魔术方法

- 分类：
  - 创建与销毁
    - `__init__` 与 `__del__`
  - hash
  - bool
  - 可视化
  - 运算符重载
  - 容器和大小
  - 可调用对象
  - 上下文管理
  - 反射
  - 描述器
  - 其他杂项

上下文管理

文件IO操作可以对文件对象使用上下文管理，使用with...as语法。

```
with open('test') as f:
    pass
```

仿照上例写一个自己的类，实现上下文管理

```
class Point:
    pass

with Point() as p: # AttributeError: __exit__
    pass
```

提示属性错误，没有 `__exit__` ，看了需要这个属性

上下文管理对象

当一个对象同时实现了 `__enter__` ()和 `__exit__` ()方法，它就属于上下文管理的对象

方法	意义

`__enter__`	进入与此对象相关的上下文。如果存在该方法，with语法会把该方法的返回值作为绑定到as子句中指定的变量上
`__exit__`	退出与此对象相关的上下文。

```
class Point:
    def __init__(self):
        print('init')
    def __enter__(self):
        print('enter')
    def __exit__(self, exc_type, exc_val, exc_tb):
        print('exit')


with Point() as f:
    print('do sth.')
```

实例化对象的时候，并不会调用enter，进入with语句块调用 `__enter__` 方法，然后执行语句体，最后离开with语句块的时候，调用 `__exit__` 方法。

with可以开启一个上下文运行环境，在执行前做一些准备工作，执行后做一些收尾工作。

上下文管理的安全性

看看异常对上下文的影响。

```
class Point:
    def __init__(self):
        print('init')

    def __enter__(self):
        print('enter')

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('exit')


with Point() as f:
    raise Exception('error')
    print('do sth.')
```

```
#init
#enter
#exit
```

可以看出在enter和exit照样执行，**上下文管理是安全的。**

极端的例子
调用sys.exit()，它会退出当前解释器。
打开Python解释器，在里面敲入sys.exit()，窗口直接关闭了。也就是说碰到这一句，Python运行环境直接退出了。

```
import sys
class Point:
    def __init__(self):
        print('init')

    def __enter__(self):
        print('enter')

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('exit')

with Point() as f:
    sys.exit(-100)
    print('do sth.')

print('outer')
```

从执行结果来看，依然执行了 `__exit__` 函数，哪怕是退出Python运行环境。
说明**上下文管理很安全。**

with语句

```
class Point:
    def __init__(self):
        print('init')

    def __enter__(self):
        print('enter')
```

```
    def __exit__(self, exc_type, exc_val, exc_tb):
        print('exit')

p = Point()
with p as f:
    print(p == f) # 为什么不相等
    print('do sth.')
```

问题在于 `__enter__` 方法上，它将自己的返回值赋给f。修改上例

```
class Point:
    def __init__(self):
        print('init')

    def __enter__(self):
        print('enter')
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('exit')

p = Point()
with p as f:
    print(p == f)
    print('do sth.')
```

`__enter__` 方法返回值就是上下文中使用的对象，with语法会把它的返回值赋给as子句的变量。

`__enter__` 方法和 `__exit__` 方法的参数

`__enter__` 方法 没有其他参数。

`__exit__` 方法有3个参数：
`__exit__(self, exc_type, exc_value, traceback)`
这三个参数都与异常有关。
如果该上下文退出时没有异常，这3个参数都为None。
如果有异常，参数意义如下
`exc_type` ，异常类型
`exc_value` ，异常的值

`traceback` ，异常的追踪信息
`__exit__` 方法返回一个等效True的值，则压制异常；否则，继续抛出异常

```
class Point:
    def __init__(self):
        print('init')

    def __enter__(self):
        print('enter')
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(exc_type)
        print(exc_val)
        print(exc_tb)
        print('exit')
        return "abc"

p = Point()
with p as f:
    raise Exception('New Error')
    print('do sth.')

print('outer')
```


练习

为加法函数计时
方法1、使用装饰器显示该函数的执行时长
方法2、使用上下文管理方法来显示该函数的执行时长

```
import time

def add(x, y):
    time.sleep(2)
    return x + y
```