
```

`num` 表示要从文件中读取的数据的长度（单位是字节），如果没有传入 `num`，那么就表示读取文件中所有的数据。

- `readlines()`

`readlines` 可以按照行的方式把整个文件中的内容进行一次性读取，并且返回的是一个列表，其中每一行的数据为一个元素。

```
f = open('test.txt')
content = f.readlines()

# ['hello world\n', 'abcdefg\n', 'aaa\n', 'bbb\n', 'ccc']
print(content)

# 关闭文件
f.close()
```

- readline()

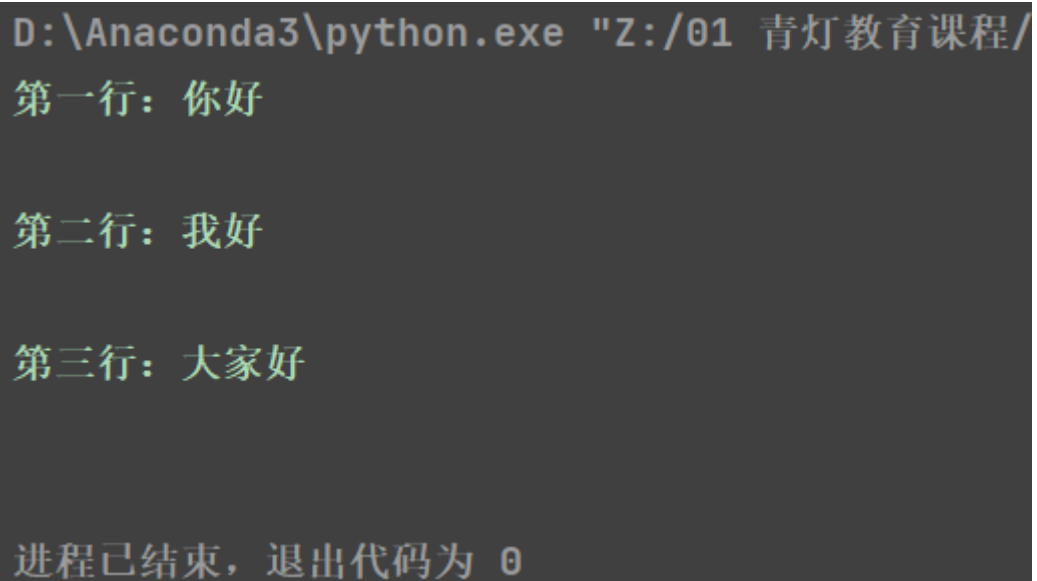
readline()一次读取一行内容。

```
f = open('test.txt')

content = f.readline()
print(f'第一行: {content}')

content = f.readline()
print(f'第二行: {content}')

# 关闭文件
f.close()
```

A terminal window with a dark background. The command prompt shows the path 'D:\Anaconda3\python.exe' followed by a file path 'Z:/01 青灯教育课程/'. The output shows three lines of text: '第一行: 你好', '第二行: 我好', and '第三行: 大家好'. At the bottom, it says '进程已结束, 退出代码为 0'.

```
D:\Anaconda3\python.exe "Z:/01 青灯教育课程/"
第一行: 你好

第二行: 我好

第三行: 大家好

进程已结束, 退出代码为 0
```

2.1.3 关闭

文件对象.close()

with 上下文管理文件

上述文件操作, 打开文件操作完后都需要关闭文件, 比较麻烦。

可以用 with 上下文管理打开的文件

```
with open('test.txt', mode='r', encoding='utf-8') as f:
    text = f.read()
```

3、文件备份

需求: 用户输入当前目录下任意文件名, 程序完成对该文件的备份功能(备份文件名为xx[备份]后缀, 例如: test[备份].txt)。

3.1 步骤

1. 接收用户输入的文件名
2. 规划备份文件名
3. 备份文件写入数据

3.2 代码实现

1. 接收用户输入目标文件名

```
old_name = input('请输入您要备份的文件名: ')
```

2. 规划备份文件名

- 2.1 提取目标文件后缀
- 2.2 组织备份的文件名, xx[备份]后缀

```
# 2.1 提取文件后缀点的下标
index = old_name.rfind('.')

# print(index) # 后缀中的下标

# print(old_name[:index]) # 源文件名 (无后缀)

# 2.2 组织新文件名 旧文件名 + [备份] + 后缀
new_name = old_name[:index] + '[备份]' + old_name[index:]

# 打印新文件名 (带后缀)
# print(new_name)
```

3. 备份文件写入数据

- 3.1 打开源文件 和 备份文件
- 3.2 将源文件数据写入备份文件
- 3.3 关闭文件

```

# 3.1 打开文件
old_f = open(old_name, 'rb')
new_f = open(new_name, 'wb')

# 3.2 将源文件数据写入备份文件
while True:
    con = old_f.read(1024)
    if len(con) == 0:
        break
    new_f.write(con)

# 3.3 关闭文件
old_f.close()
new_f.close()

```

3.3 思考

如果用户输入 `.txt`，这是一个无效文件，程序如何更改才能限制只有有效的文件名才能备份？

答：添加条件判断即可。

```

old_name = input('请输入您要备份的文件名: ')

index = old_name.rfind('.')

if index > 0:
    postfix = old_name[index:]

new_name = old_name[:index] + '[备份]' + postfix

old_f = open(old_name, 'rb')
new_f = open(new_name, 'wb')

while True:
    con = old_f.read(1024)
    if len(con) == 0:
        break
    new_f.write(con)

old_f.close()
new_f.close()

```

4、文件和文件夹的操作

在Python中文件和文件夹的操作要借助os模块里面的相关功能，具体步骤如下：

1. 导入os模块

```
import os
```

2. 使用 `os` 模块相关功能

```
os.函数名()
```

4.1 文件重命名

```
os.rename(目标文件名, 新文件名)
```

4.2 删除文件

```
os.remove(目标文件名)
```

4.3 创建文件夹

```
os.mkdir(文件夹名字)
```

4.4 删除文件夹

```
os.rmdir(文件夹名字)
```

4.5 获取当前目录

```
os.getcwd()
```

4.6 改变默认目录

```
os.chdir(目录)
```

4.7 获取目录列表

```
os.listdir(目录)
```

4.8 查看文件是否存在

```
os.path.exists('路径')
```

5、应用案例

需求：批量修改文件名，既可添加指定字符串，又能删除指定字符串。

- 步骤
 1. 设置添加删除字符串的标识
 2. 获取指定目录的所有文件
 3. 将原有文件名添加/删除指定字符串，构造新名字
 4. os.rename()重命名

- 代码

```
import os

# 设置重命名标识：如果为1则添加指定字符，flag取值为2则删除指定字符
flag = 1

# 获取指定目录
dir_name = './'

# 获取指定目录的文件列表
file_list = os.listdir(dir_name)
# print(file_list)

# 遍历文件列表内的文件
for name in file_list:

    # 添加指定字符
    if flag == 1:
        new_name = 'Python-' + name
    # 删除指定字符
    elif flag == 2:
        num = len('Python-')
        new_name = name[num:]

    # 打印新文件名，测试程序正确性
    print(new_name)

    # 重命名
    os.rename(dir_name+name, dir_name+new_name)
```

6、总结

- 文件操作步骤
 - 打开

文件对象 = `open`(目标文件, 访问模式)

- 操作

- 读

```
文件对象.read()
文件对象.readlines()
文件对象.readline()
```

- 写

```
文件对象.write()
```

- `seek()`

- 关闭

```
文件对象.close()
```

- 主访问模式

- `w`: 写, 文件不存在则新建该文件
 - `r`: 读, 文件不存在则报错
 - `a`: 追加

- 文件和文件夹操作

- 重命名: `os.rename()`
 - 获取当前目录: `os.getcwd()`
 - 获取目录列表: `os.listdir()`

深浅拷贝

对象引用、浅拷贝、深拷贝(拓展、难点、重点)

Python中, 对象的赋值, 拷贝(深/浅拷贝)之间是有差异的, 如果使用的时候不注意, 就可能产生意外的结果

其实这个是由于共享内存导致的结果

拷贝: 原则上就是把数据分离出来, 复制其数据, 并以后修改互不影响。

先看一个非拷贝的例子

使用=赋值(对象引用)

=赋值: 数据完全共享

=赋值是在内存中指向同一个对象, 如果是可变(`mutable`)类型, 比如列表, 修改其中一个, 另一个必定改变

如果是不可变类型(`immutable`), 比如字符串, 修改了其中一个, 另一个并不会变

```
>>> a = [1, 2, 3]
```

```

>>> a
[1, 2, 3]
>>> b = a
>>> b
[1, 2, 3]
>>> a[0] = 'surprise'
>>> a
['surprise', 2, 3]

>>> b
['surprise', 2, 3]
>>> b[0] = 'I hate surprises'
>>> b
['I hate surprises', 2, 3]
>>> a
['I hate surprises', 2, 3]

```

浅拷贝 (copy)

浅拷贝：数据半共享（复制其数据独立内存存放，但是只拷贝成功第一层）

```

>>> a = [1, 2, 3]
>>> b = a.copy()
>>> c = list(a)
>>> d = a[:]

>>> a[0] = 'integer lists are boring'
>>> a
['integer lists are boring', 2, 3]
>>> b
[1, 2, 3]
>>> c
[1, 2, 3]
>>> d
[1, 2, 3]

```

深拷贝 (deepcopy)

深拷贝：数据完全不共享（复制其数据完完全全放独立的一个内存，完全拷贝，数据不共享）

深拷贝就是完完全全复制了一份，且数据不会互相影响，因为内存不共享。

深拷贝的方法有

```

>>> import copy
>>> a = [1, 2, 3, [1, 2, 3]]
>>> b = copy.copy(a)
>>> a[3][0] = "surprises"
>>> b
[1, 2, 3, ['surprises', 2, 3]]

>>> c = copy.deepcopy(b)
>>> b[3][0] = "i hate surprises"
>>> c

```

```
[1, 2, 3, ['surprises', 2, 3]]  
>>> b  
[1, 2, 3, ['i hate surprises', 2, 3]]
```

总结:

copy.copy 浅拷贝 只拷贝父对象, 不会拷贝对象的内部的子对象。

copy.deepcopy 深拷贝 拷贝对象及其子对象