## 第八章 基础向应⽤的过渡 章节时长：60min

第一节 异常处理 5.5min

第二节 面向对象技术简介 （共18min）

# 8.2.1 类（class） 6min

# 8.2.2 构造函数（constructors） 7min

# 8.3.3 继承 5min

第三节 模块 11min

第四节 包（package） 7min

第五节 生成随机值 8min

第六节 使用python中的目录 6min

第七节  python官方的第三方库的仓库PyPI 5min

补充：Anaconda的介绍、安装及使⽤ 10min（估计）

* 1. 第一节：异常处理
     1. 什么是异常处理？

当代码运行时发生错误时，程序就无法继续执行，此时需要知道错误发生的位置和类型，并对代码进行纠正。Python的异常处理通常使用try/except/finally语句检测try语句块中的错误，从而让except语句捕获异常信息并处理，或无论是否发生错误都正常执行finally语句块。

* + 1. 异常处理实现
       - 1. try/except语句的语法：

try/except的语法结构如下

try:

<语句> #运行别的代码

except <错误类型>：

<语句> #如果在try部份引发了该类异常

except <错误类型>，<数据>:

<语句> #如果引发了该类异常，获得附加的数据

else:

<语句> #如果没有异常发生

* + - * 1. Try的工作原理：

尝试执行try语句后的代码，若出现与异常则执行except语句，不出现则执行else语句（如果有else语句的话）。其中，except可以以错误类型分成多个，出现指定类型的错误则执行相应的except语句。

* + - * 1. try/except语句使用举例:

try:

fh = open("testfile", "w")

fh.write("这是一个测试文件，用于测试异常!!")

except IOError:

print "Error: 没有找到文件或读取文件失败"

else:

print "内容写入文件成功"

fh.close()

代码解释：执行try后的语句，若出现错误类型为IOError的异常，则输出“Error: 没有找到文件或读取文件失败”，反正则输出“内容写入文件成功”

* + - * 1. 使用except而不带任何异常类型：

当except后不跟任何异常类型时，表示出现任何类型的异常都执except后的语句，即：

try:

正常的操作

......................

except:

发生异常，执行这块代码

......................

else:

如果没有异常执行这块代码

* + - * 1. 使用except而带多种异常类型：

若多种异常类型都要求执行同样的except代码，可以使用以下语句：

try:

正常的操作

......................

except(Exception1[, Exception2[,...ExceptionN]]]):

发生以上多个异常中的一个，执行这块代码

......................

else:

如果没有异常执行这块代码

* + - * 1. try/finally语句

try-finally 语句无论是否发生异常都将执行finally后的代码。

try:

<语句>

finally:

<语句> #退出try时总会执行

* + - * 1. 举例：

try:

fh = open("testfile", "w")

fh.write("这是一个测试文件，用于测试异常!!")

finally:

print "Error: 没有找到文件或读取文件失败"

解释：执行try后的语句，不论是否出现异常都执行finally后的语句。

Tip：使用try-finally可以使try后的代码即使出现异常也不会停止运行程序。

* 1. 第二节：面向对象技术简介
     + - 1. 面向对象简介

类（class）与对象：指用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。

类的举例：如一个班级的所有学生，该班级学生的数据都包含性别、年龄、成绩等属性，则该班级的所有学生为一个类，而其中的某个学生为一个对象。一个类中的所有对象都有共同的属性（性别、年龄、成绩等）。类似一个数据库。

* + - * 1. 创建类

使用class语句可以创建一个类：通常用法如下：

class 类的名称:

def \_\_init\_\_(self, 属性1, …,属性n):

self.属性1 =

……

self.属性n =

def 函数1(self):

def 函数n(self):

解释：定义一个类（如一个班级），包含属性1-n共n种属性（如成绩、年龄、性别等）。使用self.属性n=xxx 为第n个属性赋值（如定义self.成绩=GPA）。可以在类中定义函数对类的变量进行运算（如定义函数1为成绩X年龄，则可以通过调用函数1得到该结果）

* + - * 1. 创建类举例：

class Employee:

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self):

print "Total Employee %d" % Employee.empCount

def displayEmployee(self):

print "Name : ", self.name, ", Salary: ", self.salary

* + - * 1. 访问属性

创建好一个类后，可以通过符号点(“.”)访问类中某一对象的属性，例如想要访问班级中张三同学的成绩，若已定义students为一个类，grade为该类的其中一个属性，则语句“张三.grade()”即代表张三同学的成绩，可以使用print语句输出该属性。

举例如下（接上文代码）：

"创建 Employee 类的第一个对象"

emp1 = Employee("Zara", 2000)

"创建 Employee 类的第二个对象"

emp2 = Employee("Manni", 5000)

emp1.displayEmployee()

emp2.displayEmployee()

输出结果：

Name : Zara ,Salary: 2000

Name : Manni ,Salary: 5000

* 1. 第三节：模块
     + - 1. Python模块简介

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和Python语句。模块能定义函数，类和变量，模块里也能包含可执行的代码。

python有成熟的模块可为用户调用，其中包含许多方便直接使用的函数，方便用户实现多样化功能。

* + - * 1. 模块调用

使用某一模块及其内部的函数之前需要先调用该模块，方法有多种：

import module语句

例如：若需要使用模块math，则需先运行

import math

from module import function语句

针对模块内的某一函数调用。

例如：若需要使用模块fib 的 fibonacci 函数，则先运行

from fib import fibonacci

from module import \* 语句

需要使用某一模块中的所有函数时可以使用。

例如：需要一次性导入math模块中的所有函数为自己所用，则先运行

from math import \*

* 1. 第四节：包
     + - 1. 什么是包

包是一个文件夹，其中包含多个模块文件。它定义了一个由模块及子包，和子包下的子包等组成的 Python 的应用环境。该文件夹下必须存在 \_\_init\_\_.py 文件, 该文件的内容可以为空。\_\_init\_\_.py 用于标识当前文件夹是一个包。

* + - * 1. 包的举例

例如：在文件夹package\_runoob下有“\_\_init\_\_.py”，“runoob1.py”，“runoob2.py”三个文件，其中后两者为两个模块，分别包含多个函数。

现在假设runoob1下有runoob11,runoob12两个函数，runoob模块下有runoob21,runoob22两个函数。则

可以通过以下语句导入相应的函数：

from package\_runoob.runoob1 import runoob11

from package\_runoob.runoob1 import runoob12

from package\_runoob.runoob2 import runoob21

from package\_runoob.runoob2 import runoob22

* 1. 第五节：生成随机值

random()函数：

随机生成一个实数，它在[0,1)范围内。

uniform(x, y)：

随机生成一个实数，它在[x,y]范围内。

choice(seq)：

从序列的元素中随机挑选一个元素

如：

random.choice(range(10))

从0到9中随机挑选一个整数。

randrange ([start,] stop [,step])：

从指定范围内，按指定基数递增的集合中获取一个随机数，基数默认值为 1

如：

randrange(1,10,2)

表示在[1,3,5,7,9]中随机取一个数字

shuffle(lst)

将列表的所有元素随机排序

* 1. 第六节：使用Python中的目录
     + - 1. Python目录使用的包

pathlib 模块下的path包

使用以下命令调用

from path lib import Path(注意大小写)

* + - * 1. 两种路径表示方式

绝对路径举例：

Windows：Path(“c:\Program Files\Microsoft”)

在此电脑中文件路径下可点击直接获取路径：



Mac:Path( /usr/local/bin)

相对路径

与当前py文件同级的文件，可以省去绝对路径前的e:\program file等一系列路径

如py文件在Microsoft下，且要调用的文件也在该目录下，则可直接使用：

Path(“Microsoft\某文件”)

调用该文件

Tip：一般在调用后会将该路径赋值给一个变量，如

path=Path(“c:\Program Files\Microsoft”)

* + - * 1. path常用操作

检查路径是否存在

print(path.exists())

输出结果为布尔值，存在为True，不存在为False

创建新路径

path.mkdir()

无返回值，运行结果为创建一个与path同名的新文件或文件夹

删除路径

path.rmdir()

无返回值，运行结果为删除path同名的文件或文件夹

获取所有同类文件

path.glob(“\*.\*”)：获取所有路径下的所有文件

path.glob(“\*.py”)：获取所有路径下的py文件

path.glob(“\*.xlsx”)：获取所有路径下的xlsx文件

返回一个 生成器

* 1. 第七节：Python官方的第三方库的仓库PyPI
     + - 1. 什么是PyPI

pypi 是 Python Package Index 的首字母简写，其实表示的是 Python 的 Packag 索引，也是 Python 的官方索引。PyPI是python官方的第三方包的仓库，所有人都可以下载第三方包或上传自己开发的包到PyPI。PyPI推荐使用pip包管理器来下载第三方包。

例如：

当用户需要使用python取用某年的股票数据时，若用户自己写代码，需要一点一点写出爬虫、输出文件等一系列代码，且需要经过调试和功能完善才足够使用。而一些用户已经写好可用代码，并上传到PyPI库中，于是我们可以使用PyPI中已有的python包来实现几行代码完成取用股票数据的操作，省时省力。

* + - * 1. 如何使用PyPI

查阅过自己需要使用的python包名，在命令行中输入”pip install 包名称”，回车后等待安装完成即可使用，使用方法同包的调用。

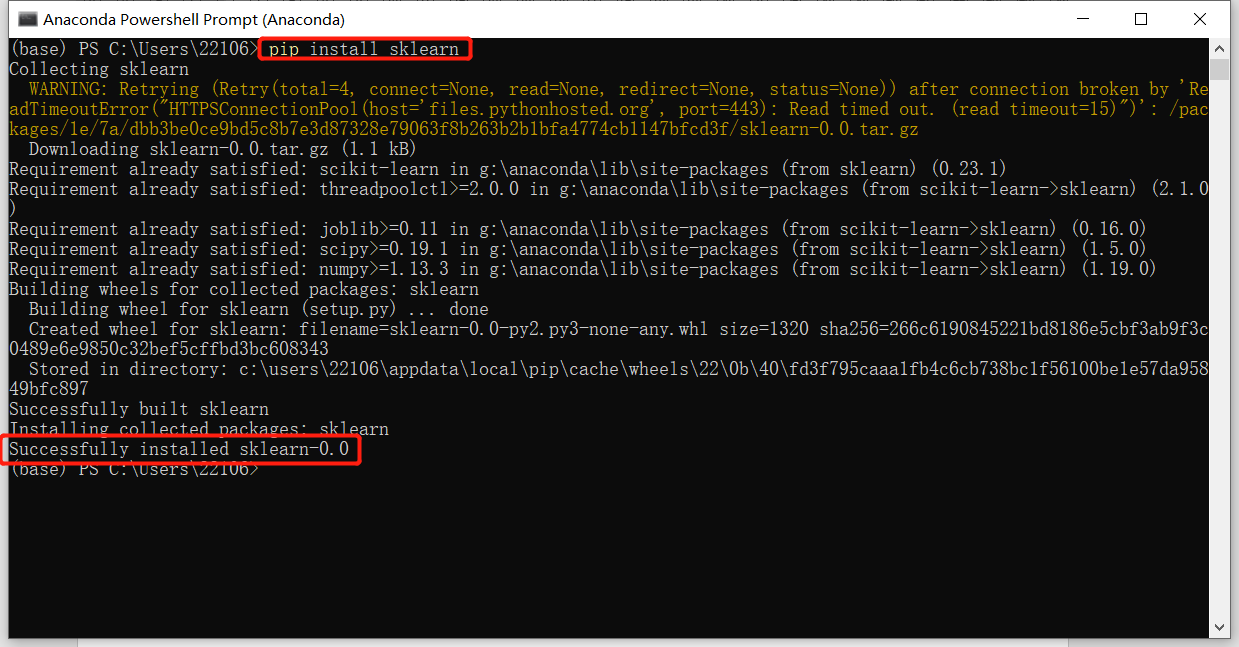
Windows系统打开命令行的方法：win+r，输入”cmd”后运行

Mac系统在终端（Terminal）运行pip命令即可

* + - * 1. 使用实例

例如需要使用sklearn包进行机器学习操作

首先，打开命令行/终端，输入pip install sklearn并运行，等待安装完成



打开python，新建文件，使用from sklearn import \*导入sklearn包

后续即可使用sklearn中内置的函数编写代码。

* 1. 补充节：Anaconda的介绍、安装及使用
     + - 1. Anaconda是什么？

是Python的一个发行版，相当于在原有python基础上添加了更多功能。主要特色是方便的包管理器及内置的使用python包。

* + - * 1. 为什么使用Anaconda

官方python编译器无法满足代码补全等方便功能，且python在实际使用中往往需要调用多种第三方包，每个都使用pip安装则不胜其烦，anaconda中内置了大量常用包，可以使用户免去安装步骤。

* + - * 1. 安装方法

进入Anaconda官网下载安装包后运行安装，该软件开源免费。

若官网下载速度过慢，可以使用清华大学开源软件镜像站下载，网址如下：

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

* + - * 1. 使用方法

安装完成后所有内置包都可以直接使用

安装完成后打开Anaconda，内置有名为“Spyder”的编译器，可作为原有python编译器的替代。

* 1. 附录：常见异常类型及解释

|  |  |
| --- | --- |
| 异常名称 | 描述 |
|  |  |
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| Exception | 常规错误的基类 |
| StopIteration | 迭代器没有更多的值 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| StandardError | 所有的内建标准异常的基类 |
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| ZeroDivisionError | 除(或取模)零 (所有数据类型) |
| AssertionError | 断言语句失败 |
| AttributeError | 对象没有这个属性 |
| EOFError | 没有内建输入,到达EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| IOError | 输入/输出操作失败 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| ImportError | 导入模块/对象失败 |
| LookupError | 无效数据查询的基类 |
| IndexError | 序列中没有此索引(index) |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于Python 解释器不是致命的) |
| NameError | 未声明/初始化对象 (没有属性) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError | Python 语法错误 |
| IndentationError | 缩进错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| ValueError | 传入无效的参数 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |