

# Python基础教程

陈加忠

计算科学与技术学院

QQ群: 904987289

Email: [jzchen@hust.edu.cn](mailto:jzchen@hust.edu.cn)

# 知识点与要求

- 命令行配置开发环境
- 命令行安装opencv
- 在IDE中设置Python解释器
- Python基本规范与语句
- 理解算法表述
- Python对算法的编程实现
- 可视化展示算法效果

# Python基础知识

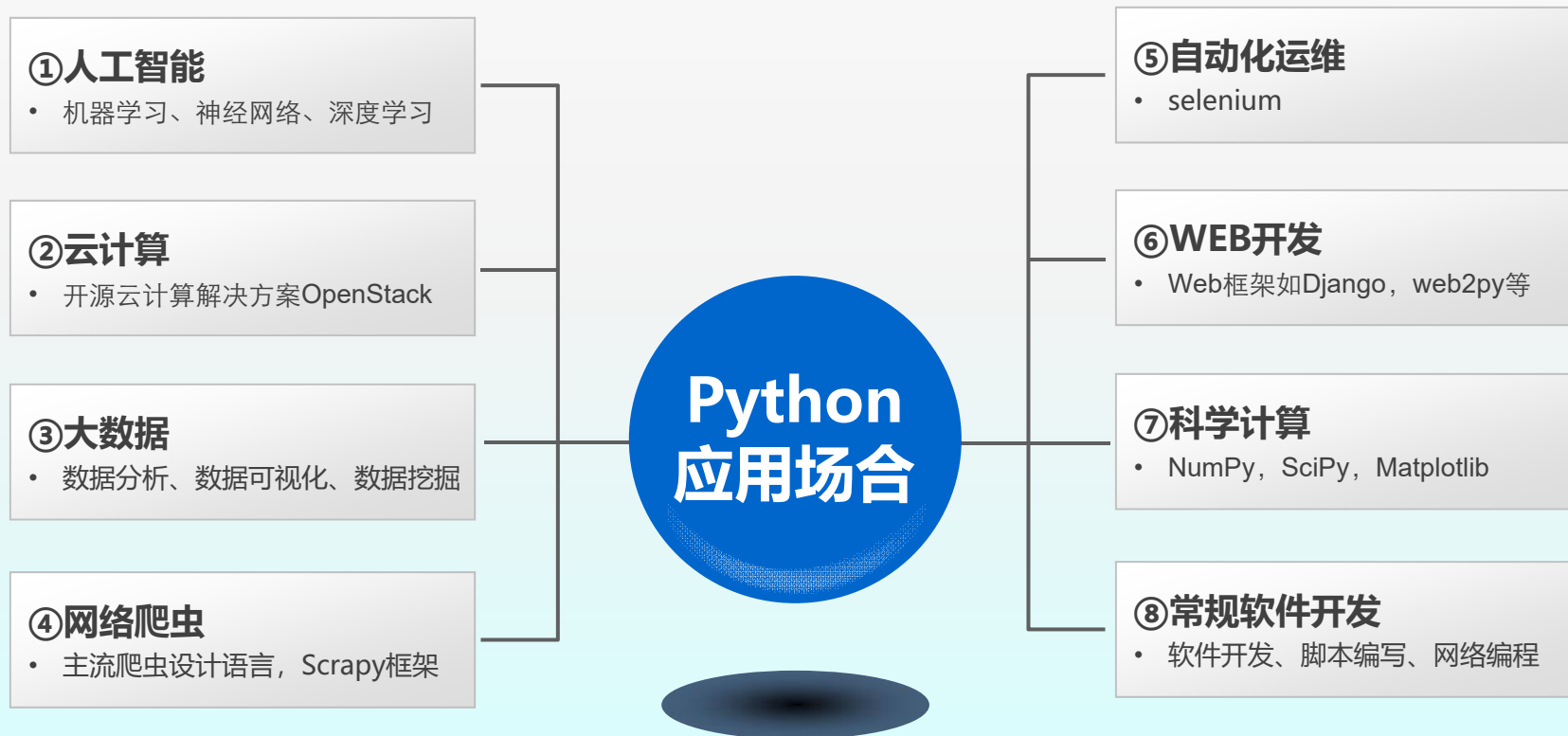
- **Python**开发环境
- Python基础语法
- Python数据类型
- 表达式
- 函数结构
- 流程控制

# 什么是Python

- Python由荷兰人吉多·范·罗苏姆于1989年编写的一个脚本解释程序, 使用一种优雅的语法, 可读性强
- Python于1991年公开发布, 早于HTTP 1.0协议5年, 早于Java语言4年. 它是一种开源的、**解释性**的, 面向对象的编程语言
- Python支持**类**和多层继承等的面向对象编程技术
- Python可运行在多种计算机平台和操作系统中, 如Unix、Windows、Mac OS、Ubuntu、OS/2等等
- Python语言具有简洁性、易读性以及可扩展性, 在国外用Python做科学计算的研究机构日益增多, 一些知名大学已经采用Python来教授程序设计课程
- 本科阶段《计算思维》、《机器学习》等课程将用到Python进行编程

# Python简介

## • Python的应用场合



# Python开发环境安装

- **Python开发环境安装**用于指导在个人笔记本上练习
- 安装pycharm与anaconda
- 学校软件网站上[software.hust.edu.cn](http://software.hust.edu.cn)是试用版, 需要激活
- 建议版本:
  - ✓ Pycharm: Anaconda3-2019.03-Windows-x86\_64
  - ✓ Anaconda: pycharm-community-2019.1.3

# Python开发环境安装

- win+s 搜索 prompt (或在安装路径) 打开 Anaconda Prompt, 配 anaconda 环境
- 或者在开始菜单查找 Anaconda Prompt
- 在 anaconda 命令行中, 敲入: `conda create -n 自选路径名字如: dl python=3.7`, 选择 anaconda 环境安装路径, 即:
- `conda create -n dl python=3.7`

# Python开发环境安装

- 配置Python安装环境
- `conda create -n dl python=3.7`

```
Anaconda Prompt - conda create -n dl python=3.7

(base) C:\Users\jzchen>conda create -n dl python=3.7
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.6.11
  latest version: 4.8.5

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\jzchen\Anaconda3\envs\dl

added / updated specs:
- python=3.7

The following packages will be downloaded:
```

package	build	
ca-certificates-2020.7.22	0	164 KB



# Python开发环境安装

- 装python, 敲y回车, 一共32M

```
Anaconda Prompt - conda create -n dl python=3.7

-----
Total: 30.2 MB

The following NEW packages will be INSTALLED:

ca-certificates  pkgs/main/win-64::ca-certificates-2020.7.22-0
certifi          pkgs/main/win-64::certifi-2020.6.20-py37_0
openssl         pkgs/main/win-64::openssl-1.1.1h-he774522_0
pip             pkgs/main/win-64::pip-20.2.2-py37_0
python          pkgs/main/win-64::python-3.7.9-h60c2a47_0
setuptools      pkgs/main/win-64::setuptools-49.6.0-py37_1
sqlite          pkgs/main/win-64::sqlite-3.33.0-h2a8f88b_0
vc              pkgs/main/win-64::vc-14.1-h0510ff6_4
vs2015_runtime  pkgs/main/win-64::vs2015_runtime-14.16.27012-hf0eaf9b_3
wheel           pkgs/main/noarch::wheel-0.35.1-py_0
wincertstore    pkgs/main/win-64::wincertstore-0.2-py37_0
zlib            pkgs/main/win-64::zlib-1.2.11-h62dcd97_4

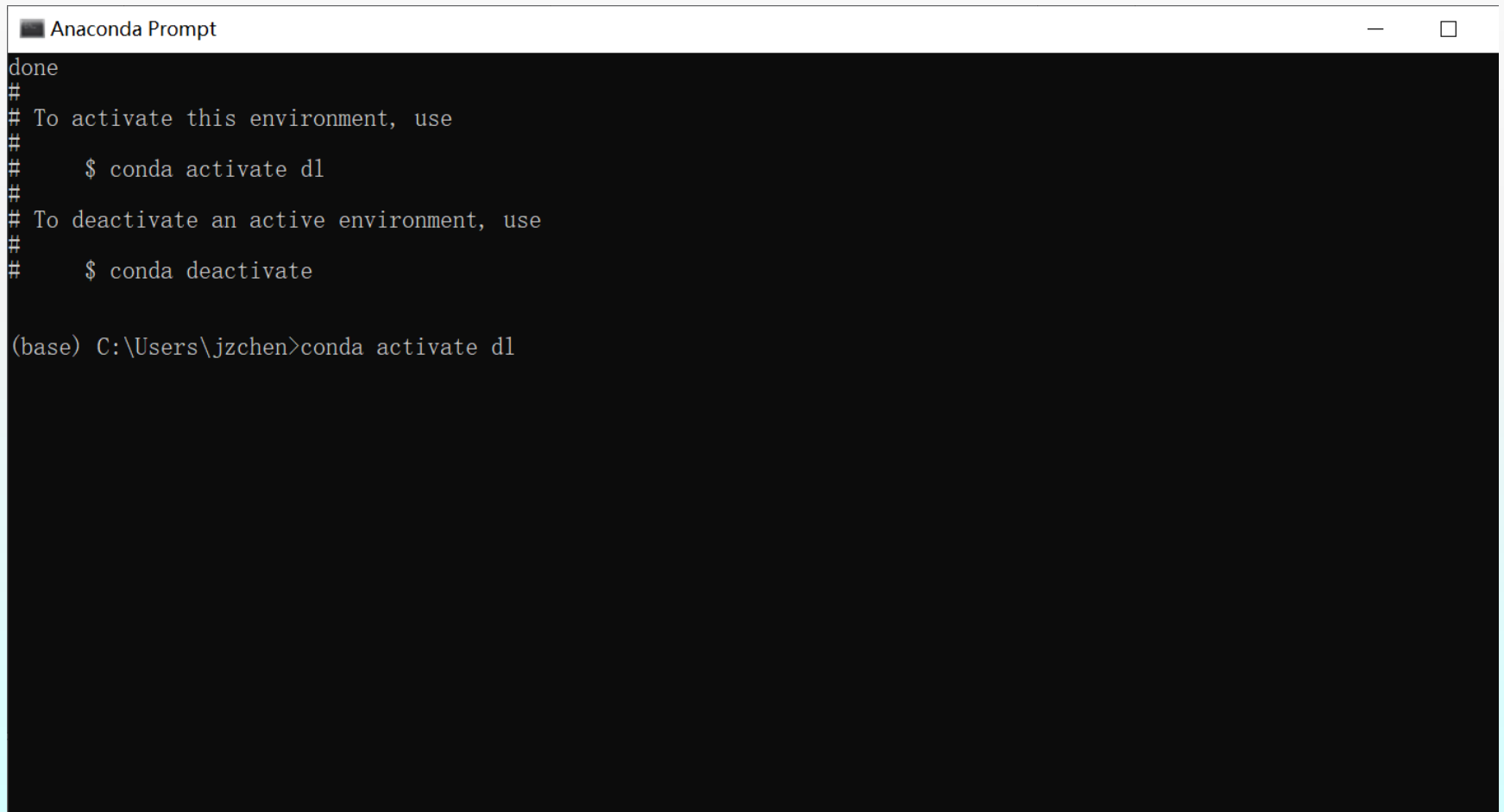
Proceed ([y]/n)? y_
```

# Python开发环境安装

- 在anaconda命令行(cmd)中, 敲入: `conda activate dl`, 激活开发环境
- 在`dl`环境中安装`opencv`:
  - ✓ 在线安装: `pip install opencv-python` 或者: `conda install -c menpo opencv`
  - ✓ 如果已经下载了whl文件, `pip install` 路径 `\opencv_python-4.4.0.44-cp37-cp37m-win_amd64.whl`, 一共32M

# Python开发环境安装

- 敲conda activate dl, 回车, 激活环境

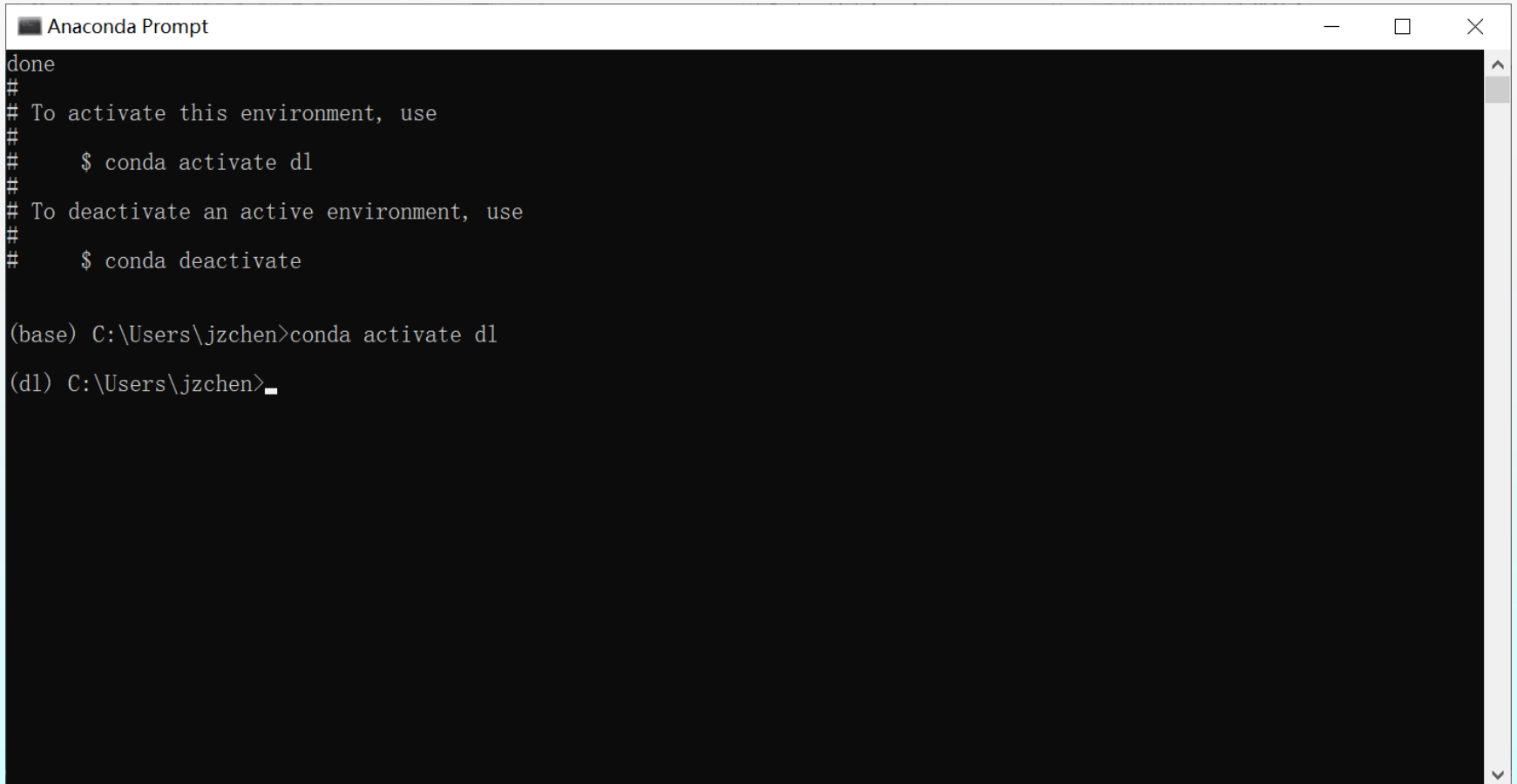


```
Anaconda Prompt
done
#
# To activate this environment, use
#
#     $ conda activate dl
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) C:\Users\jzchen>conda activate dl
```

# Python开发环境安装

- 进入创建的conda环境dl

A screenshot of the Anaconda Prompt window. The window title is "Anaconda Prompt". The terminal output shows instructions for activating and deactivating a conda environment. The user has entered the command to activate the environment named 'dl', and the prompt has changed from '(base)' to '(dl)'.

```
done
##
## To activate this environment, use
##
##     $ conda activate dl
##
## To deactivate an active environment, use
##
##     $ conda deactivate

(base) C:\Users\jzchen>conda activate dl
(dl) C:\Users\jzchen>_
```

# Python开发环境安装

- 在线装opencv, 慢
- dl环境中敲conda install -c menpo opencv

```
Anaconda Prompt - conda install -c menpo opencv

(base) C:\Users\jzchen>activate dl

(dl) C:\Users\jzchen>conda install -c menpo opencv
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.6.11
  latest version: 4.8.5

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\jzchen\Anaconda3\envs\dl

  added / updated specs:
    - opencv

The following packages will be downloaded:
```

package	build
---------	-------

# Python开发环境安装

- 敲入y, 确认安装opencv, 247M, 可能比较慢

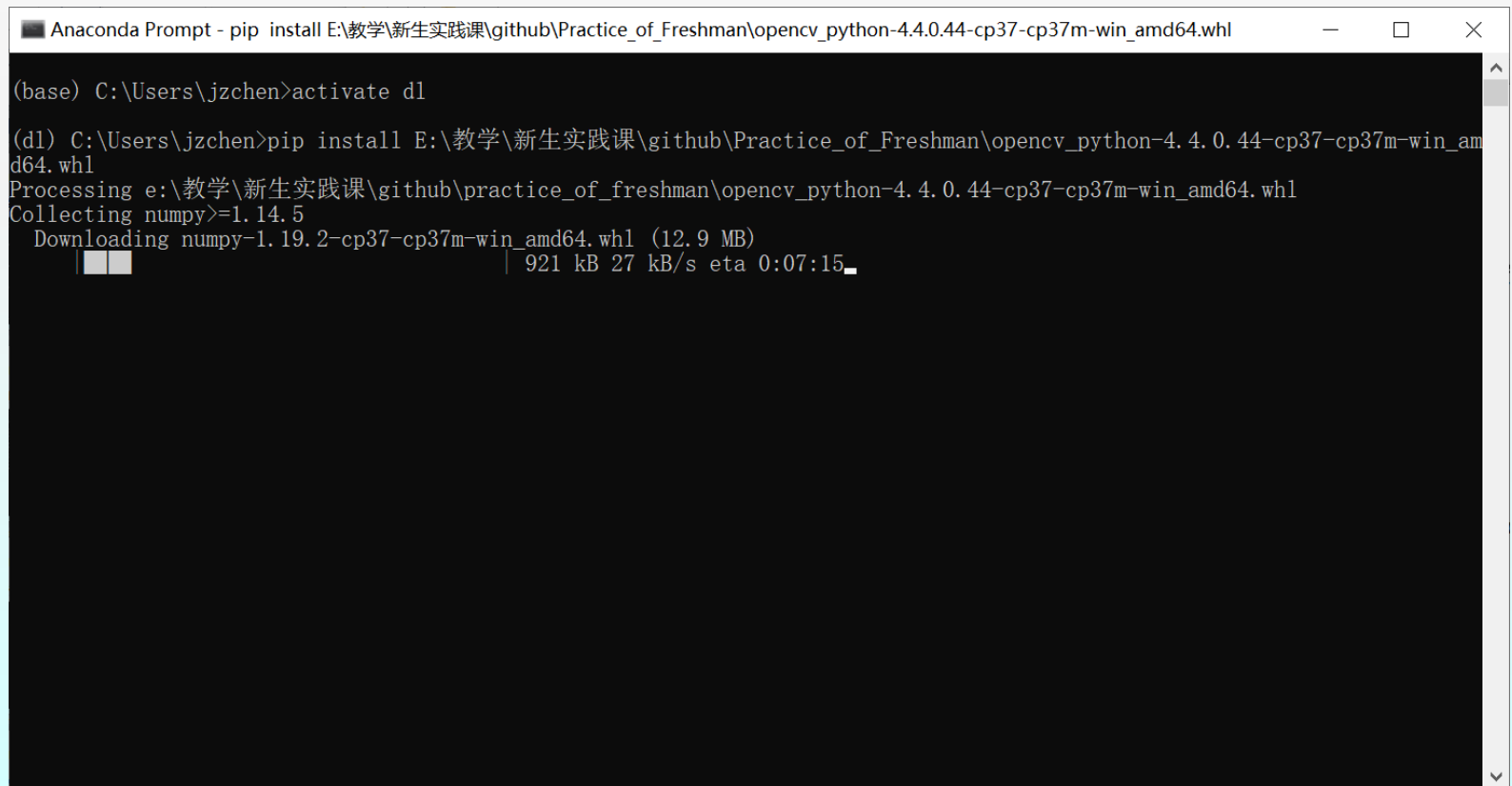
```
Anaconda Prompt - conda install -c menpo opencv
The following NEW packages will be INSTALLED:

blas                pkgs/main/win-64::blas-1.0-mkl
hdf5                pkgs/main/win-64::hdf5-1.8.20-hac2f561_1
icc_rt              pkgs/main/win-64::icc_rt-2019.0.0-h0cc432a_1
intel-openmp        pkgs/main/win-64::intel-openmp-2020.2-254
jpeg                pkgs/main/win-64::jpeg-9b-hb83a4c4_2
libopencv           pkgs/main/win-64::libopencv-3.4.2-h20b85fd_0
libpng              pkgs/main/win-64::libpng-1.6.37-h2a8f88b_0
libtiff             pkgs/main/win-64::libtiff-4.1.0-h56a325e_1
lz4-c               pkgs/main/win-64::lz4-c-1.9.2-h62dcd97_1
mkl                 pkgs/main/win-64::mkl-2020.2-256
mkl-service         pkgs/main/win-64::mkl-service-2.3.0-py37hb782905_0
mkl_fft             pkgs/main/win-64::mkl_fft-1.2.0-py37h45dec08_0
mkl_random          pkgs/main/win-64::mkl_random-1.1.1-py37h47e9c7a_0
numpy               pkgs/main/win-64::numpy-1.19.1-py37h5510c5b_0
numpy-base         pkgs/main/win-64::numpy-base-1.19.1-py37ha3acd2a_0
opencv              pkgs/main/win-64::opencv-3.4.2-py37h40b0b35_0
py-opencv           pkgs/main/win-64::py-opencv-3.4.2-py37hc319ecb_0
six                 pkgs/main/noarch::six-1.15.0-py_0
xz                  pkgs/main/win-64::xz-5.2.5-h62dcd97_0
zstd                pkgs/main/win-64::zstd-1.4.5-h04227a9_0

Proceed ([y]/n)? y
```

# Python开发环境安装

- 离线装opencv, 需事先准备好WHL文件, 快
- `pip install 路径\whl文件名`



```
Anaconda Prompt - pip install E:\教学\新生实践课\github\Practice_of_Freshman\opencv_python-4.4.0.44-cp37-cp37m-win_amd64.whl

(base) C:\Users\jzchen>activate dl

(dl) C:\Users\jzchen>pip install E:\教学\新生实践课\github\Practice_of_Freshman\opencv_python-4.4.0.44-cp37-cp37m-win_amd64.whl
Processing e:\教学\新生实践课\github\practice_of_freshman\opencv_python-4.4.0.44-cp37-cp37m-win_amd64.whl
Collecting numpy>=1.14.5
  Downloading numpy-1.19.2-cp37-cp37m-win_amd64.whl (12.9 MB)
    | 921 kB 27 kB/s eta 0:07:15_
```

# 查看Python开发环境安装

- win+s 搜索prompt (或在安装路径) 打开 Anaconda Prompt
- 先在cmd (command)窗口敲入activate dl再回车进入dl环境
- 再在cmd窗口用conda list查看安装的包及环境安装的路径与版本
- 也可以用python查看python的版本
  - ✓ 查看安装了哪些包
  - ✓ 查看包的版本
  - ✓ 查看安装路径



# 查看Python开发环境安装

- 敲入conda list查看安装的包及版本

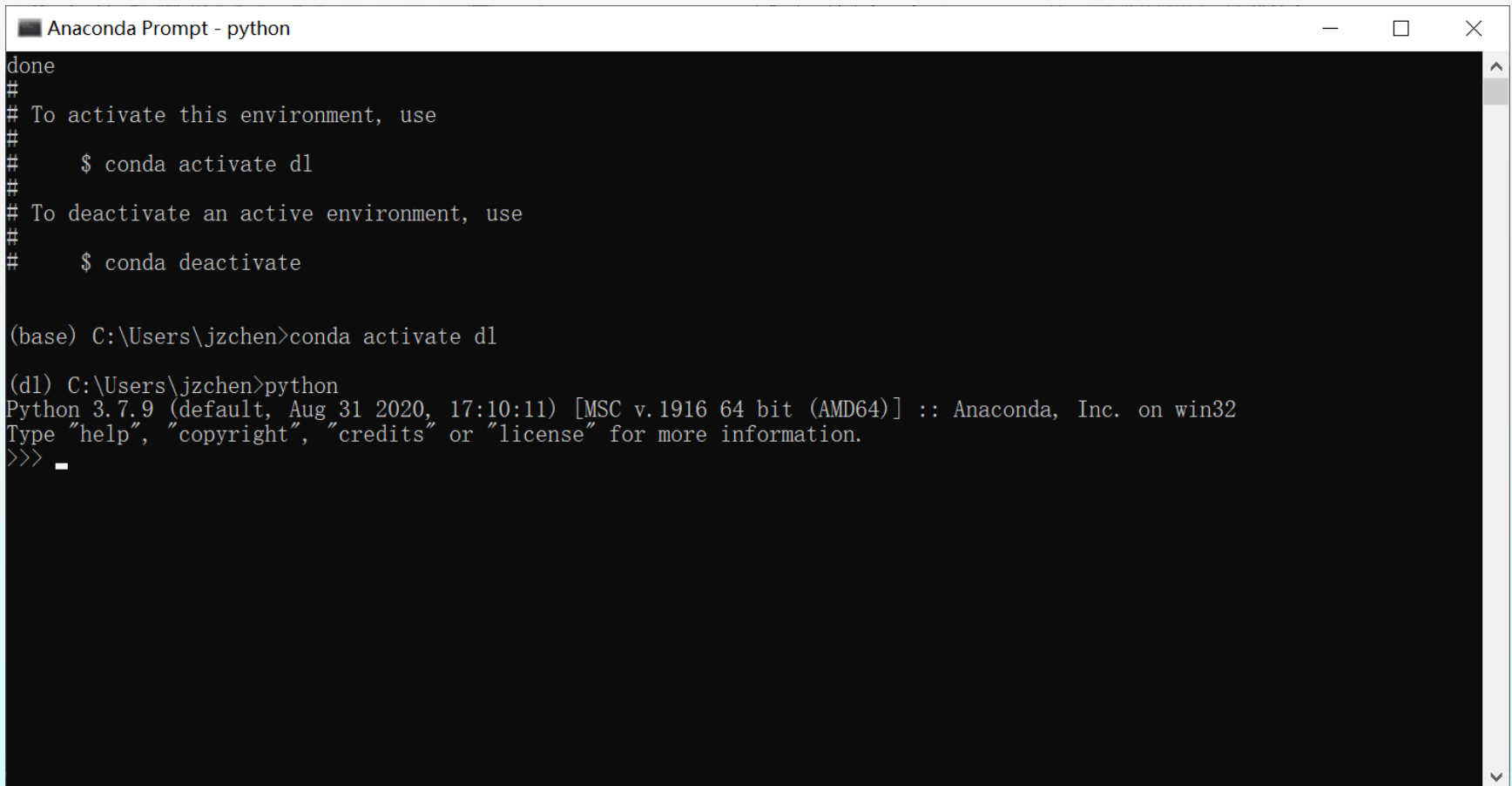
```
Anaconda Prompt

(base) C:\Users\jzchen>activate dl

(dl) C:\Users\jzchen>conda list
WARNING: The conda.compat module is deprecated and will be removed in a future release.
# packages in environment at C:\Users\jzchen\.conda\envs\dl:
#
# Name                                Version                                Build      Channel
_tflow_select                         2.1.0                                gpu
absl-py                              0.8.1                                py37_0
astor                                 0.8.0                                py37_0
blas                                  1.0                                  mkl
ca-certificates                      2020.6.24                            0
certifi                              2020.6.20                            py37_0
cloudpickle                          1.5.0                                py_0
colorama                             0.4.4                                pypi_0    pypi
cudatoolkit                          10.0.130                             0
cudnn                                 7.6.4                                cuda10.0_0
cyclor                               0.10.0                               py37_0
cytoolz                              0.10.1                               py37he774522_0
dask-core                            2.20.0                               py_0
decorator                             4.4.2                                py_0
freetype                             2.9.1                                ha9979f8_1
gast                                  0.2.2                                py37_0
google-pasta                         0.1.8                                py_0
grpcio                               1.16.1                               py37h351948d_1
h5py                                  2.8.0                                py37hf7173ca_2
hdf5                                  1.8.20                               hac2f561_1
icc_rt                               2019.0.0                             h0cc432a_1
icu                                   58.2                                 ha66f8fd_1
```

# 查看Python开发环境安装

- 敲入python, 可以看到安装的python版本



```
Anaconda Prompt - python
done
##
## To activate this environment, use
##
##     $ conda activate dl
##
## To deactivate an active environment, use
##
##     $ conda deactivate

(base) C:\Users\jzchen>conda activate dl

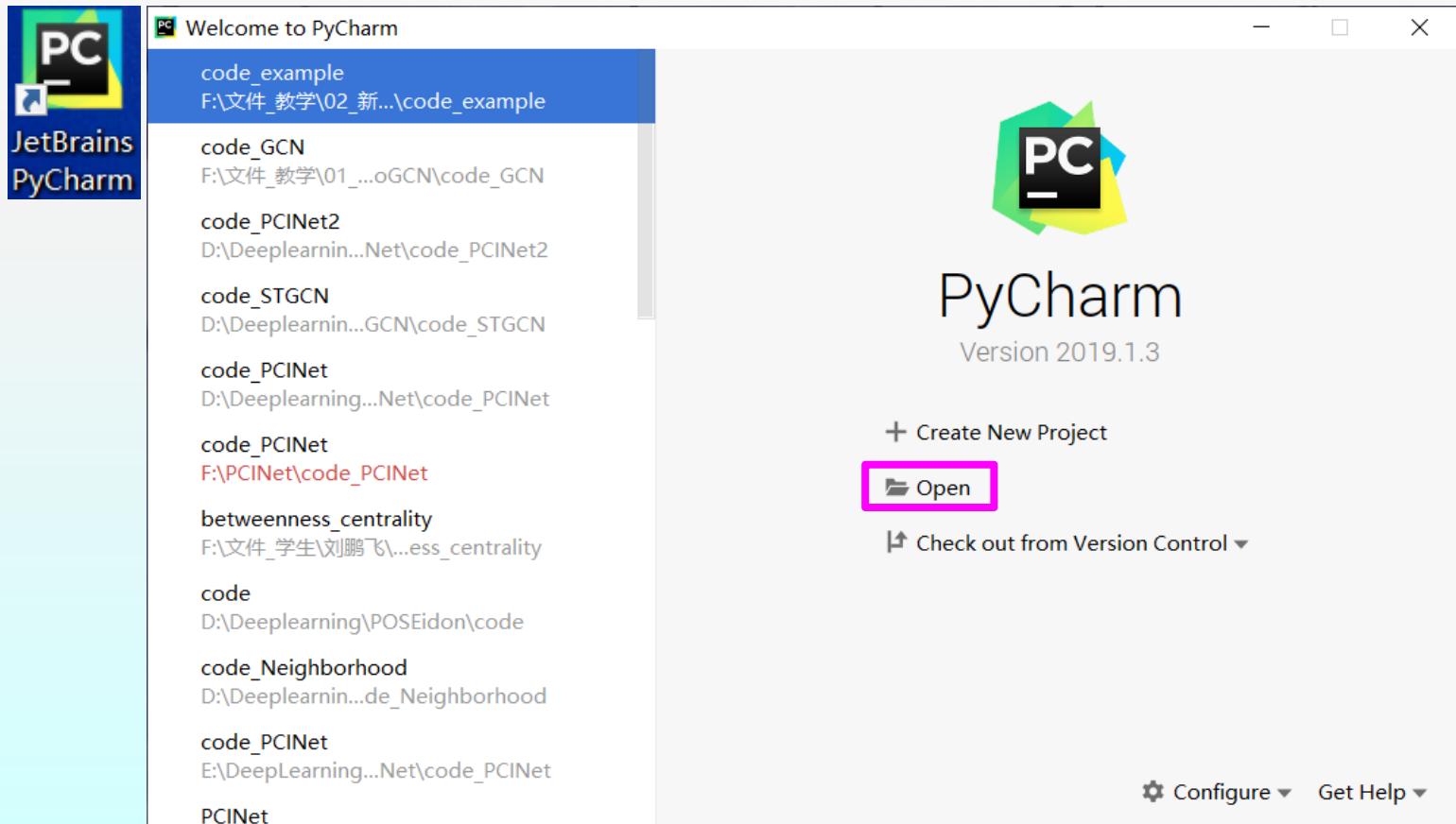
(dl) C:\Users\jzchen>python
Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> _
```

# PyCharm打开一个Project

- QQ群下载code\_example.rar压缩包到本地电脑D盘
- 解压到D盘
- 切记：必须先下载到D盘再解压，保证解压后的各个文件在D:\code\_example\中
- 如果不能，就在D盘建一个code\_example文件夹，再把解压后的各文件copy进去

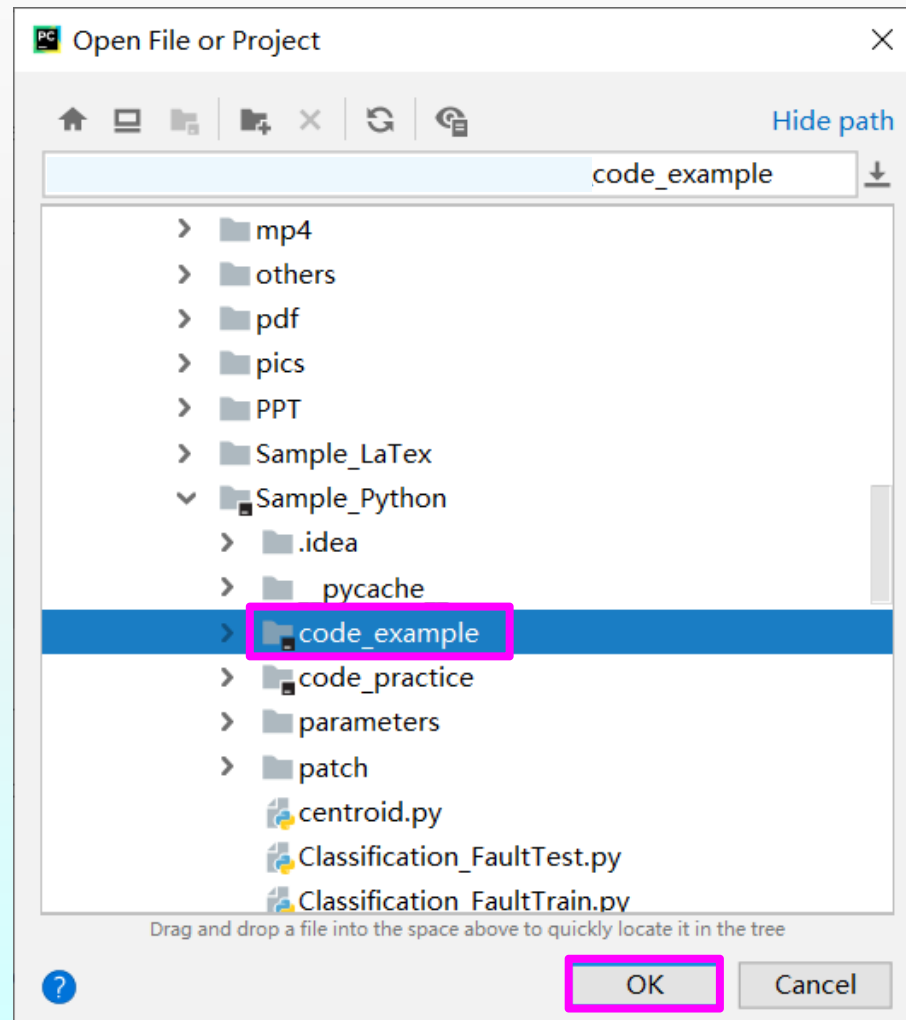
# PyCharm打开一个Project

- 桌面双击PyCharm图标, Charm: 魅力
- 在Welcome页面点**Open**



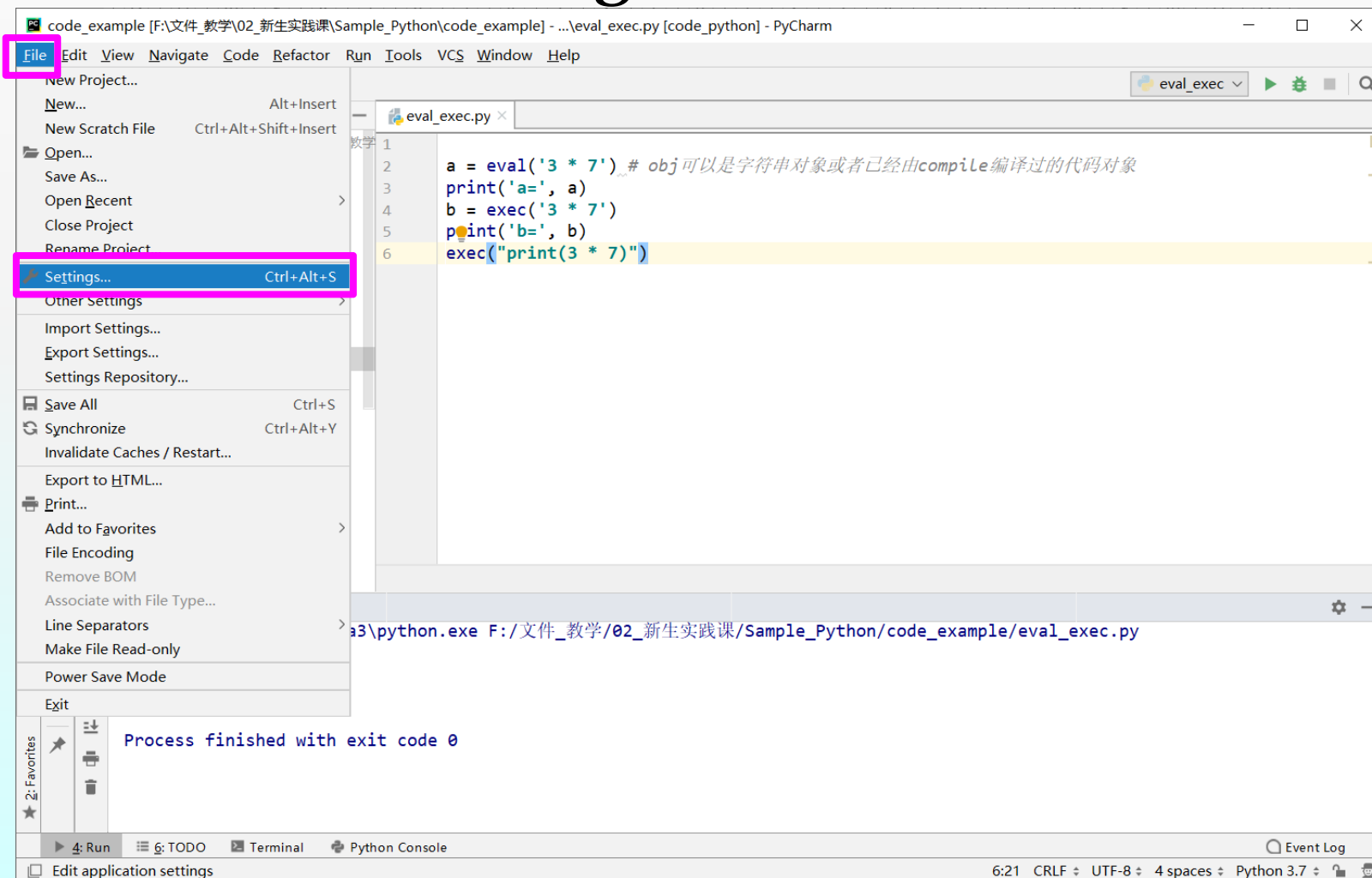
# PyCharm打开一个Project

- 找到刚才的压缩包解压路径code\_example



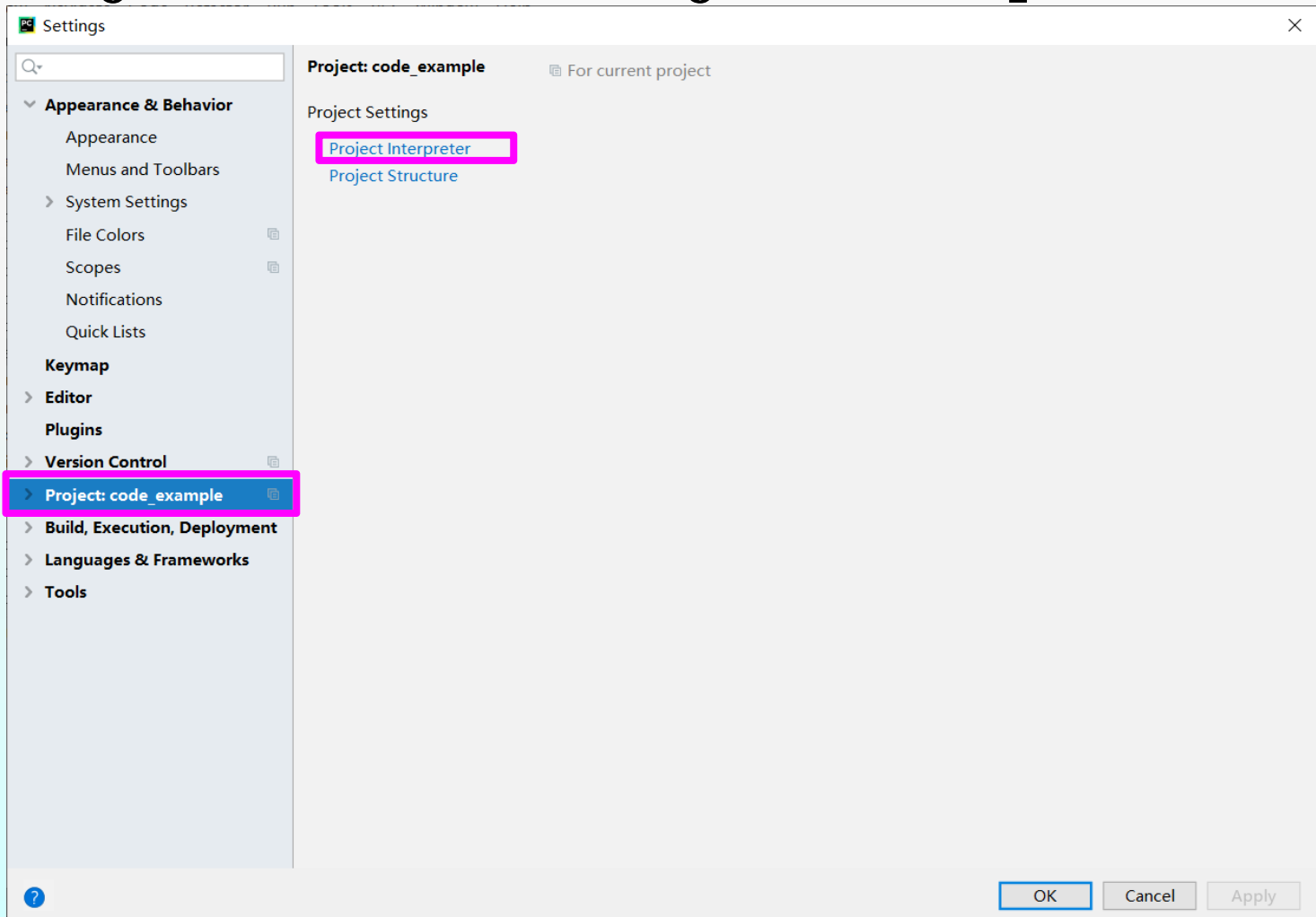
# Python解释器配置

- 点File → 点Setting...



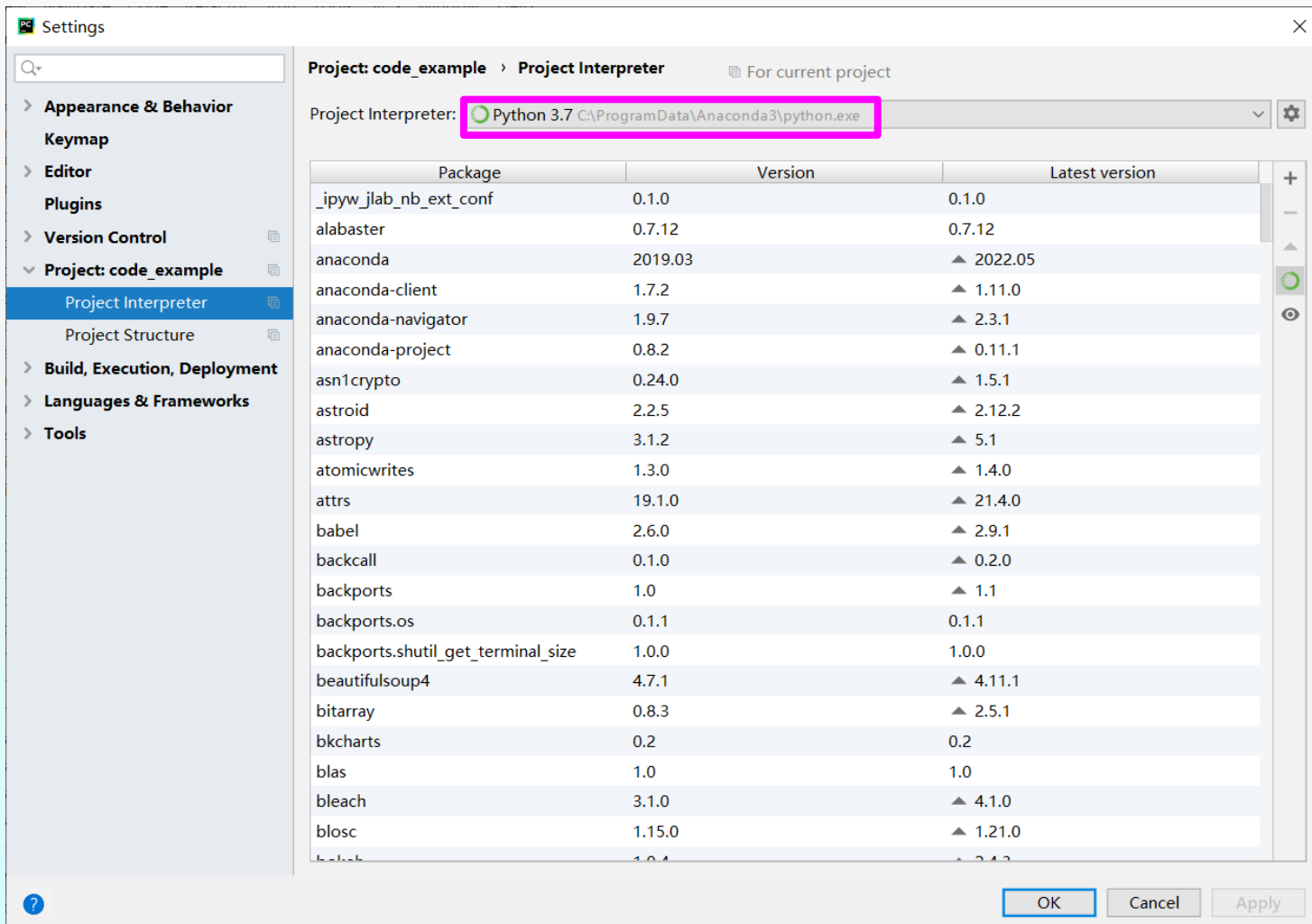
# Python解释器配置

- 点Project: xxx → 点Project Interpreter



# Python解释器配置

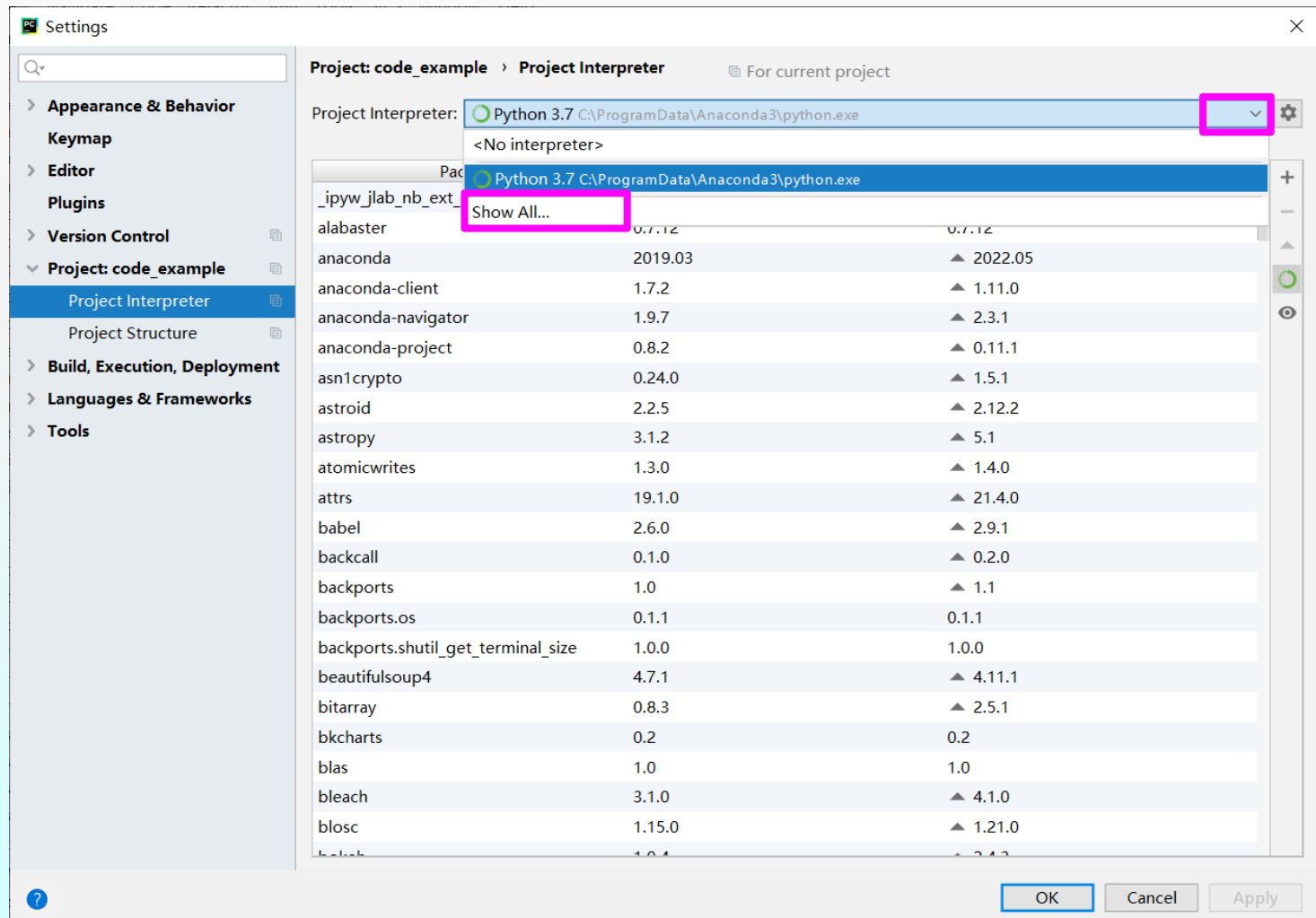
- 可能会出现默认的解释器Python 3.7, 不要点ok





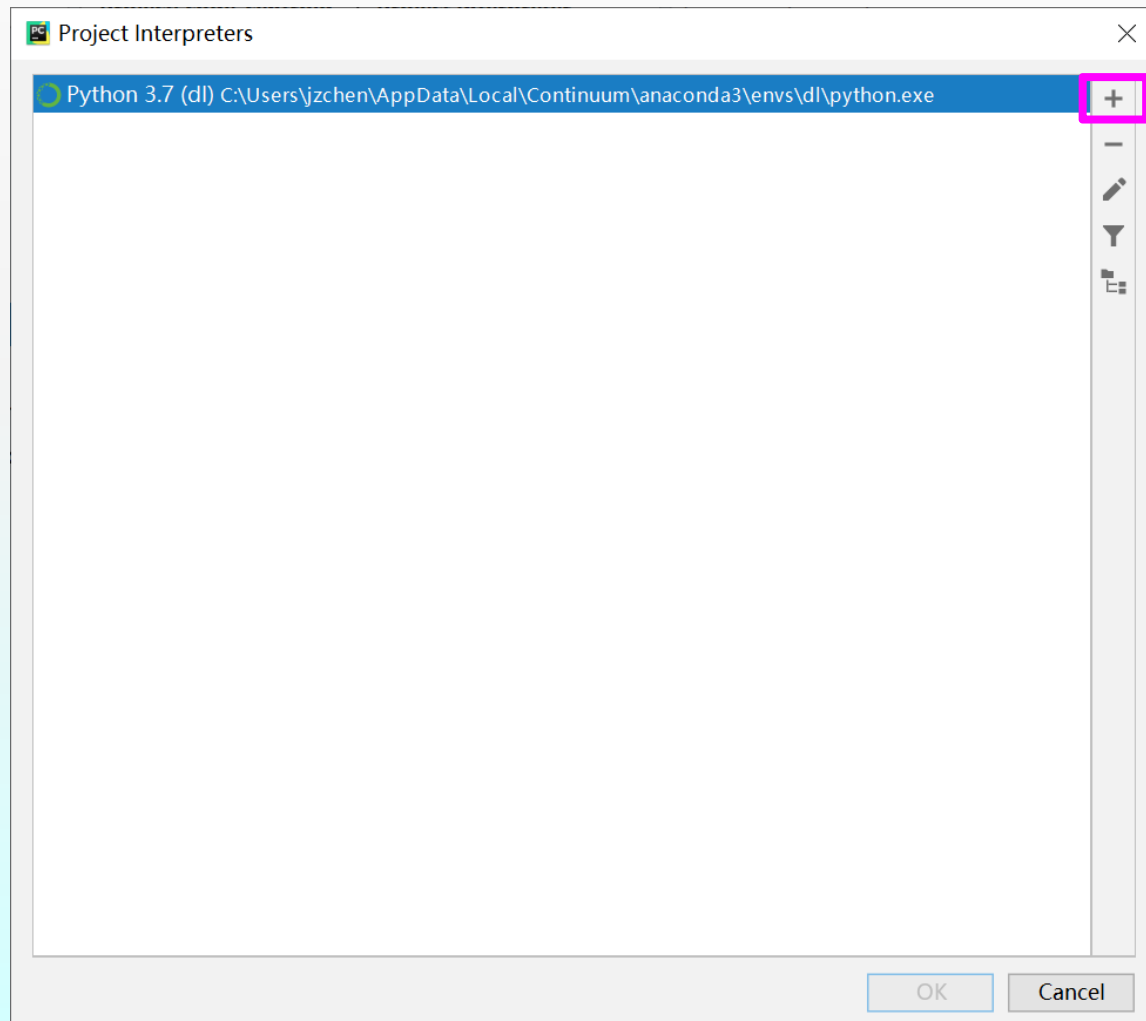
# Python解释器配置

- 点下拉箭头再点Show All...



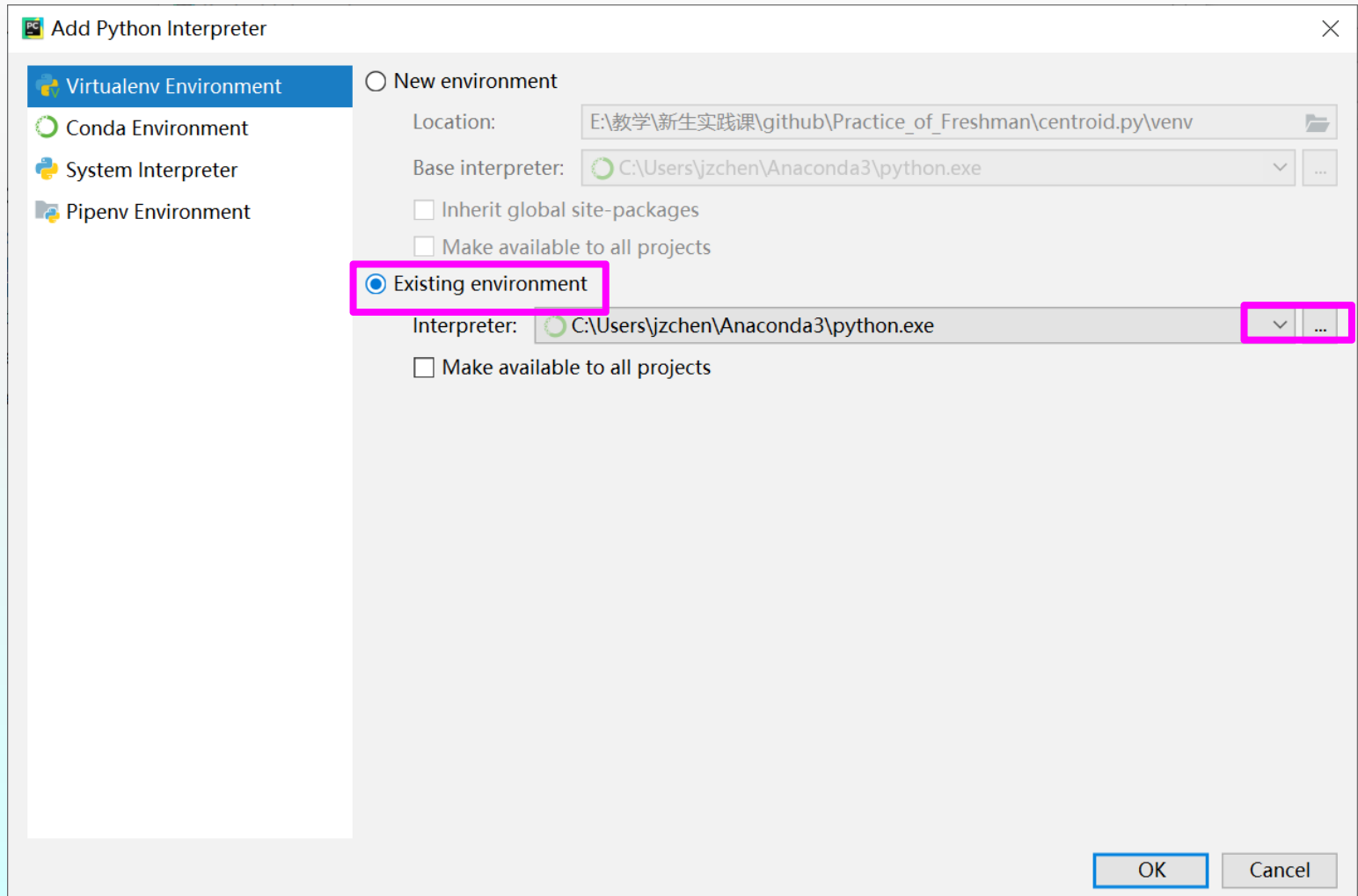
# Python解释器配置

- 点+



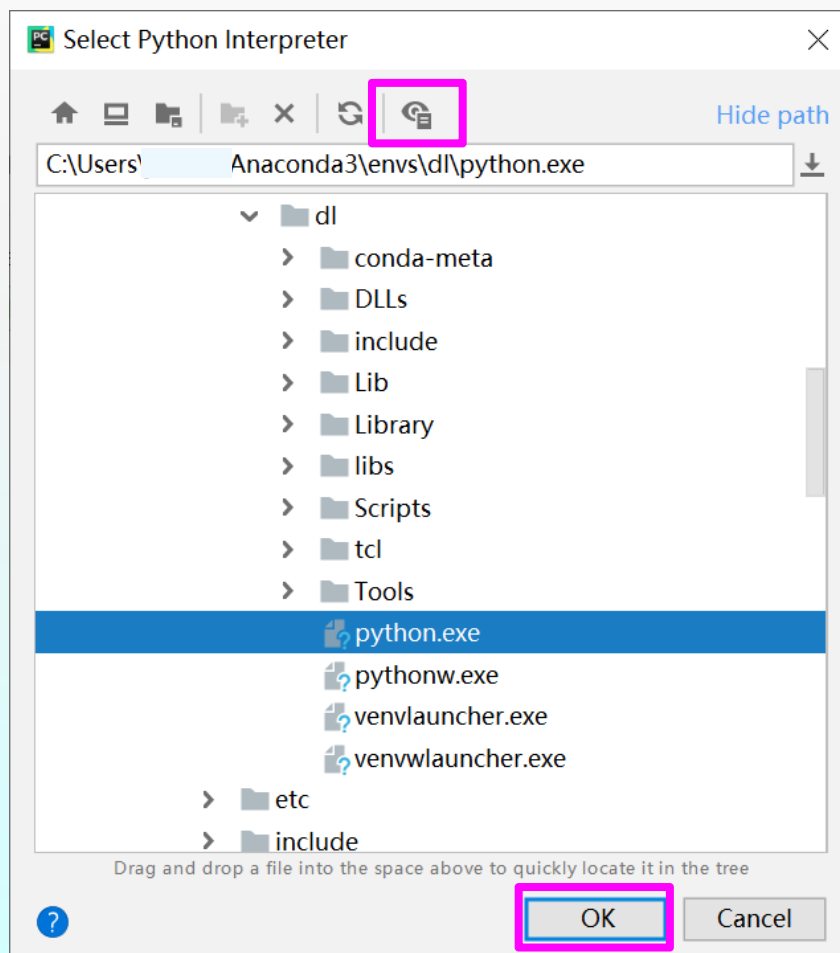
# Python解释器配置

- 选中 Existing environment, 再点...



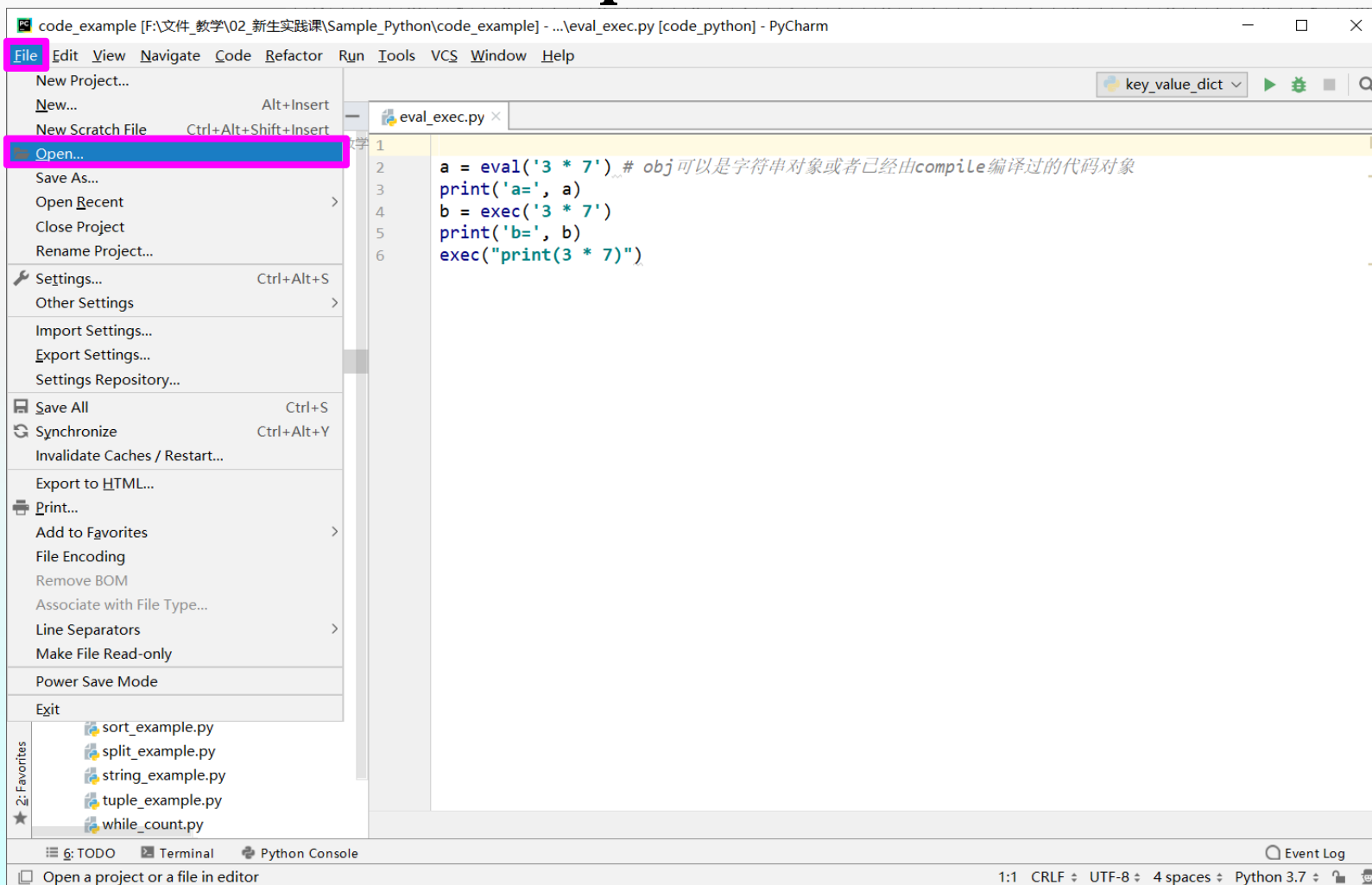
# Python解释器配置

- 击眼睛显示隐藏文件夹, 在某个路径找到dl环境的解释器, 一直点OK

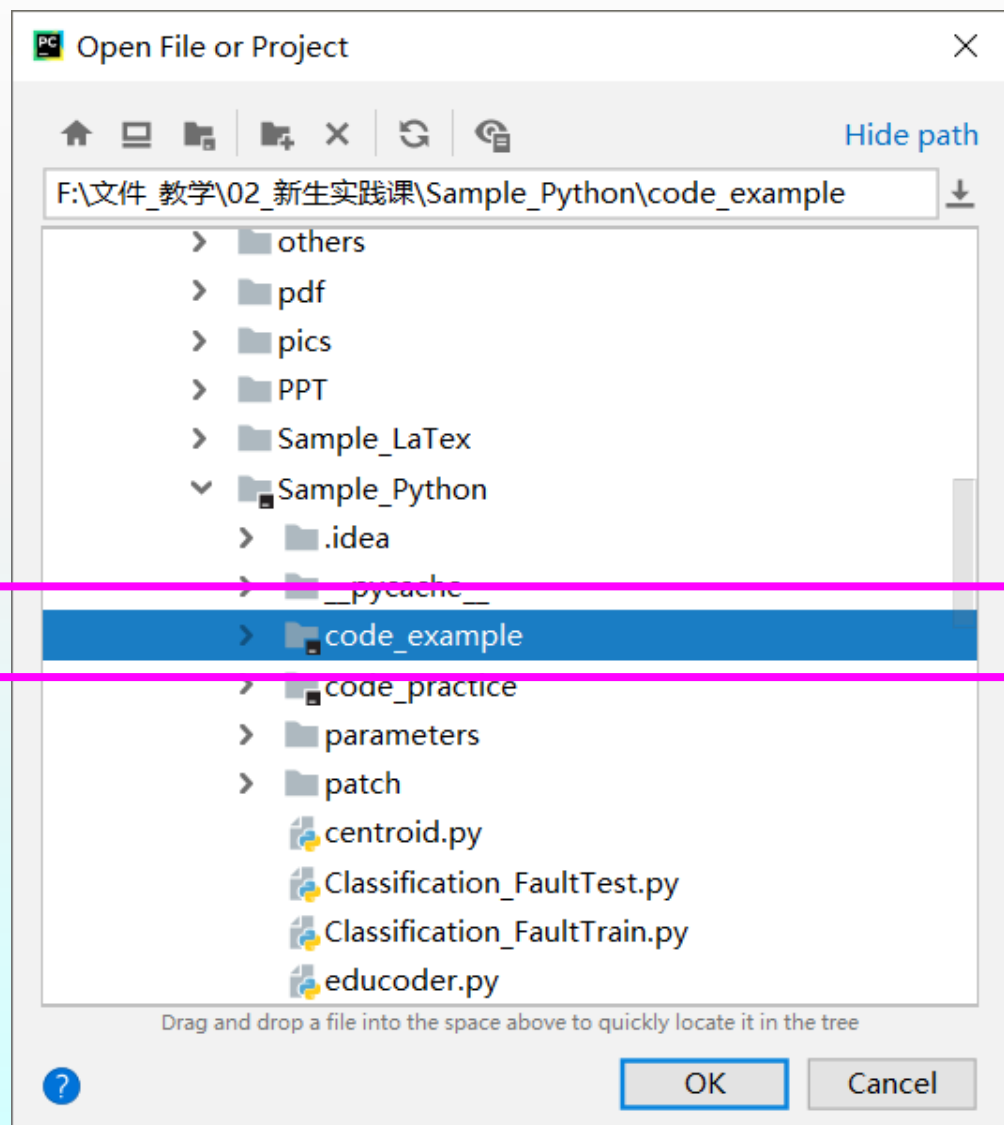


# 另外一种方式打开工作路径

- 菜单栏点File → Open

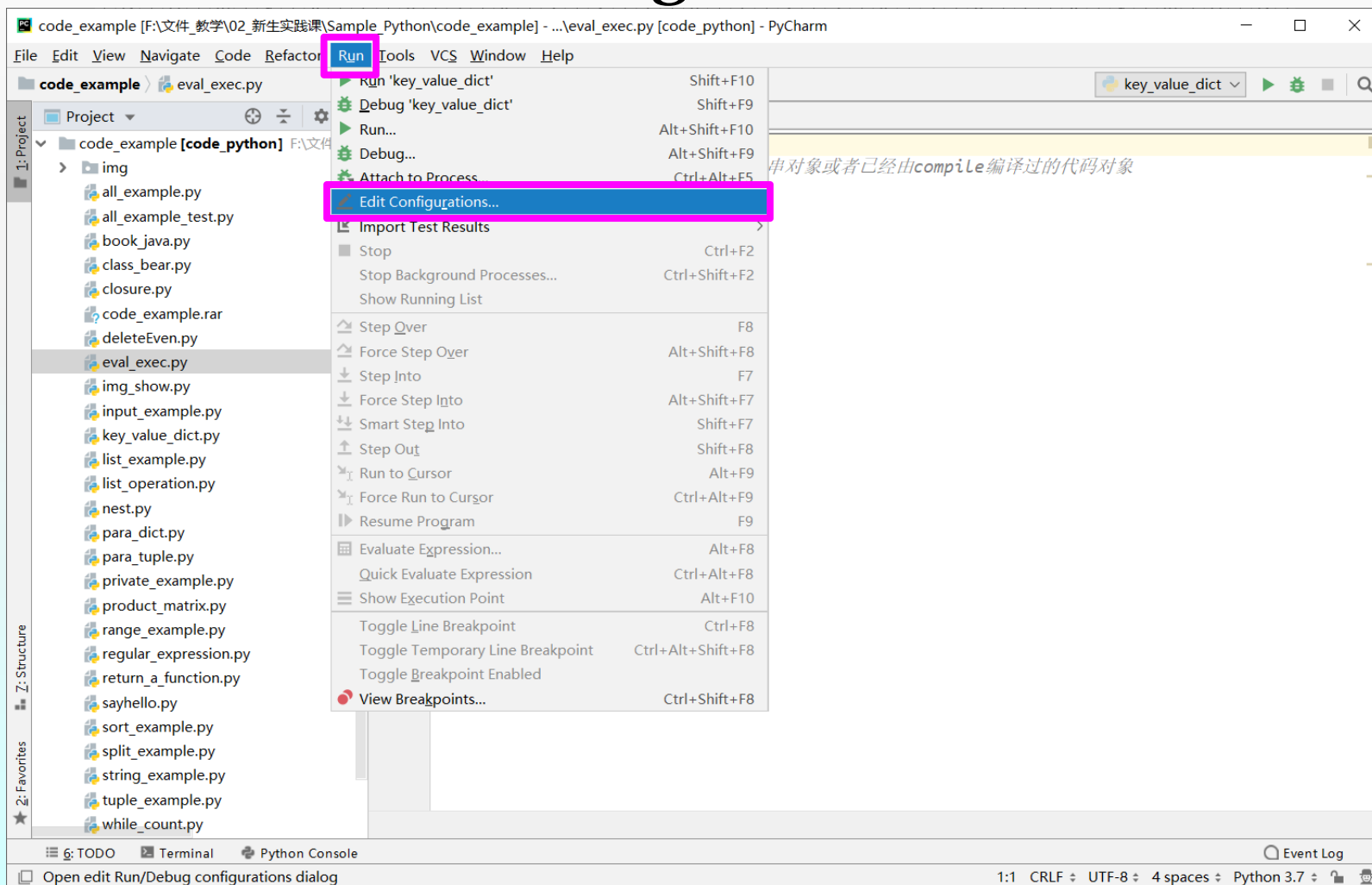


# 另外一种方式打开工作路径



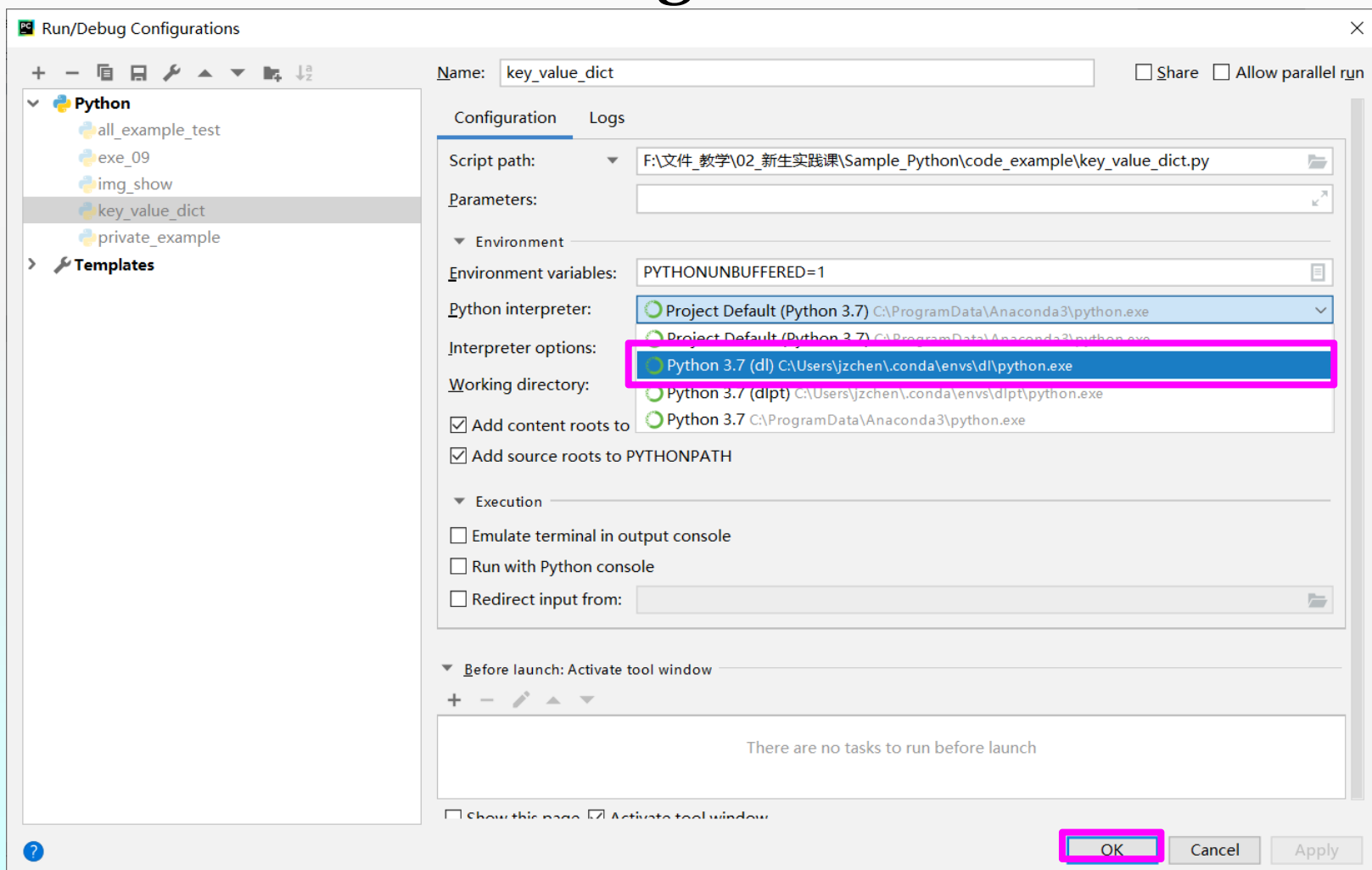
# 为当前工作路径选择解释器

- 点Run → Edit Configuration



# 为当前工作路径选择解释器

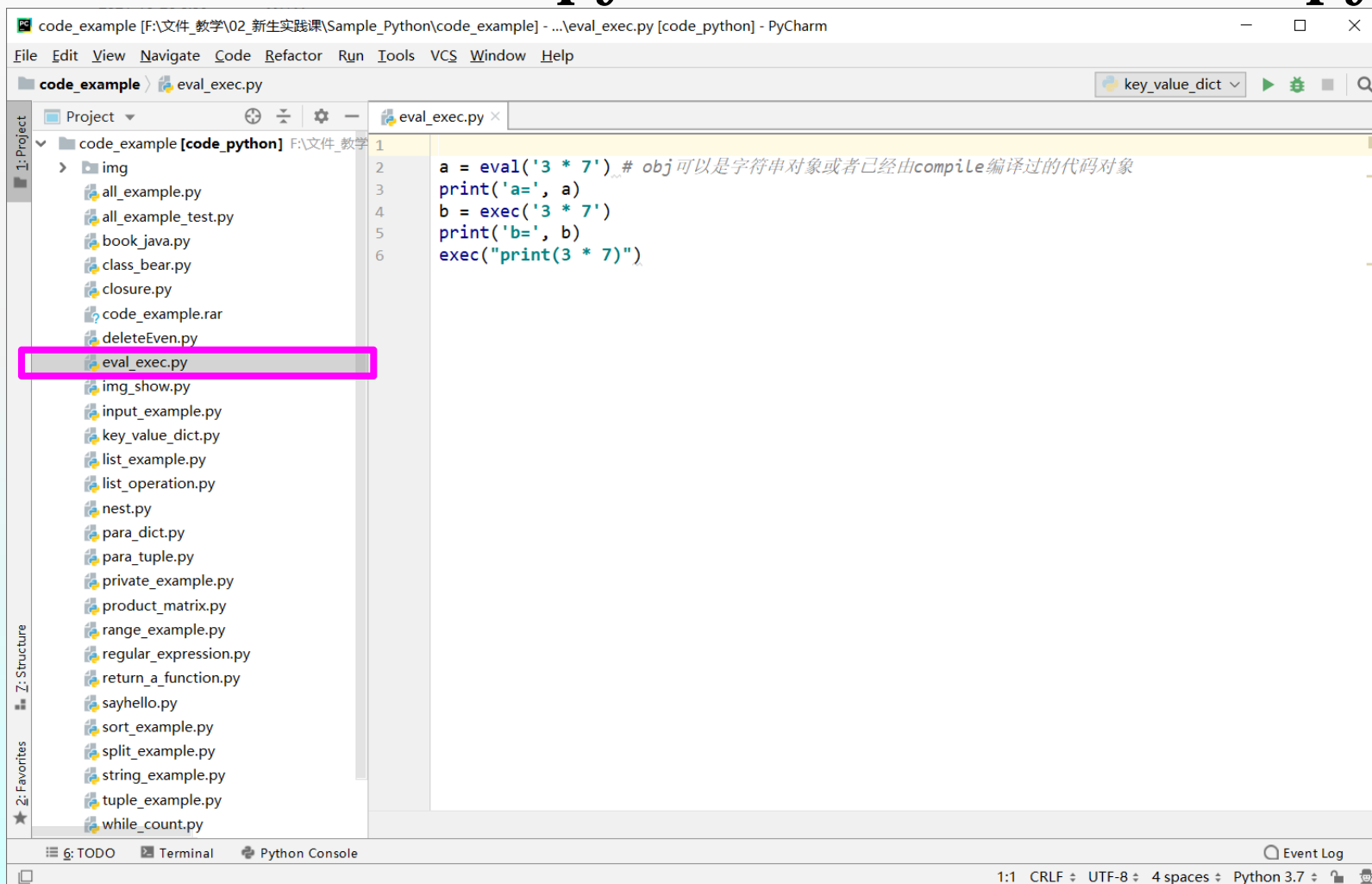
- 点Run → Edit Configuration





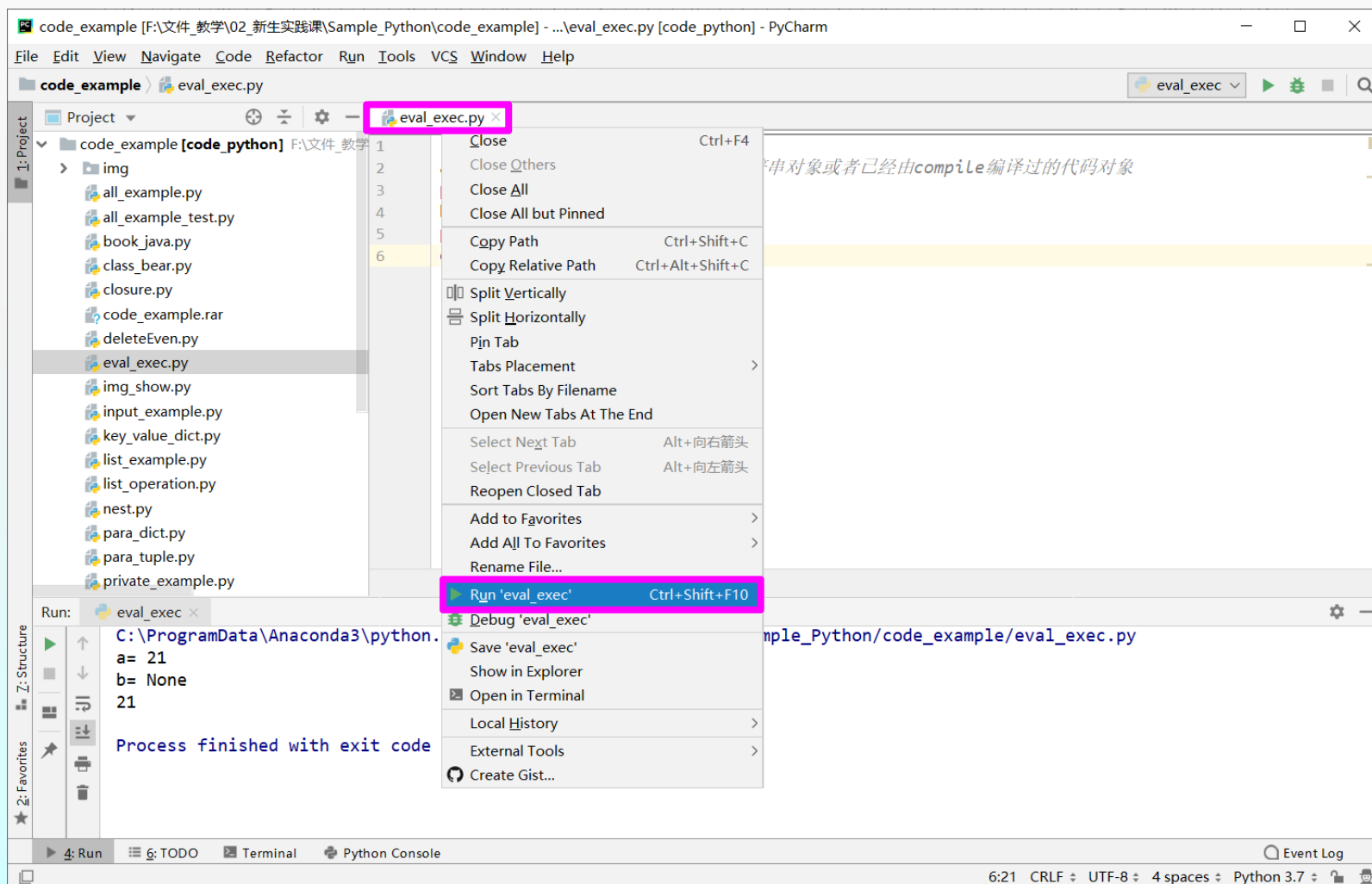
# 运行一个程序

- 双击左侧的某个.py文件, 比如eval\_exec.py



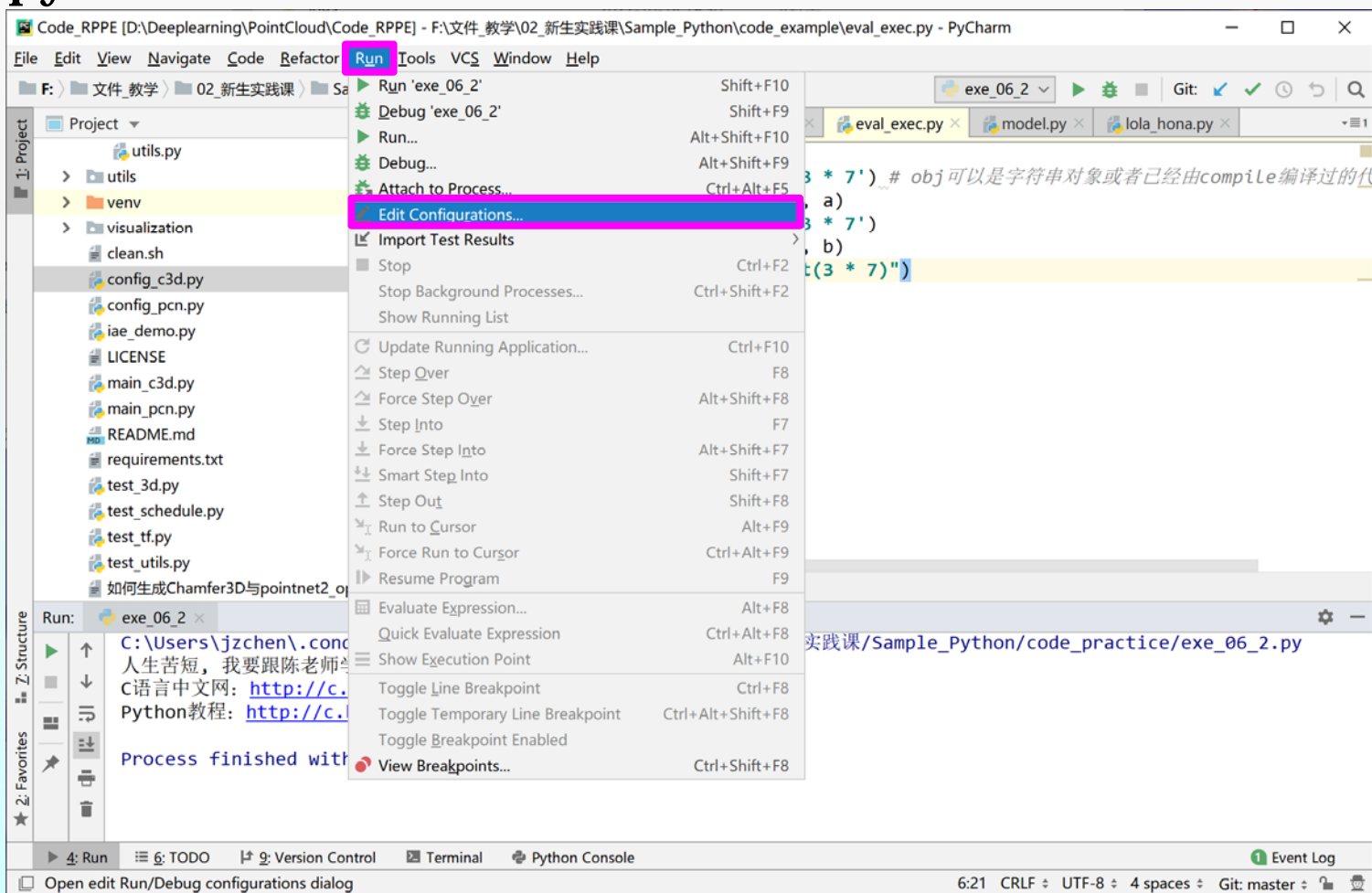
# 运行一个程序

- 光标放在eval\_exec.py上, 点鼠标右键, 点Run 'eval\_exec'



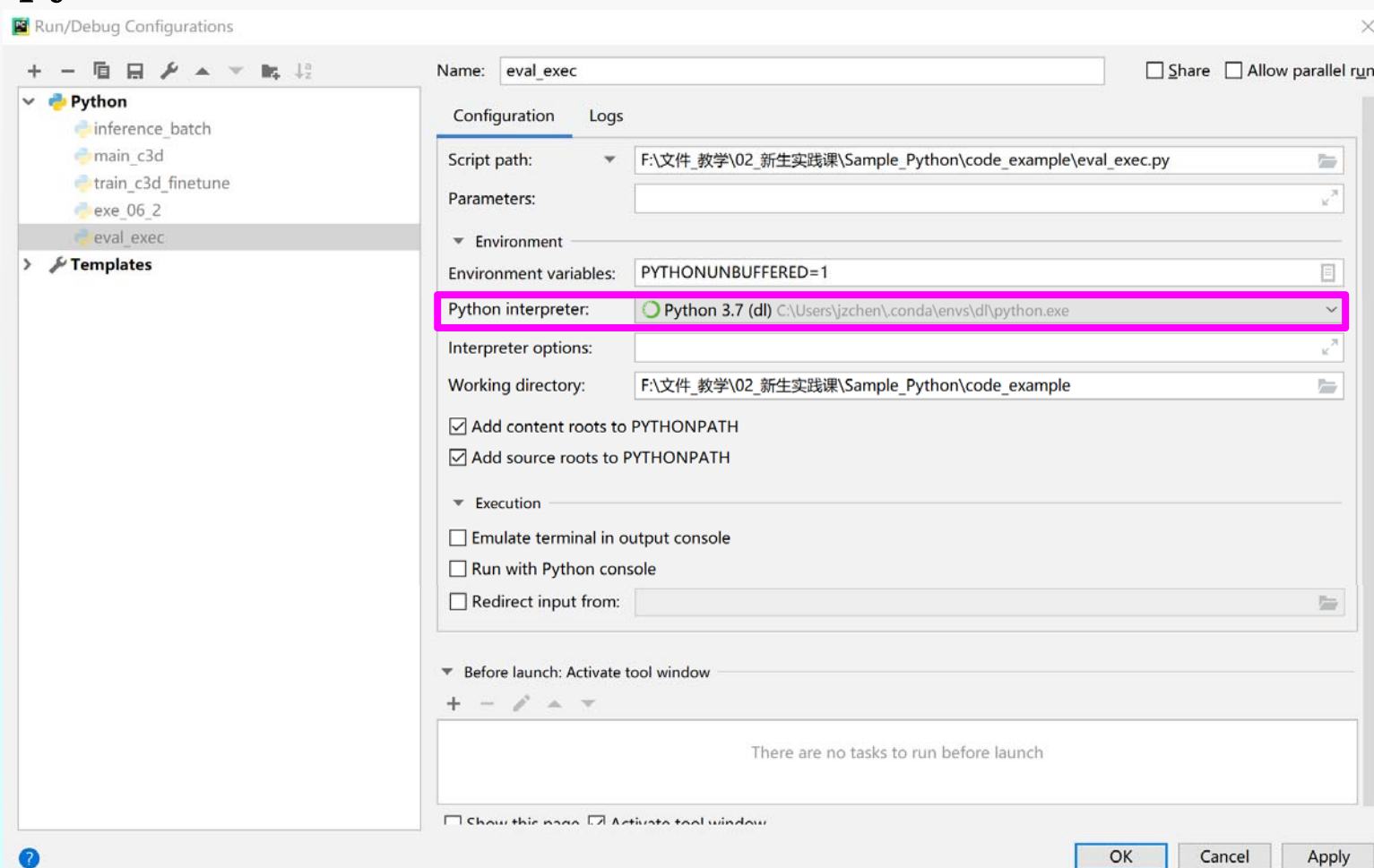
# 运行一个程序

- 可能会报错: Run→Edit Configurations, 需要为当前工程或py文件指定解释器



# 运行一个程序

- 可能会报错: Run→Edit Configurations, 需要为当前工程或py文件指定解释器



# Python基础知识

- Python开发环境
- **Python基础语法**
- Python数据类型
- 表达式
- 函数结构
- 流程控制

# Python标识符

- 在python里, 标识符有字母、数字、下划线组成
- 在python中, 所有标识符可以包括英文、数字以及下划线\_, 但不能以数字开头(**matlab也这样**). 但文件名可以
- python中的标识符是区分大小写的
- 以下划线**开头**的标识符是有特殊意义的:
  - ✓ 以单下划线开头\_**foo**的代表不能直接访问的类属性, 需要通过类提供的接口进行访问, 不能用"from xxx import \*"而导入
  - ✓ 以双下划线开头的\_\_**foo**代表类的私有成员
  - ✓ 以双下划线开头和结尾的\_\_**foo**\_\_代表python里特殊方法专用的标识, 如\_\_init\_\_()代表**类**的构造函数

# Python Enhancement Proposal #8

- 缩写为PEP 8, 它是针对Python代码格式而编订的 **风格指南**
  - ✓ 全局变量使用英文大写, 单词之间加下划线  
`SCHOOL_NAME = 'HUST'`
  - ✓ 全局变量一般只在模块内有效, 实现方法: 使用 `__all__` 机制或添加一个前置下划线:
    - ◆ 下划线开头的不会被导入
  - ✓ 私有变量使用英文小写和一个前导下划线: `_student_name`
  - ✓ 内置变量使用英文小写, 两个前导下划线和两个后置下划线: `__maker__`
  - ✓ 一般变量使用英文小写, 单词之间加下划线: `lass_name`

# Python保留字符

- **eval\_exec.py**
- 下面的列表显示了在Python中的保留字, 这些保留字不能用作常数或变数, 或任何其他标识符名称
- 所有Python的关键字只包含小写字母
- **and**, **exec**, **eval**, **not**, **assert**, **finally**
- **or**, **break**, **for**, **pass**, **class**
- **from**, **print**, **continue**, **global**, **raise**
- **def**, **return**, **import**, **del**
- **try**, **if**, **elif**, **else**, **while**, **in**
- **is**, **with**, **except**, **lambda**, **yield**
- 比较eval('3 \* 7')与exec('3 \* 7')的区别



# 行和缩进

- **Python**与其他语言最大的区别是, **Python**的代码块不使用大括号{}来控制类,函数以及其他逻辑判断, **Python**最具特色的就是用缩进来写模块
- 缩进的空白数量是可变的,但是所有代码块语句必须包含相同的缩进空白数量,这个必须严格执行
- 因此,在**Python**的代码块中必须使用相同数目的行首缩进空格数
- 建议在每个缩进层次使用单个制表符或两个空格或四个空格,切记不能混用

# 多行语句

- **Python**语句中一般以**新行**作为为语句的结束符
- 但是我们可以使用斜杠(\)将一行的语句分为多行显示(**LaTeX**用\\), 如下所示:

```
item_one = 'one+'
item_two = 'two+'
item_three = 'three'
total = item_one + \
        item_two + \
        item_three
print(total)
```

- 语句中包含[], {} 或 () 括号就**不需要**使用多行连接符, 如下实例:

```
days = ['Monday', 'Tuesday', 'Wednesday', 'thursday', 'Friday',
        'Saturday', 'Sunday']
print(days)
```

# Python引号和注释

- **Python** 接收单引号'、双引号"、三引号""""来表示字符串,引号的开始与结束必须是相同类型
- **Python** 中单行注释采用#开头
- **Python** 中多行注释使用三个单引号('')或三个双引号(""")

```
1
2 a = eval('3 * 7') # obj可以
3 print('a=', a)
4 b = exec('3 * 7')
5 print('b=', b)
6 exec("print(3 * 7)")
```

```
1
2 """
3 a = eval('3 * 7') # obj可以是字符串
4 print('a=', a)
5 b = exec('3 * 7')
6 print('b=', b)
7 exec("print(3 * 7)")
8 """
```

# 多个语句构成代码组

- 像if、while、def和class这样的复合语句, 首行以关键字开始, 以冒号 : 结束, 该行之后的一行或多行代码构成代码组
- 缩进相同的一组语句构成一个代码块, 可称之为代码组
- 语句中不要用中文标点
- 将首行及后面的代码组称为一个子句 (clause)
- 实例如下:

```
you_love_LaTeX = 1
you_dont_love_LaTeX = 1 - you_love_LaTeX
if you_love_LaTeX == 1:
    score = 95
    print(score)
elif you_dont_love_LaTeX == 1:
    print('Please work hard on LaTeX')
else:
    score = 65
    print(score)
```

# 输入输出

- **input\_example.py**
- 输入函数: `input()`
- 输出函数: `print()`
- 实例如下:
  - ✓ `name=input("你的姓名:")`
  - ✓ `print("你的姓名是%s "%name)`
  - ✓ `%s`的作用是将对象传到`str()`方法中进行处理, 输出字符串

# Python基础知识

- Python开发环境
- Python基础语法
- **Python数据类型**
- 表达式
- 函数结构
- 流程控制

# Python数据类型

- Python语言是面向对象(Object)的编程语言,可以说在Python中**一切皆对象**
- 对象是某类型具体**实例**中的某一个,每个对象都有身份、类型和值
  - ✓ 身份(Identity)与对象都是唯一对应关系,每一个对象的身份产生后就都是独一无二的,并无法改变.对象的ID是对象在内存中获取的一段地址的**标识**
  - ✓ 类型(Type)是决定对象将以哪种数据类型进行存储
  - ✓ 值(Value)存储对象的数据,某些情况下可以修改值,某些对象**声明**值过后就不可以修改了

# Python数据类型

- 在Python中, 指向对象的值的名称就是变量, 也就是一种标识符, 是对内存中的存储位置的命名
- 变量使用等号赋值以后会被创建, 定义完成后可以直接使用
- **Python**有6个标准的数据类型:
  - ✓ **Numbers** (数值)
  - ✓ **String** (字符串)
  - ✓ **List** (列表)
  - ✓ **Tuple** (元组)
  - ✓ **Dictionary** (字典)
  - ✓ **Sets**(集合类型)



# 变量的定义

- 在**Python**中, **变量类型**是由赋给它的**数值类型**定义的, 例如:
  - ✓ `q = 7` #q其为数值型变量
  - ✓ `q = "Seven"` #q为字符串型变量
  - ✓ `q = [3, 5, "moon"]` #q为列表型变量

# Python支持的四种不同数值类型

- 整型(Int): 通常被称为是整型或整数, 是正或负整数, 不带小数点
- 布尔型(Booleans): 布尔值是整型的子类, 用于逻辑判断真(True)或假(False), 用数值1和0分别代表常量True和False. 在Python语言中, False可以是数值为0、对象为None或者是序列中的空字符串、空列表、空元组
- 浮点型(floating point real values): 浮点型由整数部分与小数部分组成, 浮点型也可以使用科学计数法表示( $2.5e2 = 2.5 \times 10^2 = 250$ )
- 复数(complex numbers): 由实数部分和虚数部分构成, 可以用 $a + bj$ , 或者`complex(a, b)`表示, 复数的实部a和虚部b都是浮点型

# Python整型Integers

- 整数类型(int)简称为整型, 表示整数, 包括正负的整数, 如: 0110、-123、123456789
- Python的整型是长整型, 能表达的数的范围是无限的, 内存足够大, 就能表示足够多的数
- 在使用整型的数还包括其它进制
  - ✓ 0b开始的是二进制(**b**inary), 0o开始的是八进制(**o**ctonary)
  - ✓ 0x开始的十六进制(**hex**adecimal)
  - ✓ 进制之间可以使用函数进行转换, 使用时需要注意数值符合进制

# Python字符串

- **ilovepython.py**
- 字符串(String)是由数字、字母、下划线组成的一串字符
- Python的字符串列表有2种取值顺序:
  - ✓ 从左到右**索引**, 从**0**开始, 最大是字符串长度减1
  - ✓ 从右到左**索引**, 从**-1**开始, 最大范围是字符串开头
- 切片: 如果要取得一段子串的话, 可以用**切片**[头下标:尾下标:**步长**]方式
- 步长默认为1, 可以是正数或负数, 下标为空表示取到头或尾
- 注意从“头下标”开始, 到“尾下标”的前一位结束 (不包含尾下标本身)
  - ✓ 例如: `s = 'ilovepython'`, 字符串的范围从左到右为**0~10**
    - ◆ 则`s[2]`的结果为o
    - ◆ `s[1:5:2]`的结果是lv
    - ◆ `s[-1::-1]`的结果是nohtypevoli

# 加号+与星号\*

- **string\_example.py**
- 加号(+)是字符串连接运算符, 星号(\*)是重复操作, 实例如下:

```
3  str = "Hello World!"
4  print(str)
5  print(str[0])
6  print(str[2:5])
7  print(str[2:])
8  print(str*2)
9  print(str+" I am coming!")
```

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World! I am coming!
```

# 其它常用字符串函数

- **split\_example.py**
- `split()`: 字符串分割, 经常用于修改文件名的后缀!
- `len()`: 取字符串长度, 例如: `len(str)`
- `upper()`: 源字符串转换为大写, 例如: `str.upper()`
- `lower()`: 源字符串转换为小写, 例如: `str.lower()`
- `title()`: 字符串所有单词的首字母变成大写, 例如: `str.title()`
- `strip()`: 去除字符串首尾空格, 例如: `str.strip()`
- `find()`: 字符串查找
- `replace()`: 字符串替换

# Python列表

- **list\_example.py**
- List(列表)是 Python 中使用最频繁的数据类型
- 列表可以完成大多数集合类的数据结构实现,它支持字符、数字、字符串甚至可以包含列表 (所谓嵌套)
- 列表用[]标识,列表中的切片方法和前面的字符串相同,采用[头下标:尾下标:步长],就可以截取相应的列表
- 加号(+)是列表连接运算符,星号(\*)是重复操作,实例如下:

```
2 list = ['kk', 768, 2.23, 'json', 70.2]
3 tinlist = [123, 'json']
4 print(list)                ['kk', 768, 2.23, 'json', 70.2]
5 print(list[0])              kk
6 print(list[1:3])            [768, 2.23]
7 print(list[2:])             [2.23, 'json', 70.2]
8 print(tinlist*2)            [123, 'json', 123, 'json']
9 print(list+tinlist)         ['kk', 768, 2.23, 'json', 70.2, 123, 'json']
```

# 列表基本操作

- **list\_operation.py**
- 若: `guests=['Zhang san', 'Li si', 'Wang wu', 'Zhao liu']`,
- 添加元素:
  - ✓ `insert()`: 在列表 **指定位置** 添加元素, 例如 `guests.insert(1, 'hu qi')`
  - ✓ `append()`: 在列表 **尾部** 添加元素, 例如 `guests.append('hu qi')`
- 按索引修改列表元素:
  - ✓ `guests[2]='wang fang'`
- 删除列表元素:
  - ✓ `del()`: 按索引删除指定元素, 例如 `del guests[2]`
  - ✓ **pop()**: **按索引删除** 并返回元素, 例如 `name=guests.pop(2)`
  - ✓ `remove()`: **按值** 删除元素, 例如 `guests.remove('Li si')`
- 排序: `sort(reverse=True)`, **深度学习中读入训练集时常用到**
  - ✓ 不填参数或参数设为 **False** 则正向排序, 否则反向排序



# 列表基本操作

- **range\_example.py**
- 创建数字列表: `range(lower_limit, upper_limit, step)`
  - ✓ **lower\_limit**: 生成系列整数的下限整数, **不填该参数则默认为从0开始**, 生成的整数从此数开始, 包括该数
  - ✓ **upper\_limit**: 生成系列整数的上限整数, **必填**参数, 生成的整数要小于该上限
  - ✓ **Step**: 在下限和上限之间生成系列整数之间的间隔步长, **不填该参数则默认步长为1**

```
1 # range_example.py
2 print(range(10)) # 默认从0开始
3 print(range(0, 10))
4 numbers = []
5 for i in range(10):
6     number = i**2
7     numbers.append(number) # 用append追加列表的新元素
8
9 print(numbers)
10
```

# 列表基本操作

- 使用list()函数和range()函数创建数字列表
  - ✓ `data_list = list(range(lower_limit, upper_limit, step))`
  - ✓ 例如, 我们要生成并输出1~5的数字列表
    - ◆ 1. `data_list = list(range(1,6))`
    - ◆ 2. `print(data_list)`
- 对数字列表进行简单的统计运算:
  - ✓ `max()`: 求最大值
  - ✓ `min()`: 求最小值
  - ✓ `sum()`: 求和

# Python元组

- **tuple\_example.py**
- 元组tuple是另一个数据类型, 类似于list(列表)
- 元组用**()**标识, 内部元素用逗号隔开, **但是元组不能二次赋值, 相当于只读列表**

```
2 tuple_in = ('kk', 768, 2.23, 'json', 70.2)
3 tintuple = (123, 'json')
4 print(tuple_in)           ('kk', 768, 2.23, 'json', 70.2)
5 print(tuple_in[0])        kk
6 print(tuple_in[1:3])      (768, 2.23)
7 print(tuple_in[2:])       (2.23, 'json', 70.2)
8 print(tintuple*2)          (123, 'json', 123, 'json')
9 print(tuple_in+tintuple)  ('kk', 768, 2.23, 'json', 70.2, 123, 'json')
```

# Python元组(续)

- 元组tuple的基本操作:
  - ✓ len(): 求元素个数
  - ✓ max(): 求最大值
  - ✓ min(): 求最小值
  - ✓ tuple(): 将列表转换为元组

# Python字典

- **key\_value\_dict.py**
- 列表是有序的对象结合,字典(dictionary)是**无序**的对象集合
- 字典用“{ }”标识,字典由索引(**键key**)和它对应**值value**组成,其基本操作方法和列表类似
- 字典中的元素是通过**键**来存取的,而不是通过偏移存取. Python**为字典**类型提供了items()方法, items()方法会将字典里的所有的**键与值**一起返回,也可用keys()或values()方法仅访问**索引**或**值**,例如:

```
2 # 创建并初始化名叫menu的字典
3 menu = {'fish': '40', 'pork': '30', 'patato': '20', 'lamb': '50'}
4 # 利用item()方法便利输出键key和值value
5 for key, value in menu.items():
6     print('key:' + key + ', ', 'value:' + value)
```

Run: key\_value\_dict ×

C:\ProgramData\Anaconda3\python.exe F:/教学/新生实践课/  
key:fish, value:40  
key:pork, value:30  
key:patato, value:20  
key:lamb, value:50

# Python数据类型转换

- 有时候,需要对数据内置的类型进行转换,数据类型的转换,只需要将数据类型作为函数名即可. 以下几个内置的函数可以执行数据类型之间的转换,这些函数返回一个新的对象,表示转换的值

函数	描述
<code>int(x, [base])</code> , <code>int('01010101',2)</code> , 结果为85	将x转换为一个整数
<code>long(x, [base])</code>	将x转换为一个长整数
<code>float(x)</code>	将x转换到一个浮点数
<code>complex(real [,imag])</code>	创建一个复数
<code>str(x)</code>	将对象 x 转换为字符串
<code>repr(x)</code>	将对象 x 转换为表达式字符串
<code>eval(str)</code>	计算在字符串中的有效表达式,并返回一个对象
<code>tuple(s)</code>	将序列 s 转换为一个元组
<code>list(s)</code>	将序列 s 转换为一个列表
<code>set(s)</code>	转换为可变集合
<code>dict(d)</code>	创建一个字典, d 必须是一个序列 (key,value) 元组
<code>frozenset(s)</code>	转换为不可变集合
<code>chr(x)</code> 不是char(x)	将一个整数转换为一个字符
<code>unichr(x)</code>	将一个整数转换为Unicode字符
<code>ord(x)</code>	将一个字符转换为它的整数值
<code>hex(x)</code>	将一个整数转换为一个十六进制字符串
<code>oct(x)</code>	将一个整数转换为一个八进制字符串
<code>type(对象)</code> , 如: <code>type(1)</code> 返回 <code>&lt;class 'int'&gt;</code>	内建的用来查看变量类型的函数

# Python基础知识

- Python基础知识
- Python基础语法
- Python数据类型
- 表达式
- 函数结构
- 流程控制

# 算术运算符

- 算术运算符主要是用于数字类型的数据基本运算, python 支持直接进行计算, 也就是可以将python shell当计算器来使用

运算符	说明	表达式	结果
+	加: 把数据相加	10 + 24	34
-	减: 把数据相减	34 - 10	10
*	乘: 把数据相乘	34*10	340
/	除: 把数据相除	34/10	3.4
%	取模: 除法运算求余数	34 % 10	4
**	幂: 返回 x 的 y 次幂	2**4	16
//	取整除: 返回商整数部分	34 // 10	3

\*可以返回重复若干次的字符串



# 比较运算符

- 比较运算符用于判断同类型的对象是否相等, 比较运算的结果是布尔值 **True**或**False**, 比较时因数据类型不同比较的依据不同
- 复数不可以比较大小, 但可以比较是否相等
- 在**Python**中比较的值相同时, 也不一定是同一个对象

运算符	说明	表达式	结果
<b>==</b>	等于: 判断是否相等	1 == 1	True
<b>!=</b>	不等于: 判断是否不相等	1 != 1	False
>	大于: 判断是否大于	1 > 2	False
<	小于: 判断是否小于	1 < 2	True
>=	大于等于: 判断是否大于等于	1 >= 2	False
<=	小于等于: 判断是否小于等于	1 <= 2	True

# 逻辑运算符

- 逻辑运算符为and (与)、or (或)、not (非)用于逻辑运算判断表达式的True或者False, 通常与流程控制一起使用

运算符	表达式	x	y	结果	说明
and	x and y	True	True	True	表达式一边有 False 就会返回 False, 当两边都是 True 时返回 True。
		True	False	False	
		False	True	False	
		False	False	False	
or	x or y	True	True	True	表达式一边 True 就会返回 True, 当两边都是 False 时返回 False。
		True	False	True	
		False	True	True	
		False	False	False	
not	not x	True	/	False	表达式取反, 返回值与原值相反。
		False	/	True	

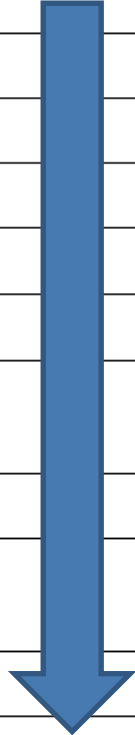
# 复合赋值运算符

- 复合赋值运算符是将一个变量参与运算的运算结果赋值给改变量,即x参加了该运算,运算完成后结果赋值给x. **运算符与等号间不能有空格**

运算符	说明	表达式	等效表达式
=	直接赋值	$x = y + z$	$x = z + y$
+=	加法赋值	$x += y$	$x = x + y$
-=	减法赋值	$x -= y$	$x = x - y$
*=	乘法赋值	$x *= y$	$x = x * y$
/=	除法赋值	$x /= y$	$x = x / y$
%=	取模赋值	$x \% = y$	$x = x \% y$
**=	幂赋值	$x ** = y$	$x = x ** y$
//=	整除赋值	$x //= y$	$x = x // y$

# 运算符优先级

- 由数值、变量、运算符组合的表达式和数学上相同,是有运算符优先级的,优先级高的运算符先进行运算,同级运算符,自左向右运算,遵从**小括号优先**原则
- 只有同级运算符时时从左到右结合,如 $1+2+3*5+5$
- 出现赋值的时候一般是右结合,如: `priority=1+2`, 先算出 $1+2$ 的值以后再赋值给`priority`



优先级	类别	运算符	说明
最高	算术运算符	**	指数, 幂
高	位运算符	+X, -X, ~X	正取反, 负取反, 按位取反
	算术运算符	*, /, %, //	乘, 除, 取模, 取整
	算术运算符	+, -	加, 减
	位运算符	>>, <<	右移, 左移运算符
	位运算符	&	按位与, 集合并
	位运算符	^	按位异或, 集合对称差
	位运算符		按位或, 集合并
	比较运算符	<=, <, >, >=	小于等于, 小于, 大于, 大于等于
	比较运算符	==, !=	等于, 不等于
	赋值运算符	=, %=, /=, //, -=, +=, *=, **=	赋值运算
	逻辑运算符	not	逻辑“非”
	逻辑运算符	and	逻辑“与”
低	逻辑运算符	or	逻辑“或”

# Python基础知识

- Python开发环境
- Python基础语法
- Python数据类型
- 表达式
- 函数结构
- 流程控制

# Python函数

- 函数是组织好的、可重复使用的、用来实现单一,或相关联功能的**代码段**
- 函数能提高应用的模块性,和代码的重复利用率
- Python提供了许多**内建**函数,比如print();可以自己创建函数,这被叫做用户自定义函数
- 可以定义一个由自己想要功能的函数,以下是简单的规则:
  - ✓ 函数代码块以 **def** 关键词开头,后接函数**标识符**名称和**圆括号()**
  - ✓ 任何传入的参数和自变量必须放在圆括号中
  - ✓ 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明
  - ✓ 函数内容以**冒号**起始,并且缩进
  - ✓ **return** [表达式] 结束函数,选择性地返回一个值给调用方
  - ✓ 不带表达式的**return**相当于返回 **None**

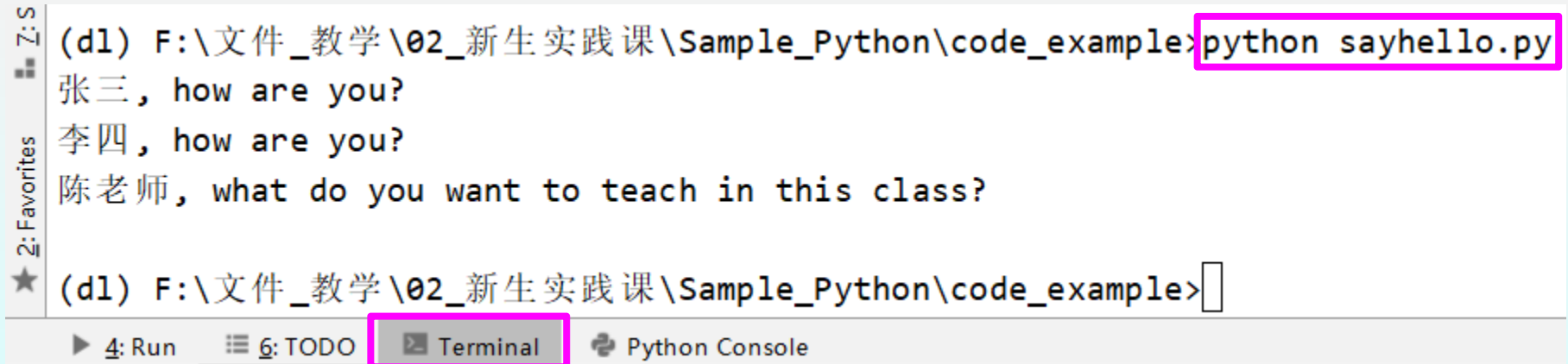
# Python函数定义

- **sayhello.py**
- 默认情况下, 参数值和参数名称是按函数声明中定义的顺序匹配起来的
- 以下为一个简单的Python函数, 它将一个字符串作为传入参数, 再打印到标准显示设备上

```
2  def printsayhello(name):
3      if name == "陈老师":
4          name = name + ", what do you want to teach in this class?"
5      else:
6          name = name + ", how are you?"
7      print(name)
8      return name
9
10 printsayhello("张三")
11 printsayhello("李四")
12 printsayhello("陈老师")
```

# Python函数调用

- 定义一个函数只给了函数一个名称, 指定了函数里包含的参数, 和代码块结构
- 这个函数的基本结构完成以后, 你可以通过另一个函数调用执行, 也可以直接从Python提示符执行
- 如下实例调用前页定义的printsayhello()函数:



The screenshot shows a Python IDE interface. On the left, there is a sidebar with icons for 'Favorites', 'Z: S', and a star icon. The main area is a terminal window with a light blue background. The terminal text is as follows:

```
(d1) F:\文件_教学\02_新生实践课\Sample_Python\code_example>python sayhello.py
张三, how are you?
李四, how are you?
陈老师, what do you want to teach in this class?

(d1) F:\文件_教学\02_新生实践课\Sample_Python\code_example>
```

At the bottom of the terminal window, there is a tab bar with four tabs: '4: Run', '6: TODO', 'Terminal', and 'Python Console'. The 'Terminal' tab is currently selected and highlighted with a pink border.



# 函数参数类型

- 必选参数

- ✓ 定义加法函数plus, 参数a, b就是必选参数
- ✓ 调用函数plus时, 必须给参数a, b传递值
- ✓ 调用时候, 函数后面不能加冒号

```
def plus(a,b):  
    c = a + b  
    return c  
  
d = plus(1,2)  
print(d)
```

# 函数参数类型

- 默认参数

- ✓ 定义加法函数PLUS, a是必选参数, b是默认值为2的参数
- ✓ 调用函数PLUS时, 则必须给a传递值, 若不给b传递值, 则b默认为2
- ✓ 如果默认a, 怎么给b传递值?

```
def PLUS(a,b=2):  
    c = a + b  
    return c  
  
d = PLUS(1)  
print(d)
```

```
def PLUS(a=1,b=2):  
    c = a + b  
    return c  
  
d = PLUS(b=1)  
print(d)
```

# 函数参数类型

- **para\_tuple.py**
- 可变参数, 参数汇集成元组

```
1  # para_tuple.py
2  # *numbers表示输入参数的个数可以是任意指
3  def myplus(*numbers):
4      add = 0
5      for i in numbers:
6          add += i
7      return add
8
9  # 调用3次plus函数, 每次参数个数都不相同
10
11  d1 = myplus(1,2,3)
12  d2 = myplus(1,2,3,4)
13  d3 = myplus(1,3,5,7,9)
14
15  # 向函数中可以传递0个参数
16  d4 = myplus()
17  print("d1=",d1,"d2=",d2,"d3=",d3,"d4=",d4)
```

# 函数参数类型

- **para\_dict.py**
- 当我们声明一个诸如 **\*\*number** 的双星号参数时, 从此处开始直至结束的所有关键字参数都将被收集并汇集成一个名为 **number** 的字典

```
2  #把赋值的等号两边变成key和value
3  def myplus(**number):
4      return number
5
6  d1 = myplus() # 向函数中可以传递0个参数
7  d2 = myplus(x=1)
8  d3 = myplus(x1=1, y1=2)
9
10 print("d1=", d1, "d2=", d2, "d3=", d3)
11
```

```
↑ C:\Users\jzchen\.conda\envs\dl\python.exe F:
↓ d1= {} d2= {'x': 1} d3= {'x1': 1, 'y1': 2}
⇅ Process finished with exit code 0
```

# 函数返回值return

- **return\_a\_function.py**
- 用return返回一个值
- 用return返回多个值(即一个元组)
  - ✓ 例如: return 1, 'abc', '1234'
- 用return返回函数

```
1  # return_a_function.py
2  # 定义求和函数, 返回的并不是求和结果, 而是计算求和的函数
3  def lazy_plus(*args):
4      def plus():
5          s = 0
6          for n in args:
7              s = s + n
8          return s
9      return plus
10
11 # 调用函数f时, 得到真正求和的结果
12 f = lazy_plus(1, 2, 3, 4, 5)
13 print(f())
```

# 嵌套调用

- **nest.py**
- 允许在函数内部创建另一个函数, 这种函数叫内嵌函数或者内部函数
- 内嵌函数的作用域在其**内部**, 如果内嵌函数的作用域超出了这个范围就不起作用, 如下所示代码:

```
def function_1():
```

```
    print('正在调用function_1()...')
```

```
    def function_2():
```

```
        print('正在调用function_2()...')
```

```
    function_2()
```

```
function_1()
```

```
function_2() #报错
```

# 使用闭包

- **closure.py**
- 闭包是函数式编程的一个重要的语法结构, 它是能够读取外部函数内的变量的函数, 外层函数的变量持久地保存在内存中
- 从表现形式上定义为, 如果在一个内部函数里对一个外部作用域(但不是在 全局作用域)的变量进行引用, 那么内部函数就认为是闭包, 如下所示代码:

```
def Fun_sub(a):
```

```
    def Fun_sub2(b):
```

```
        return a-b #内部函数里对一个外部作用域的变量进行引用
```

```
    return Fun_sub2
```

```
i = float(input('请输入减数: '))
```

```
j = float(input('请输入被减数: '))
```

```
print(Fun_sub(i)(j))
```

```
Fun_sub2(23) # NameError: name 'Fun_sub2' is not defined
```

# 函数的作用域

- 在Python中, 正常的函数和变量名是公开的(public), 都是可以被直接引用的, 比如: `abs()`、`max()` 等
- 类似`__xxx__`这种格式的变量是特殊变量, 允许被直接引用, 但是会被用作特殊用途, 比如`__author__`、`__name__`就是属于特殊变量
- 类似`_xxx`(一条下划线)和`__xxx` (两条下划线)这种格式的函数和变量就是非公开的(private) 不应该被直接引用
  - ✓ `_xxx`的函数和变量是protected, 我们直接从外部访问不会产生异常
  - ✓ `__xxx`的函数和变量是private, 我们直接从外部访问会报异常, 我们要注意前缀符号的区别



# 函数的作用域

- **private\_example.py**
- **private**函数和变量是"不应该"被直接引用,而不是"不能"被直接引用,这是因为在Python中并没有一种方法可以真正完全限制访问**private**函数或变量,但是我们为了养成良好的编程习惯,是不应该引用**private**函数或变量的
- **private**函数的作用是隐藏函数的内部逻辑,让函数有更好的封装性,我们一般都限定函数的使用范围,把外部需要使用的函数定义为**public**函数
- 把只在内部使用,而外部不需要引用的函数定义成**private**函数

```
1      # private.py
2      def _private_1(name):
3          return 'Hello, %s' % name
4
5      def _private_2(name):
6          return 'Hi, %s' % name
7
8      def greeting(name):
9          if len(name) > 3:
10             return _private_1(name)
11          else:
12             return _private_2(name)
13      name = "陈加忠"
14      print(greeting(name))
```

# 函数的import \*

- **all\_example.py**
- **all\_example\_test.py**
- 某个名为all\_example的模块, 如果定义了 `__all__`, 那么
  - ✓ 在执行 `from all_example import *` 的时候, all\_example模块中不在 `__all__` `["xxx", "yyy"]` 中的不会被导入
  - ✓ `from all_example import *`, 表示从 `from all_example import` 导入所有允许导入的函数、变量等对象
- 如果导入了单下划线开头的 `_zzz`
  - ✓ 如 `__all__` `[xxx, yyy, _zzz]`, 则也能执行 `_zzz` 函数
  - ✓ 没有 `__all__` 语句, 则不能执行 `_zzz` 函数