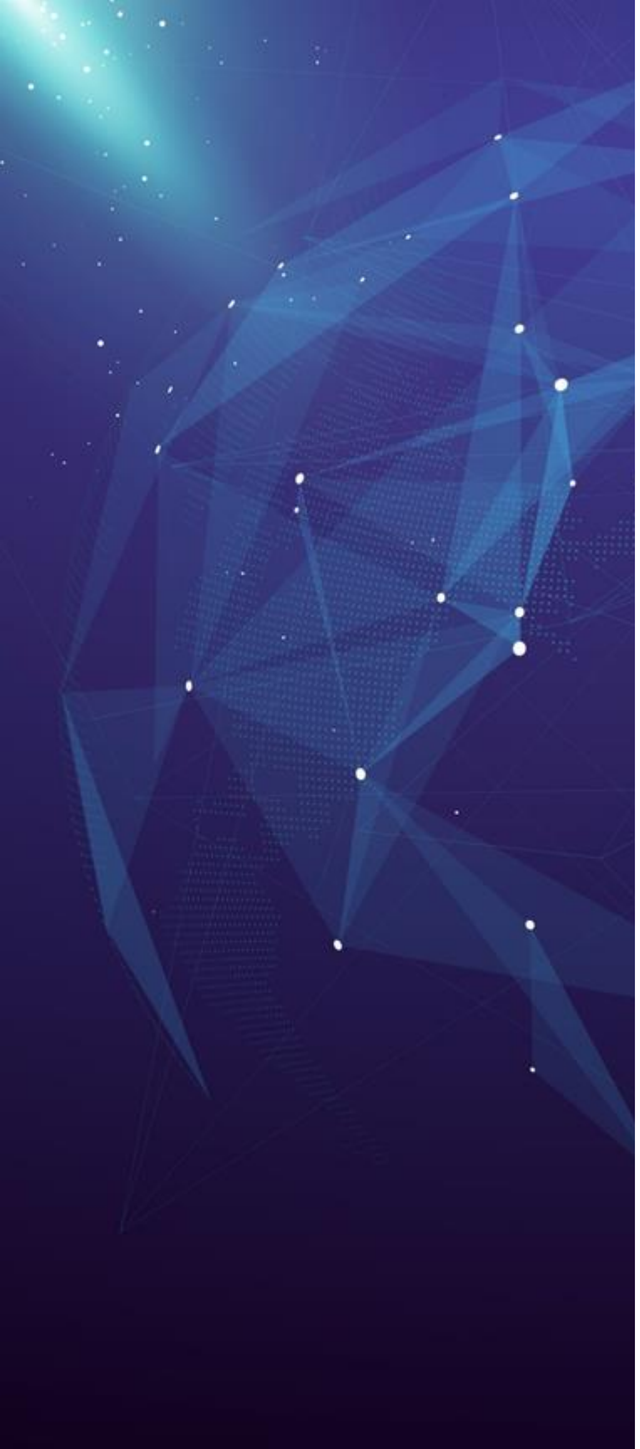
蓝鲸智云 腾讯云

Python基础（二）

蓝鲸智云讲师



课程内容

- 异常处理
- Python函数
- 函数式编程
- 装饰器函数
- 类的高阶用法
- 编码规范

异常处理

异常处理 -- 思考

- 思考：如何处理 0 / 0的问题？

```
In [3]: 0 / 0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-3-b761d17a0499> in <module>()
----> 1 0 / 0

ZeroDivisionError: division by zero

In [4]:
```

- 重点：
  - 解析器会尝试抛出异常
  - 如果有捕获，则处理之
  - 否则一直上浮到解析器出，程序退出

异常处理 -- 基本语法

- 基本语法回顾

```
try:
    foo()

except Exception as error:
    # 处理异常
    do_exception_work()

else:
    # 如果无异常抛出
    success_work()

finally::
    # 必定会执行的代码块
    clean_work()
```

异常处理 -- 内置异常

- 内置的异常类

1. 属性异常：找不到对应的属性

```
In [4]: import os

In [5]: os.hahaha
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-08fa713dc0f5> in <module>()
----> 1 os.hahaha

AttributeError: module 'os' has no attribute 'hahaha'
```

异常处理 -- 内置异常

- 内置的异常类

2. 索引异常：索引值超过了数组的长度

```
In [6]: a = []

In [7]: a[1]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-7-3ef3908cabc7> in <module>()
----> 1 a[1]

IndexError: list index out of range
```

异常处理 -- 内置异常

- 内置的异常类

3. 键值异常：字典中找不到需要的键值

```
In [8]: a = {}

In [9]: a['1']
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-9-390e40a6dcaf> in <module>()
----> 1 a['1']

KeyError: '1'
```

```
In [10]: a.get('1', 'default')
Out[10]: 'default'
```

异常处理 -- 内置异常

- 内置的异常类

4. 引入异常：找不到希望引入的模块

```
In [11]: import hahah
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-11-2a193ae048ee> in <module>()
----> 1 import hahah

ModuleNotFoundError: No module named 'hahah'
```

py3

```
In [68]: import hahah
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-68-2a193ae048ee> in <module>()
----> 1 import hahah

ImportError: No module named hahah
```

py2

异常处理 -- 内置异常

- 内置的异常类

5. 语法错误：文本格式不符合python语法（唯一不是运行时异常）

```
In [70]: def :
  File "<ipython-input-70-f200e4e0bbe8>", line 1
    def :
        ^
SyntaxError: invalid syntax
```

异常处理 -- 内置异常

- 内置的异常类

6. 编码错误：编码或者解码字符集不能完成任务

```
In [71]: u'哈哈哈'.encode('ascii')
---------------------------------------------------------------------------
UnicodeEncodeError                        Traceback (most recent call last)
<ipython-input-71-4a1c34c0984c> in <module>()
----> 1 u'哈哈哈'.encode('ascii')

UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-2: ordinal not in range(128)

In [72]: '哈哈哈'.decode('ascii')
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
<ipython-input-72-762fe36513ad> in <module>()
----> 1 '哈哈哈'.decode('ascii')

UnicodeDecodeError: 'ascii' codec can't decode byte 0xe5 in position 0: ordinal not in range(128)
```

异常处理 -- 内置异常

- 内置的异常类

7. 输入输出错误：磁盘/文件操作异常（ Python3 归入到系统异常中 ）

```
In [69]: open('hahaha', 'r')
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
<ipython-input-69-d876a7b715c2> in <module>()
----> 1 open('hahaha', 'r')

IOError: [Errno 2] No such file or directory: 'hahaha'
```

异常处理 -- 内置异常

- 内置的异常类

8. windows错误：windows平台专有的异常，从系统API调用中产生

exception **WindowsError**

Raised when a Windows-specific error occurs or when the error number does not correspond to an **errno** value. The **winerror** and **strerror** values are created from the return values of the **GetLastError()** and **FormatMessage()** functions from the Windows Platform API. The **errno** value maps the **winerror** value to corresponding `errno.h` values. This is a subclass of **OSError**.

New in version 2.0.

Changed in version 2.5: Previous versions put the **GetLastError()** codes into **errno**.

Python函数

Python函数 -- 思考

- 思考：如何复用我的代码？

```
import os

# 希望可以找到一个路径下的所有日志文件
count = 0
for file_name in os.listdir('./'):

    if file_name.endswith('.log'):
        print('find log file: %s.' % file_name)
        count += 1

print('total %s log file(s) found' % count)
```

```
# 希望可以找到一个路径下的所有XML文件
count = 0
for file_name in os.listdir('./'):

    if file_name.endswith('.xml'):
        print('find xml file: %s.' % file_name)
        count += 1

print('total %s log file(s) found' % count)
```

两段代码是否高度相似？

Python函数 -- 思考

- 函数可以协助你将代码进行管理

```
import os

def foo(file_type):

    count = 0
    for file_name in os.listdir('./'):

        if file_name.endswith('.%s' % file_type):
            print('find %s file: %s.' % (file_type, file_name))
            count += 1

    print('total %s %s file(s) found' % (count, file_type))

    return count

log_result = foo('log')
xml_result = foo('xml')
```

Python函数 -- 函数组成

- 函数的组成

```
import os

def foo(file_type):

    count = 0
    for file_name in os.listdir('./'):

        if file_name.endswith('.%s' % file_type):
            print('find %s file: %s.' % (file_type, file_name))
            count += 1

    print('total %s %s file(s) found' % (count, file_type))

    return count

log_result = foo('log')
xml_result = foo('xml')
```

函数名

参数列表

函数体

返回值

Python函数 -- 参数赋值

- 参数的赋值方式

```
log_result = foo('log')
```

标准调用

```
xml_result = foo(file_type='xml')

file_type = 'log'
log_result_2 = foo(file_type=file_type)
```

关键字调用

Python函数 -- 参数赋值

- 参数的赋值方式

```
def foo(file_type, file_path='./'):

    count = 0
    for file_name in os.listdir(file_path):

        if file_name.endswith('.%s' % file_type):
            print('find %s file: %s.' % (file_type, file_name))
            count += 1

    print('total %s %s file(s) found' % (count, file_type))

    return count

log_result_0 = foo('.log')
log_result_1 = foo('.log', './')
log_result_2 = foo('.log', file_path='./')
log_result_3 = foo(file_path='./', file_type='.log')
```

Python函数 -- 参数赋值

- args 与 kwargs

```
def foo(*args, **kwargs):

    print('*args type->[%s] value->[%s]' % (type(args), args))
    print('**kwargs type->[%s] value->[%s]' % (type(kwargs), kwargs))

foo(1, 2, 3, haha=123)
# *args type->[<class 'tuple'>] value->[(1, 2, 3)]
# **kwargs type->[<class 'dict'>] value->[{'haha': 123}]
```

Python函数 -- 返回值

- 返回值

```
def foo(a, b):

    return a + b

result = foo(1, 2)
print('result: %s' % result)
# result: 3
```

Python函数 -- 返回值

- 返回值

```
def foo(a, b):

    print('result: %s' % (a + b))

result = foo(1, 2)
print('result: %s' % result)
# result: None
```

Python函数 -- 返回值

- 返回值

```
def foo(a, b):

    add_result = a + b
    div_result = a / b

    return add_result, div_result

result = foo(1, 2)
print('result: %s, type: %s' % (result, type(result)))
# result: (3, 0.5), type: <class 'tuple'>
```

Python函数 -- 返回值

- 返回值 --- 坑！

```
def foo(a, b):

    add_result = a + b

    return add_result,

result = foo(1, 2)
print('result: %s, type: %s' % (result, type(result)))
# result: (3,), type: <class 'tuple'>
```

函数式编程

函数式编程 -- 思考

- 思考：是否必须完整的写出一个函数？

```
Help on built-in function filter in module __builtin__:

filter(...)
    filter(function or None, sequence) -> list, tuple, or string

    Return those items of sequence for which function(item) is true.  If
    function is None, return the items that are true.  If sequence is a tuple
    or string, return the same type, else return a list.
```

```
def foo(a):

    return a % 2 == 0
```

得不偿失？

```
a = [0, 1, 2, 3, 4, 5]
result = filter(foo, a)
print('result: %s' % list(result))
# result: [0, 2, 4]
```

函数式编程 -- 用途

- 我们可以使用匿名函数来替代

匿名函数

```
a = [0, 1, 2, 3, 4, 5]
filter_result = filter(lambda x: x % 2 == 0, a)
```

- 优势：结构体简单，函数体简短场景适用
  即用即弃

函数式编程 -- 使用场景

- 其他的一些常见应用场景

```
from functools import reduce

a = [0, 1, 2, 3, 4, 5]
filter_result = list(filter(lambda x: x % 2 == 0, a))
map_result = list(map(lambda x: x ** x, a))
reduce_result = reduce(lambda a, b: a + b, a, 0)
print('filter_result: %s, map_result: %s, reduce_result: %s' % (filter_result, map_result, reduce_result))
# filter_result: [0, 2, 4], map_result: [1, 1, 4, 27, 256, 3125], reduce_result: 15
```

- 注意：
  - 不需要return
  - 多个参数的方法

装饰器函数

装饰器函数 -- 思考

- 思考：如何去改变别人已经写好的函数？

  我希望看到函数执行开始和结束时间

```
import time

def fun():

    print('haha, this is fun~')

print('start time: %s' % time.time())
fun()
print('end time: %s' % time.time())
# start time: 1516458317.0160131
# haha, this is fun~
# end time: 1516458317.016094
```

能否有类似函数的方法解决？
减少重复代码？

装饰器函数 -- 思考

- 思考：如何去改变别人已经写好的函数？

  我希望看到函数执行开始和结束时间

```
import time

def fun():

    print('haha, this is fun~')

def time_fun(real_fun):

    print('start time: %s' % time.time())
    real_fun()
    print('end time: %s' % time.time())

time_fun(fun)
```

- 将函数作为参数传入，解决了重复问题
- 但是需要两个函数名，是否有更优方案？

装饰器函数 -- 装饰器原理

- 装饰器函数

```
def time_fun(real_fun):

    def wrapper():
        print('start time: %s' % time.time())
        real_fun()
        print('end time: %s' % time.time())

    return wrapper

@time_fun
def fun():

    print('haha, this is fun~')

fun()
```

```
def time_fun(real_fun):

    def wrapper():
        print('start time: %s' % time.time())
        real_fun()
        print('end time: %s' % time.time())

    return wrapper

def fun():

    print('haha, this is fun~')

fun = time_fun(fun)
```

装饰器函数 -- 高阶用法

- 装饰器函数 --- 高阶用法

1. 修饰器不断累加

```
@dec_1
@dec_2
def fun():
    # do something
    pass
```

2. 修饰器接受参数

```
def dec_1(val):
    def wrapper(fun):
        # do something with val
        def real_fun():
            print 'got val : %s ' % val
            fun()
        return real_fun
    return wrapper

@dec_1(123)
def fun():
    # do something
    pass
fun = dec_1(val)(fun)
```

类的高阶用法

类的高阶用法 -- 思考

- 思考：希望类可以有一个属性，返回某个属性的绝对值

```
class MyClass(object):

    def __init__(self, a):

        self.a = a

    def abs_a(self):
        return abs(self.a)

a = MyClass(-3)
print('result: %s' % MyClass.abs_a())
# result: 3
```

是否必须使用方法暴露？

类的高阶用法 -- property

- Property修饰器可以将函数变为一个属性使用

```
class MyClass(object):

    def __init__(self, a):

        self.a = a

    @property
    def abs_a(self):
        return abs(self.a)

    @abs_a.setter
    def abs_a(self, new_a):
        self.a = new_a

a = MyClass(-3)
print('result: %s' % MyClass.abs_a)
a.a = -10
print('result: %s' % MyClass.abs_a)
# result: 3
# result: 10
```

注意：

1. 可以直接像使用属性一样访问abs_a
2. 通过实现setter方法，可以像操作属性一样进行赋值

类的高阶用法 -- classmethod

- Classmethod可以将方法作为类方法而非实例方法使用

```
class MyClass(object):

    def __init__(self, a, b):

        self.a = a
        self.b = b

    @classmethod
    def make_my_class(cls):

        return cls(1, 2)

c = MyClass.make_my_class()
```

注意：

1. 方法的第一个参数改为cls，指向类本身
2. 通过类来调用而非实例调用

类的高阶用法 -- staticmethod

- Classmethod可以将方法作为类方法而非实例方法使用

```
class MyClass(object):

    def __init__(self, a, b):

        self.a = a
        self.b = b

    @staticmethod
    def add_result():

        return 1 + 2

print('result: %s' % MyClass.add_result())
# result: 3
```

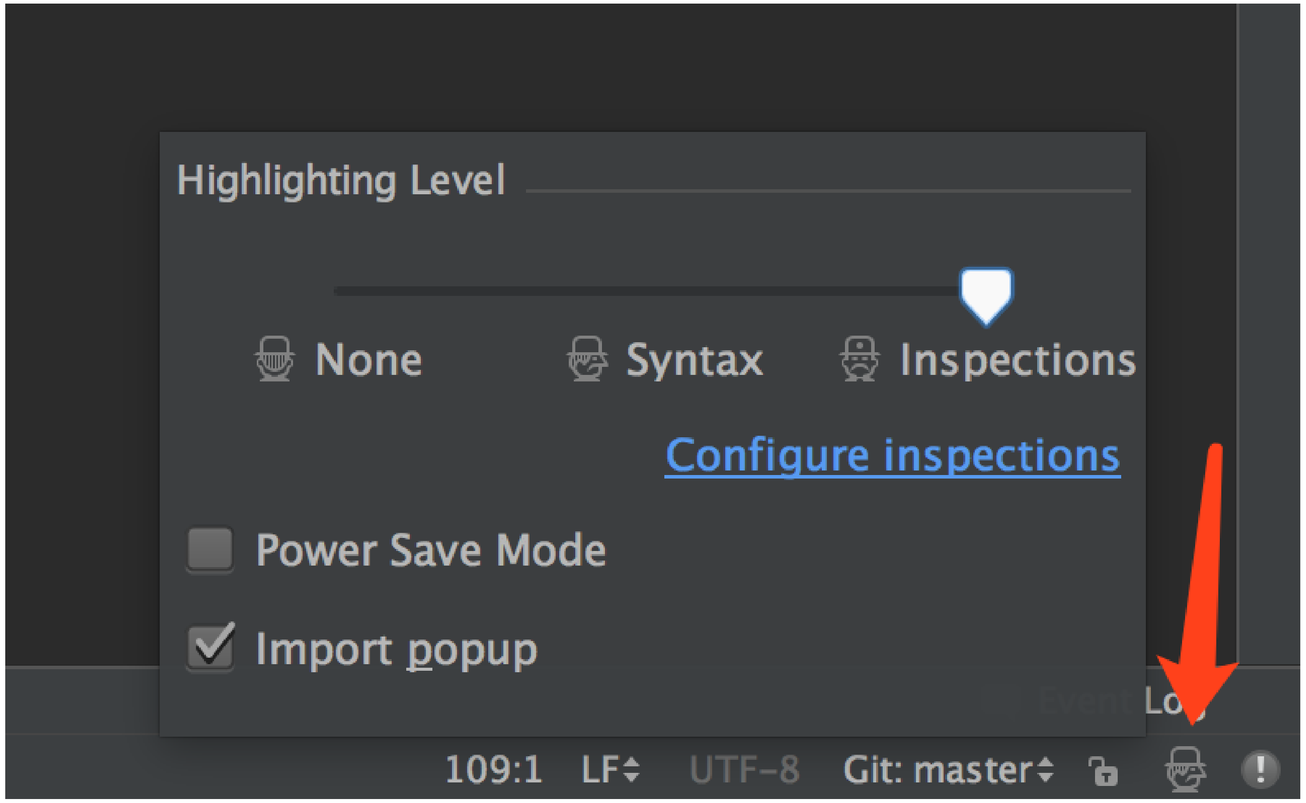
注意：

1. 方法不再接受任何默认参数
2. 可以通过实例或者类来触发调用

Python编码规范

Python编码规范 -- Pycharm设置

- PyCharm中自带有PEP8的检查



Python编码规范 -- 编辑规范

- 常见的Python编码规范
  - 分号：不要在行尾加分号, 也不要用分号将两条命令放在同一行
  - 行长度：每行不超过80个字符，这会让我们的代码更急凑，更可读
  - 缩进：使用4个空格来缩进代码，不要使用tab，或者tab空格混用
  - 空行：函数或者类定义之间空2行， 方法定义之间空1行
  - 导入格式：每个导入应该独占一行
  - 语句：通常每个语句应该独占一行

Python编码规范 -- 命名规范

- 命名：
- module_name
- package_name
- method_name
- instance_var_name
- function_parameter_name
- local_var_name
- function_name
- ClassName
- ExceptionName
- GLOBAL_VAR_NAME

- 约定：
- 所谓”内部(Internal)”表示仅模块内可用, 或者, 在类内是保护或私有的.

- 用单下划线(_)开头表示模块变量或函数是protected的(使用from package import * 时不会包含).

- 用双下划线(__)开头的实例变量或方法表示类内私有.

- 将相关的类和顶级函数放在同一个模块里. 不像Java, 没必要限制一个类一个模块.

- 对类名使用大写字母开头的单词(驼峰命名), 但是模块名应该用小写加下划线的方式(如lower_with_under.py). 尽管已经有很多现存的模块使用类似于CapWords.py这样的命名, 但现在已经不鼓励这样做, 因为如果模块名碰巧和类名一致, 这会让人困扰.

学习资料

- 异常：
  - Py2： https://docs.python.org/2/tutorial/errors.html
  - Py3： https://docs.python.org/3.6/tutorial/errors.html
- Six模块：
  - https://pythonhosted.org/six/
- Python进阶： 《Python Cookbook》

作业：Python基础（二）

1. 小明是业务运维，经常需要在服务器上找文件。请帮他实现一个函数，可以接受两个参数：路径及文件后缀名。然后函数按照参数的要求，遍历该路径下的所有符合该后缀名的文件，并打印到屏幕显示。

2. 后面发现小明的函数太危险，可能删除系统路径下的文件。在上述函数的基础上，实现一个修饰器，修饰器可以接受一个参数：禁止遍历的路径。以防止误操作。

- 要求：
  1. 需要考虑各种异常情况
  2. 需要考虑良好的扩展和通用性
  3. 注意平台的差异性（windows & linux）

下集预告

前端基础（一）

- WEB 基本知识
- HTML页面结构
- CSS样式
- 蓝鲸 MagicBox 实战

蓝鲸智云

腾讯云

蓝鲸智云公众号

蓝鲸高校版交流群