

文件的读写

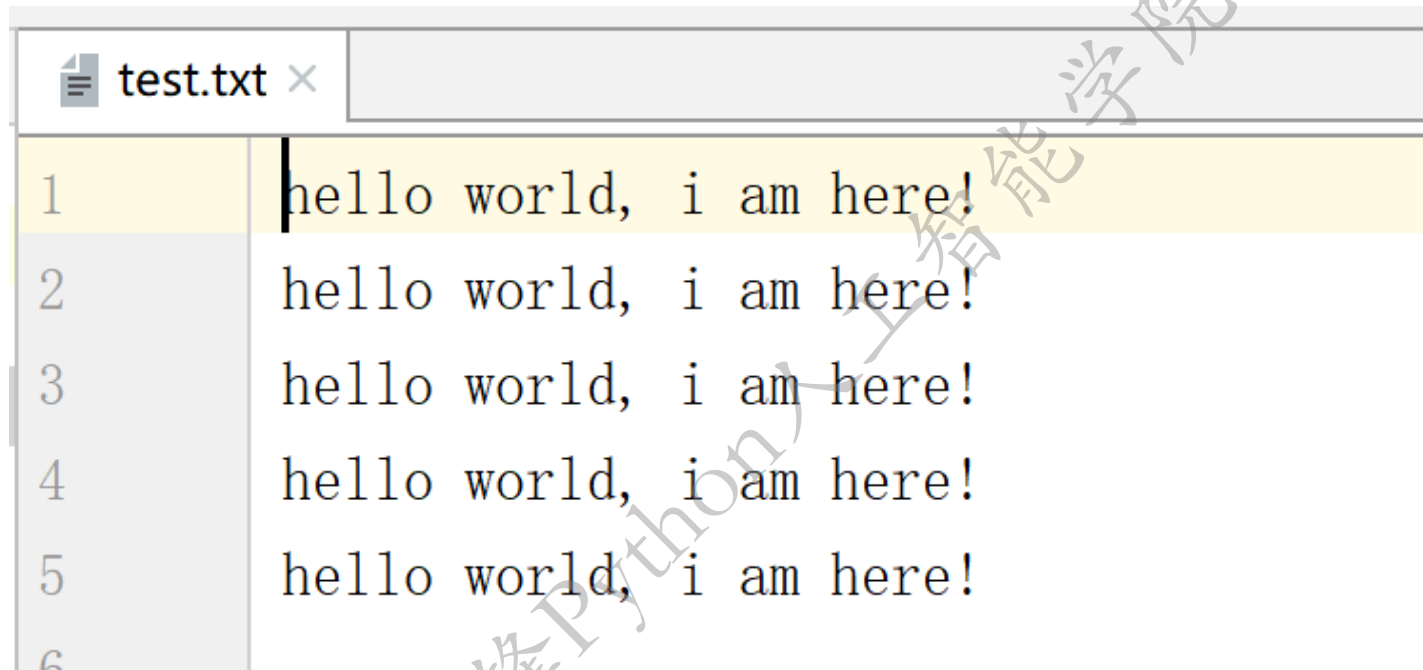
<1>写数据(write)

使用write()可以完成向文件写入数据

demo: 新建一个文件 `file_write_test.py` ,向其中写入如下代码:

```
f = open('test.txt', 'w')
f.write('hello world, i am here!\n' * 5)
f.close()
```

运行之后会在 `file_write_test.py` 文件所在的路径中创建一个文件 `test.txt` ,并写入内容, 运行效果显示如下:



```
test.txt x
1 hello world, i am here!
2 hello world, i am here!
3 hello world, i am here!
4 hello world, i am here!
5 hello world, i am here!
6
```

注意:

- 如果文件不存在, 那么创建; 如果存在那么就先清空, 然后写入数据

<2>读数据(read)

使用read(num)可以从文件中读取数据, 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

demo: 新建一个文件 `file_read_test.py` , 向其中写入如下代码:

```
f = open('test.txt', 'r')
content = f.read(5) # 最多读取5个数据
print(content)

print("-"*30) # 分割线, 用来测试

content = f.read() # 从上次读取的位置继续读取剩下的所有的数据
print(content)

f.close() # 关闭文件, 这个可是个好习惯哦
```

运行现象:

```
hello
-----
world, i am here!
```

注意:

- 如果用open打开文件时, 如果使用的"r", 那么可以省略 `open('test.txt')`

<3>读数据 (readline)

readline只用来读取一行数据。

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

content = f.readline()
print("1:%s" % content)

content = f.readline()
print("2:%s" % content)

f.close()
```

<4>读数据 (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, 其中每一行为列表的一个元素。

```
f = open('test.txt', 'r')
content = f.readlines()
print(type(content))

for temp in content:
    print(temp)

f.close()
```

指针定位

- `tell()` 方法用来显示当前指针的位置

```
f = open('test.txt')
print(f.read(10)) # read 指定读取的字节数
print(f.tell())  # tell()方法显示当前文件指针所在的文字
f.close()
```

- `seek(offset, whence)` 方法用来重新设定指针的位置。

- `offset`:表示偏移量
- `whence`:只能传入012中的一个数字。
- 0表示从文件头开始
- 1表示从当前位置开始
- 2 表示从文件的末尾开始

```
f = open('test.txt', 'rb') # 需要指定打开模式为rb,只读二进制模式

print(f.read(3))
print(f.tell())

f.seek(2, 0) # 从文件的开头开始, 跳过两个字节
print(f.read())

f.seek(1, 1) # 从当前位置开始, 跳过一个字节
print(f.read())

f.seek(-4, 2) # 从文件末尾开始, 往前跳过四个字节
print(f.read())

f.close()
```