Python异常、模块与包

目录

Contents

学习目标

Learning Objectives

1. 了解异常的概念

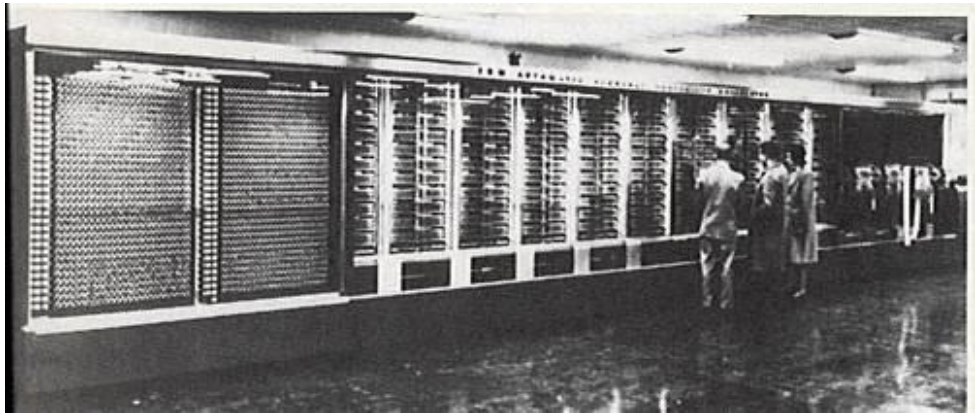
什么是异常

当检测到一个错误时，Python解释器就无法继续执行了，反而出现了一些错误的提示，这就是所谓的“异常”，也就是我们常说的BUG

bug单词的诞生

早期计算机采用大量继电器工作，马克二型计算机就是这样的。

1945年9月9日，下午三点，马克二型计算机无法正常工作了，技术人员试了很多办法，最后定位到第70号继电器出错。负责人哈珀观察这个出错的继电器，发现一只飞蛾躺在中间，已经被继电器打死。她小心地用摄子将蛾子夹出来，用透明胶布帖到“事件记录本”中，并注明“第一个发现虫子的实例。”自此之后，引发软件失效的缺陷，便被称为Bug。



早期的马克型计算机

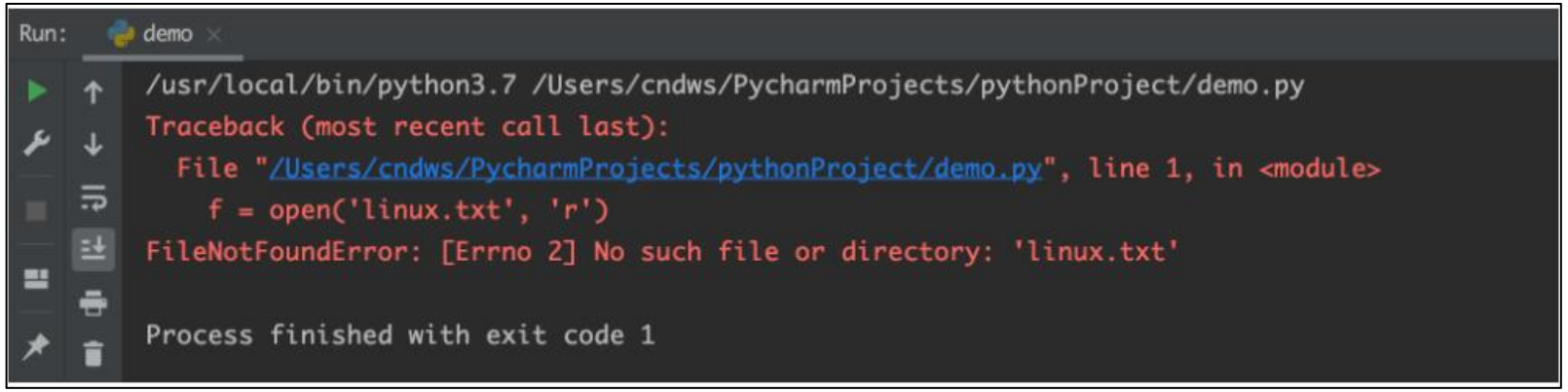
bug：程序错误

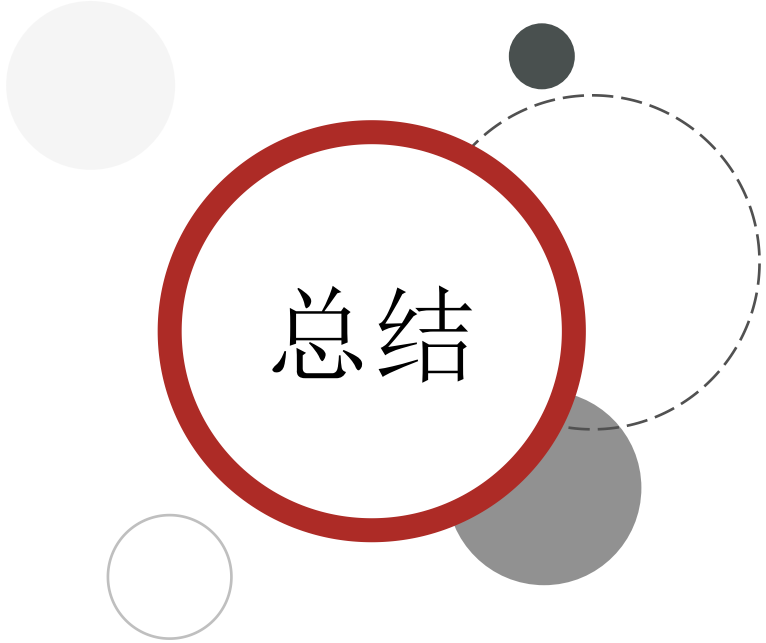
异常演示

例如：以`r`方式打开一个不存在的文件。

```
f = open('linux.txt', 'r')
```

执行结果：





1. 什么是异常：

异常就是程序运行的过程中出现了错误

2. bug是什么意思：

bug就是指异常的意思，因为历史因为小虫子导致计算机失灵的案例，所以延续至今，bug就代表软件出现错误。

目录

Contents

学习目标

Learning Objectives

1. 知道为什么要捕获异常
2. 掌握捕获异常的语法格式

为什么要捕获异常

世界上没有完美的程序，任何程序在运行的过程中，都有可能出现：异常，也就是出现bug

导致程序无法完美运行下去。

我们要做的，不是力求程序完美运行。

而是在力所能及的范围内，对可能出现的bug，进行提前准备、提前处理。

这种行为我们称之为：异常处理（捕获异常）

为什么需要捕获异常

当我们的程序遇到了BUG，那么接下来有两种情况：

① 整个程序因为一个BUG停止运行

② 对BUG进行提醒，整个程序继续运行

显然在之前的学习中，我们所有的程序遇到BUG就会出现①的这种情况，也就是整个程序直接奔溃.

但是在真实工作中，我们肯定不能因为一个小的BUG就让整个程序全部奔溃，也就是我们希望的是达到② 的这种情况

那这里我们就需要使用到**捕获异常**

捕获异常的作用在于：提前假设某处会出现异常，做好提前准备，当真的出现异常的时候，可以有后续手段。

捕获常规异常

基本语法：

```
try:
    可能发生错误的代码
except:
    如果出现异常执行的代码
```

快速入门

需求：尝试以`r`模式打开文件，如果文件不存在，则以`w`方式打开。

```
try:
    f = open('linux.txt', 'r')
except:
    f = open('linux.txt', 'w')
```

捕获指定异常

基本语法：

```
try:
    print(name)
except NameError as e:
    print('name变量名称未定义错误')
```

注意事项

① 如果尝试执行的代码的异常类型和要捕获的异常类型不一致，则无法捕获异常。

② 一般try下方只放一行尝试执行的代码。

捕获多个异常

当捕获多个异常时，可以把要捕获的异常类型的名字，放到except 后，并使用元组的方式进行书写。

```
try:
    print(1/0)
except (NameError, ZeroDivisionError):
    print('ZeroDivision错误...')
```

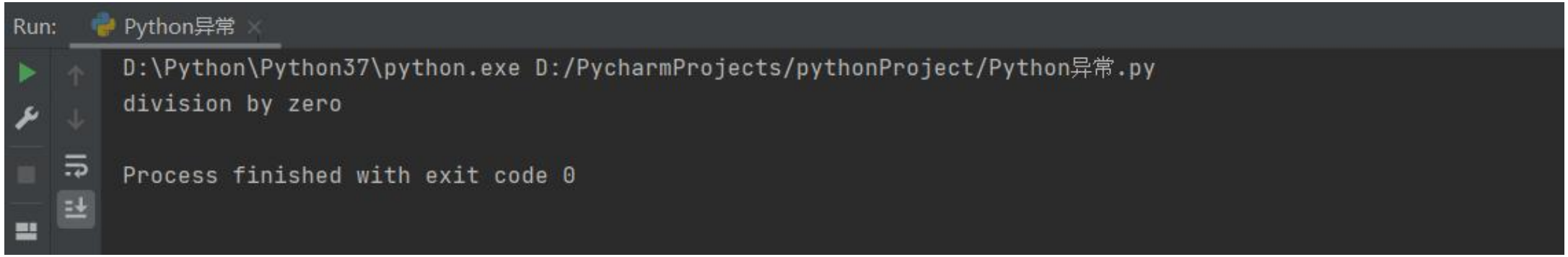
执行结果：

捕获异常并输出描述信息

基本语法：

```
try:
    print(num)
except (NameError, ZeroDivisionError) as e:
    print(e)
```

执行结果：

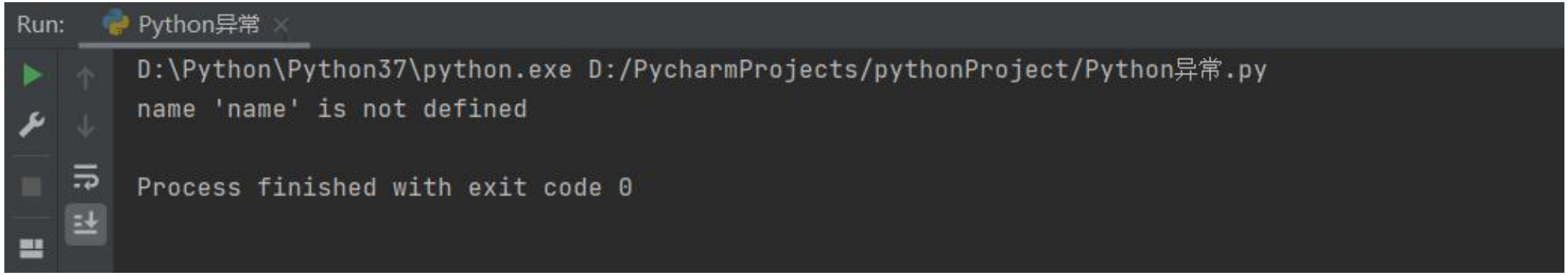


捕获所有异常

基本语法：

```
try:
    print(name)
except Exception as e:
    print(e)
```

执行结果：

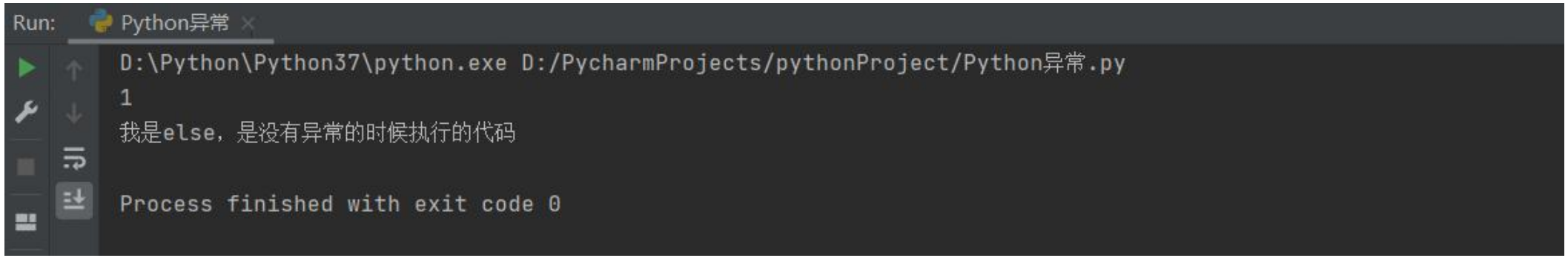


异常else

else表示的是如果没有异常要执行的代码。

```
try:
    print(1)
except Exception as e:
    print(e)
else:
    print('我是else，是没有异常的时候执行的代码')
```

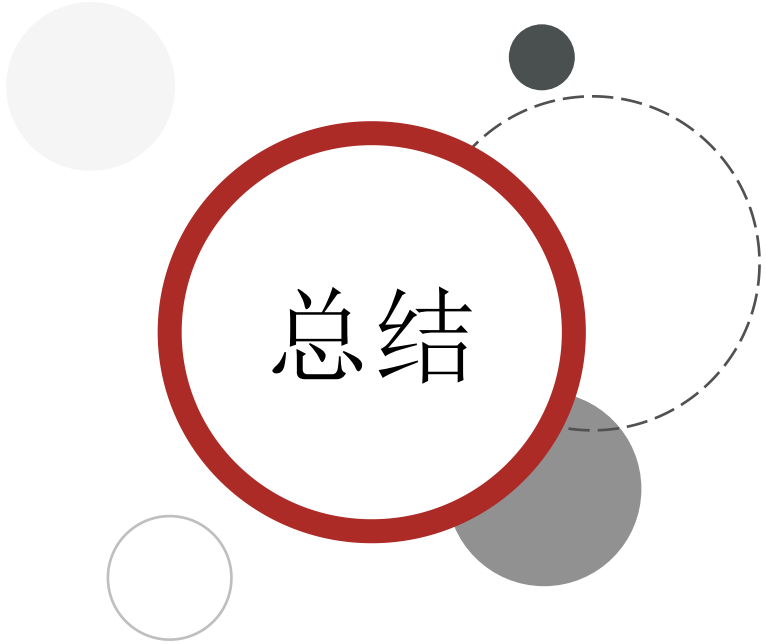
执行结果：



异常的finally

finally表示的是无论是否异常都要执行的代码，例如关闭文件。

```
try:
    f = open('test.txt', 'r')
except Exception as e:
    f = open('test.txt', 'w')
else:
    print('没有异常，真开心')
finally:
    f.close()
```



1. 为什么要捕获异常？

在可能发生异常的地方，进行捕获。当异常出现的时候，提供解决方式，而不是任由其导致程序无法运行。

2. 捕获异常的语法？

```
try:
    可能要发生异常的语句
except[异常 as 别名:]
    出现异常的准备手段
[else:]
    未出现异常时应做的事情
[finally:]
    不管出不出现异常都会做的事情
```

3. 如何捕获所有异常？

异常的种类多种多样，如果想要不管什么类型的异常都能捕获到，那么使用：

- except:
- except Exception:
- 两种方式捕获全部的异常

目录

Contents

学习目标

Learning Objectives

1. 知道异常具有传递性

异常的传递

异常是具有传递性的

当函数func01中发生异常，并且没有捕获处理这个异常的时候，异常会传递到函数func02，当func02也没有捕获处理这个异常的时候main函数会捕获这个异常， 这就是异常的传递性.

提示:

当所有函数都没有捕获异常的时候，程序就会报错

```
def func01():  异常在func01中没有被捕获
    print("这是func01开始")
    num = 1 / 0
    print("这是func01结束")

            异常在func02中没有被捕获
def func02():
    print("这是func02开始")
    func01()
    print("这是func02结束")


def main():异常在mian中被捕获
    try:
        func02()
    except Exception as e:
        print(e)


main()
```

异常的传递

```
def func01():  异常在func01中没有被捕获
    print("这是func01开始")
    num = 1 / 0
    print("这是func01结束")


           异常在func02中没有被捕获
def func02():
    print("这是func02开始")
    func01()
    print("这是func02结束")


def main():异常在mian中被捕获
    try:
        func02()
    except Exception as e:
        print(e)


main()
```

利用异常具有传递性的特点，当我们想要保证程序不会因为异常崩溃的时候，就可以在main函数中设置异常捕获，由于无论在整个程序哪里发生异常，最终都会传递到main函数中，这样就可以确保所有的异常都会被捕获

目录

Contents

学习目标

Learning Objectives

1. 了解什么是模块
2. 掌握导入Python内置的模块

什么是模块

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

模块的作用： python中有很多各种不同的模块，每一个模块都可以帮助我们快速的实现一些功能，比如实现和时间相关的功能就可以使用time模块

我们可以认为一个模块就是一个工具包，每一个工具包中都有各种不同的工具供我们使用进而实现各种不同的功能.

大白话：模块就是一个Python文件，里面有类、函数、变量等，我们可以拿过来用（导入模块去使用）

模块的导入方式

模块在使用前需要先导入 导入的语法如下:

```
[from 模块名] import [模块 | 类 | 变量 | 函数 | *] [as 别名]
```

常用的组合形式如:

- import 模块名
- from 模块名 import 类、变量、方法等
- from 模块名 import *
- import 模块名 as 别名
- from 模块名 import 功能名 as 别名

import模块名

基本语法：

```
import 模块名
import 模块名1，模块名2

模块名.功能名()
```

案例：导入time模块

```
# 导入时间模块
import time

print("开始")
# 让程序睡眠1秒(阻塞)
time.sleep(1)
print("结束")
```

from 模块名 import 功能名

基本语法：

```
from 模块名 import 功能名

功能名()
```

案例：导入time模块中的sleep方法

```
# 导入时间模块中的sleep方法
from time import sleep

print("开始")
# 让程序睡眠1秒(阻塞)
sleep(1)
print("结束")
```

from 模块名 import *

基本语法：

```
from 模块名 import *

功能名()
```

案例：导入time模块中所有的方法

```
# 导入时间模块中所有的方法
from time import *

print("开始")
# 让程序睡眠1秒(阻塞)
sleep(1)
print("结束")
```

as定义别名

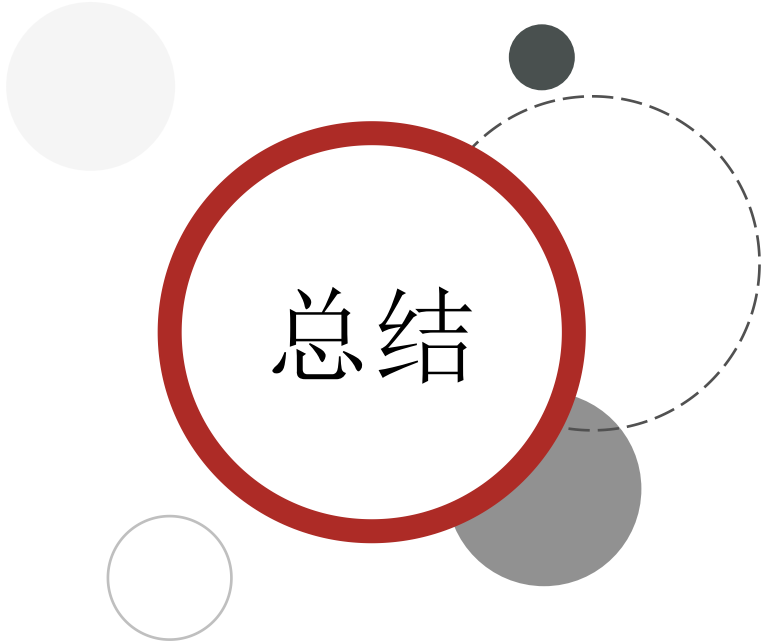
基本语法：

```
# 模块定义别名
import 模块名 as 别名

# 功能定义别名
from 模块名 import 功能 as 别名
```

案例：

```
# 模块别名
import time as tt

tt.sleep(2)
print('hello')
```

```
# 功能别名
from time import sleep as sl
sl(2)
print('hello')
```



1. 什么是模块？

模块就是一个Python代码文件，内含类、函数、变量等，我们可以导入进行使用。

2. 如何导入模块

```
[from 模块名] import [模块 | 类 | 变量 | 函数 | *] [as 别名]
```

3. 注意事项：

- from可以省略，直接import即可
- as别名可以省略
- 通过”.”来确定层级关系
- 模块的导入一般写在代码文件的开头位置

目录

Contents

学习目标

Learning Objectives

1. 了解如何自定义模块并使用
2. 了解__main__变量的作用

制作自定义模块

Python中已经帮我们实现了很多的模块. 不过有时候我们需要一些个性化的模块，这里就可以通过自定义模块实现，也就是自己制作一个模块

案例：新建一个Python文件，命名为my_module1.py，并定义test函数

my_module1.py

```
1  def test(a, b):
2      print(a + b)
3
```

my_module1.py | text_my_module.py

```
1  import my_module1
2
3  my_module1.test(10, 20)
4
5
6
```

注意：

每个Python文件都可以作为一个模块，模块的名字就是文件的名字. 也就是说自定义模块名必须要符合标识符命名规则

测试模块

在实际开发中，当一个开发人员编写完一个模块后，为了让模块能够在项目中达到想要的效果，

这个开发人员会自行在py文件中添加一些测试信息，例如，在my_module1.py文件中添加测试代码test(1,1)

```
def test(a, b):
    print(a + b)


test(1, 1)
```

问题：

此时，无论是当前文件，还是其他已经导入了该模块的文件，在运行的时候都会自动执行`test`函数的调用

解决方案：

```
def test(a, b):
    print(a + b)

# 只在当前文件中调用该函数，其他导入的文件内不符合该条件，则不执行test函数调用
if __name__ == '__main__':
    test (1, 1)
```

注意事项

```python
# 模块1代码
def my_test(a, b):
    print(a + b)

# 模块2代码
def my_test(a, b):
    print(a - b)

# 导入模块和调用功能代码
from my_module1 import my_test
from my_module2 import my_test

# my_test函数是模块2中的函数
my_test(1, 1)
```

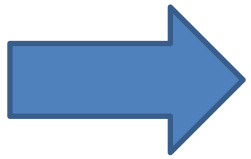
注意事项：当导入多个模块的时候，且模块内有同名功能，当调用这个同名功能的时候，调用到的是后面导入的模块的功能

__all__

如果一个模块文件中有`__all__`变量，当使用`from xxx import *`导入时，只能导入这个列表中的元素

my_module1.py

```
__all__ = ['test_A']


def test_A():
    print('testA')


def test_B():
    print('testB')

```



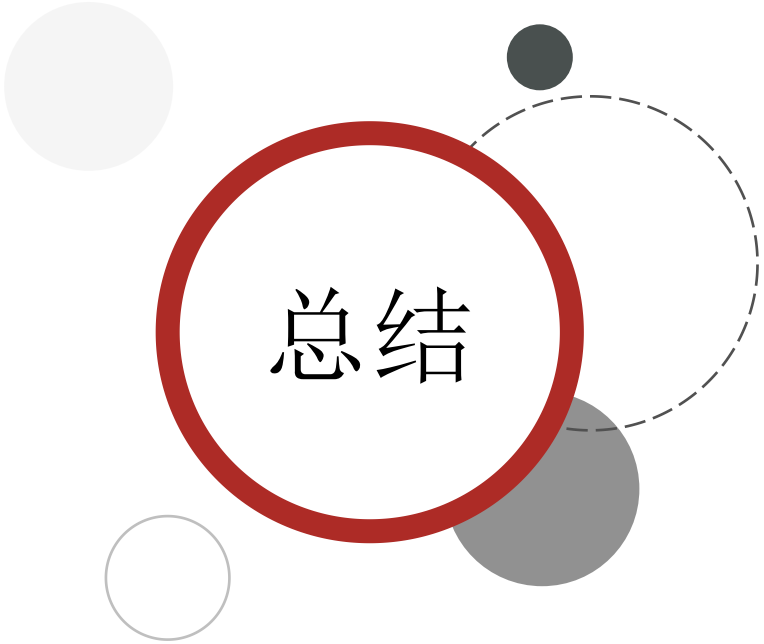
text_my_module.py

```
from my_module1 import *

test
```

这里只能使用test_A函数

test_A() my_module1

^↓ and ^↑ will move caret down and up in the editor Next Tip



1. 如何自定义模块并导入?

在Python代码文件中正常写代码即可，通过import、from关键字和导入Python内置模块一样导入即可使用。

2. __main__变量的功能是?

if __main__ == “__main__”表示，只有当程序是直接执行的才会进入if内部，如果是被导入的，则if无法进入

3. 注意事项

- 不同模块，同名的功能，如果都被导入，那么后导入的会覆盖先导入的
- __all__变量可以控制import *的时候哪些功能可以被导入

目录

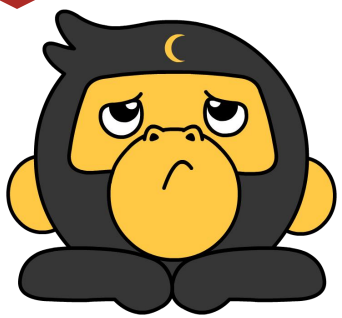
Contents

学习目标

Learning Objectives

1. 了解什么是Python包
2. 掌握如何自定义包

思考



基于Python模块，我们可以在编写代码的时候，导入许多外部代码来丰富功能。

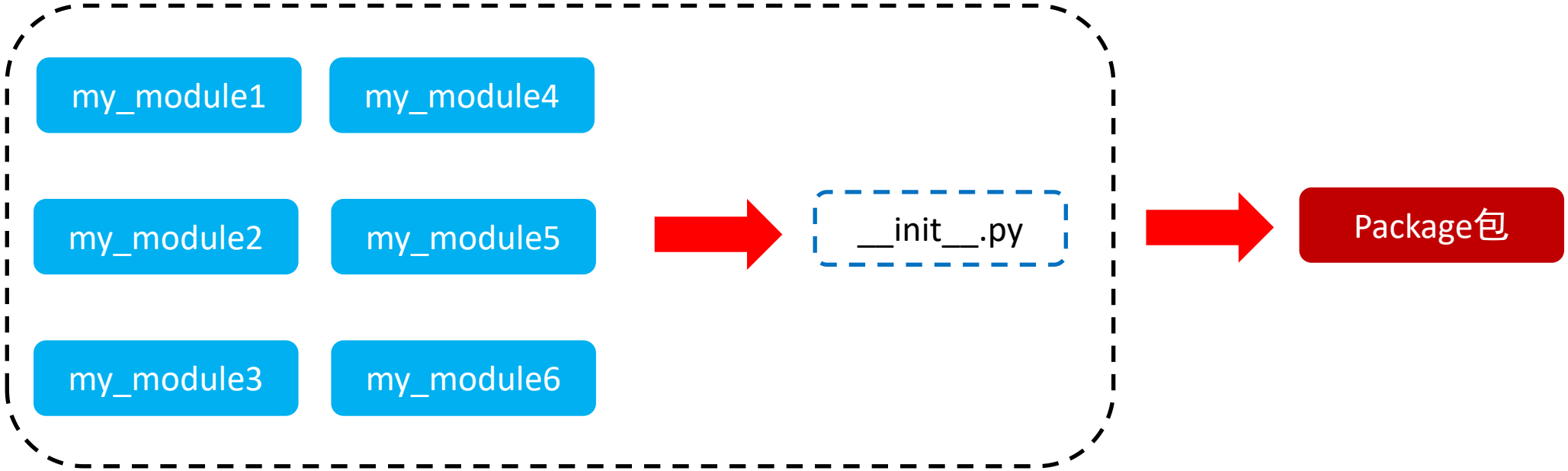
但是，如果Python的模块太多了，就可能造成一定的混乱，那么如何管理呢？

通过Python包的功能来管理。

什么是Python包

从物理上看，包就是一个文件夹，在该文件夹下包含了一个 __init__.py 文件，该文件夹可用于包含多个模块文件

从逻辑上看，包的本质依然是模块



包的作用：

当我们的模块文件越来越多时，包可以帮助我们管理这些模块，包的作用就是包含多个模块，但包的本质依然是模块

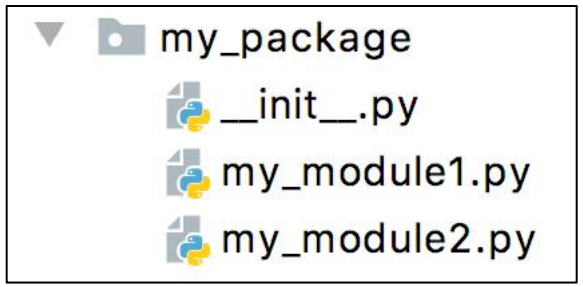
快速入门

步骤如下:

① 新建包`my_package`

② 新建包内模块: `my_module1` 和 `my_module2`

③ 模块内代码如下



my_module1.py

```
# my_module1模块中
print(1)


def info_print1():
    print('my_module1')

```

my_module2.py

```
# my_module2模块中
print(2)


def info_print2():
    print('my_module2')

```

Pycharm中的基本步骤:

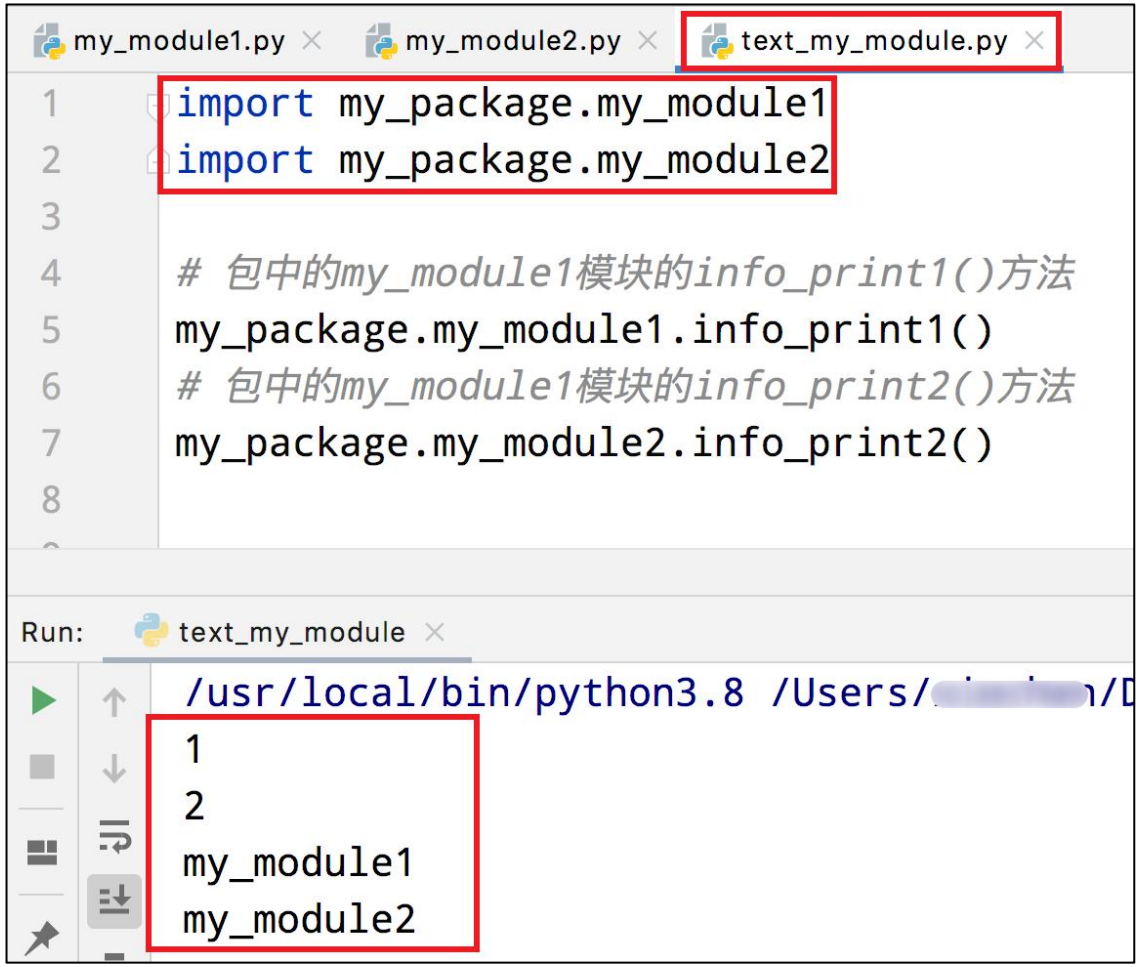
[New] ➔ [Python Package] ➔ 输入包名 ➔ [OK] ➔ 新建功能模块(有联系的模块)

注意: 新建包后，包内部会自动创建`__init__.py`文件，这个文件控制着包的导入行为

导入包

方式一：

```
import 包名.模块名

包名.模块名.目标
```



导入包

方式二：

注意：必须在`__init__.py`文件中添加`__all__ = []`，控制允许导入的模块列表

```
from 包名 import *
模块名.目标
```

my_module1.py | my_module2.py | text_my_module.py | __init__.py

```
1  # 包中的__all__和模块中的__all__一样有着控制的功能
2  __all__ = ["my_module2"]
3
```

包中可以用的模块的名字

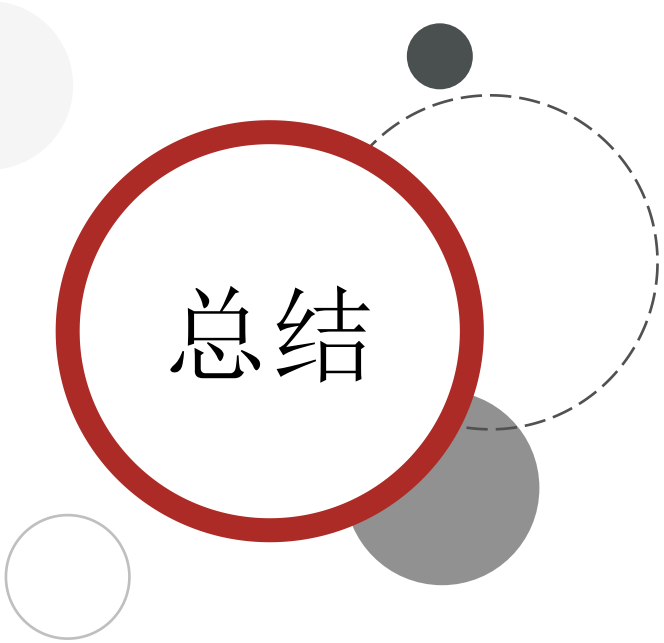
my_module1.py | my_module2.py | text_my_module.py | __init__.py

```
1  from my_package import *
2
3  # 包中的my_module1模块的info_print1()方法
4  my_module1.info_print1()
5  # 包中的my_module1模块的info_print2()方法
6  my_module2.info_print2()
```

my_module1报红证明不可用

注意：

__all__针对的是 ’ from ... import * ‘ 这
对 ‘ import xxx ’ 这种方式无效



1. 什么是Python的包？

包就是一个文件夹，里面可以存放许多Python的模块（代码文件），通过包，在逻辑上将一批模块归为一类，方便使用。

2. __init__.py文件的作用？

创建包会默认自动创建的文件，通过这个文件来表示一个文件夹是Python的包，而非普通的文件夹。

3. __all__变量的作用？

同模块中学习到的是一个作用，控制 import * 能够导入的内容

目录

Contents

学习目标

Learning Objectives

1. 了解什么是第三方包
2. 掌握使用pip安装第三方包

什么是第三方包

我们知道，包可以包含一堆的Python模块，而每个模块又内含许多的功能。

所以，我们可以认为：一个包，就是一堆同类型功能的集合体。

在Python程序的生态中，有许多非常多的第三方包（非Python官方），可以极大的帮助我们提高开发效率，如：

- 科学计算中常用的：numpy包
- 数据分析中常用的：pandas包
- 大数据计算中常用的：pyspark、apache-flink包
- 图形可视化常用的：matplotlib、pyecharts
- 人工智能常用的：tensorflow
- 等

这些第三方的包，极大的丰富了Python的生态，提高了开发效率。

但是由于是第三方，所以Python没有内置，所以我们需要安装它们才可以导入使用哦。

安装第三方包 - pip

第三方包的安装非常简单，我们只需要使用Python内置的pip程序即可。

打开我们许久未见的：命令提示符程序，在里面输入：

pip install 包名称

即可通过网络快速安装第三方包

pip的网络优化

由于pip是连接的国外的网站进行包的下载，所以有的时候会速度很慢。

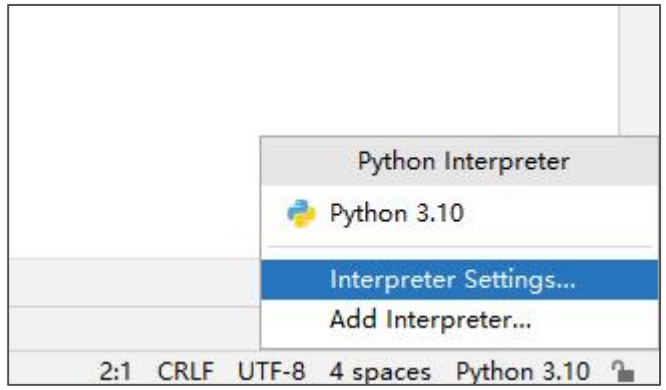
我们可以通过如下命令，让其连接国内的网站进行包的安装：

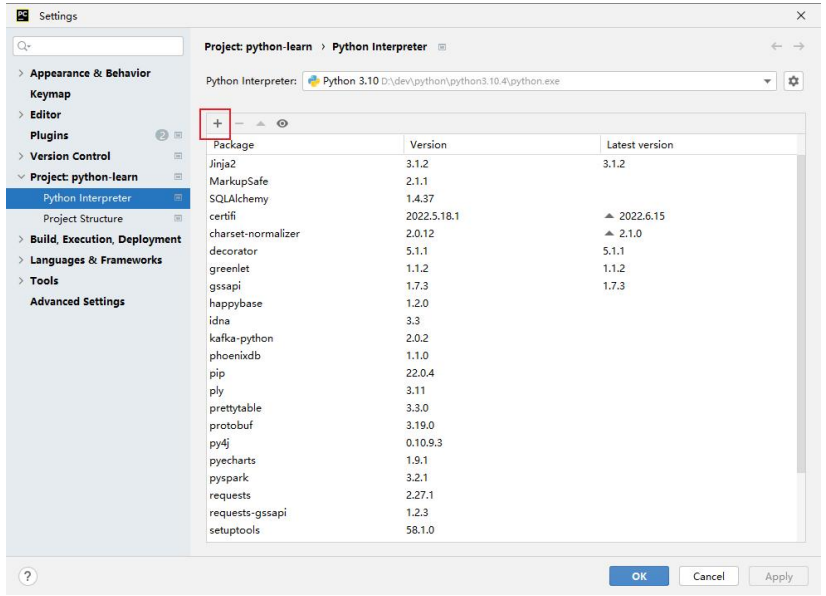
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple 包名称

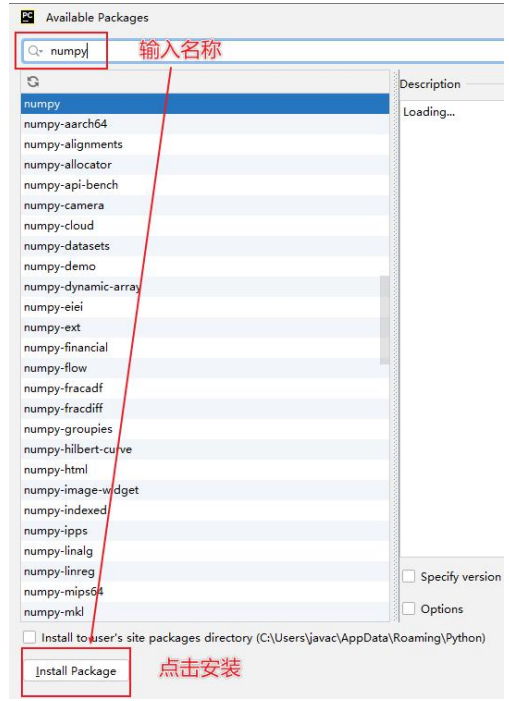
https://pypi.tuna.tsinghua.edu.cn/simple 是清华大学提供的一个网站，可供pip程序下载第三方包

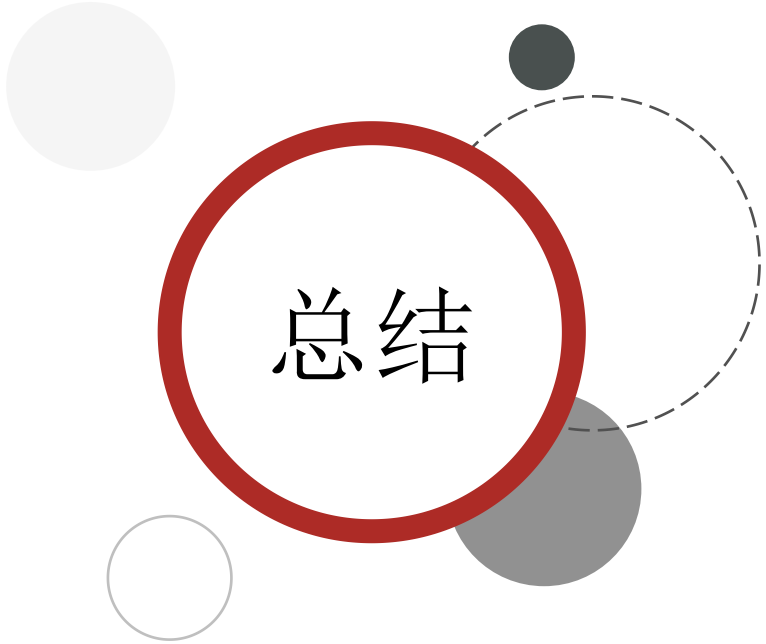
安装第三方包 - PyCharm

PyCharm也提供了安装第三方包的功能：









1. 什么是第三方包？有什么作用？

第三方包就是非Python官方内置的包，可以安装它们扩展功能，提高开发效率。

2. 如何安装？

- 在命令提示符内：
  - **pip install 包名称**
  - **pip install -i https://pypi.tuna.tsinghua.edu.cn/simple 包名称**
- 在PyCharm中安装

Python异常、模块、包：综合案例

练习

练习案例：自定义工具包

创建一个自定义包，名称为：my_utils （我的工具）

在包内提供2个模块

- str_util.py （字符串相关工具，内含：）
  - **函数：str_reverse(s)，接受传入字符串，将字符串反转返回**
  - **函数：substr(s, x, y)，按照下标x和y，对字符串进行切片**
- file_util.py（文件处理相关工具，内含：）
  - **函数：print_file_info(file_name)，接收传入文件的路径，打印文件的全部内容，如文件不存在则捕获异常，输出提示信息，通过finally关闭文件对象**
  - **函数：append_to_file(file_name, data)，接收文件路径以及传入数据，将数据追加写入到文件中**

构建出包后，尝试着用一用自己编写的工具包。

传智教育旗下高端IT教育品牌