



4.6 上下文管理器



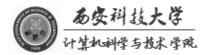
■ 异常处理

```
try:
    f = open("mypython.txt")
    print(f.read())
except IOError as e:
    print (e)
finally:
    f.close()
```

■ with语句

```
with open("mypython.txt") as f:
    print(f.read())
```

- 使用with语句替代try-finally 语句,代码更加的简洁清晰。
- 对于需要**对资源进行访问**的任务,无论在代码运行过程中,是否发生异常,都会执行必要的清理操作,**释放资源**。



N



■ with语句

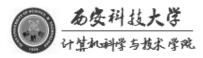
```
with open("mypython.txt") as f:
    print(f.read())
```

运行结果:

Python3.0

在with语句完成时,会**自动关闭文件**。如果再次读取这个文件,就会出现文件关闭的错误提示信息。

print(f.read())





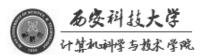
■上下文管理器

- □ 上下文管理器是Python中的一种<mark>协议</mark>,它保证了每次代码执行的一致性
- □ 一旦进入上下文管理器,就一定会按照规定的步骤退出
- □ 如果合理的设计了退出上下文管理器的步骤,就能够很好的处理异常。
- □ 上下文管理器被最多用到的场景是资源清理操作。
- □ 实现上下文管理器,只要在类定义时,实现 enter ()方法和 exit ()方法即可

```
class A():
    def __init__(self, val_a):
        self.a=val_a

def __enter__(self):
        print("calss A's __enter__function.")

def __exit__(self, exc_type, exc_val, exc_tb):
        print("calss A's __exit__function.")
```





□ 实现上下文管理器

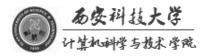
```
class A():
    def __init__(self, val_a):
        self.a=val_a

    def __enter__(self):
        print("calss A's __enter__function.")

    def __exit__(self, exc_type, exc_val, exc_tb):
        print("calss A's __exit__function.")
```

如果with语句块正常结束,那么 这三个参数全部是None。如果 发生异常,那么这三个参数的值 分别异常的类、实例和跟踪记录。

- 在with语句块执行前,首先会执行enter()方法
- 当with语句块执行结束,无论是否出现异常,都会调用 exit ()方法
- 通常将清除、释放资源的操作写在 exit ()方法中



上下文管理器对象,或函数(返回值是上下文管理器对象)

with 上下文管理器表达式 [as 变量]: 语句块

with open("mypython.txt") as f:
 print(f.read())



例:模拟实现一个文件类

```
class File():
   def init (self, filename, mode):
       self.filename = filename
       self.mode = mode
   def enter (self):
       print("执行__enter__()方法")
       self.f = open(self.filename, self.mode)
       return self.f
   def exit (self, *args):
       print("执行__exit__方法")
       self.f.close()
```

N

4.6 上下文管理器



以读的方式打开当前目录下的mypython文件,并输出文件内容。

```
with File('mypython.txt', 'r') as f:
    print(f.read())
```

运行结果: 执行__enter__()方法 Python3.∅

执行__exit__方法

- 在执行with语句块之前,首先执行了__enter()__方法,然后再执行with语句块,最后执行__exit()__方法。
- 采用这种方式读取文件时,即使执行过程中出现了异常,__exit()__方法也会被执行,完成关闭文件的操作。

