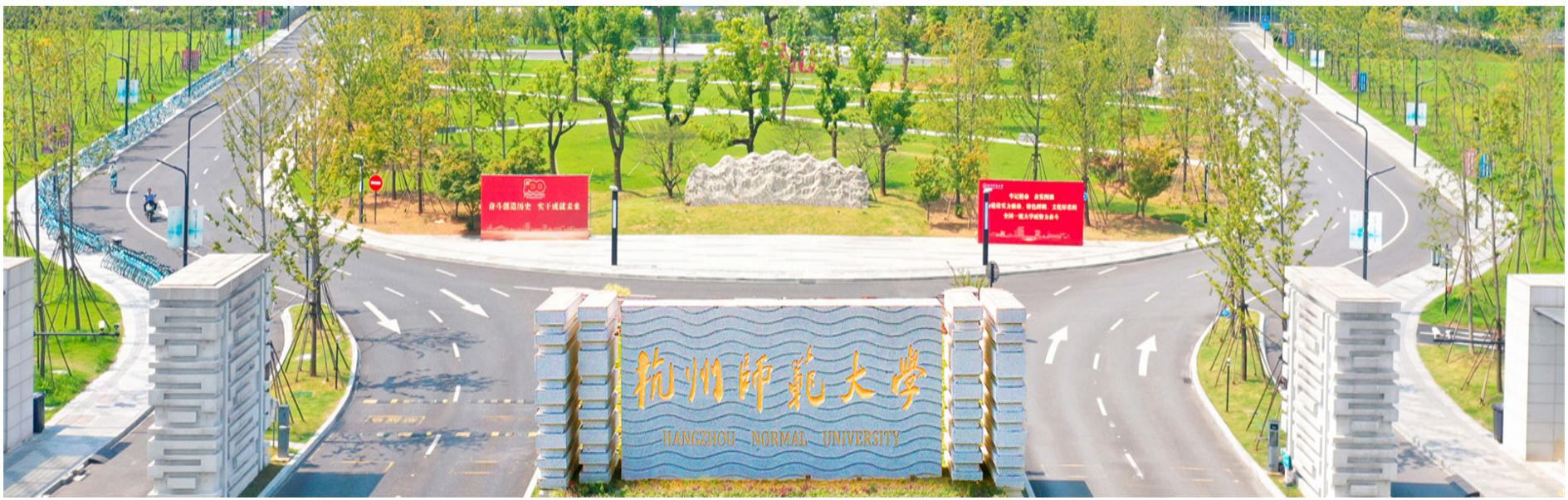
第六讲 - 代码组织与异常处理

张建章

阿里巴巴商学院

杭州师范大学

2022-09

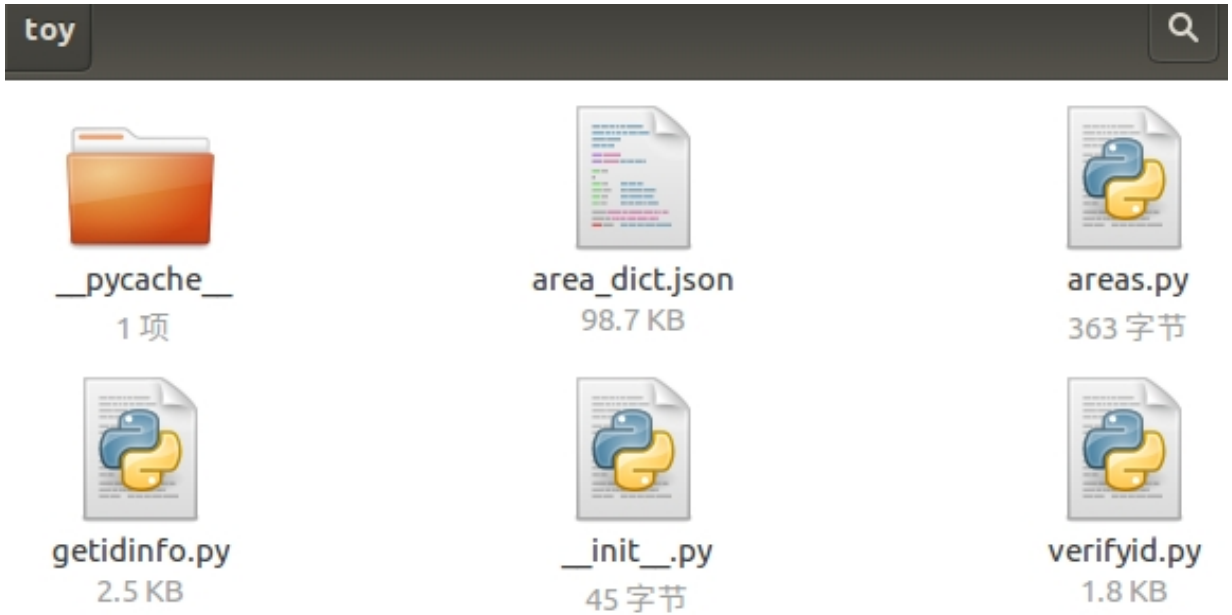


目录 I

模块 (module): 把实现相关功能的代码放到一个 py 文件中，就是一个模块，其中可以包含类、函数、变量、可执行语句、导入其他模块等。

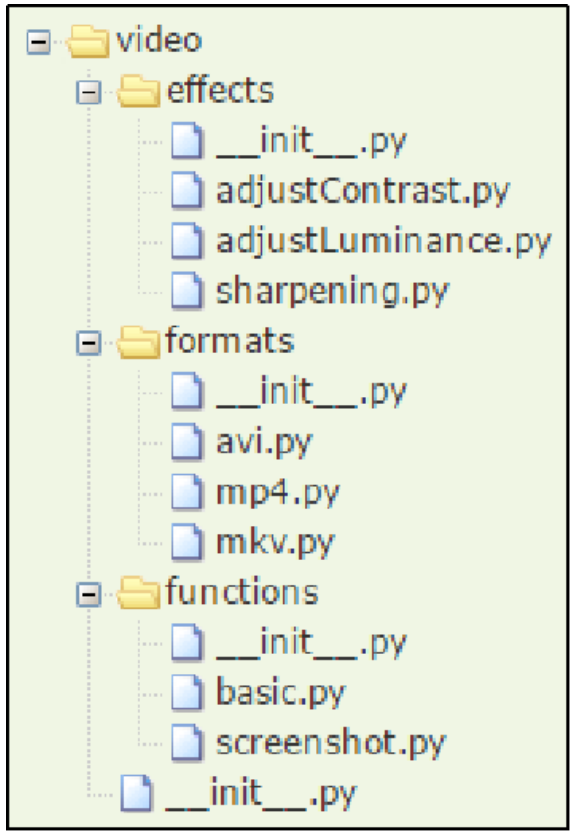
包 (package): 把多个功能模块放到同一个文件夹构成一个包。

例如，把验证身份证号合法性的代码 (多个函数) 放到名为 verifyid.py 的文件中，把获取身份证号信息的代码 (多个函数) 放到名为 getidinfo.py 的文件中，就创建了两个名为 verifyid 和 getidinfo 的模块；把这两个模块 (两个 py 文件) 放到名为 toy 的文件夹中，就创建了一个名为 toy 的文件夹。如下图所示：



1. 模块与包的定义

模块与包的关系



包的结构 (包-子包-模块)

以 Anaconda 为例，Python 的内置 (自带) 的模块与包的存放路径为 `anaconda3/lib/python3.9/`，自己安装的第三方包的路径为 `anaconda3/lib/python3.9/site-packages`，也可自己编写包或模块。

使用 `pip` 进行包管理，可在PyPI和GitHub查找所需第三方包，常用命令 (在命令行运行，如需要在 jupyter 中运行，则在命令前加上!) 如下：

```
pip install <Package Name> # 安装包
pip install <Package Name>==<Version> # 安装包的特定版本
# 使用清华镜像源安装包，速度更快，默认使用国外的镜像，速度较慢
pip install <Package Name> -i
↪  https://pypi.tuna.tsinghua.edu.cn/simple
pip show # 查看已安装的包信息
pip list # 列出已安装的所有包
pip list --outdated # 列出需要更新的包
pip install --upgrade <Package Name> # 升级包
pip uninstall <Package Name> # 卸载包
```

实操: 使用清华镜像源安装中文分词包 jieba。

Python 导入包或模块 (及其中的内容) 有如下两种方式，其中 something 为“点式结构”，别名是可选的。

第一种: `import something [as alias]`

第二种: `from something import something [as alias]`

包是一种用“点式模块名”构造 Python 模块命名空间的方法，使用“点”连接包结构中各层级的内容 (看下面代码示例)。

```
# 导入包
import toy
# 访问包中的变量，包名 点 模块名 点 变量名
toy.areas.AREA_DICT
# 调用模块中的函数，模块名 点 函数名
toy.verifyid.verify_id('330282200202184l0X') # VALID
# 将函数赋值给变量vi，这样调用函数时，就不用写长长的名字了
vi = verifyid.verify_id
vi('999') # INVALID
```

模块的常见属性有 `__doc__` 和 `__name__`，前者为 py 文件头部的注释，用于说明该模块实现的功能、用法等，后者为模块的名字。

```
# 导入包中的一个模块，目标明确，速度快
from toy import verifyid
# 导入包中的所有模块，速度慢
from toy import *
# 导入模块中的一个函数
from toy.verifyid import verify_id
verify_id('999') # INVALID
# 导入模块中的全部内容（所有函数、变量等）
from toy.verifyid import *
# 导入包并起一个别名
import toy as t
# 查看模块的说明文档
print(t.verifyid.__doc__) # check if a ID number is valid.
# 查看模块的名字
print(t.verifyid.__name__) # toy.verifyid
```

注意：上面以 toy 模块为例的示例代码也适用于导入模块中的内容。

在自己编写包的时候，同一个包内，模块之间互相调用的方式有：

绝对引用：`from toy.areas import area_dict`

相对引用：`from .areas import area_dict`

在本课件配套的 toy 包中 verifyid 模块中使用了绝对引用方式，调用 areas 模块中的变量 area_dict，getidinfo 模块中使用了相对引用方式，调用 areas 模块中的变量 area_dict。

语法错误: 代码书写不符合 Python 语法，如选择、循环结构不写冒号。

异常: 代码语法正确，运行时出现错误，如，除 0 错误。

自定义异常: 为便于程序调试，Python 允许用户自定义异常，并使用关键字 `raise` 主动抛出异常。

```
# SyntaxError, 循环结构不写冒号
for i in range(3)
  print(i)
# ZeroDivisionError, 除0错误
10 % 0
# 自定义异常
class CustomError(Exception):
  def __init__(self, message, status):
    super().__init__(message, status)
    self.message = message
    self.status = status

# 主动抛出自定义异常
raise CustomError('Connected Failed', 404)
```

BaseException 类是所有异常类的基类，其有四个子类，除 Exception 类外，其他三个均为系统级异常，Exception 类是所有内置异常类和用户自定义异常类的基类，Python 中异常的分类如下图：

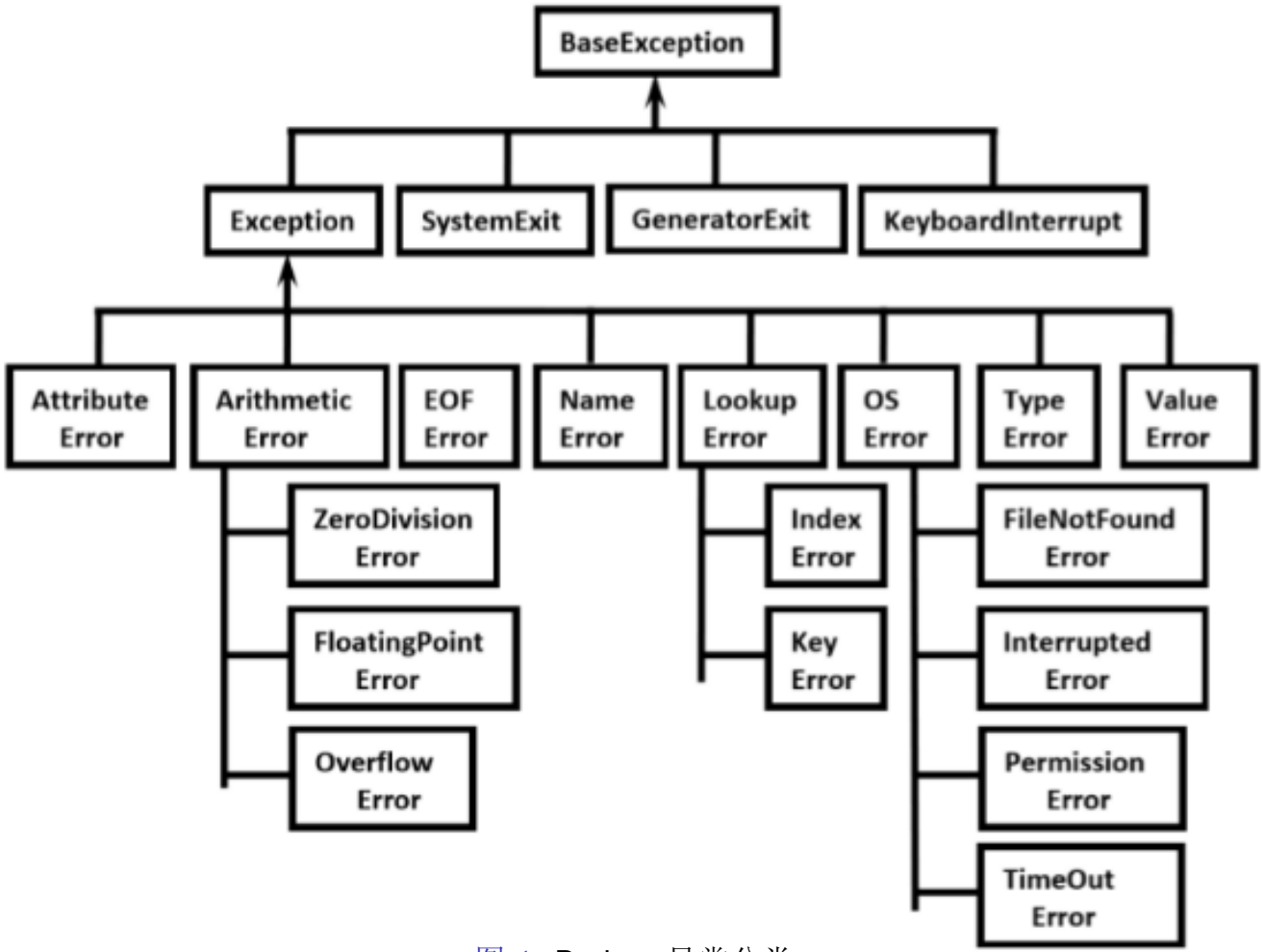


图 1: Python 异常分类

Table 1: 常见 Python 内置异常列表

异常名称	描述	异常名称	描述
Exception	普通错误的基类	AttributeError	对象没有这个属性
IOError	输入/输出操作失败	IndexError	序列中没有此索引
KeyError	映射中没有这个键	NameError	未声明/初始化对象
SyntaxError	Python语法错误	SystemError	一般解释器系统错误
ValueError	传入无效参数	ZeroDivisionError	除0异常
ImportError	导入模块异常	TypeError	类型异常
ReferenceError	引用不存在对象异常	AssertionError	assert语句触发的异常

```
4 + spam*3 # NameError: name 'spam' is not defined
'2' + 2 # TypeError: can only concatenate str (not "int") to str
import kkkkk # ModuleNotFoundError: No module named 'kkkkk'
d = {};d['abc'] # KeyError: 'abc'
l = [1, 2, 3]; l[100] # IndexError: list index out of range
s = 'kkk';s.llower() # AttributeError: 'str' object has no
↪  attribute 'llower'
```

使用 `try-except` 语句捕获并处理异常，格式如下：

```
# 下面是伪代码不要直接运行
try:
  statements # 从这些语句中捕获可能的异常
except[ (ErrorType1, ErrorType2, ...)]:
  statements # 对捕获到的指定异常进行处理
```

```
l = list('Python')
try:
  for i in range(10):
    print(l[i],end = ',')
  print('循环顺利结束.')
except IndexError:
  print('\nERROR: 索引超出范围啦...')
```

try 下面的语句块执行过程中发生异常时，则跳过剩余部分，执行 except 子句，匹配遇到的异常类型，如匹配成功，则执行 except 子句下面的异常处理代码，然后离开 try-except 结构，否则程序报错。

except 子句可以有多个 (类似于多选结构里的多个条件)，try 捕获到异常后，**依次**匹配每个 except 子句后的异常类型，一旦匹配成功，就执行相应的异常处理代码，然后**离开** try-except 结构。

```
l = list('Python')
try:
  for i in range(10):
    print(l[i],end = ',')
  print('循环顺利结束.')
except NameError:
  print('\nERROR: 命名错误...')
except IndexError:
  print('\nERROR: 索引超出范围啦...')
  print(10/0)
```

注意: except 子句下面的异常处理代码在运行时也可能出现新的异常，运行上例后，在处理 IndexError 的过程中又引发了 ZeroDivisionError。

except 后面可以放置多个异常类型 (放在圆括号内，以逗号分割)，表明若多个异常中至少发生一个，则执行该部分异常处理代码，若不放置任何异常类型，则代表可匹配所有的异常类型。

```
l = list('Python')
try:
  for i in range(10):
    print(l[i],end = ',')
  print('循环顺利结束.')
except (NameError, IndexError):
  print('\nERROR: 命名错误或索引错误')

l = list('Python')
try:
  for i in range(10):
    print(l[i],end = ',')
  print('循环顺利结束.')
except:
  print('\nERROR: There is an error.')
```

图 2: 完整的 try-except 结构

try 和 except 必须成对出现，else 子句和 finally 子句是可选的。

下面是使用完整的 try-except 结构的一个示例，其中，except 子句中使用 as 关键字捕获该异常类的示例，便于查看具体异常信息。

```
i = 0
while i < 3:
  try:
    x_input = input("请输入一个数字: ")
    x_int = int(x_input)
    i = 3
  except ValueError as e:
    print("您输入的不是整数，请再次尝试输入！")
    print("具体错误信息如下: {}\n".format(e))
    i += 1
    if i == 3:
      print('三次机会已经用完，明天再试吧.')
  else:
    print("恭喜你，输入正确！")
  finally:
    print("你的输入为: {}".format(x_input))
```

自定义异常

- Python 允许用户自定义异常，描述内置异常未涉及的异常情况，以便于程序调试；
- 通过定义一个继承 Exception 类的派生类来创建自定义异常；
- Python 不会自动抛出或处理任何自定义异常，需要使用 raise 语句在合理的场合手工触发异常；
- 在使用自定义异常时，经常需要在捕获异常的同时获取该异常的实例 (如上页例子中的 e)，以获取存储在异常实例中的数据，在 except 子句中使用 as 关键字加实例名即可。

  通过下页实例来理解上面的知识点。

```
class MyEnameError(Exception):
  def __init__(self, value):
    self.value = value
  def __str__(self):
    return str('''English name can only include alphabet letters
    ↪  and space, your input is: {}'''.format(self.value))

import string
i = 3
while i > 0:
  try:
    ename = input("请输入你的英文名: ")
    if set(ename) - set(string.ascii_letters + ' '):
      raise MyEnameError(ename)
    else:
      break
  except MyEnameError as e:
    print(e)
    i -= 1
    print("你还有{}次输入机会".format(i))
```

断言用于判断一个表达式是否满足，在表达式返回 False 时触发 AssertionError 异常，显示错误提示信息，语法如下：

```
assert expression [, arguments]
```

断言可以在条件不满足程序运行的情况下直接返回错误，而不必等待程序运行时出现错误，例如，我们的代码只能在满足特点条件的机器上运行时，可以先判断当前及其是否符合条件。

```
import sys
assert ('linux' in sys.platform), "该代码只能在 Linux 下执行"

assert 1==1 # 条件为True正常执行，没有指定错误提示信息
print('继续')

salary = -100
assert salary > 0, '工资只能为正数' # 指定了错误提示信息

salary = -100
if not salary > 0:
  raise AssertionError('工资只能为正数')
```

THE END