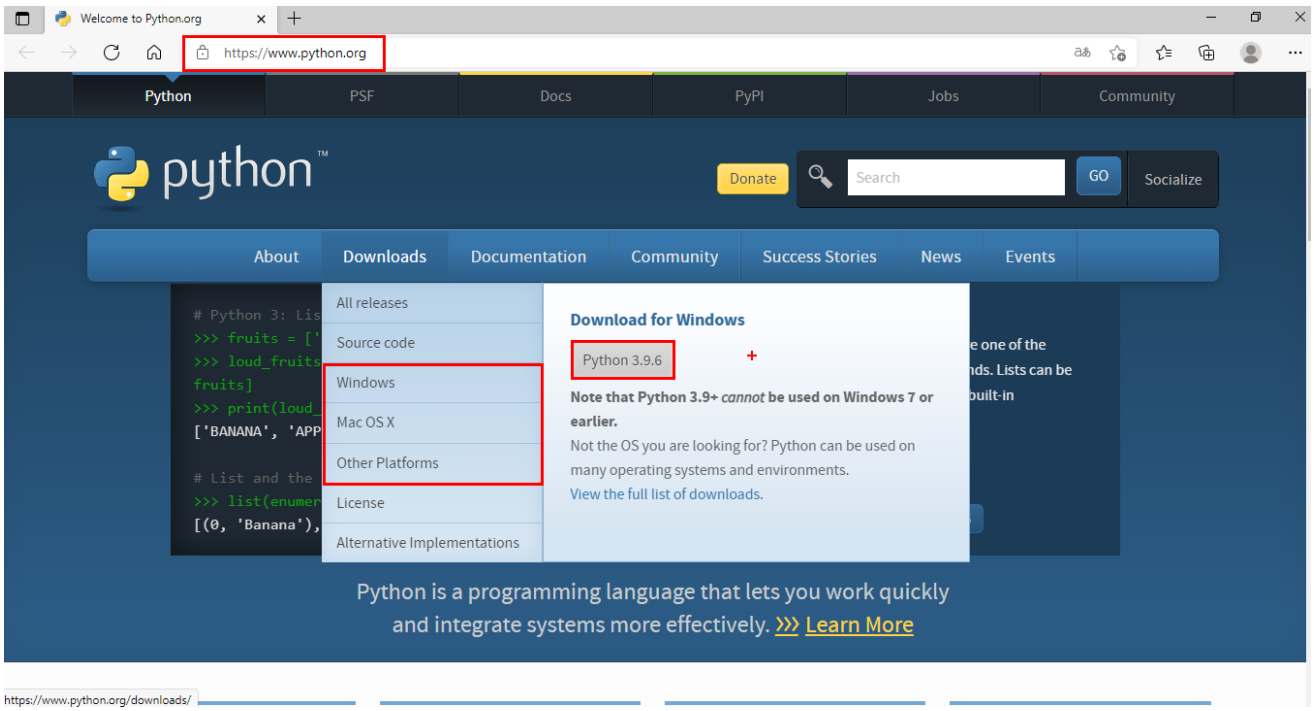
一、Python环境的安装

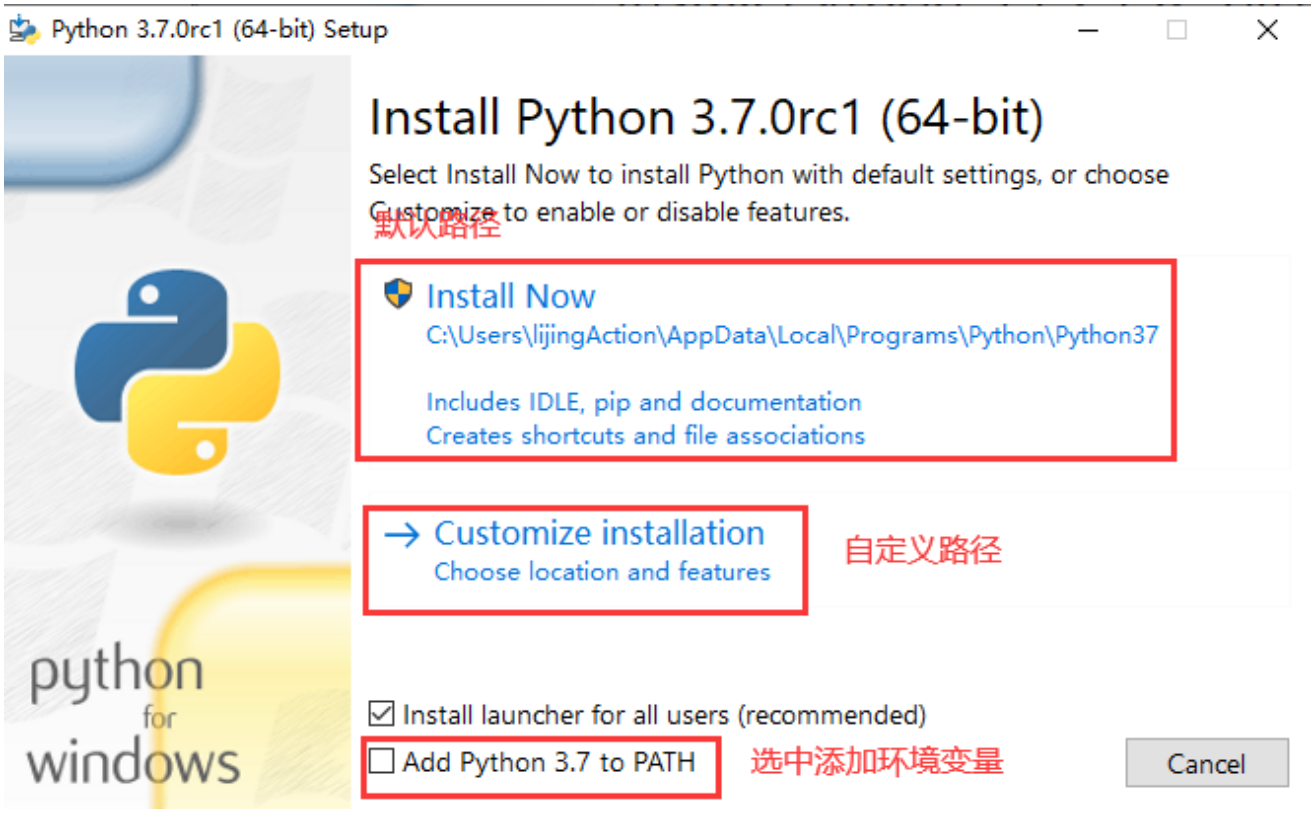
1. 下载Python

- 访问Python官网: https://www.python.org/
- 点击downloads按钮，在下拉框中选择系统类型(windows/Mac OS/Linux等)
- 选择下载最新版本的Python



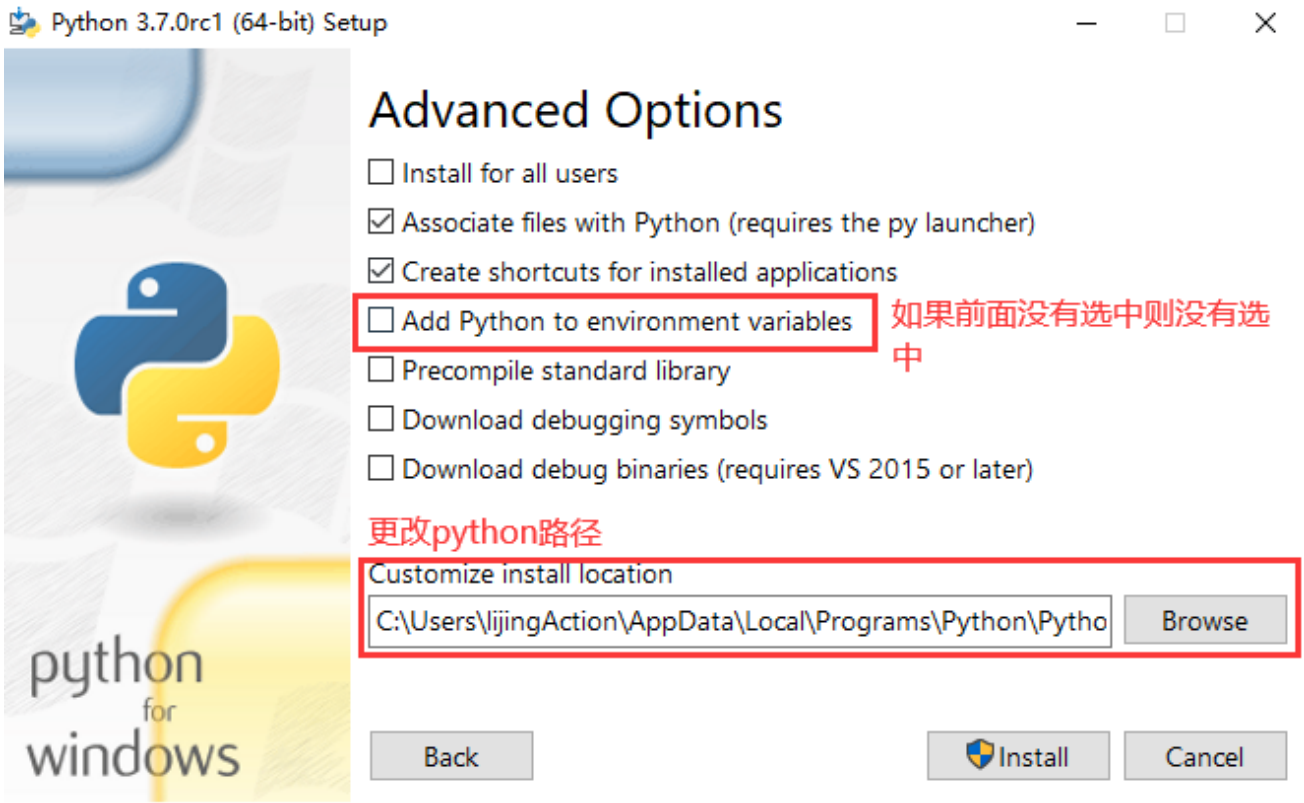
2. 安装Python

- 双击下载好的Python安装包
- 勾选左下角`Add Python 3.7 to PATH`选项，然后选择`Install now`立刻安装Python.
- 默认安装


Python 3.7.0rc1 (64-bit) Setup
Install Python 3.7.0rc1 (64-bit)
Select Install Now to install Python with default settings, or choose
Customize to enable or disable features.
默认路径
Install Now
C:\Users\lijingAction\AppData\Local\Programs\Python\Python37
Includes IDLE, pip and documentation
Creates shortcuts and file associations
Customize installation
Choose location and features
自定义路径
python
for
windows
Install launcher for all users (recommended)
Add Python 3.7 to PATH
选中添加环境变量
Cancel

- 自定义安装

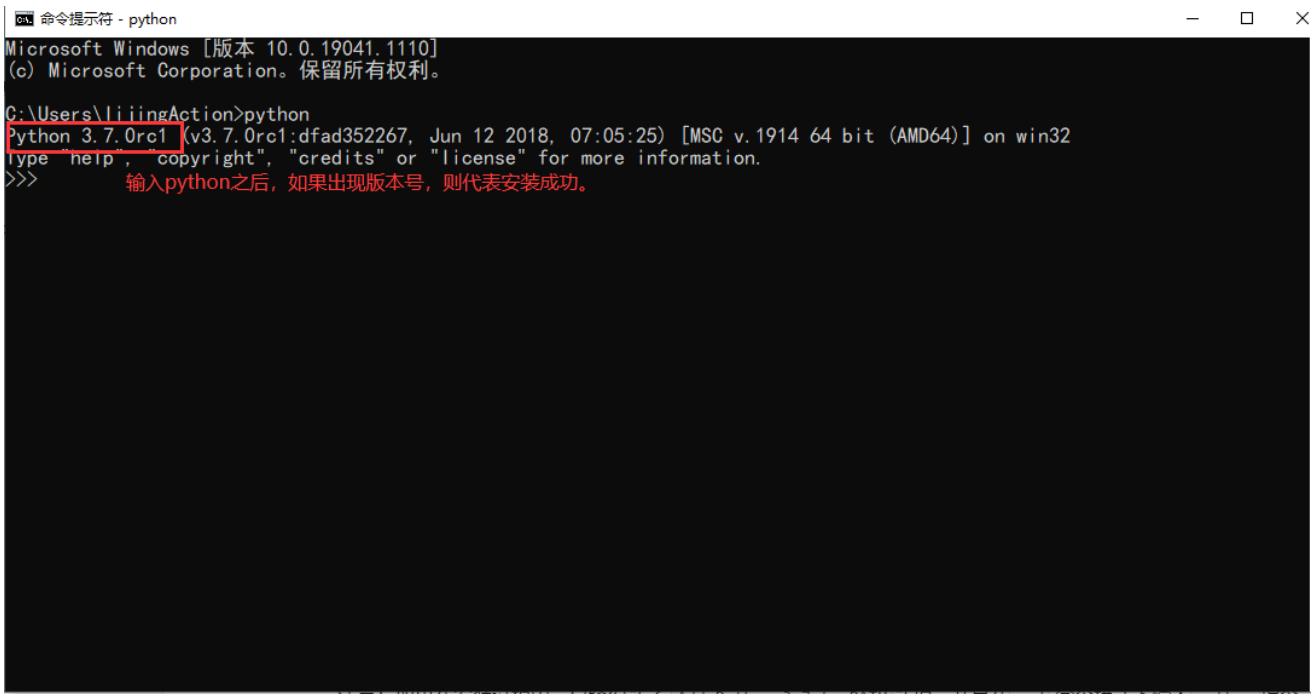

Python 3.7.0rc1 (64-bit) Setup
Optional Features
Documentation
加载文档文件
Installs the Python documentation file.
pip
pip指令，用于安装python包
Installs pip, which can download and install other Python packages.
tcl/tk and IDLE
图形化界面显示安装
Installs tkinter and the IDLE development environment.
Python test suite
测试工具
Installs the standard library test suite.
py launcher
for all users (requires elevation)
py文件加载器
Installs the global 'py' launcher to make it easier to start Python.
全部选择**
python
for
windows
Back
Next
Cancel



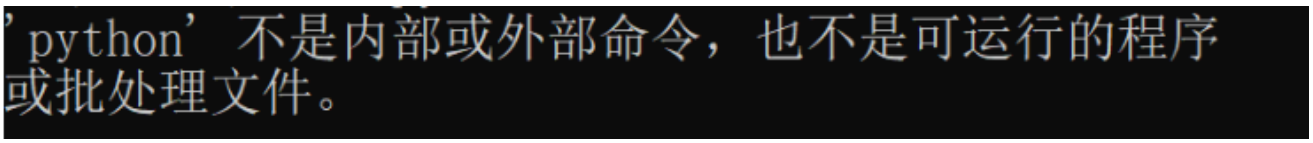
- 安装完成

3. 测试是否安装成功

- 点击电脑左下角开始按钮，输入`cmd`进入到windows的命令行模式。
- 在命令行中输入Python,正确显示Python版本，即表示Python安装成功



- 如果在命令行中输入python出现如下错误

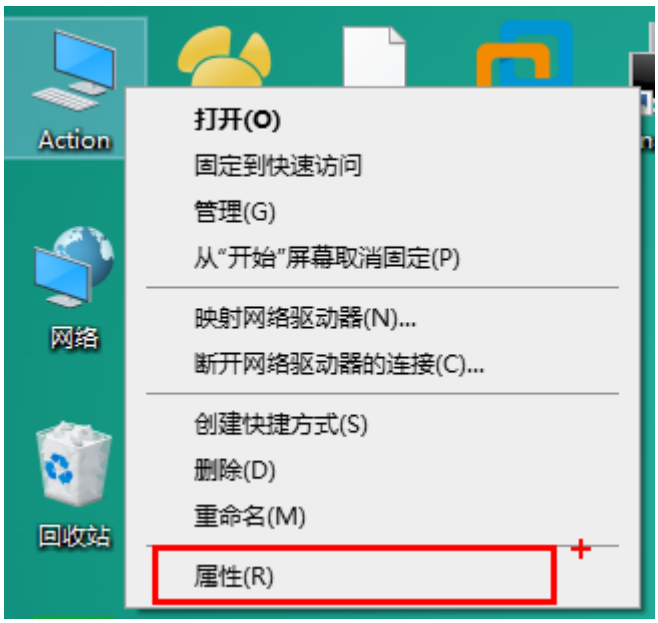


可能是因为在安装Python的过程中没有勾选 `Add Python 3.7 to PATH` 选项，此时需要手动对Python进行配置。

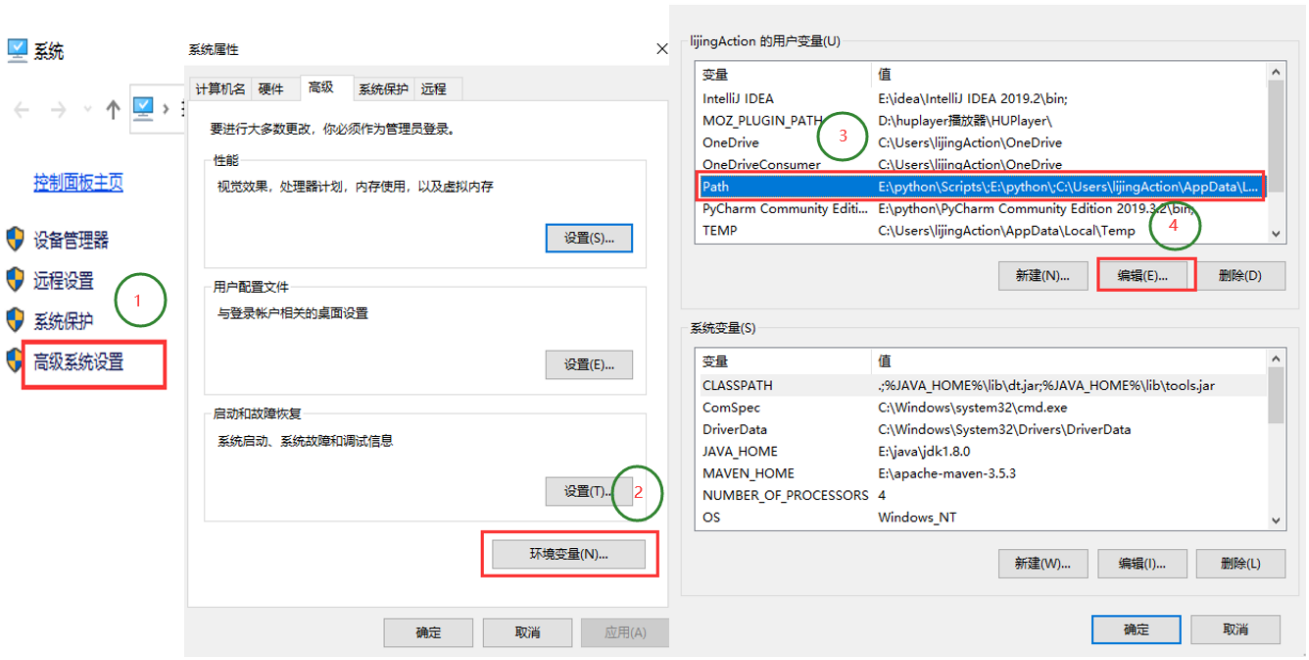
4. 手动配置Python

注意：如果在安装过程中，已经勾选了 `Add Python 3.7 to PATH` 选项，并且在 `cmd` 命令模式下输入 `python` 指令不报错，就不需要再手动的配置Python.

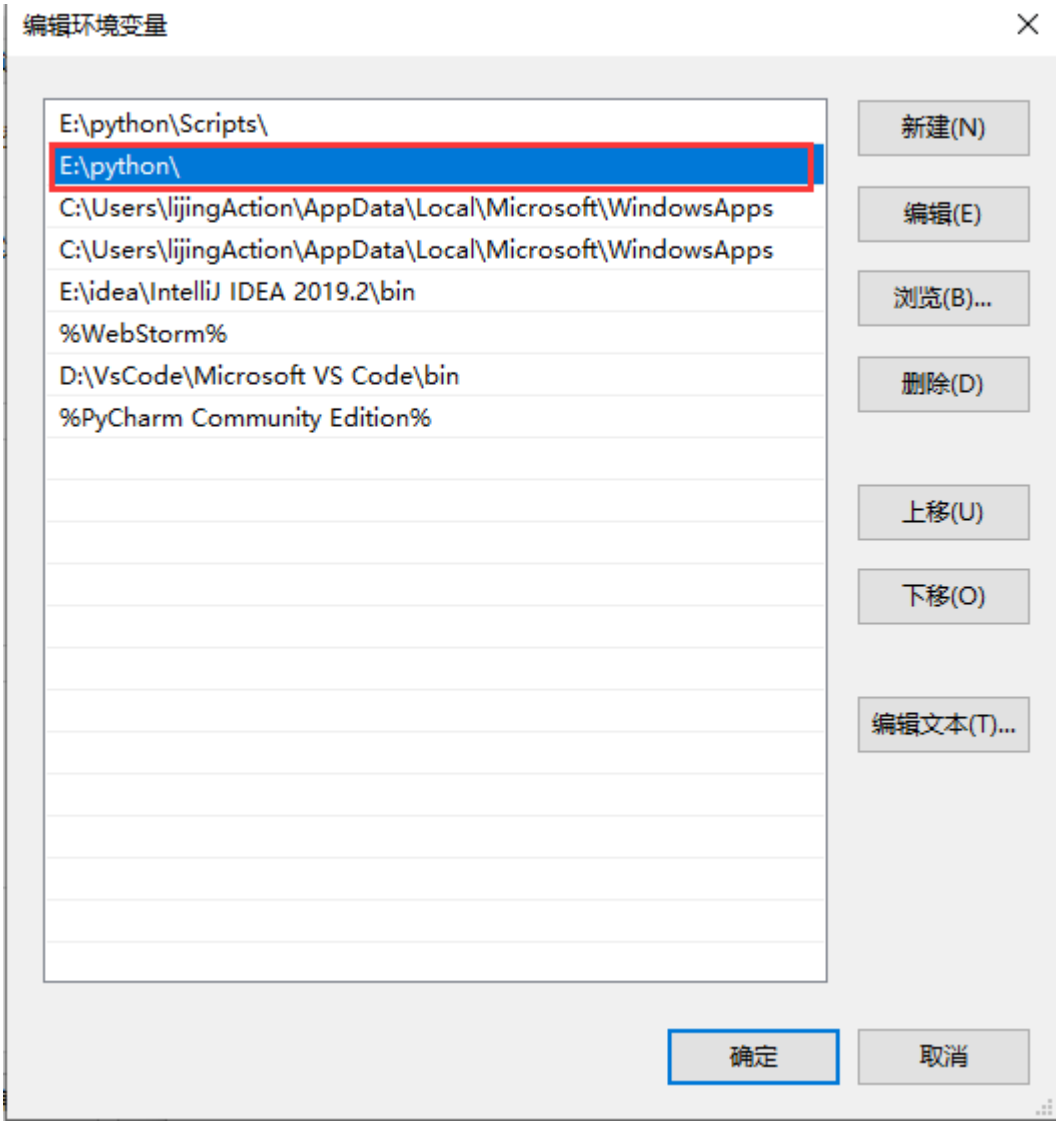
- 右键 `此电脑` --> 选择 `属性`



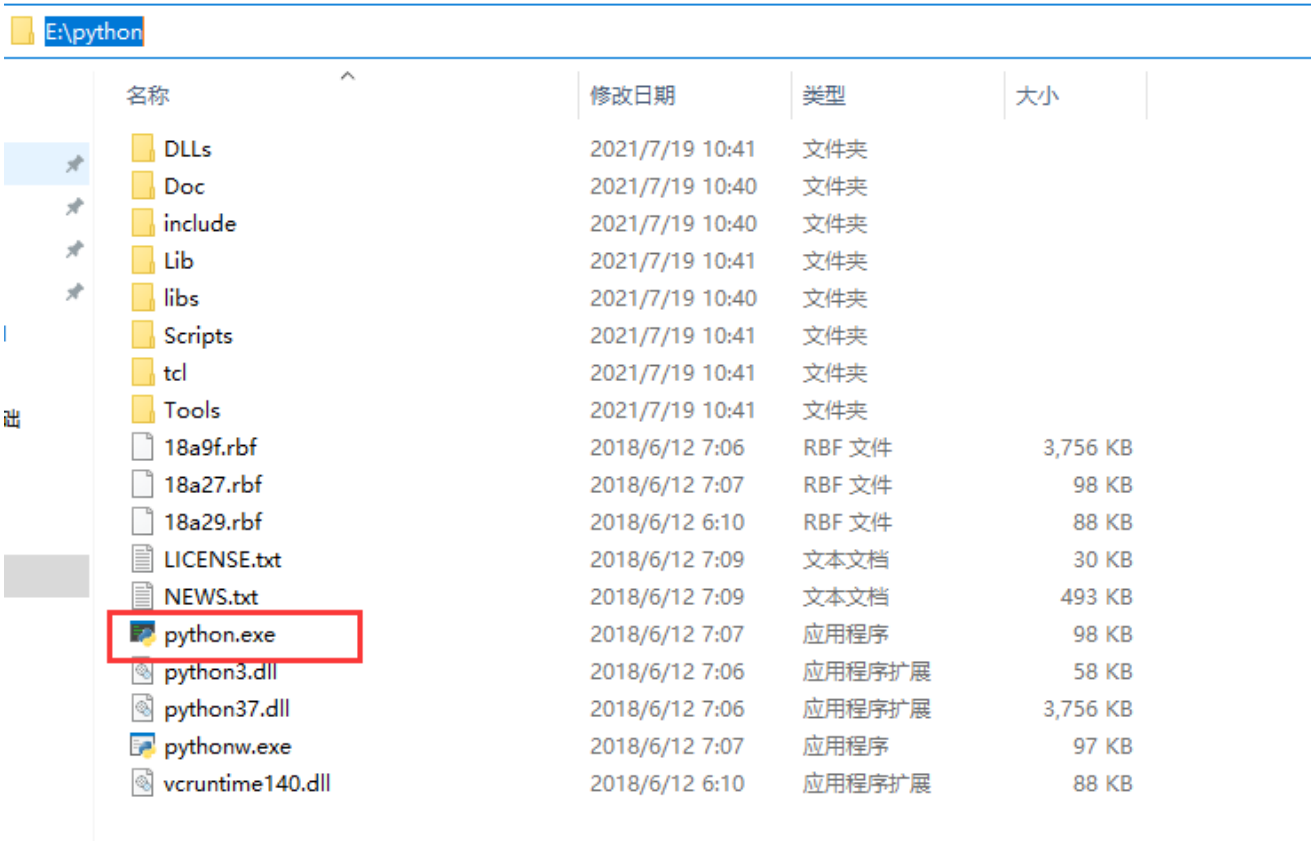
- 选择 `高级系统设置` --> `环境变量` -->找到并且双击 `Path`



- 双击 `Path` ,在弹框里点击新建，找到Python的安装目录，把路径添加进去



- 这里新添加的路径 `E:\python` 是Python安装好以后， `Python.exe` 这个可执行文件所在的目录。



二、pip的使用

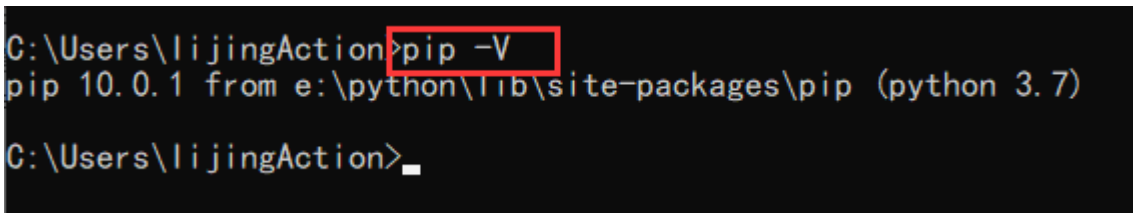
> pip 是一个现代的，通用的Python包管理工具。提供了对 Python 包的查找、下载、安装、卸载的功能，便于我们对Python的资源包进行管理。

1. 安装

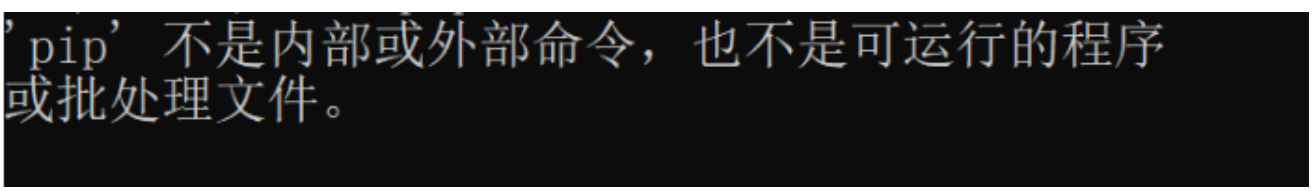
在安装Python时，会自动下载并且安装pip.

2. 配置

- 在windows命令行里，输入 `pip -V` 可以查看pip的版本。

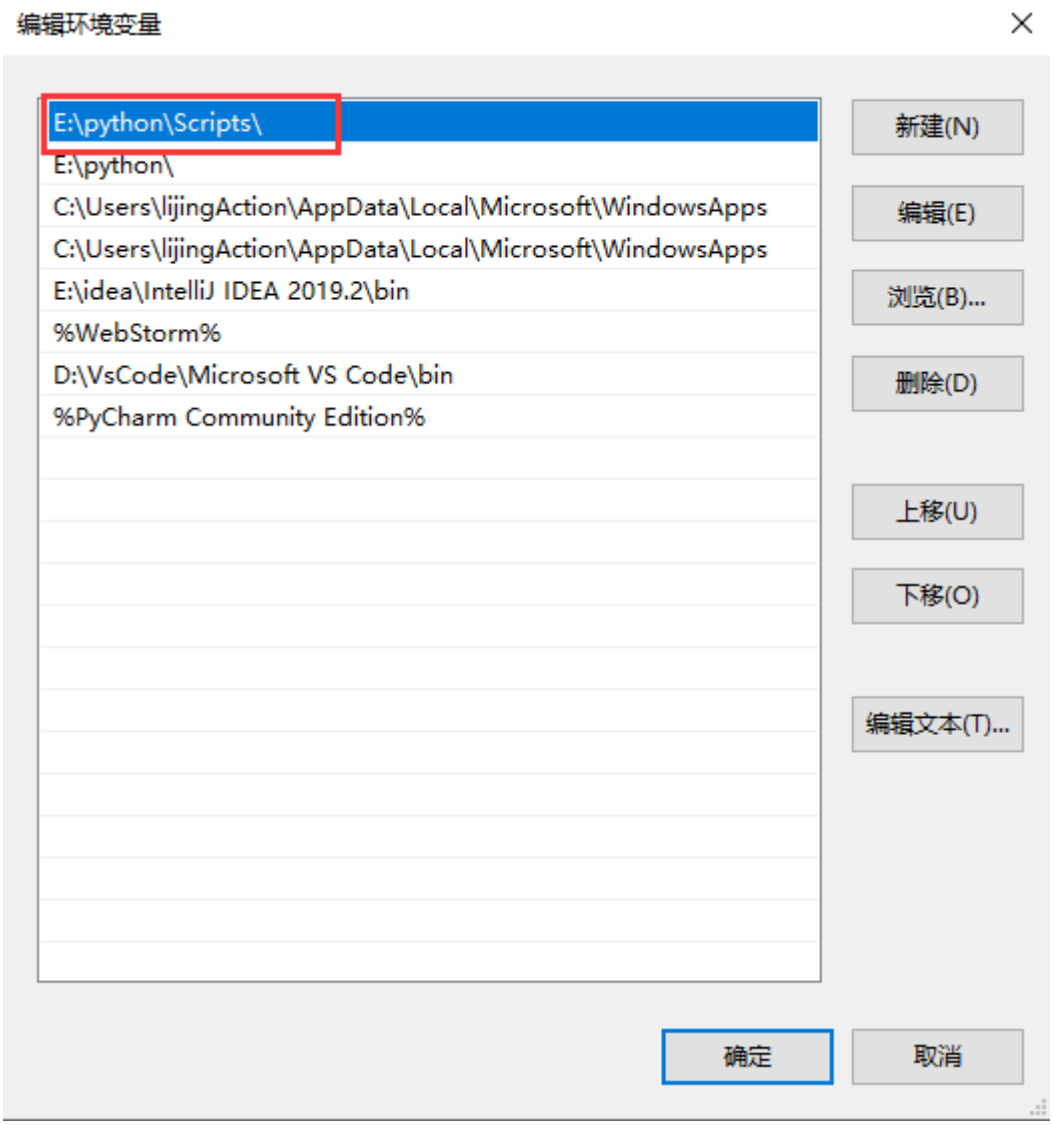


- 如果在命令行里，运行 `pip -V`,出现如下提示:

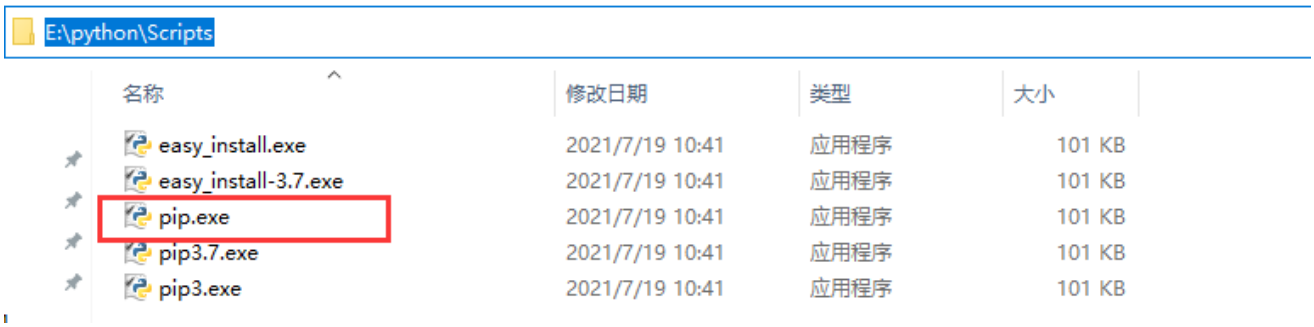


可能是因为在安装python的过程中未勾选 `Add Python 3.7 to PATH` 选项，需要手动的配置pip的环境变量。

- 右键`此电脑`-->`环境变量`-->找到并且双击`Path`-->在弹窗里点击新建-->找到pip的安装目录，把路径添加进去。



- 这里新添加的路径`E:\python\Scripts`是Python安装好以后，`pip.exe`这个可执行文件所在的目录。



3. 使用pip管理Python包

- pip install `<包名>` 安装指定的包
- pip uninstall `<包名>` 删除指定的包
- pip list 显示已经安装的包
- pip freeze 显示已经安装的包，并且以指定的格式显示

4. 修改pip下载源

运行pip install 命令会从网站上下载指定的python包，默认是从 `https://files.pythonhosted.org/` 网站上下载。这是个国外的网站，遇到网络情况不好的时候，可能会下载失败，我们可以通过命令，修改pip现在软件时的源。 格式:

`pip install 包名 -i 国内源地址`

示例: `pip install ipython -i https://pypi.mirrors.ustc.edu.cn/simple/` 就是从中国科技大学(ustc)的服务器上下载requests(基于python的第三方web框架)

国内常用的pip下载源列表:

- 阿里云 http://mirrors.aliyun.com/pypi/simple/
- 中国科技大学 https://pypi.mirrors.ustc.edu.cn/simple/
- 豆瓣(douban) http://pypi.douban.com/simple/
- 清华大学 https://pypi.tuna.tsinghua.edu.cn/simple/
- 中国科学技术大学 http://pypi.mirrors.ustc.edu.cn/simple/

三、运行Python程序

1. 终端运行

1. 直接在python解释器中书写代码

```
C:\Users\lijingAction>python
Python 3.7.0rc1 (v3.7.0rc1:dfad352267, Jun 12 2018, 07:05:25) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('苍茫的天涯是我的爱')
苍茫的天涯是我的爱
>>>
```

- 退出python环境
  - exit()
  - ctrl + z ==>enter

2. 使用ipython解释器编写代码

   使用pip命令，可以快速的安装IPython.

   ```
   pip install ipython
   ```

```
C:\Users\lijingAction>ipython
Python 3.7.0rc1 (v3.7.0rc1:dfad352267, Jun 12 2018, 07:05:25) [MSC v.1914
64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.25.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('绵绵的青山脚下花正开')
绵绵的青山脚下花正开

In [2]:
```

2. 运行python文件

使用python指令运行后缀为.py的python文件

```
C:\Users\lijingAction>python C:\Users\lijingAction\Desktop\test.py
This is my first python demo

C:\Users\lijingAction>_
```

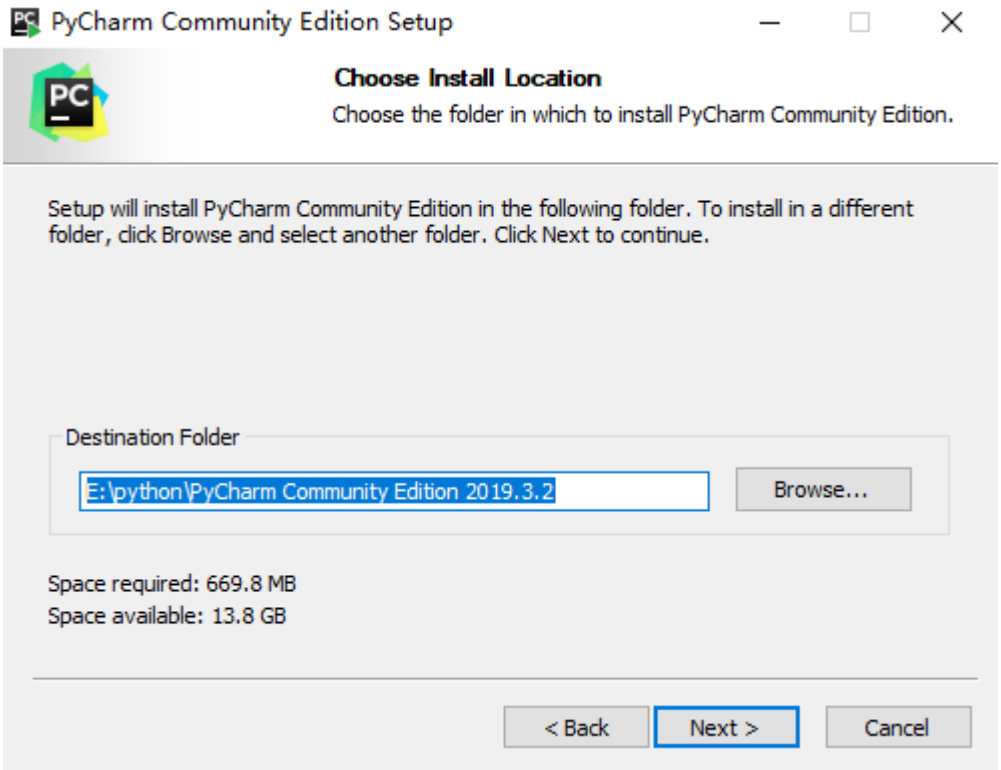
3. Pycharm

尽管上面介绍的方法已经能够提高我们的编码速度，但是仍然无法应对我们开发中更加复杂的要求。一般情况下，我们都需要借助工具来辅助我们快速的搭建环境，编写代码以及运行程序。

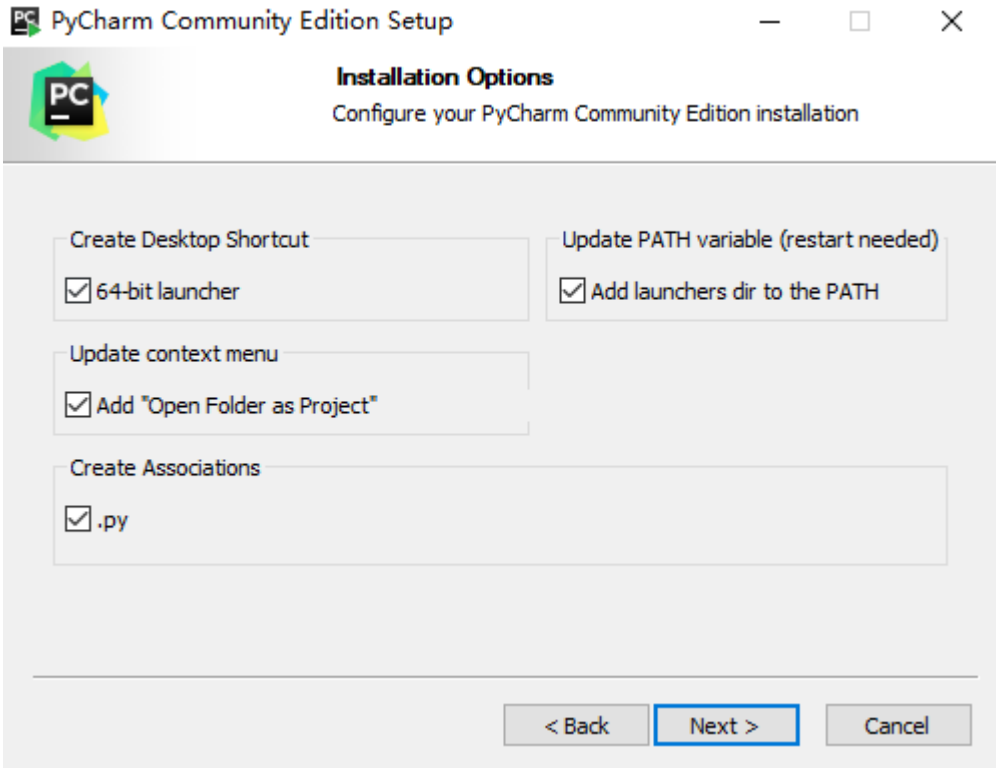
- IDE的概念
  IDE(Integrated Development Environment)又被称为**集成开发环境**。说白了，就是有一款图形化界面的软件，它集成了编辑代码，编译代码，分析代码，执行代码以及调试代码等功能。在我们Python开发中，最常用的IDE是Pycharm.

pycharm由捷克公司JetBrains开发的一款IDE,提供代码分析、图形化调试器，集成测试器、集成版本控制系统等，主要用来编写Python代码。
下载地址:http://www.jetbrains.com/pycharm/download

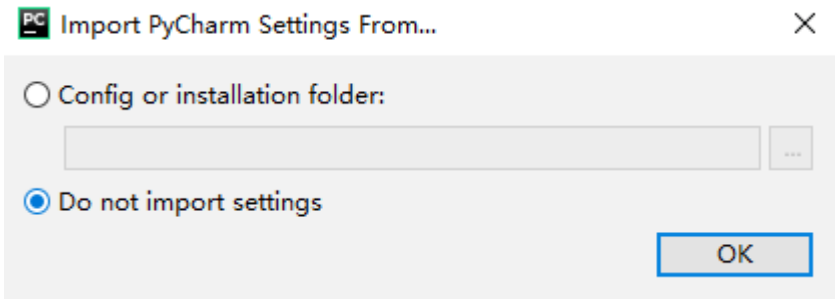
- Pycharm的安装
  - 双击安装文件
  - 自定义安装路径（可以不用设置



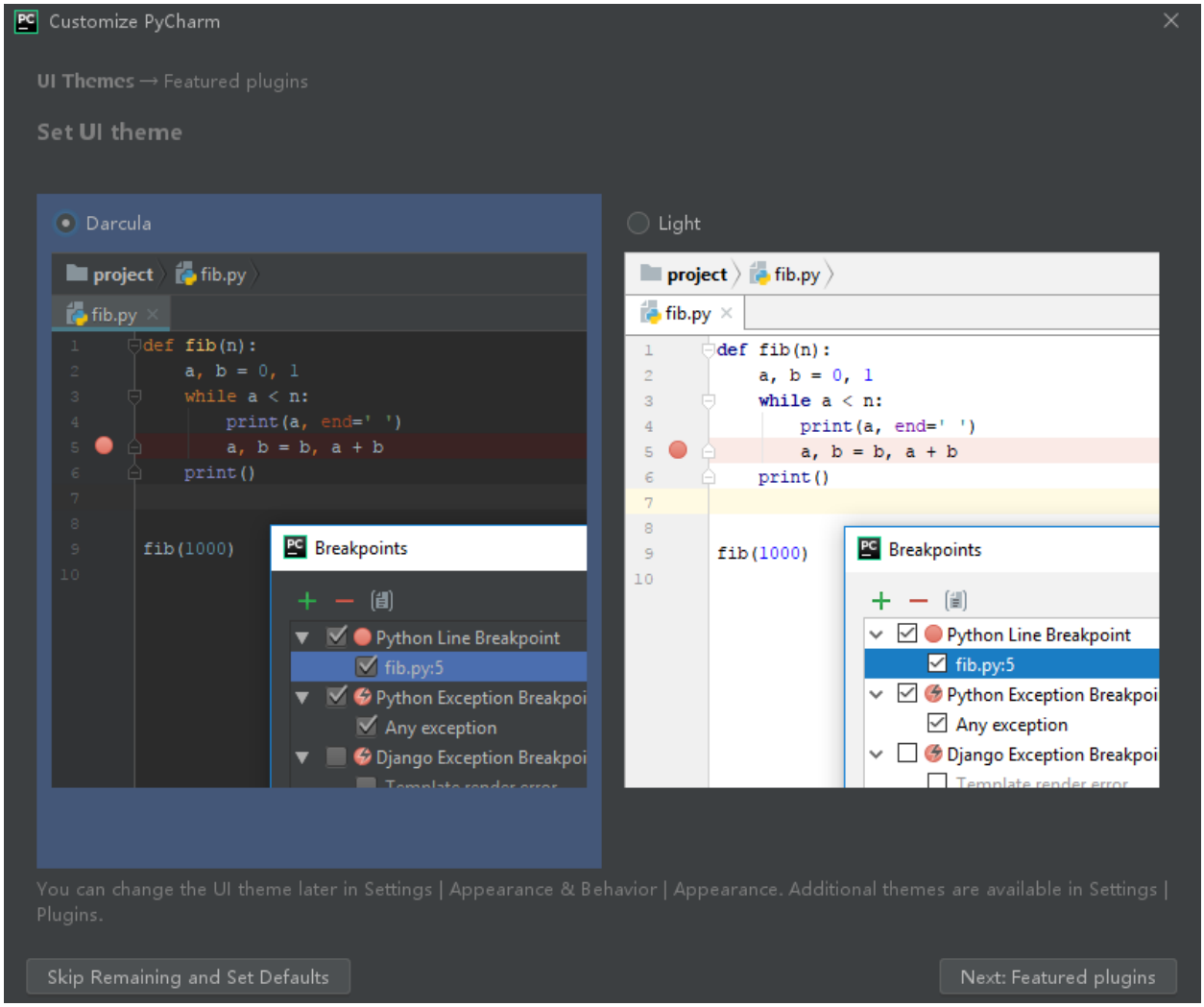
  - 编辑设置（全部选中）



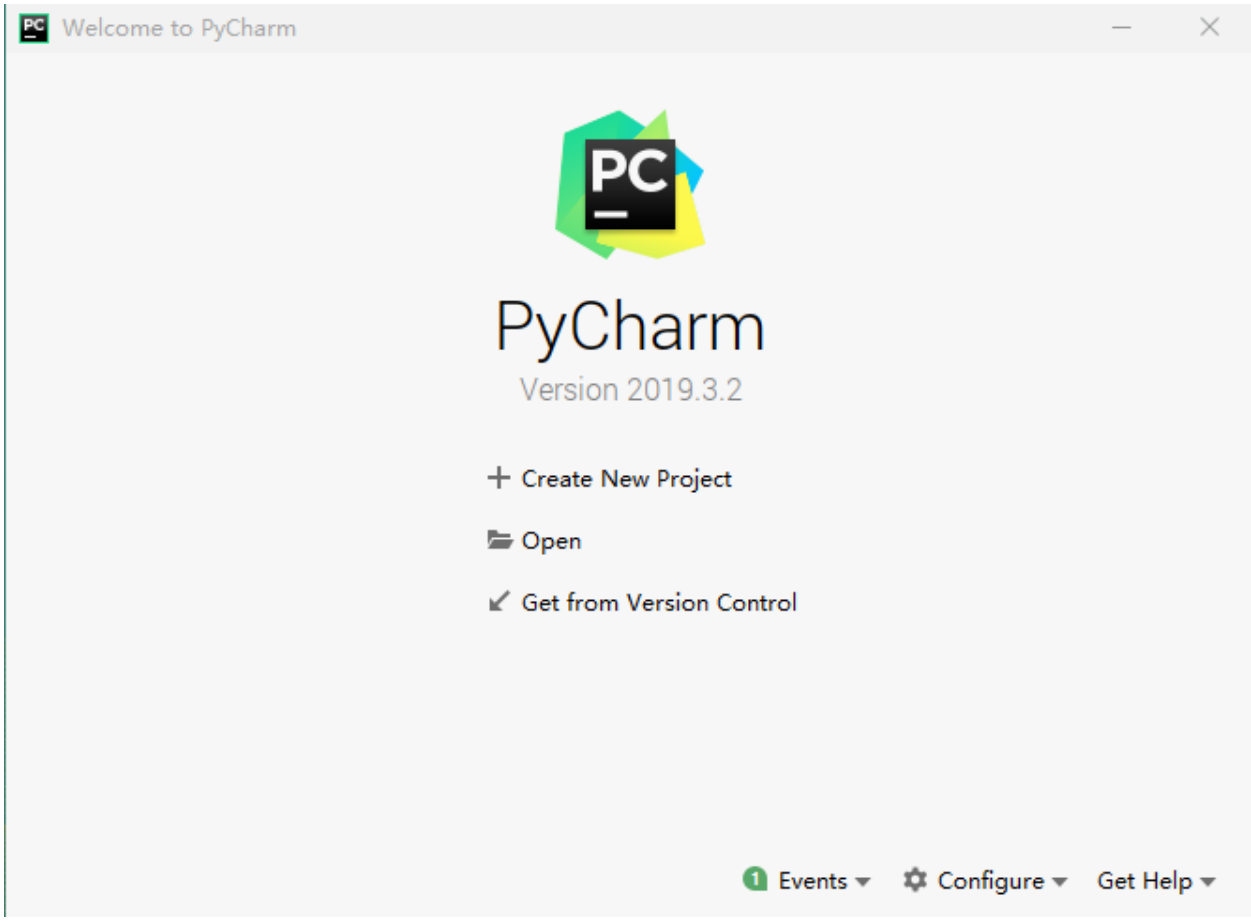
- 安装完成后双击



- 设置主题

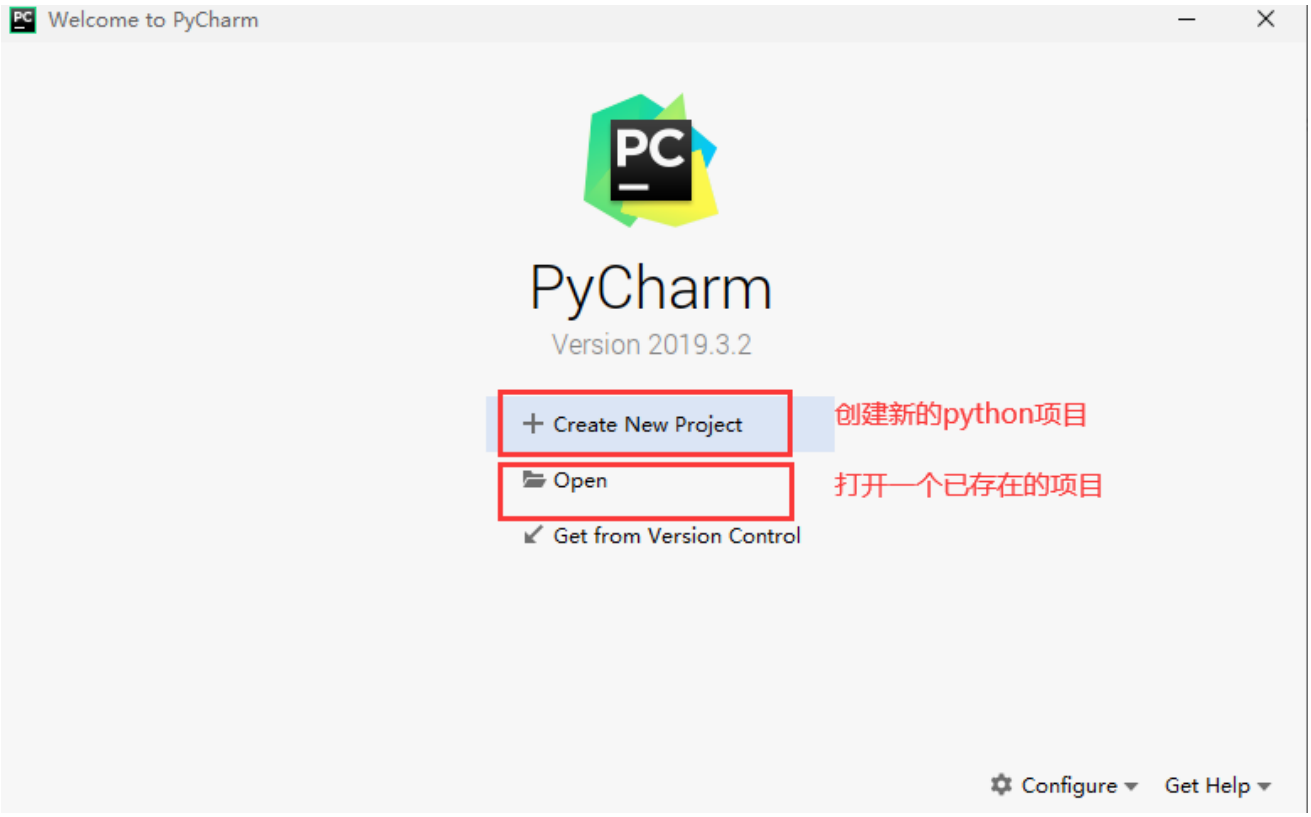

Customize PyCharm
UI Themes → Featured plugins
Set UI theme
Darcula
Light
project
fib.py
def fib(n):
a, b = 0, 1
while a < n:
print(a, end=' ')
a, b = b, a + b
print()
fib(1000)
Breakpoints
Python Line Breakpoint
fib.py:5
Python Exception Breakpoi
Any exception
Django Exception Breakpoi
You can change the UI theme later in Settings | Appearance & Behavior | Appearance. Additional themes are available in Settings | Plugins.
Skip Remaining and Set Defaults
Next: Featured plugins

- 启动软件

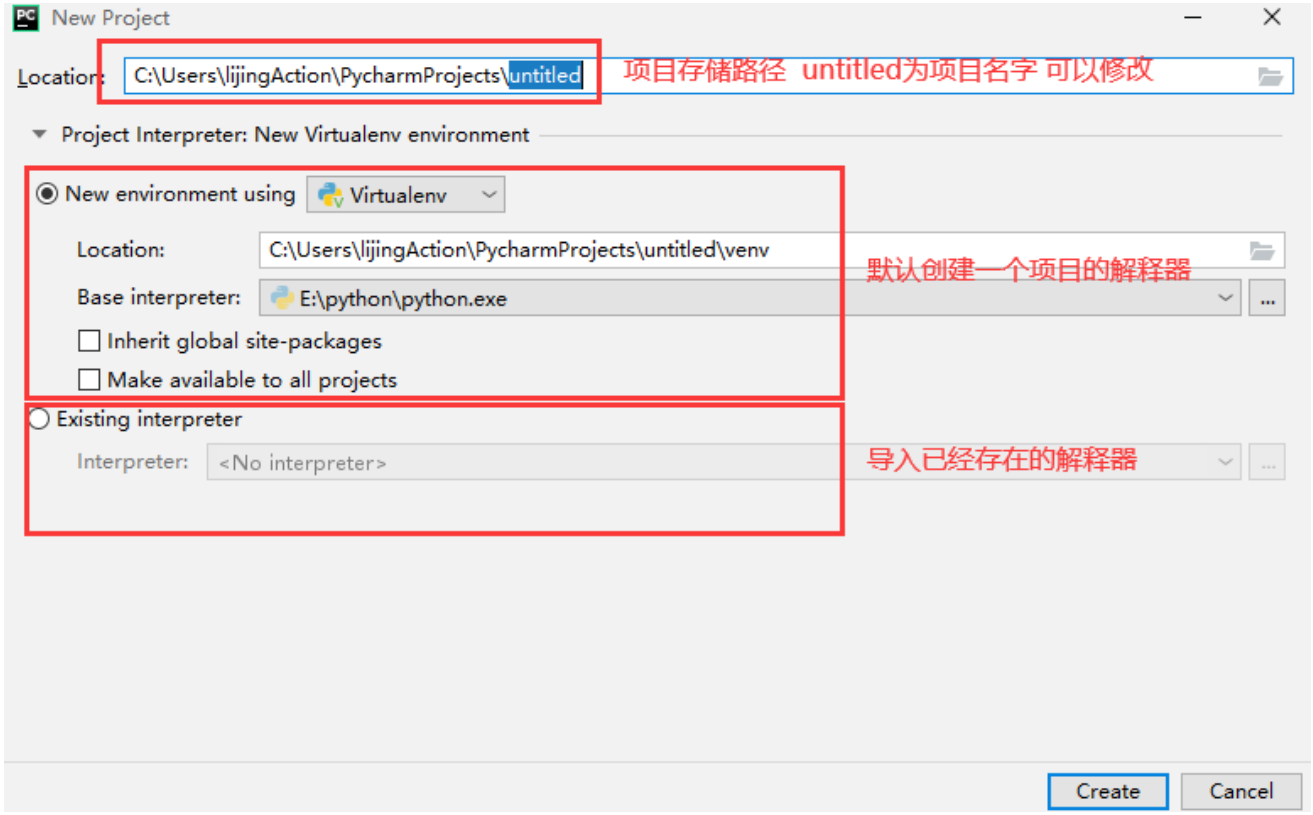


4. Pycharm的使用介绍

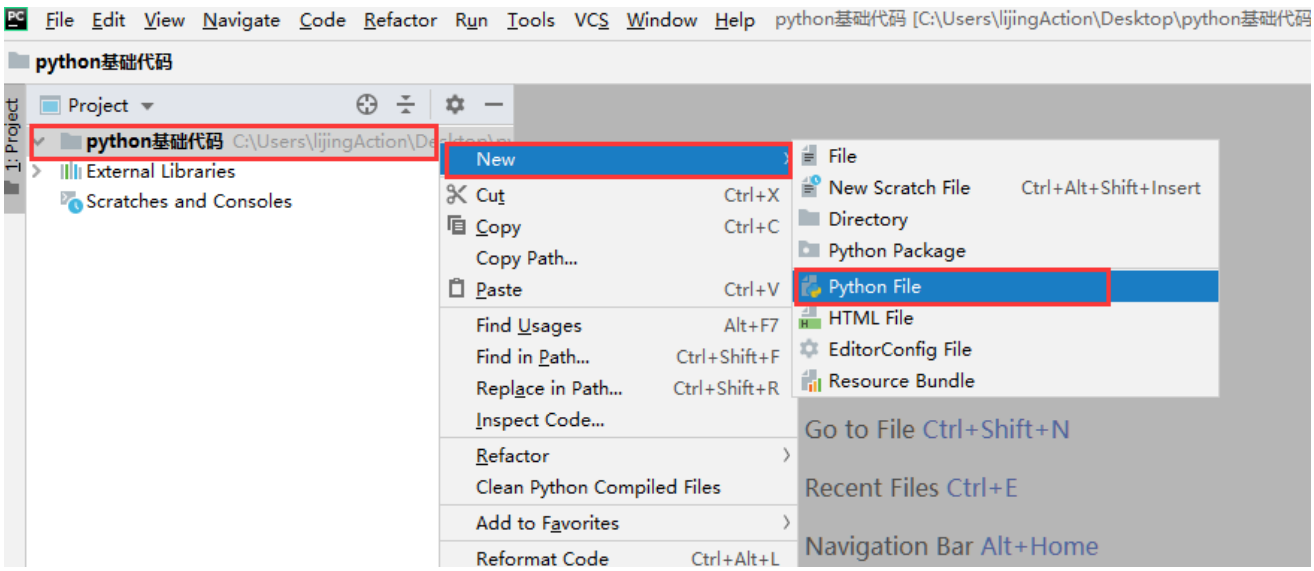
- 运行Pycharm,选择`Create New Project`,创建一个新的Python工程。



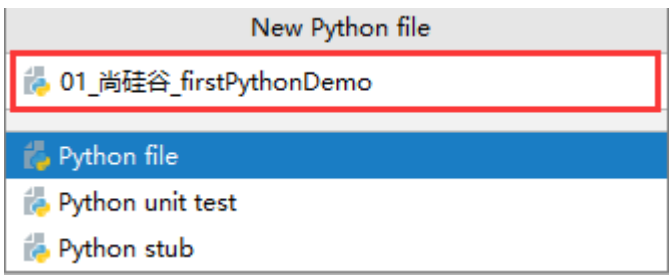
- 选择'Pure Python'创建一个新的纯Python工程项目，`Location`表示该项目的保存路径，`Interpreter`用来指定Python解释器的版本。



- 右击项目，选择`New`，再选择`Python File`



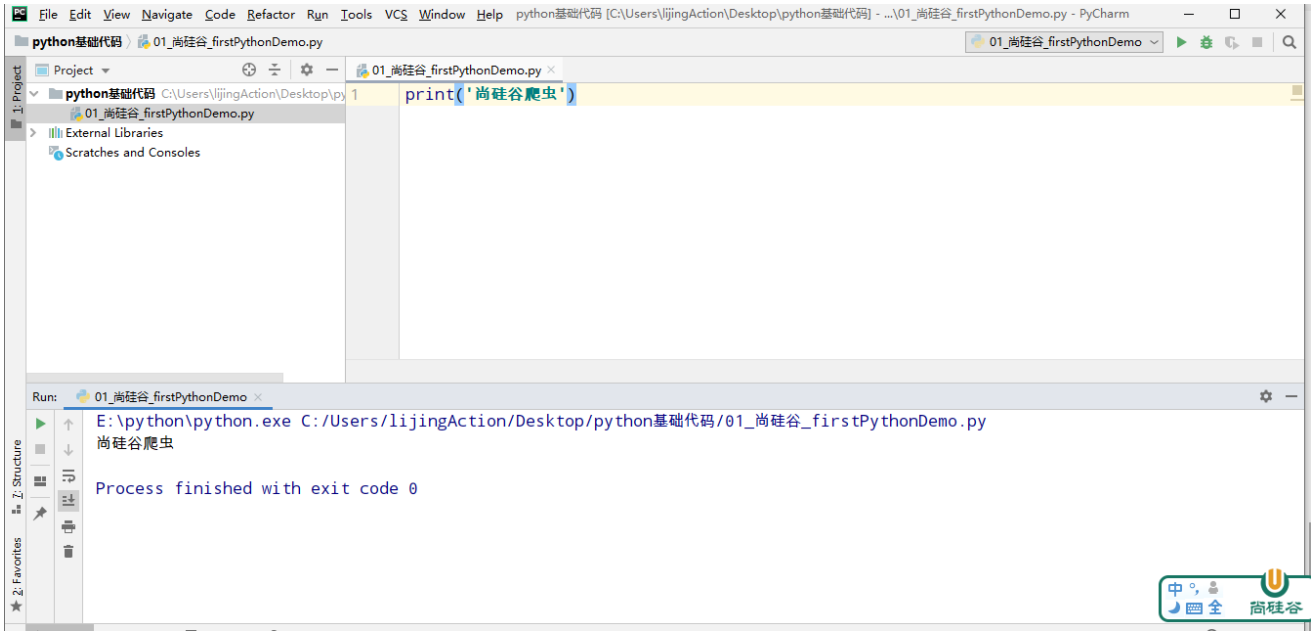
- 在弹出的对话框中输入的文件名`HelloPython`，点击OK，表示创建一个Python程序的文本文件，文本文件后缀名默认.py

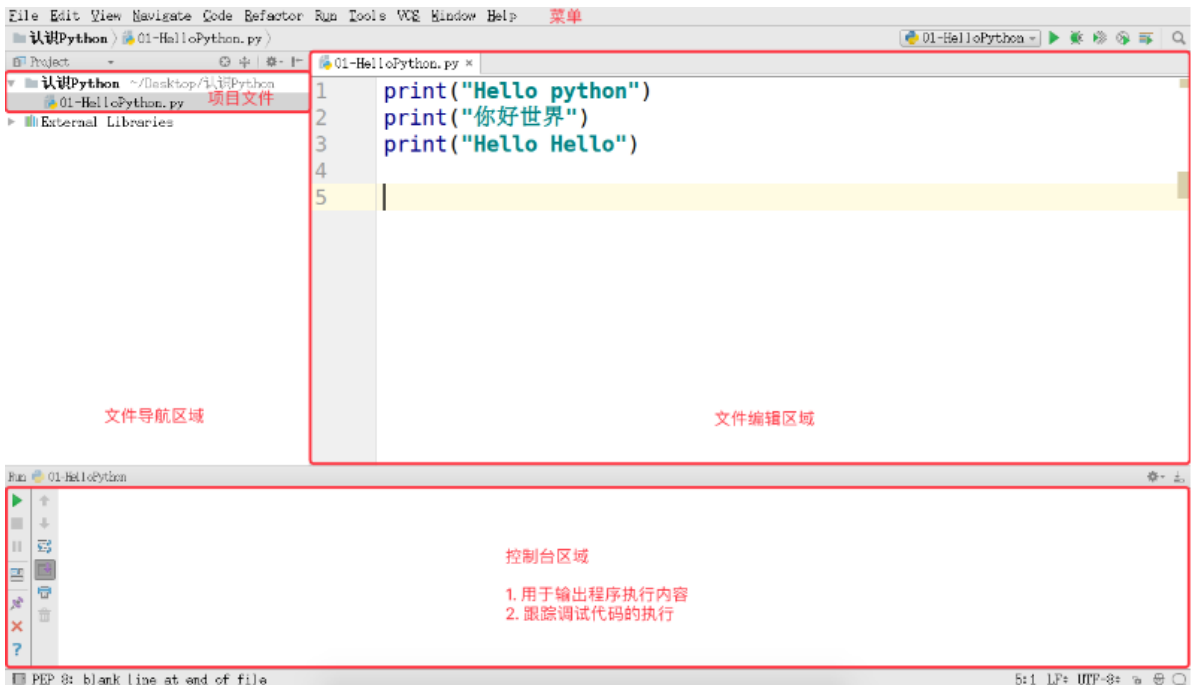


- 在新建的`01_尚硅谷_firstPythonDemo.py`文件里，输入以下代码，并在空白处右键选择`Run`运行，表示输出一段`尚硅谷爬虫`字符串。

```
print('尚硅谷爬虫')
```

- 运行成功后，Pycharm Console窗口将显示我们的输出结果。



- 页面布局介绍
  - 

- **文件导航区域**能够 **浏览 / 定位 / 打开** 项目文件
- **文件编辑区域** 能够 **编辑** 当前打开的文件
- **控制台区域** 能够：
  - 输出程序执行内容

- 跟踪调试代码的执行

四、Python

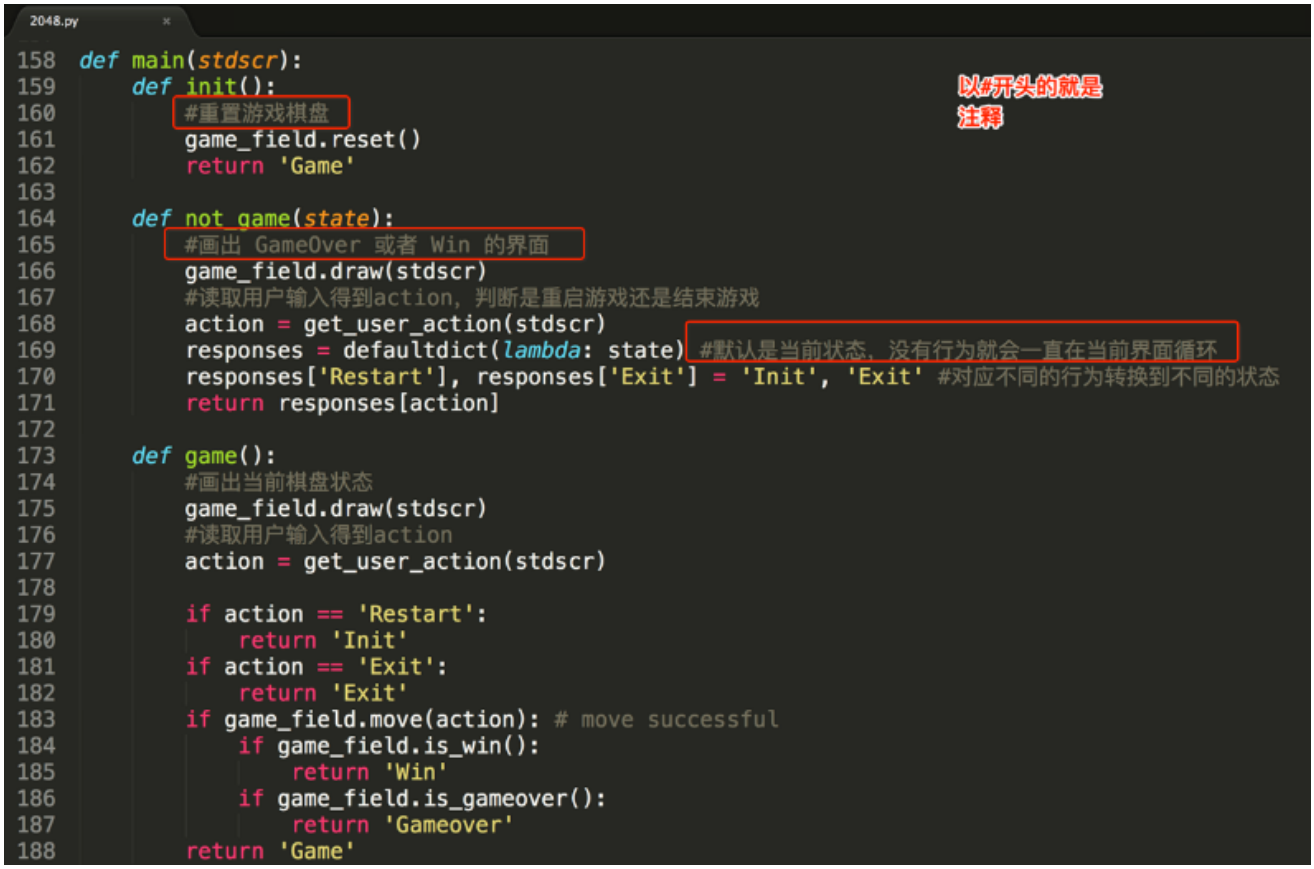
1.1 注释

1.1 注释介绍

在我们工作编码的过程中，如果一段代码的逻辑比较复杂，不是特别容易理解，可以适当的添加注释，以辅助自己或者其他编码人员解读代码。

没注释的代码

有注释的代码



注意：注释是给程序员看的，为了让程序员方便阅读代码，解释器会忽略注释。使用自己熟悉的语言，适当的对代码进行注释说明是一种良好的编码习惯。

1.2 注释的分类

在Python中支持单行注释和多行注释。

单行注释

以#开头，#右边的所有东西当做说明，而不是真正要执行的程序，起辅助说明作用。

```
# #开头右边的都是注释，解析器会忽略注释
print('hello world')  #我的作用是在控制台输出hello world
```

多行注释

以`'''`开始，并以`'''`结束，我们称之为多行注释。

```
'''
                            _ooOoo_
                           o8888888o
                           88" . "88
                           (| -_- |)
                           O\  =  /O
                        ____/`---'\____
                      .'  \\|     |//  `.
                     /  \\|||  :  |||//  \
                    /  _|||||  -:-  |||||-  \
                    |   | \\\  -  /// |   |
```

```
                    | \_|  ''\---/''  |   |
                    \  .-\__  `-`  ___/-. /
                  ___`. .'  /--.--\  `. . __
               ."" '<  `.___\_<|>_/___.'  >'"".
              | | :  `- \`.;`\ _ /`;.`/ - ` : | |
              \  \ `-.   \_ __\ /__ _/   .-` /  /
        ======`-.____`-.___\_____/___.-`____.-'======
                           `=---='
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                    佛祖保佑          永无BUG
          佛曰:
                  写字楼里写字间，写字间里程序员；
                  程序人员写程序，又拿程序换酒钱。
                  酒醒只在网上坐，酒醉还来网下眠；
                  酒醉酒醒日复日，网上网下年复年。
                  但愿老死电脑间，不愿鞠躬老板前；
                  奔驰宝马贵者趣，公交自行程序员。
                  别人笑我忒疯癫，我笑自己命太贱；
                  不见满街漂亮妹，哪个归得程序员？
'''
```

2. 变量以及数据类型

2.1 变量的定义

思考下列代码有什么问题?

```
print("今天天气真好")
print("今天天气真好")
print("今天天气真好")
print("今天天气真好")
print("今天天气真好")
print("今天天气真好")
print("今天天气真好")
print("今天天气真好")
print("今天天气真好")
```

> 不使用变量打印九次 "今天天气真好",如果需要变成打印 "今天天气不好" 需要修改九行代码

对于重复使用，并且经常需要修改的数据，可以定义为变量，来提高编程效率。

定义变量的语法为: `变量名 = 变量值`。(这里的 = 作用是赋值。)

定义变量后可以使用变量名来访问变量值。

```
# 定义一个变量表示这个字符串。如果需要修改内容，只需要修改变量对应的值即可
weather = "今天天气真好"
print(weather)  # 注意，变量名不需要使用引号包裹
print(weather)
print(weather)
```

说明:

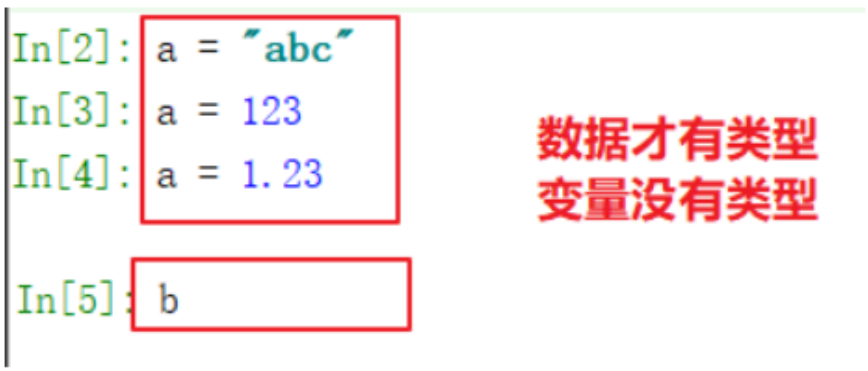
- 变量即是可以变化的量，可以随时进行修改。
- 程序就是用来处理数据的，而变量就是用来存储数据的。

2.2 变量的类型

程序中: 在 **Python 里为了应对不同的业务需求，也把数据分为不同的类型。** 如下图所示:

2.3 查看数据类型

- 在python中，只要定义了一个变量，而且它有数据，那么它的类型就已经确定了，不需要咱们开发者主动的去说明它的类型，系统会自动辨别。也就是说在使用的时候 **"变量没有类型，数据才有类型"**。
- 比如下面的示例里，a 的类型可以根据数据来确认,但是我们没法预测变量 b 的类型。



- 如果临时想要查看一个变量存储的数据类型，可以使用 **type(变量的名字)**，来查看变量存储的数据类型。

3. 标识符和关键字

计算机编程语言中，标识符是用户编程时使用的名字，用于给变量、常量、函数、语句块等命名，以建立起名称与使用之间的关系。

1. 标识符由字母、下划线和数字组成，且数字不能开头。
2. 严格区分大小写。
3. 不能使用关键字。

> 思考：下面的标识符哪些是正确的，哪些不正确为什么

```
fromNo12
from#12
my_Boolean
my-Boolean
Obj2
2ndObj
myInt
test1
Mike2jack
My_tExt
_test
```

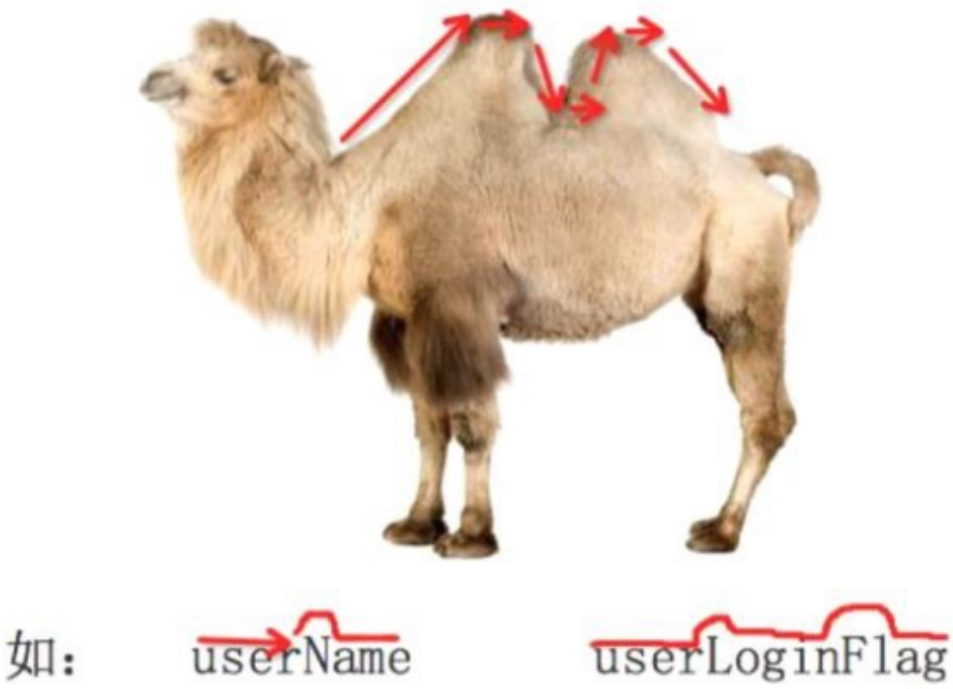
3.1 命名规范

- 标识符命名要做到顾名思义。

  > 起一个有意义的名字，尽量做到看一眼就知道是什么意思(提高代码可 读性) 比如: 名字 就定义为 name，定义学生 用 student

  ```
  a = "zhangsan"  # bad
  name = "zhangsan"  # good
  b = 23   # bad
  age = 23 # good
  ```

- 遵守一定的命名规范。
  - 驼峰命名法，又分为大驼峰命名法和小驼峰命名法。



- 小驼峰式命名法（lower camel case）： 第一个单词以小写字母开始；第二个单词的首字母大写，例如：myName、aDog
- 大驼峰式命名法（upper camel case）： 每一个单字的首字母都采用大写字母，例如：FirstName、LastName.

- 还有一种命名法是用下划线“_”来连接所有的单词，比如send_buf.

  Python的命令规则遵循PEP8标准

3.2 关键字

- 关键字的概念
  一些具有特殊功能的标识符，这就是所谓的关键字。
  关键字，已经被python官方使用了，所以不允许开发者自己定义和关键字相同名字的标识符。
- 关键字

```
False      None     True     and      as       assert   break    class
continue   def      del      elif     else     except   finally  for
from       global   if       import   in       is       lambda   nonlocal
not        or       pass     raise    return   try      while    with
yield
```

关键字的学习以及使用，咱们会在后面的课程中依次一一进行学习。

4. 类型转换

函数	说明
int(x)	将x转换为一个整数
float(x)	将x转换为一个浮点数
str(x)	将对象 x 转换为字符串
bool(x)	将对象x转换成为布尔值

- 转换成为整数

```
print(int("123"))  # 123 将字符串转换成为整数

print(int(123.78))  # 123 将浮点数转换成为整数

print(int(True))  # 1 布尔值True转换成为整数是 1
print(int(False)) # 0 布尔值False转换成为整数是 0

# 以下两种情况将会转换失败
'''
123.456 和 12ab 字符串，都包含非法字符，不能被转换成为整数，会报错
print(int("123.456"))
print(int("12ab"))
'''
```

- 转换成为浮点数

```
f1 = float("12.34")
print(f1)   # 12.34
print(type(f1)) # float 将字符串的 "12.34" 转换成为浮点数 12.34

f2 = float(23)
print(f2)  # 23.0
print(type(f2)) # float 将整数转换成为了浮点数
```

- 转换成为字符串

```
str1 = str(45)
str2 = str(34.56)
str3 = str(True)
print(type(str1),type(str2),type(str3))
```

- 转换成为布尔值

```
print(bool(''))
print(bool(""))
print(bool(0))
print(bool({}))
print(bool([]))
print(bool(()))
```

5. 运算符

5.1 算数运算符

下面以a=10 ,b=20为例进行计算

运算符	描述	实例
+	加	两个对象相加 a + b 输出结果 30
-	减	得到负数或是一个数减去另一个数 a - b 输出结果 -10
*	乘	两个数相乘或是返回一个被重复若干次的字符串 a * b 输出结果 200
/	除	b / a 输出结果 2
//	取整除	返回商的整数部分 9//2 输出结果 4 , 9.0//2.0 输出结果 4.0
%	取余	返回除法的余数 b % a 输出结果 0
**	指数	a**b 为10的20次方
()	小括号	提高运算优先级，比如: (1+2) * 3

> 注意：混合运算时，优先级顺序为： `**` 高于 `* / % //` 高于 `+ -` ，为了避免歧义，建议使用 () 来处理运算符优先级。 并且，不同类型的数字在进行混合运算时，整数将会转换成浮点数进行运算。

```
>>> 10 + 5.5 * 2
21.0
>>> (10 + 5.5) * 2
31.0
```

算数运算符在字符串里的使用

- 如果是两个字符串做加法运算，会直接把这两个字符串拼接成一个字符串。

```
In [1]: str1 ='hello'

In [2]: str2 = 'world'

In [3]: str1+str2
Out[3]: 'helloworld'

In [4]:
```

- 如果是数字和字符串做加法运算，会直接报错。

```
In [1]: str1 = 'hello'

In [2]: a = 2

In [3]: a+str1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-993727a2aa69> in <module>
----> 1 a+str1

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- 如果是数字和字符串做乘法运算，会将这个字符串重复多次。

```
In [4]: str1 = 'hello'

In [5]: str1*10
Out[5]: 'hellohellohellohellohellohellohellohellohellohello'
```

5.2 赋值运算符

运算符	描述	实例
=	赋值运算符	把 = 号右边的结果 赋给 左边的变量，如 num = 1 + 2 * 3，结果num的值为7

```
# 单个变量赋值
>>> num = 10
>>> num
10

# 同时为多个变量赋值(使用等号连接)
>>> a = b = 4
>>> a
4
>>> b
4
>>>

# 多个变量赋值(使用逗号分隔)
>>> num1, f1, str1 = 100, 3.14, "hello"
>>> num1
100
>>> f1
3.14
>>> str1
"hello"
```

5.3 复合赋值运算符

运算符	描述	实例
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
//=	取整除赋值运算符	c //= a 等效于 c = c // a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a

```
# 示例：+=
>>> a = 100
>>> a += 1  # 相当于执行 a = a + 1
>>> a
101

# 示例：*=
>>> a = 100
>>> a *= 2  # 相当于执行 a = a * 2
>>> a
200

# 示例：*=，运算时，符号右侧的表达式先计算出结果，再与左边变量的值运算
>>> a = 100
>>> a *= 1 + 2  # 相当于执行 a = a * (1+2)
>>> a
300
```

5.4 比较运算符

以下假设变量a为10，变量b为20：

运算符	描述	实例
==	等于:比较对象是否相等	(a == b) 返回 False
!=	不等于:比较两个对象是否不相等	(a != b) 返回 true
>	大于:返回x是否大于y	(a > b) 返回 False
>=	大于等于:返回x是否大于等于y	(a >= b) 返回 False
<	小于:返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价	(a < b) 返回 true
<=	小于等于:返回x是否小于等于y	(a <= b) 返回 true

5.5 逻辑运算符

运算符	逻辑表达式	描述	实例
and	x and y	只要有一个运算数是False，结果就是False; 只有所有的运算数都为True时，结果才是True 做取值运算时，取第一个为False的值，如果所有的值都为True,取最后一个值	True and True and False--> 结果为False True and True and True--> 结果为True
or	x or y	只要有一个运算数是True，结果就是True; 只有所有的运算数都为False时，结果才是False 做取值运算时，取第一个为True的值，如果所有的值都为False,取最后一个值	False or False or True-->结果为True False or False or False-->结果为False
not	not x	布尔"非" - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。	not True --> False

性能提升

面试题：一下代码的输出结果是什么，为什么会有这样的输出。

```
a = 34

a > 10 and print('hello world')
a < 10 and print('hello world')

a >10 or print('你好世界')
a <10 or print('你好世界')
```

思考：

1. 逻辑运算的短路问题
2. 逻辑与运算和逻辑或运算取值时为什么是那种规则。

6. 输入输出

6.1 输出

普通输出

python中变量的输出

```
print('吴亦凡火了')
```

格式化输出

比如有以下代码:

```
pirnt("我今年10岁")
pirnt("我今年11岁")
pirnt("我今年12岁")
```

- 想一想:

> 在输出年龄的时候，用了多次"我今年xx岁"，能否简化一下程序呢？？？

- 答:

> 字符串格式化

看如下代码:

```
age = 10
print("我今年%d岁" % age)

age += 1
print("我今年%d岁" % age)

age += 1
print("我今年%d岁" % age)
```

在程序中，看到了%这样的操作符，这就是Python中格式化输出。

```
age = 18
name = "红浪漫晶哥"
print("我的姓名是%s，年龄是%d" % (name, age))
```

6.2 输入

在Python中，获取键盘输入的数据的方法是采用 input 函数（至于什么是函数，咱们以后的章节中讲解），那么这个 input 怎么用呢?

看如下示例:

```
password = input("请输入密码:")
print('您刚刚输入的密码是:%s' % password)
```

运行结果:

注意:

- input()的小括号中放入的是提示信息，用来在获取数据之前给用户的一个简单提示
- input()在从键盘获取了数据以后，会存放到等号右边的变量中
- input()会把用户输入的任何值都作为字符串来对待

7. 流程控制语句

7.1 if判断语句

- if语句是用来进行判断的，其使用格式如下：

```
    if 要判断的条件:
        条件成立时，要做的事情
```

- demo1:

```
    age = 30
    if age >= 18:
        print("我已经成年了")
```

- 运行结果:

```
    我已经成年了
```

- demo2:

```
    age = 16
    if age >= 18:
        print("我已经成年了")
```

- 运行结果:

```
```

小总结:

- 以上2个demo仅仅是age变量的值不一样，导致结果却不同；能够看得出if判断语句的作用：就是当满足一定条件时才会执行代码块语句，否则就不执行代码块语句。
- 注意：代码的缩进为一个tab键，或者4个空格

练一练

要求：从键盘获取自己的年龄，判断是否大于或者等于18岁，如果满足就输出“哥，已成年，网吧可以去了”

> 1. 使用input从键盘中获取数据，并且存入到一个变量中
> 2. 使用if语句，来判断 age>=18是否成立

7.2 if else

> 想一想：在使用if的时候，它只能做到满足条件时要做的事情。那万一需要在不满足条件的时候，做某些事，该怎么办呢?
>
> 答：使用 if-else

if-else的使用格式

```
if 条件:
    满足条件时的操作
else:
    不满足条件时的操作
```

demo1

```
age = 18
if age >= 18:
    print("我可以去红浪漫了")
else:
    print("未成年，不允许去")
```

结果1：大于等于18的情况

```
我可以去红浪漫了
```

结果2：小于18的情况

```
未成年，不允许去
```

练一练

要求：从键盘输入身高，如果身高没有超过150cm，则进动物园不用买票，否则需要买票。

7.3 elif

- 想一想:

  > 如果有这样一种情况：当条件A满足时做事情1；当条件A不满足、条件B满足时做事情2；当条件B不满足、条件C满足时做事情3，那该怎么实现呢?

- 答:

  > elif

elif的功能

elif的使用格式如下:

```
if xxx1:
    事情1
elif xxx2:
    事情2
elif xxx3:
    事情3
```

说明:

- 当xxx1满足时，执行事情1，然后整个if结束
- 当xxx1不满足时，那么判断xxx2，如果xxx2满足，则执行事情2，然后整个if结束
- 当xxx1不满足时，xxx2也不满足，如果xxx3满足，则执行事情3，然后整个if结束

demo:

```
score = 77

if score>=90:
    print('本次考试，等级为A')
elif score>=80:
    print('本次考试，等级为B')
elif score>=70:
    print('本次考试，等级为C')
elif score>=60:
    print('本次考试，等级为D')
elif score<60:
    print('本次考试，等级为E')
```

7.4 for

在Python中 for循环可以遍历任何序列的项目，如一个列表或者一个字符串等。

for循环的格式

```
for 临时变量 in 列表或者字符串等可迭代对象:
    循环满足条件时执行的代码
```

for循环的使用

- 遍历字符串:

```
for s in "hello":
    print(s)
```

输出结果:

```
h
e
l
l
o
```

- 打印数字

```
for i in range(5):
    print(i)
```

输出结果:

```
0
1
2
3
4
```

练习

使用for循环，计算1~100的和

7.5 range

range 可以生成数字供 for 循环遍历,它可以传递三个参数，分别表示 起始、结束和步长。

```
>>> range(2, 10, 3)
[2, 5, 8]
>>> for x in range(2, 10, 3):
...     print(x)
...
2
5
8
```

8. 数据类型高级

8.1 字符串高级

字符串的常见操作包括:

- 获取长度:len　　len函数可以获取字符串的长度。
- 查找内容:find　　查找指定内容在字符串中是否存在，如果存在就返回该内容在字符串中第一次出现的开始位置索引值，如果不存在，则返回-1.

- 判断:startswith,endswith　　判断字符串是不是以谁谁谁开头/结尾
- 计算出现次数:count　　返回 str在start和end之间 在 mystr里面出现的次数
- 替换内容:replace　　替换字符串中指定的内容，如果指定次数count，则替换不会超过count次。
- 切割字符串:split　　通过参数的内容切割字符串
- 修改大小写:upper,lower　　将字符串中的大小写互换
- 空格处理:strip　　去空格
- 字符串拼接:join　　字符串拼接

8.2 列表高级

列表的增删改查

添加元素

添加元素有一下几个方法:

- append 在末尾添加元素
- insert 在指定位置插入元素
- extend 合并两个列表

append

append会把新元素添加到列表末尾

```
    #定义变量A，默认有3个元素
    A = ['xiaoWang','xiaoZhang','xiaoHua']

    print("-----添加之前，列表A的数据-----A=%s" % A)

    #提示、并添加元素
    temp = input('请输入要添加的学生姓名:')
    A.append(temp)

    print("-----添加之后，列表A的数据-----A=%s" % A)
```

insert

insert(index, object) 在指定位置index前插入元素object

```
strs = ['a','b','m','s']
strs.insert(3,'h')
print(strs)  # ['a', 'b', 'm', 'h', 's']
```

extend

通过extend可以将另一个列表中的元素逐一添加到列表中

```
a = ['a','b','c']
b = ['d','e','f']
a.extend(b)
print(a)  # ['a', 'b', 'c', 'd', 'e', 'f'] 将 b 添加到 a 里
print(b) # ['d','e','f'] b的内容不变
```

修改元素

我们是通过指定下标来访问列表元素，因此修改元素的时候，为指定的列表下标赋值即可。

```
#定义变量A，默认有3个元素
A = ['xiaoWang','xiaoZhang','xiaoHua']

print("-----修改之前，列表A的数据-----A=%s" % A)

#修改元素
A[1] = 'xiaoLu'

print("-----修改之后，列表A的数据-----A=%s" % A)
```

查找元素

所谓的查找，就是看看指定的元素是否存在，主要包含一下几个方法：

- in 和 not in

in, not in

python中查找的常用方法为：

- in（存在）,如果存在那么结果为true，否则为false
- not in（不存在），如果不存在那么结果为true，否则false

```
#待查找的列表
nameList = ['xiaoWang','xiaoZhang','xiaoHua']

#获取用户要查找的名字
findName = input('请输入要查找的姓名:')

#查找是否存在
if findName in nameList:
    print('在列表中找到了相同的名字')
else:
    print('没有找到')
```

说明：

> in的方法只要会用了，那么not in也是同样的用法，只不过not in判断的是不存在

删除元素

类比现实生活中，如果某位同学调班了，那么就应该把这个条走后的学生的姓名删除掉；在开发中经常会用到删除这种功能。

列表元素的常用删除方法有：

- del：根据下标进行删除
- pop：删除最后一个元素
- remove：根据元素的值进行删除

del

```
movieName = ['加勒比海盗','骇客帝国','第一滴血','指环王','霍比特人','速度与激情']
print('------删除之前------movieName=%s' % movieName)
del movieName[2]
print('------删除之后------movieName=%s' % movieName)
```

pop

```
movieName = ['加勒比海盗','骇客帝国','第一滴血','指环王','霍比特人','速度与激情']
print('------删除之前------movieName=%s' % movieName)
movieName.pop()
print('------删除之后------movieName=%s' % movieName)
```

remove

```
movieName = ['加勒比海盗','骇客帝国','第一滴血','指环王','霍比特人','速度与激情']
print('------删除之前------movieName=%s' % movieName)
movieName.remove('指环王')
print('------删除之后------movieName=%s' % movieName)
```

8.3 元组高级

Python的元组与列表类似，不同之处在于**元组的元素不能修改。**元组使用小括号，列表使用方括号。

```
>>> aTuple = ('et',77,99.9)
>>> aTuple
('et',77,99.9)
```

访问元组

```
>>> tuple=('hello',100,3.14)
>>> tuple[0]
'hello'
>>> tuple[1]
100
>>> tuple[2]
3.14
>>>
```

修改元组

```
>>> tuple[2]=188
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

python中不允许修改元组的数据，包括不能删除其中的元素。

定义只有一个数据的元组

定义只有一个元素的元组，需要**在唯一的元素后写一个逗号**

```
>>> a = (11)
>>> a
11
>>> type(a)
int
>>> a = (11,)  # 只有一个元素的元组，必须要在元素后写一个逗号
>>> a
(11,)
>>> type(a)
tuple
```

8.4 切片

切片是指对操作的对象截取其中一部分的操作。**字符串、列表、元组**都支持切片操作。

切片的语法：[起始:结束:步长]，也可以简化使用 [起始:结束]

注意：选取的区间从"起始"位开始，到"结束"位的前一位结束（不包含结束位本身），步长表示选取间隔。

```
# 索引是通过下标取某一个元素
# 切片是通过下标去某一段元素

s = 'Hello World!'
print(s)

print(s[4])  # o 字符串里的第4个元素

print(s[3:7])  # lo W 包含下标 3，不含下标 7

print(s[1:]) # ello World! 从下标为1开始，取出 后面所有的元素（没有结束位）

print(s[:4])  # Hell 从起始位置开始，取到 下标为4的前一个元素（不包括结束位本身）

print(s[1:5:2]) # el 从下标为1开始，取到下标为5的前一个元素，步长为2（不包括结束位本身）
```

8.5 字典高级

查看元素

除了使用key查找数据，还可以使用get来获取数据

```
info = {'name':'班长','age':18}

print(info['age']) # 获取年龄
# print(info['sex']) # 获取不存在的key，会发生异常

print(info.get('sex')) # 获取不存在的key，获取到空的内容，不会出现异常
print(info.get('sex',  '男'))  # 获取不存在的key，可以提供一个默认值。
```

修改元素

字典的每个元素中的数据是可以修改的，只要通过key找到，即可修改

demo:

```
info = {'name':'班长', 'id':100}

print('修改之前的字典为 %s:' % info)

info['id'] = 200  # 为已存在的键赋值就是修改

print('修改之后的字典为 %s:' % info)
```

结果:

```
修改之前的字典为 {'name': '班长', 'id': 100}
修改之后的字典为 {'name': '班长', 'id': 200}
```

添加元素

如果在使用 **变量名['键'] = 数据** 时，这个“键”在字典中，不存在，那么就会新增这个元素

demo:添加新的元素

```
info = {'name':'班长'}

print('添加之前的字典为:%s' % info)

info['id'] = 100  # 为不存在的键赋值就是添加元素

print('添加之后的字典为:%s' % info)
```

结果:

```
添加之前的字典为:{'name': '班长'}
添加之后的字典为:{'name': '班长', 'id': 100}
```

删除元素

对字典进行删除操作，有一下几种：

- del
- clear()

demo:del删除指定的元素

```
info = {'name':'班长', 'id':100}

print('删除前,%s' % info)

del info['name']  # del 可以通过键删除字典里的指定元素

print('删除后,%s' % info)
```

结果

```
删除前,{'name': '班长', 'id': 100}
删除后,{'id': 100}
```

del删除整个字典

```
info = {'name':'monitor', 'id':100}

print('删除前,%s'%info)

del info  # del 也可以直接删除变量

print('删除后,%s'%info)
```

结果

```
删除前,{'name': 'monitor', 'id': 100}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'info' is not defined
```

clear清空整个字典

```
info = {'name':'monitor', 'id':100}

print('清空前,%s'%info)

info.clear()

print('清空后,%s'%info)
```

结果

```
清空前,{'name': 'monitor', 'id': 100}
清空后,{}
```

字典的遍历

遍历字典的key（键）

```
>>> dict = {"name":'zhangsan','sex':'m'}
>>> for key in dict.keys():
...     print key
...
name
sex
>>>
```

遍历字典的value（值）

```
>>> dict = {"name":'zhangsan','sex':'m'}
>>> for value in dict.values():
...     print value
...
zhangsan
m
>>>
```

遍历字典的项（元素）

```
>>> dict = {"name":'zhangsan','sex':'m'}
>>> for item in dict.items():
...     print item
...
('name', 'zhangsan')
('sex', 'm')
>>>
```

遍历字典的key-value（键值对）

```
>>> dict = {"name":'zhangsan','sex':'m'}
>>> for key,value in dict.items():
...     print("key=%s,value=%s"%(key,value))
...
key=name,value=zhangsan
key=sex,value=m
>>>
```

9. 函数

思考：下列代码的问题

```
print('欢迎马大哥光临红浪漫')
print('男宾2位')

print('欢迎马大哥光临红浪漫')
print('男宾2位')

print('欢迎马大哥光临红浪漫')
print('男宾2位')

print('欢迎马大哥光临红浪漫')
print('男宾2位')
```

9.1 定义函数

定义函数的格式如下：

```
def 函数名():
    代码
```

示例：

```
# 定义一个函数，能够完成打印信息的功能
def f1():
    print('欢迎马大哥光临红浪漫')
    print('男宾2位')
```

9.2 调用函数

定义了函数之后，就相当于有了一个具有某些功能的代码，想要让这些代码能够执行，需要调用它

调用函数很简单的，通过 **函数名()** 即可完成调用

```
# 定义完函数后，函数是不会自动执行的，需要调用它才可以
f1()
```

> 函数定义好以后，函数体里的代码并不会执行，如果想要执行函数体里的内容，需要手动的调用函数。
>
> 每次调用函数时，函数都会从头开始执行，当这个函数中的代码执行完毕后，意味着调用结束了。

9.3 函数参数

思考一个问题，如下：

> 现在需要定义一个函数，这个函数能够完成2个数的加法运算，并且把结果打印出来，该怎样设计？下面的代码可以吗？有什么缺陷吗？

```
def add2num():
  a = 11
  b = 22
  c = a+b
  print(c)
```

为了让一个函数更通用，即想让它计算哪两个数的和，就让它计算哪两个数的和，在定义函数的时候可以让函数接收数据，就解决了这个问题，这就是 函数的参数

定义、调用带有参数的函数

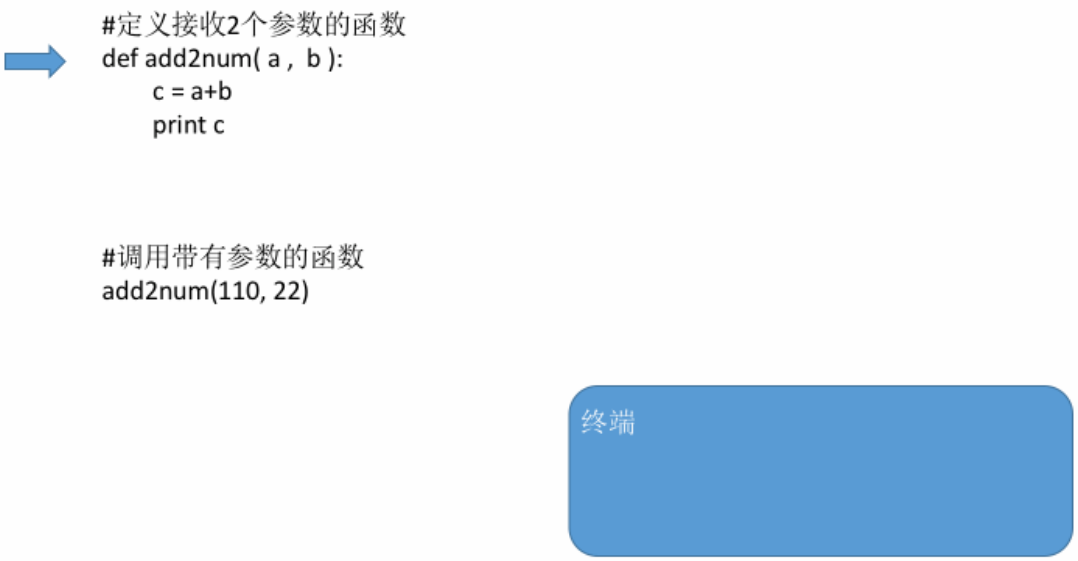
定义一个add2num(a, b)函数，来计算任意两个数字之和:

```
def add2num(a, b):
    c = a+b
    print c

add2num(11, 22) # 调用带有参数的函数时，需要在小括号中，传递数据
```

注意点：

- 在定义函数的时候，小括号里写等待赋值的变量名
- 在调用函数的时候，小括号里写真正要进行运算的数据

调用带有参数函数的运行过程：



调用函数时参数的顺序

```
>>> def test(a,b):
...     print(a,b)
...
>>> test(1,2)  # 位置参数
1 2
>>> test(b=1,a=2)  # 关键字参数
2 1
```

> - 定义时小括号中的参数，用来接收参数用的，称为“形参”
> - 调用时小括号中的参数，用来传递给函数用的，称为“实参”

9.4 函数返回值

“返回值”介绍

现实生活中的场景：

> 我给女儿10块钱，让她给我买个冰淇淋。这个例子中，10块钱是我给女儿的，就相当于调用函数时传递到参数，让女儿买冰淇淋这个事情最终的目标，我需要让她把冰淇淋带回来，此时冰淇淋就是返回值

综上所述：

- 所谓“返回值”，就是程序中函数完成一件事情后，最后给调用者的结果

带有返回值的函数

想要在函数中把结果返回给调用者，需要在函数中使用return

如下示例:

```
def add2num(a, b):
    c = a+b
    return c  # return 后可以写变量名
```

或者

```
def add2num(a, b):
    return a+b  # return 后可以写计算表达式
```

保存函数的返回值

在本小节刚开始的时候，说过的“买冰淇淋”的例子中，最后女儿给你冰淇淋时，你一定是从女儿手中接过来 对么，程序也是如此，如果一个函数返回了一个数据，那么想要用这个数据，那么就需要保存

保存函数的返回值示例如下:

```
#定义函数
def add2num(a, b):
    return a+b

#调用函数，顺便保存函数的返回值
result = add2num(100,98)

#因为result已经保存了add2num的返回值，所以接下来就可以使用了
print(result)
```

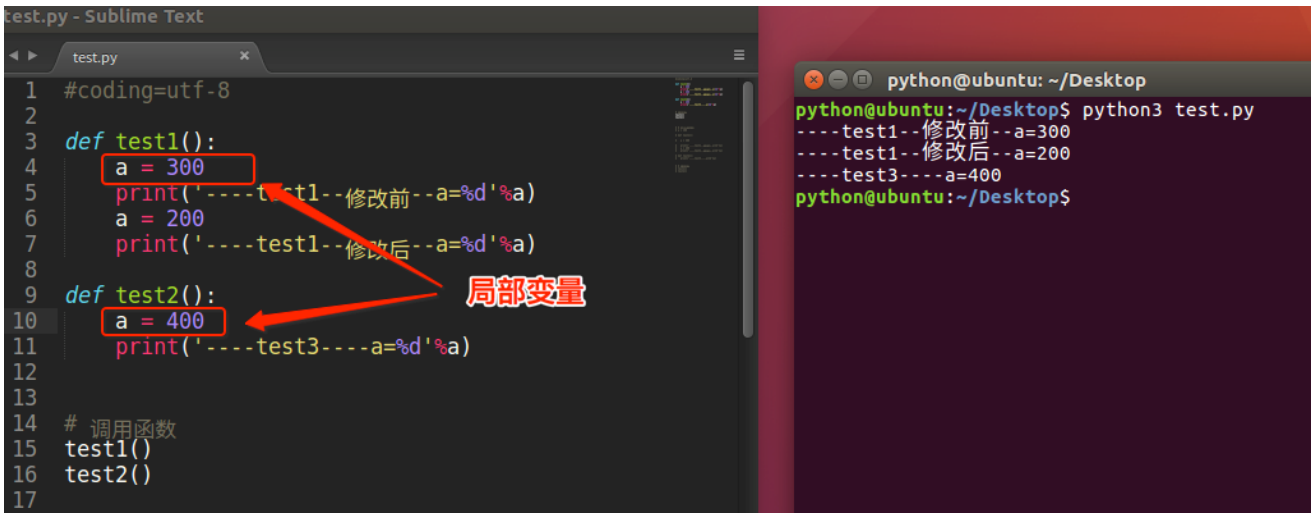
结果:

```
198
```

9.5 局部变量

什么是局部变量

如下图所示:



- 局部变量，就是在函数内部定义的变量
- 其作用范围是这个函数内部，即只能在这个函数中使用，在函数的外部是不能使用的

9.6 全局变量

如果一个变量，既能在一个函数中使用，也能在其他的函数中使用，这样的变量就是全局变量

```
# 定义全局变量
a = 100

def test1():
    print(a)  # 虽然没有定义变量a但是依然可以获取其数据

def test2():
    print(a)  # 虽然没有定义变量a但是依然可以获取其数据

# 调用函数
test1()
test2()
```

- 在函数外边定义的变量叫做`全局变量`
- 全局变量能够在所有的函数中进行访问

10. 文件

10.1 文件的打开与关闭

打开文件/创建文件

在python，使用open函数，可以打开一个已经存在的文件，或者创建一个新文件

open(文件路径，访问模式)

示例如下：

```
f = open('test.txt', 'w')
```

说明:

文件路径

- 绝对路径：指的是绝对位置，完整地描述了目标的所在地，所有目录层级关系是一目了然的。
  - 例如：`E:\python`，从电脑的盘符开始，表示的就是一个绝对路径。
- 相对路径：是从当前文件所在的文件夹开始的路径。
  - `test.txt`，是在当前文件夹查找 `test.txt` 文件
  - `./test.txt`，也是在当前文件夹里查找`test.txt`文件， `./` 表示的是当前文件夹。
  - `../test.txt`，从当前文件夹的上一级文件夹里查找 `test.txt` 文件。 `../` 表示的是上一级文件夹
  - `demo/test.txt`，在当前文件夹里查找 `demo`这个文件夹，并在这个文件夹里查找 `test.txt`文件。

访问模式：

访问模式	说明
r	以只读方式打开文件。文件的指针将会放在文件的开头。如果文件不存在，则报错。**这是默认模式。**
w	打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
r+	打开一个文件用于读写。文件指针将会放在文件的开头。
w+	打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
a+	打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
ab+	以二进制格式打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

关闭文件

示例如下：

```
# 新建一个文件，文件名为:test.txt
f = open('test.txt', 'w')

# 关闭这个文件
f.close()
```

10.2 文件的读写

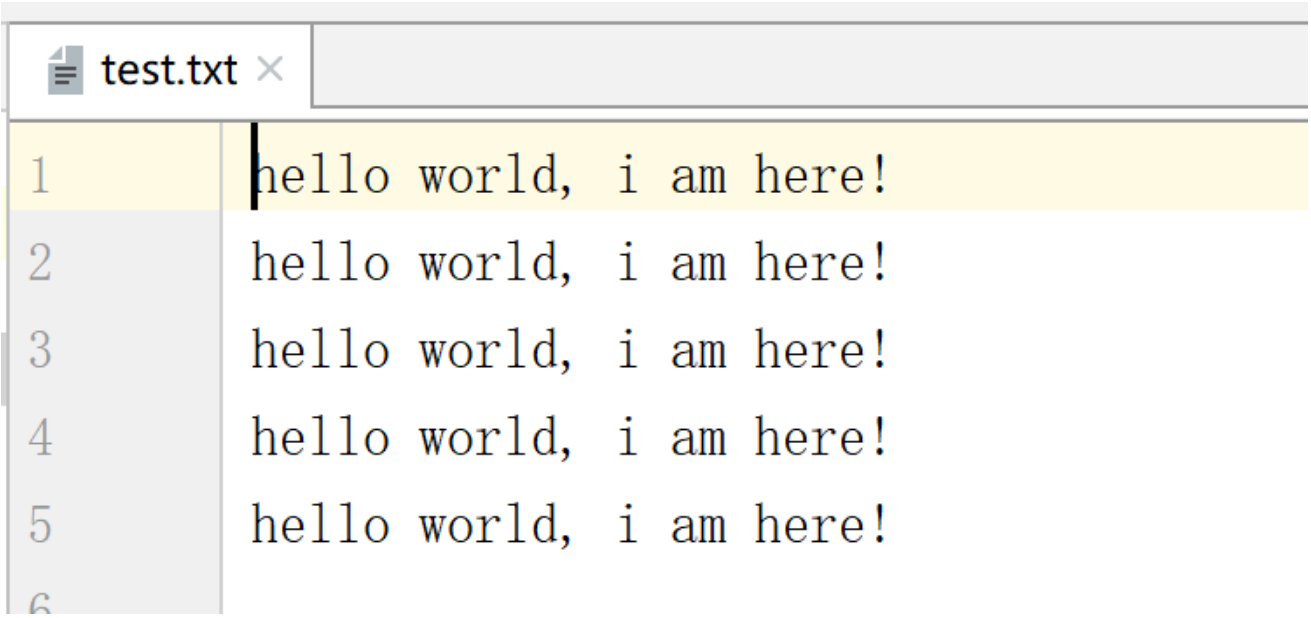
写数据(write)

使用write()可以完成向文件写入数据

demo: 新建一个文件 `file_write_test.py` ,向其中写入如下代码:

```
f = open('test.txt', 'w')
f.write('hello world, i am here!\n' * 5)
f.close()
```

运行之后会在`file_write_test.py`文件所在的路径中创建一个文件`test.txt`,并写入内容，运行效果显示如下:



注意:

- 如果文件不存在，那么创建；如果存在那么就先清空，然后写入数据

读数据(read)

使用read(num)可以从文件中读取数据，num表示要从文件中读取的数据的长度（单位是字节），如果没有传入num，那么就表示读取文件中所有的数据

demo: 新建一个文件`file_read_test.py`，向其中写入如下代码:

```
f = open('test.txt', 'r')
content = f.read(5)  # 最多读取5个数据
print(content)

print("-"*30)  # 分割线，用来测试

content = f.read()  # 从上次读取的位置继续读取剩下的所有的数据
print(content)

f.close()  # 关闭文件，这个可是个好习惯哦
```

运行现象：

```
hello
------------------------------
 world, i am here!
```

注意：

- 如果用open打开文件时，如果使用的"r"，那么可以省略 `open('test.txt')`

读数据（readline）

readline只用来读取一行数据。

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

content = f.readline()
print("1:%s" % content)

content = f.readline()
print("2:%s" % content)


f.close()
```

读数据（readlines）

readlines可以按照行的方式把整个文件中的内容进行一次性读取，并且返回的是一个列表，其中每一行为列表的一个元素。

```
f = open('test.txt', 'r')
content = f.readlines()
print(type(content))

for temp in content:
    print(temp)

f.close()
```

10.3 序列化和反序列化

通过文件操作，我们可以将字符串写入到一个本地文件。但是，如果是一个对象(例如列表、字典、元组等)，就无法直接写入到一个文件里，需要对这个对象进行序列化，然后才能写入到文件里。

设计一套协议，按照某种规则，把内存中的数据转换为字节序列，保存到文件，这就是序列化，反之，从文件的字节序列恢复到内存中，就是反序列化。

对象---》字节序列 === 序列化

字节序列--》对象 ===反序列化

Python中提供了JSON这个模块用来实现数据的序列化和反序列化。

JSON模块

JSON(JavaScriptObjectNotation, JS对象简谱)是一种轻量级的数据交换标准。JSON的本质是字符串。

使用JSON实现序列化

JSON提供了dump和dumps方法，将一个对象进行序列化。

dumps方法的作用是把对象转换成为字符串，它本身不具备将数据写入到文件的功能。

```
import json
file = open('names.txt', 'w')
names = ['zhangsan', 'lisi', 'wangwu', 'jerry', 'henry', 'merry', 'chris']
# file.write(names)  出错，不能直接将列表写入到文件里

# 可以调用 json的dumps方法，传入一个对象参数
result = json.dumps(names)

# dumps 方法得到的结果是一个字符串
print(type(result))  # <class 'str'>

# 可以将字符串写入到文件里
file.write(result)

file.close()
```

dump方法可以在将对象转换成为字符串的同时，指定一个文件对象，把转换后的字符串写入到这个文件里。

```
import json

file = open('names.txt', 'w')
names = ['zhangsan', 'lisi', 'wangwu', 'jerry', 'henry', 'merry', 'chris']

# dump方法可以接收一个文件参数，在将对象转换成为字符串的同时写入到文件里
json.dump(names, file)
file.close()
```

使用JSON实现反序列化

使用loads和load方法，可以将一个JSON字符串反序列化成为一个Python对象。

loads方法需要一个字符串参数，用来将一个字符串加载成为Python对象。

```python
import json

# 调用loads方法，传入一个字符串，可以将这个字符串加载成为Python对象
result = json.loads('["zhangsan", "lisi", "wangwu", "jerry", "henry", "merry", "chris"]')
print(type(result))  # <class 'list'>
```

load方法可以传入一个文件对象，用来将一个文件对象里的数据加载成为Python对象。

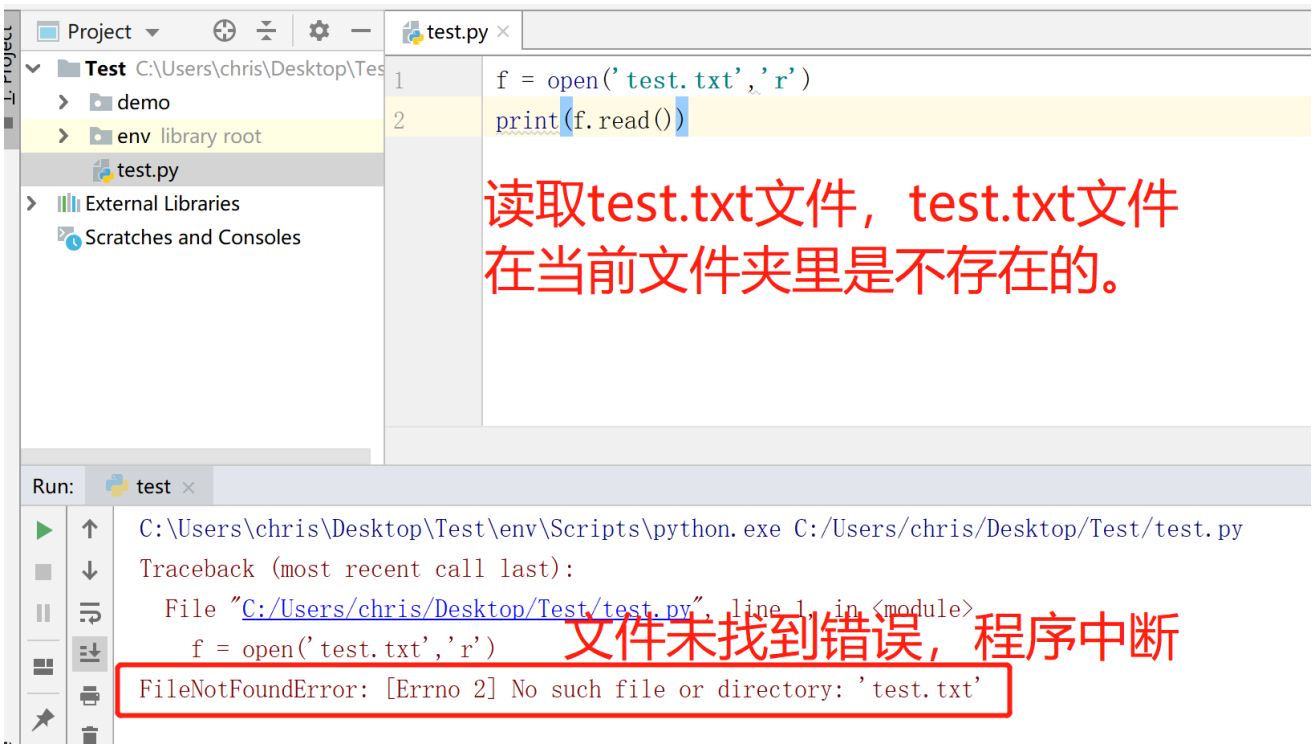
```python
import json

# 以可读方式打开一个文件
file = open('names.txt', 'r')

# 调用load方法，将文件里的内容加载成为一个Python对象
result = json.load(file)

print(result)
file.close()
```

11. 异常

程序在运行过程中，由于我们的编码不规范，或者其他原因一些客观原因，导致我们的程序无法继续运行，此时，程序就会出现异常。如果我们不对异常进行处理，程序可能会由于异常直接中断掉。为了保证程序的健壮性，我们在程序设计里提出了异常处理这个概念。

11.1 读取文件异常

在读取一个文件时，如果这个文件不存在，则会报出`FileNotFoundError`错误。



11.2 try...except语句

try...except语句可以对代码运行过程中可能出现的异常进行处理。 语法结构:

```
try:
    可能会出现异常的代码块
except 异常的类型:
    出现异常以后的处理语句
```

示例:

```
try:
    f = open('test.txt', 'r')
    print(f.read())
except FileNotFoundError:
    print('文件没有找到,请检查文件名称是否正确')
```