

Python语言程序设计

# 第1章 程序设计基本方法

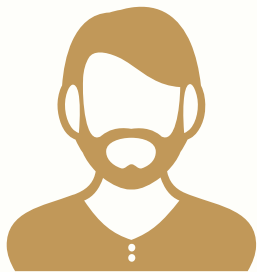
---





# 本课概要

# 第1章 Python基本语法元素



- **1.1** 程序设计基本方法
- **1.2** Python开发环境配置
- **1.3** 实例1: 温度转换
- **1.4** Python程序语法元素分析



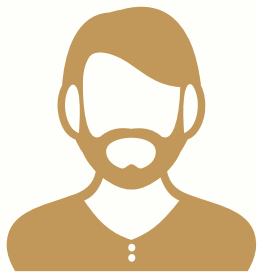
# 第1章 Python基本语法元素

## 方法论

- 程序的基本编写方法：IPO

## 实践能力

- 看懂10行左右简单Python代码



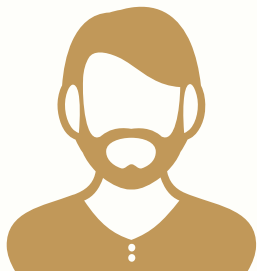
Python语言程序设计

# 1.1 程序设计基本方法

---



# 程序设计基本方法



- 计算机与程序设计
- 编译和解释
- 程序的基本编写方法
- 计算机编程





# 计算机的概念

**计算机是根据指令操作数据的设备**

- **计算的功能性**

对数据的操作，表现为数据计算、输入输出处理和结果存储等

- **操作的可编程性**

根据一系列指令自动地、可预测地、准确地完成操作者的意图



# 计算机的发展

**计算机的硬件发展参照摩尔定律，表现为指数方式**

- **计算机硬件所依赖的集成电路规模参照摩尔定律发展**
- **计算机运行速度因此也接近几何级数快速增长**
- **计算机高效支撑的各类运算功能不断丰富发展**

计算机技术的发展主要围绕计算的功能性和可编程性展开

一方面，计算机**硬件发展**变现为指数模式

另一方面，表达计算机可编程性的程序设计语言也在经历从机器、汇编到高级语言的发展过程

# 程序设计

## 程序设计是计算机可编程性的体现

- 程序设计，亦称编程，深度应用计算机的主要手段
- 程序设计已经成为当今社会需求量最大的职业技能之一
- 很多岗位都将被计算机程序接管，程序设计将是生存技能

计算机技术的发展主要围绕计算的功能性和可编程性展开

一方面，计算机硬件发展变现为指数模式

另一方面，表达计算机可编程性的程序设计语言也在经历从机器、汇编到高级语言的发展过程

# 程序设计语言

**程序设计语言是一种用于交互(交流)的人造语言**

- **程序设计语言，亦称编程语言，程序设计的具体实现方式**
- **编程语言相比自然语言更简单、更严谨、更精确**
- **编程语言主要用于人类和计算机之间的交互**

程序设计语言是计算机能够理解和识别用户操作意图的一种交互体系，它按照特定规则组织计算机指令，使计算机能够自动进行各种运算处理

# 程序设计语言

**编程语言种类很多，但生命力强劲的不多**

- 编程语言有超过600种，绝大部分都不再被使用**
- C语言诞生于1972年，它是第一个被广泛使用的编程语言**
- Python语言诞生于1990年，它是目前流行好用的编程语言**



# 编译和解释

# 编程语言的执行方式

## 计算机执行源程序的两种方式：编译和解释

- **源代码**：采用某种编程语言编写的计算机程序，人类可读

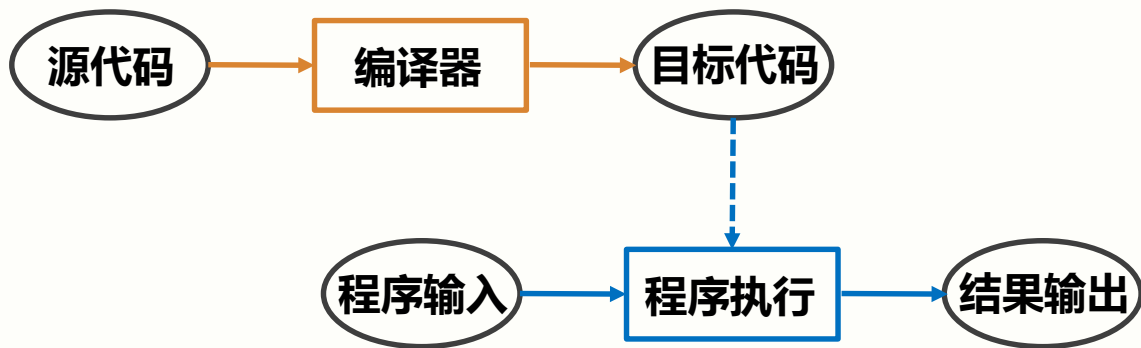
例如：`result = 2 + 3`

- **目标代码**：计算机可直接执行，机器语言代码

例如：`11010010 00111011`

# 编译

将源代码一次性转换成目标代码的过程

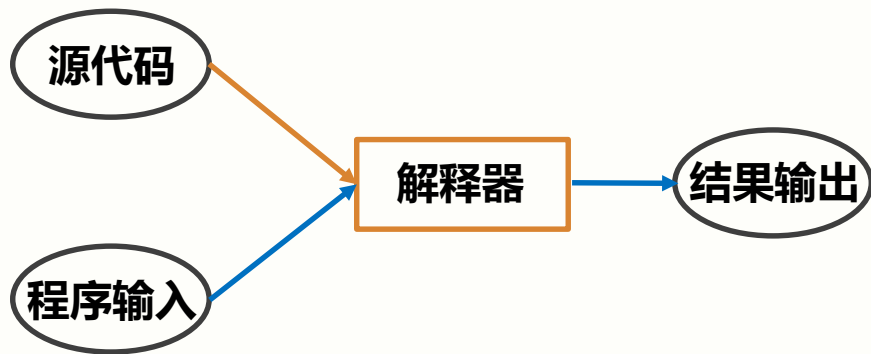


执行编译过程的程序叫作编译器

编译器将源代码转化成目标代码，计算机可以立即或稍后运行这个目标代码

# 解释

将源代码逐条转换成目标代码同时逐条运行目标代码的过程

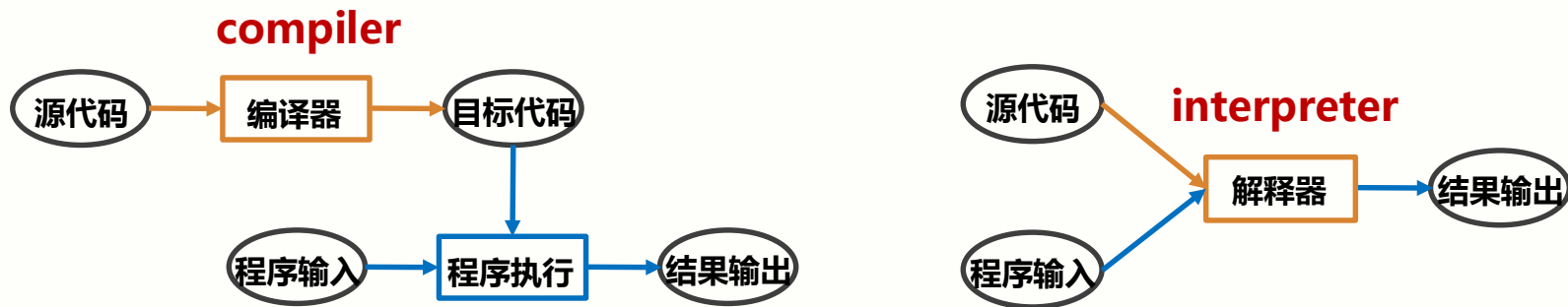


**执行解释过程的程序叫作解释器**

源代码和数据一同输入给解释器，然后运行输出结果



# 编译和解释



- 编译：一次性翻译，之后不再需要源代码（类似英文翻译）
- 解释：每次程序运行时随翻译随执行（类似实时的同声传译）

# 静态语言和脚本语言

根据执行方式不同，编程语言分为两类

- **静态语言：使用编译执行的编程语言**

C/C++语言、Java语言

- **动态（脚本）语言：使用解释执行的编程语言**

Python语言、JavaScript语言、PHP语言

# 静态语言和脚本语言

**执行方式不同，优势各有不同**

- **静态语言：编译器一次性生成目标代码，优化更充分**  
程序运行速度更快，具备更好的执行效率
- **动态（脚本）语言：执行程序时需要源代码，维护更灵活**  
源代码维护灵活、跨多个操作系统平台、可移植性好



# 程序的基本编写方法

# IPO

## 程序的基本编写方法

- **I : Input 输入，程序的输入**
- **P : Process 处理，程序的主要逻辑**
- **O : Output 输出，程序的输出**

每个程序都有统一的运算模式，即输入数据、处理数据和输出数据，这种朴素的运算模式形成了程序的基本编写方法：IPO

# 理解IPO

## 输入 ( Input )

- 程序的输入

文件输入、网络输入、控制台输入、交互界面输入、内部参数输入等

- 输入是一个程序的开始

# 理解IPO

## 输出 ( Output )

- **程序的输出**

控制台输出、图形输出、文件输出、网络输出、操作系统内部变量输出等

- **输出是程序展示运算结果的方式**

# 理解IPO

## 处理 ( Process )

- 处理是程序对输入数据进行计算产生输出结果的过程
- 处理方法统称为算法，它是程序最重要的部分
- 算法是一个程序的灵魂



# 理解问题的计算部分

**一个待解决问题中，可以用程序辅助完成的部分**

- **计算机只能解决计算问题，即问题的计算部分**
- **一个问题可能有多种角度理解，产生不同的计算部分**
- **问题的计算部分一般都有输入、处理和输出过程**

# 编程解决问题的步骤

## 6个步骤 (1-3)

- **分析问题：分析问题的计算部分，想清楚**

首先明确，计算机只能解决计算问题，即解决一个问题的计算部分

- **划分边界：划分问题的功能边界，规划IPO**

计算机只能完成确定性的计算功能，需要精确定义或描述问题的功能边界，即明确问题的输入、输出和对处理的要求

- **设计算法：设计问题的求解算法，关注算法**

在明确处理功能的基础上，如何实现程序功能呢？需要设计问题的求解算法

编写程序的目的是：使用计算机解决问题

# 使用计算机解决问题

## 6个步骤 (4-6)

- **编写程序：编写问题的计算程序，编程序**

选择一门编程语言，将程序结构和算法设计用编程语言来实现

- **调试测试：调试程序使正确运行，运行调试**

运行程序，通过单元测试和集成测试评估程序运行结果的正确性

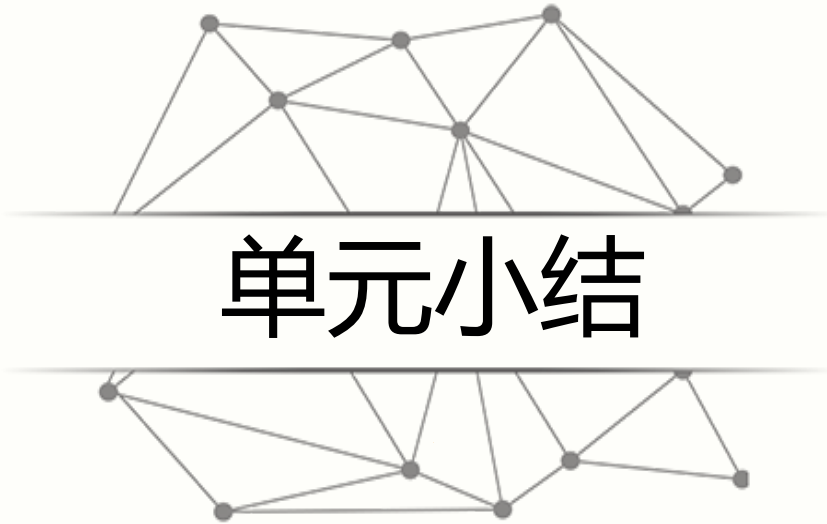
- **升级维护：适应问题的升级维护，更新完善**

随着功能、计算和应用需求的不断变化，程序不断升级维护

# 求解计算问题的精简步骤

## 3个精简步骤

- **确定IPO：明确计算部分及功能边界**
- **编写程序：将计算求解的设计变成现实**
- **调试程序：确保程序能够按照正确逻辑正确运行**



# 单元小结

# 程序设计基本方法

- 计算机的功能性和可编程性
- 编译和解释、静态语言和动态语言
- IPO、理解问题的计算部分
- 掌握计算机编程的价值



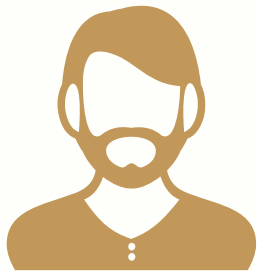
Python语言程序设计

## 1.2 Python开发环境配置

---



# Python开发环境配置



- Python语言概述
- Python基本开发环境IDLE
- Python程序编写与运行
- Python高级开发环境VSCode ( 了解下 )







# Python语言概述



**Python [ˈpaɪθən] , 译为 “蟒蛇”**

**Python语言拥有者是Python Software Foundation(PSF)**

**PSF是非盈利组织 , 致力于保护Python语言开放、开源和发展**

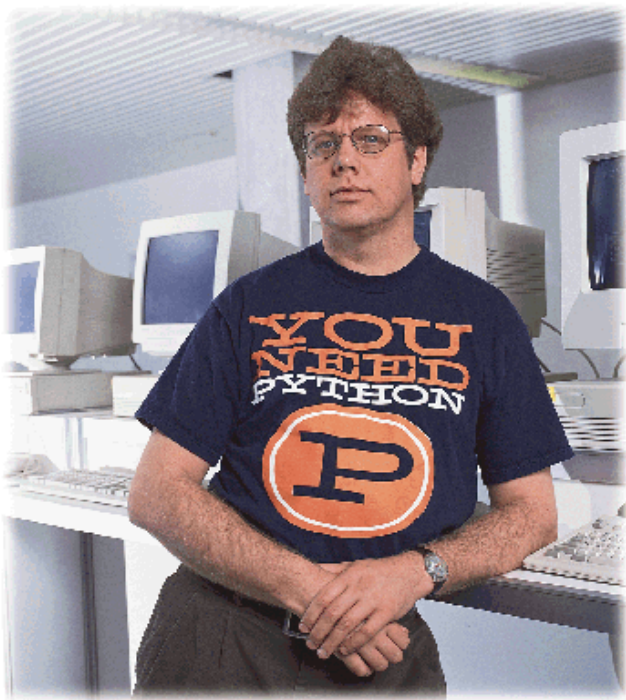
# Python语言的诞生

**Guido van Rossum ( 荷兰 )**

**1989年 , 1991年 , 诞生**

**2000年10月 , Python 2.x**

**2008年12月 , Python 3.x**





**Python语言是一个由编程牛人领导设计并开发的编程语言**

**Python语言是一个有开放、开源精神的编程语言**

**Python语言应用于火星探测、搜索引擎、引力波分析等众多领域**



# Python基本开发环境IDLE

# Python基本开发环境IDLE

**Python官方提供 适用于小规模程序开发**

- **Python官方环境：Python解释器 + IDLE开发环境**
- **轻量级：只有几十MB大小，使用灵活**
- **功能丰富：编辑器+交互环境+标准库+库安装工具...**

# Python基本开发环境IDLE

Python官方提供 适用于小规模程序开发

- 下载地址：[www.python.org/downloads](http://www.python.org/downloads)

具体配置方法

- 参考网址：<https://www.jianshu.com/p/9c46282e13f2>



# Python程序编写与运行



## C 语言实例 - 输出 "Hello, World!"



[C 语言实例](#)

使用 printf() 输出 "Hello, World!"。

### 实例

```
#include <stdio.h>
int main()
{
    // printf() 中字符串需要引号
    printf("Hello, World!");
    return 0;
}
```

## Python Hello World 实例



[Python3 实例](#)

以下实例为学习Python的第一个实例，即如何输出"Hello World!"：

### 实例

```
# -*- coding: UTF-8 -*-

# Filename : helloworld.py
# author by : www.runoob.com

# 该实例输出 Hello World!
print('Hello World!')
```

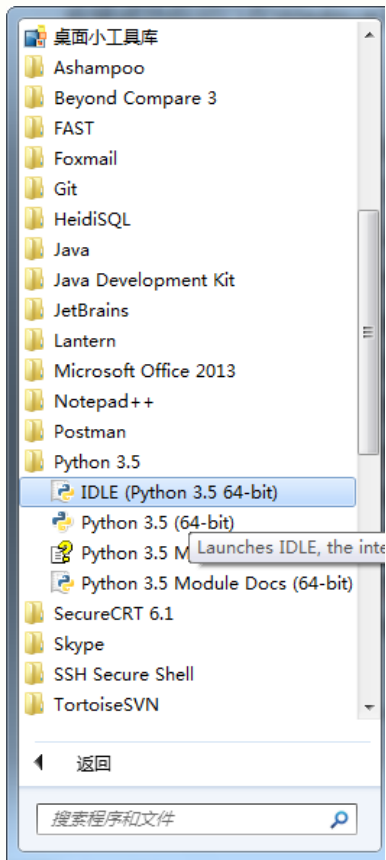
# Python的两种编程方式

## 交互式和文件式

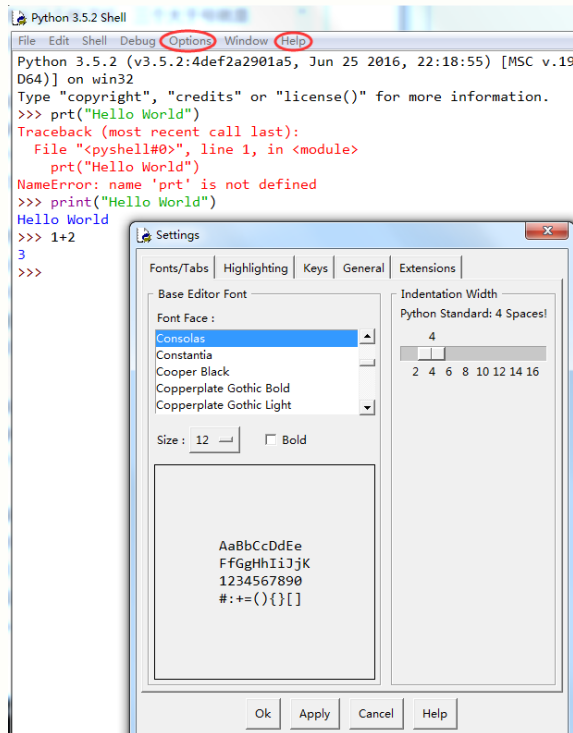
- **交互式：对每个输入语句即时运行结果，适合语法练习**
- **文件式：批量执行一组语句并运行结果，编程的主要方式**

交互式：python解释器即时响应用户输入的每条代码，给出输出结果

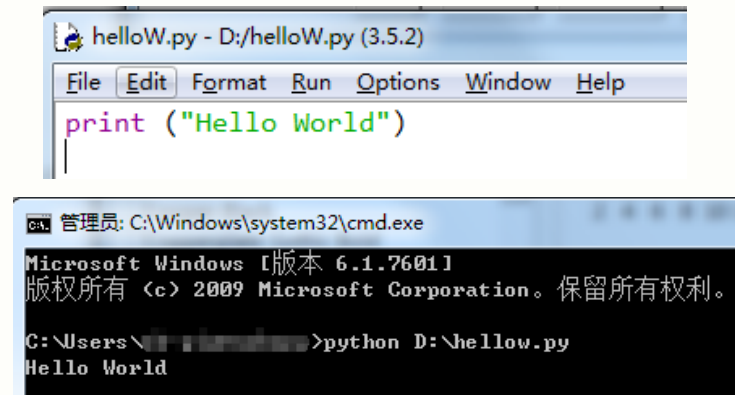
文件式：用户将python代码程序写在一个或多个文件中，然后启动python解释器批量执行文件中的代码



## 交互式



## 文件式



# 实例1: 圆面积的计算

## 根据半径r计算圆面积

命令提示符

```
>>> r = 25
>>> area = 3.1415 * r * r
>>> print(area)
1963.4375000000002
>>> print("{:.2f}".format(area))
1963.44
```

交互式

# 实例1: 圆面积的计算

## 根据半径r计算圆面积

```
r = 25
area = 3.1415 * r * r
print(area)
print("{:.2f}".format(area))
```

输出结果如下：

```
1963.4375000000002
1963.44
```

保存为CalCircle.py文件并运行

文件式



# Python高级开发环境VSCode

# Python高级开发环境VSCode

## VSCode : Visual Studio Code

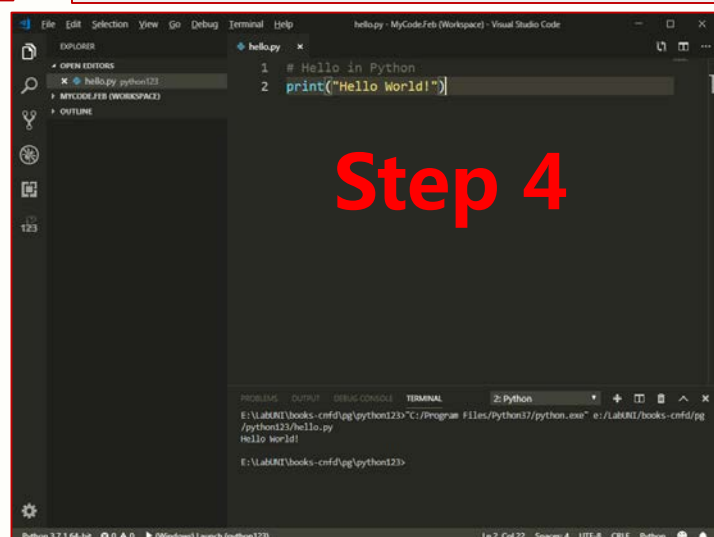
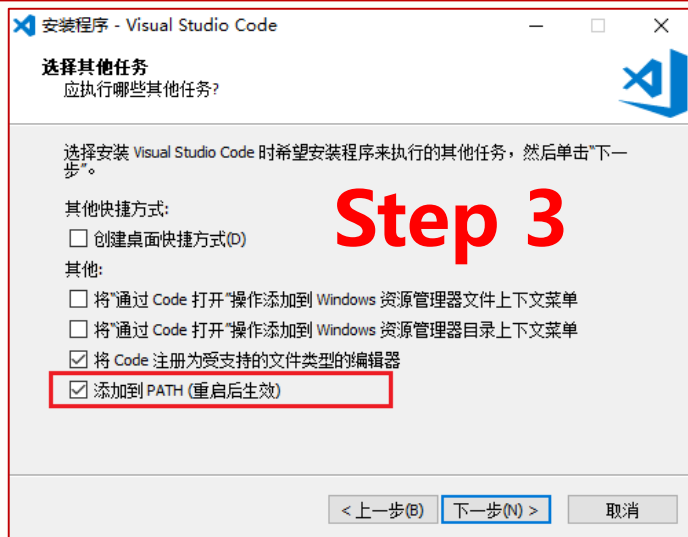
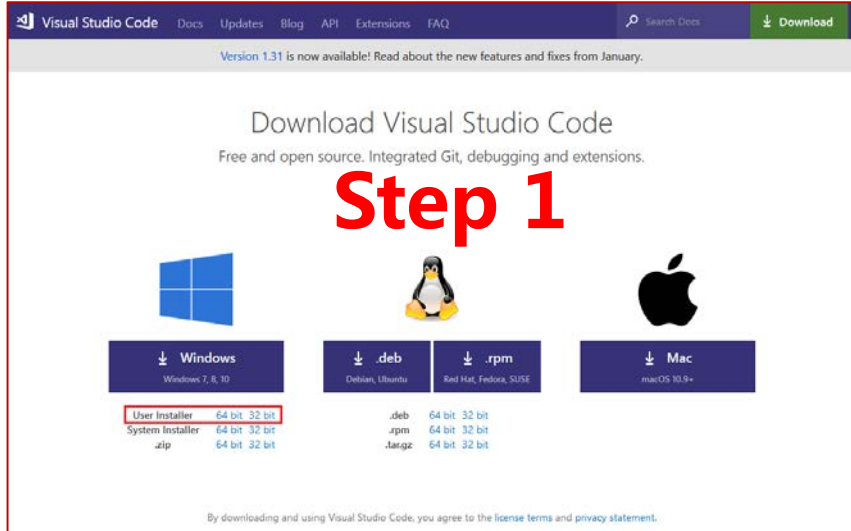
- 微软出品，与Visual Studio同质量的**专业级**开发工具
- 跨平台**免费**工具：支持Windows/Linux/MacOS
- **编辑器**模式：轻量级、功能丰富、可扩展性强...

# Python高级开发环境VSCode

**第一步：安装IDLE环境；第二步：安装VSCode**

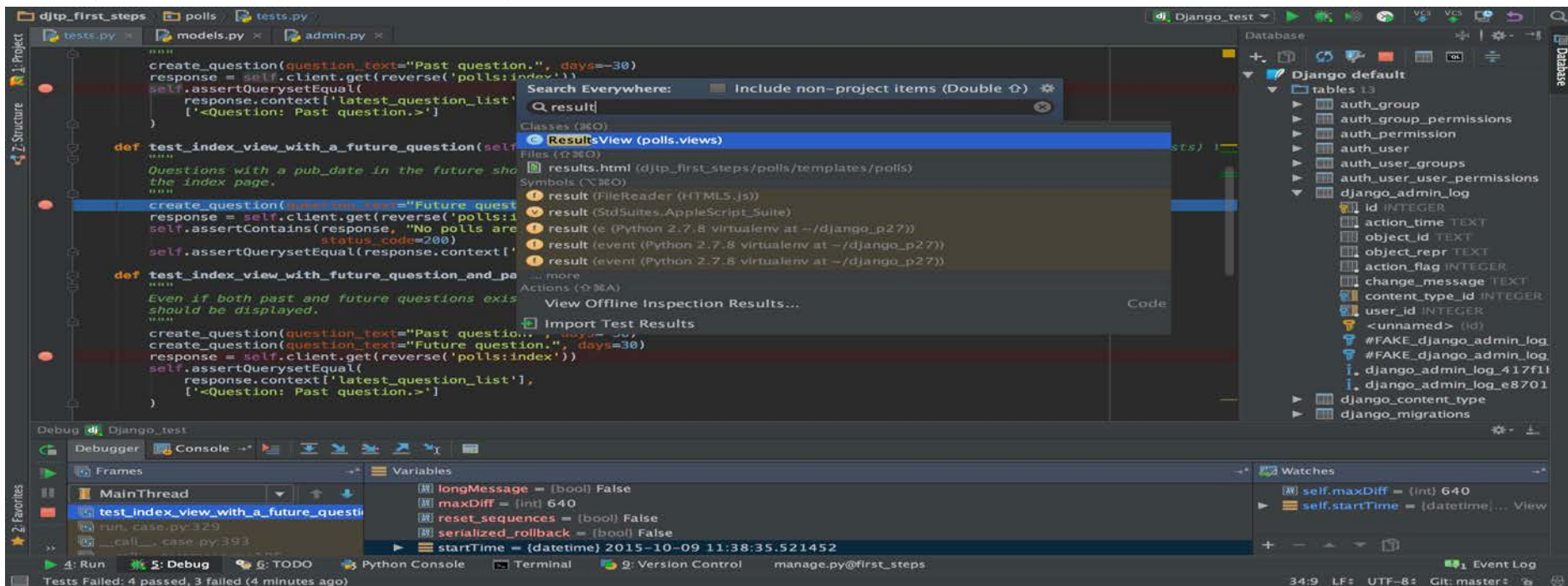
- **下载地址：** <https://code.visualstudio.com>
- **工具大小约 50MB**





# Python高级开发环境PyCharm

## PyCharm : The Python IDE for Professional Developers

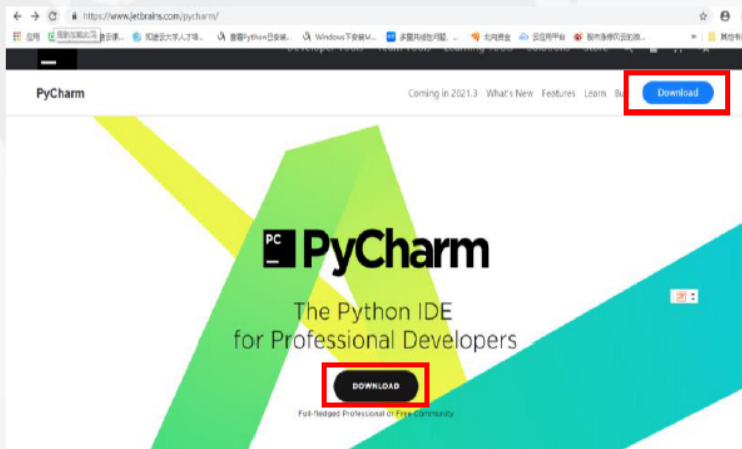


# Pycharm的安装

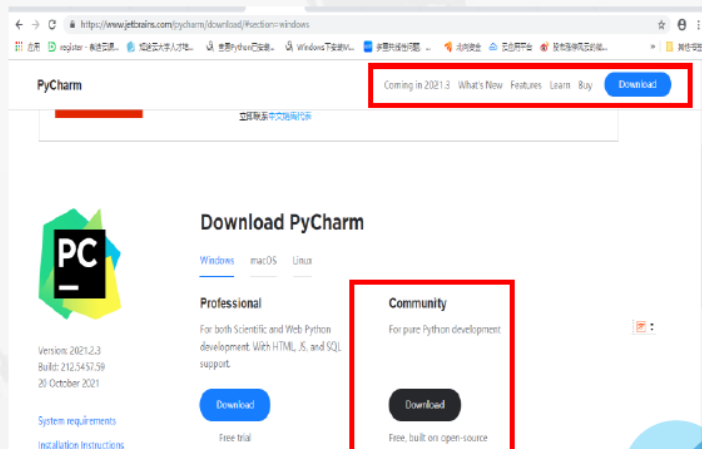
## 1. 下载Pycharm社区版安装文件

① 打开网页: <http://www.jetbrains.com/pycharm/>

② 选择 "download"



③ 选择 "download"



# Pycharm的使用

## 1. 启动Pycharm

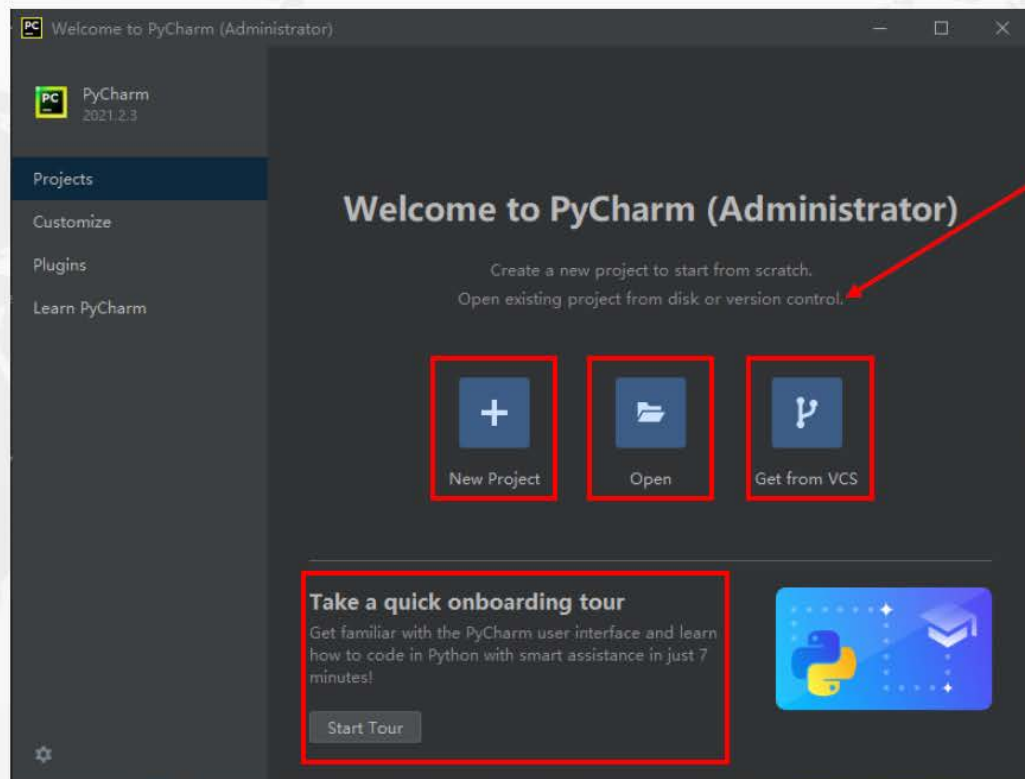


双击桌面的图标

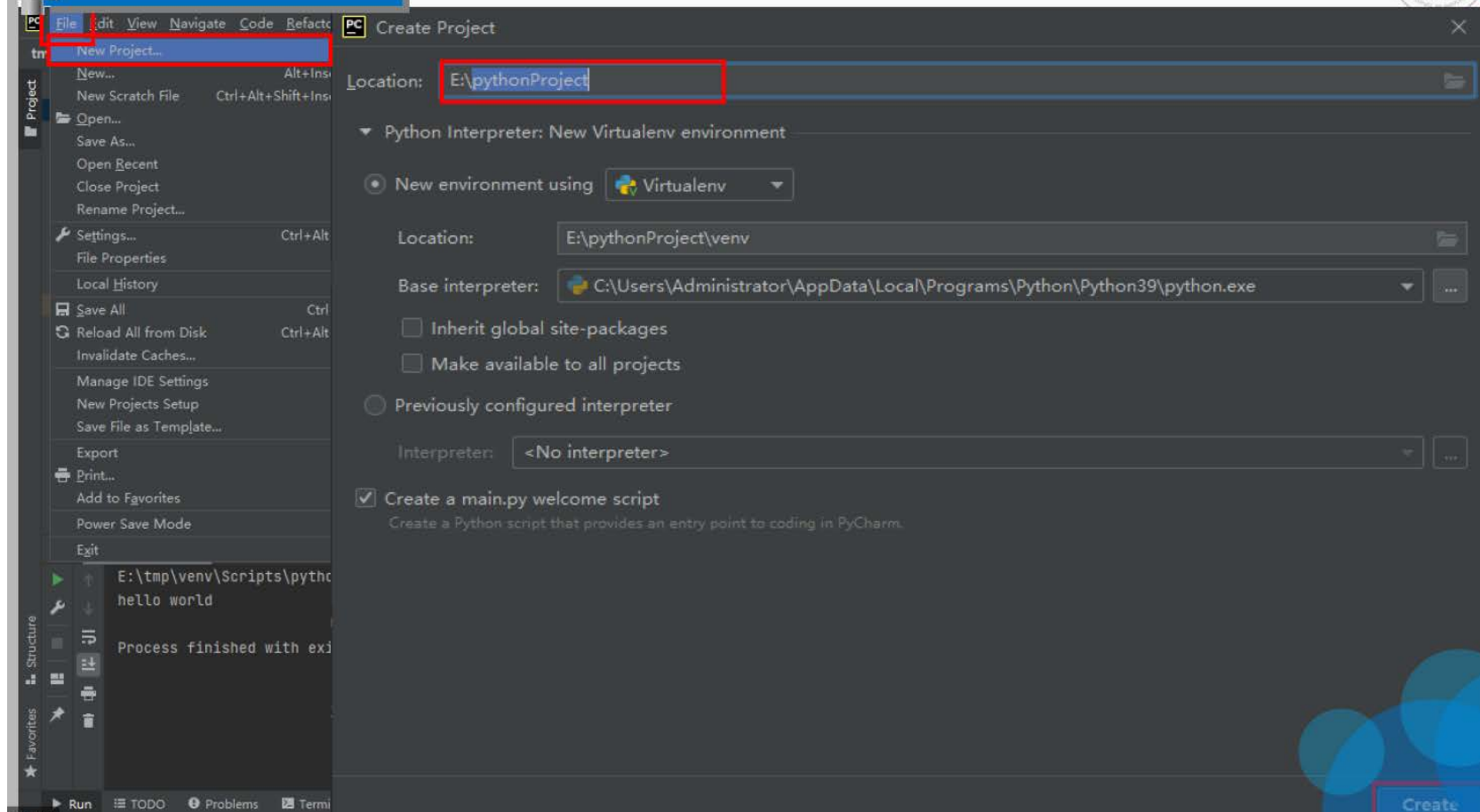
OR

“开始”菜单 → “程序” → “JetBrains--  
Pycharm Community Edition 2021.2.3”

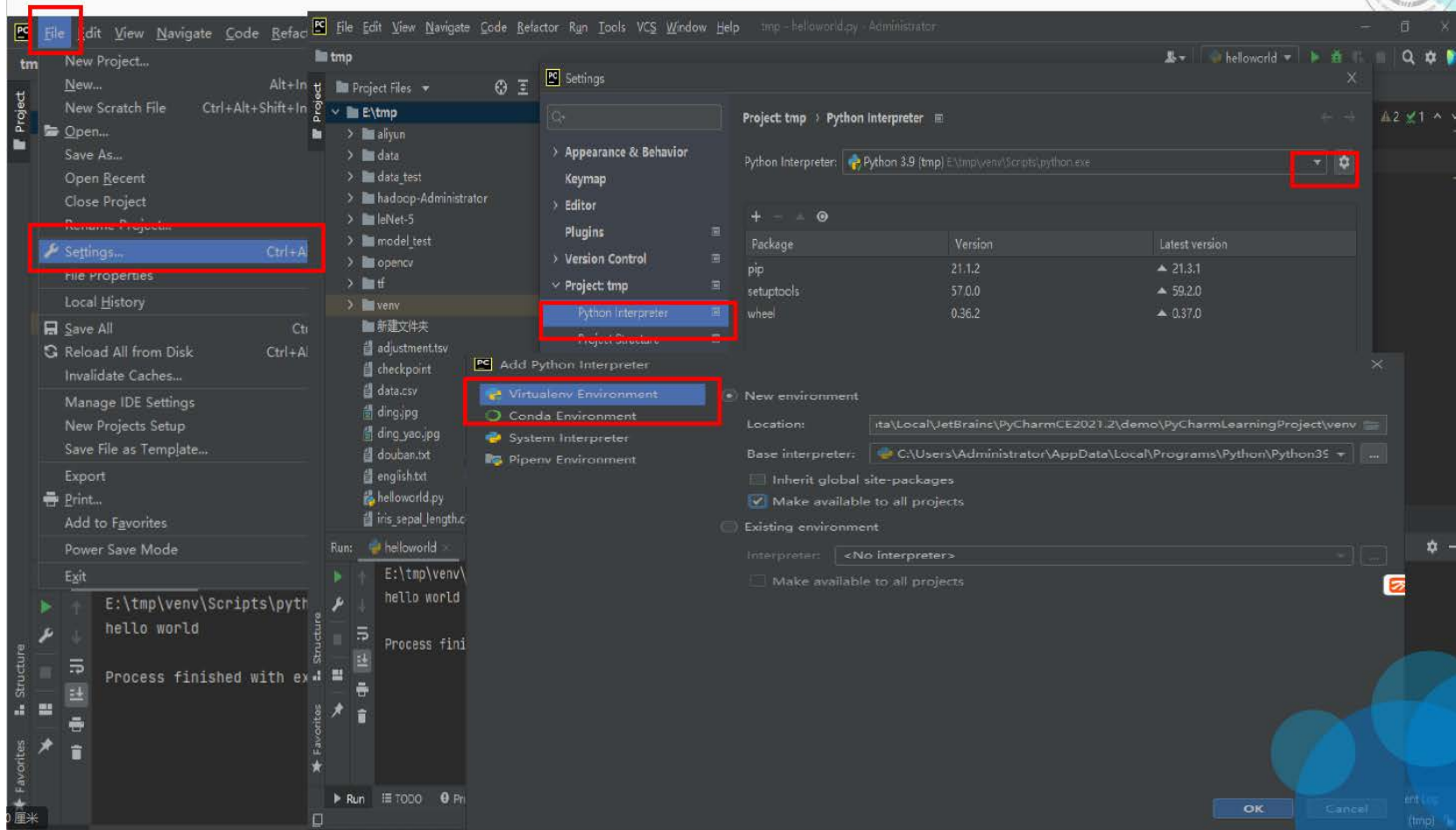
# Pycharm的使用



可以选择其中一个继续

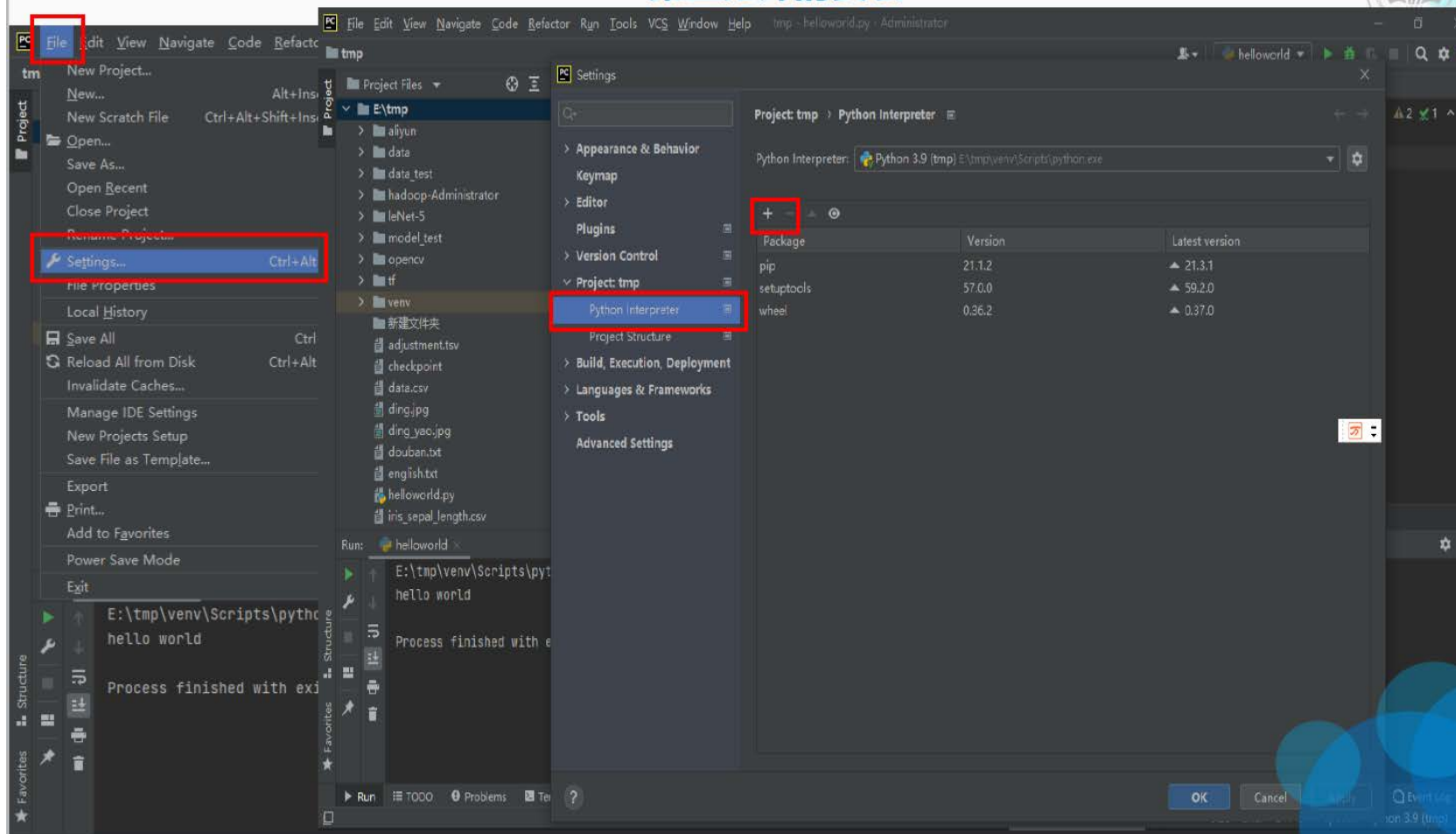


## 2. 环境设置

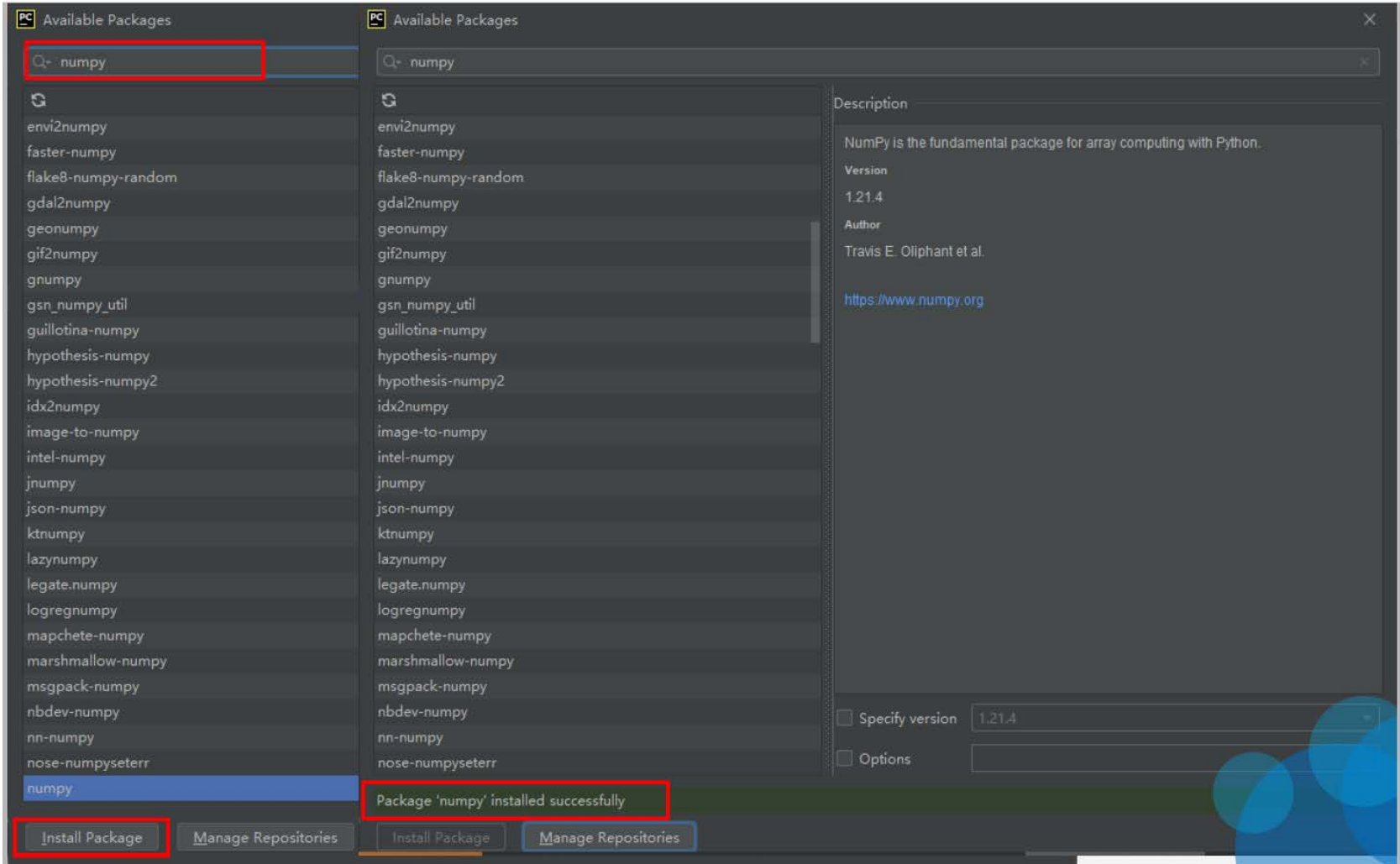




### 3. 第三方库的安装







## 4. 新建文件



The screenshot shows the PyCharm IDE interface. The 'Project' view on the left shows a project named 'tmp'. The 'File' menu is open, and the 'New' option is selected. The 'New Python file' dialog is open, showing options like 'helloworld', 'Python file', 'Python unit test', and 'Python stub'. The 'Python file' option is selected. The main editor window shows the code for 'helloworld.py':

```
1 #first program
2
3 print('hello world')
```

The 'Run' view at the bottom shows the output of the program: 'hello world'.

Creates a Python file from the specified template

## 5. 文件运行



The screenshot shows an IDE window titled 'tmp - helloworld.py - Administrator'. The file 'helloworld.py' is open, showing the following code:

```
1 #first programmer
2
3 print("hello world")
```

A context menu is open over the code, with the 'Run \'helloworld\'' option highlighted. The menu also includes options like 'Paste', 'Copy / Paste Special', 'Column Selection Mode', 'Find Usages', 'Refactor', 'Folding', 'Go To', 'Generate...', 'Debug \'helloworld\'', 'Modify Run Configuration...', 'Open In', 'Local History', 'Execute Line in Python Console', 'Run File in Python Console', 'Compare with Clipboard', and 'Create Gist...'.

The Run window at the bottom shows the command 'E:\tmp\venv\Scripts\python.exe E:/tmp/helloworld.' and the output 'hello world'. The process finished with exit code 0.

## Pycharm的使用小结

1. 新建\*.py文件：右击项目名称，选择快捷菜单的“Python file”，输入文件名。
2. 输入文件内容。
3. 运行\*.py文件：①利用快捷键：CTRL+SHIFT+F10；②“Project”资源管理器中找到文件单击右键，选择“Run \*.py”；③编辑窗口上绿色三角形按钮，单击运行（初次运行时不出现此按钮）。
4. 结果查看：Pycharm的运行结果在窗口的左下角。
5. 常用快捷键：见下表。

快捷键	功能说明
Ctrl + Enter	在下方新建行，但不移动光标
Shift + Enter	在下方新建行，并移到新行行首
Ctrl + /	注释(取消注释)选择的行
Ctrl + Alt + L	格式化代码(与QQ锁定热键冲突关闭QQ的热键)
Ctrl + Shift ++	展开所有的代码块
Ctrl + Shift +-	收缩所有的代码块
Ctrl + Alt + I	自动缩进行
Alt + Enter	优化代码添加包
Ctrl + Shift + F	高级查找
Alt + Shift + Q	更新代码到远程服务器
Ctrl+D:	对光标所在行的代码进行复制
Alt + Shift + F10	运行模式配置
Shift + F10	运行
Ctrl + Shift + F10	运行编辑器配置

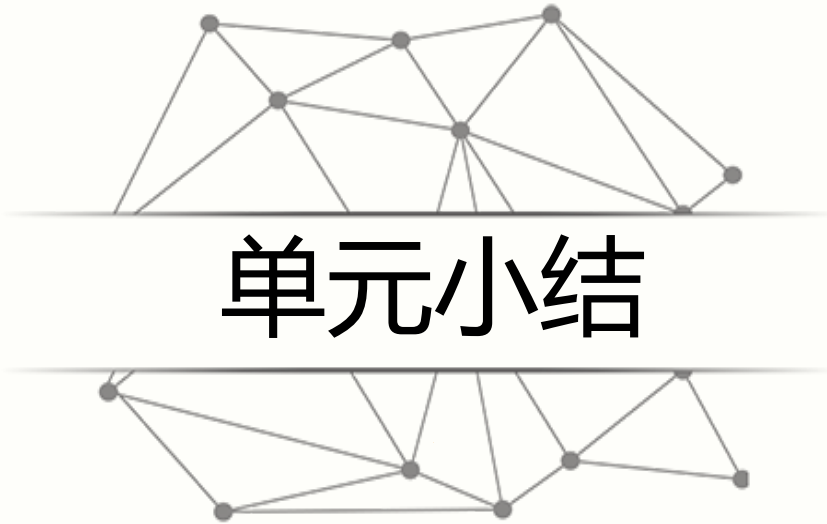
## Pycharm的安装及使用参考

<https://www.jianshu.com/p/a359299f1ab8>

<https://www.cnblogs.com/du-hong/p/10244304.html>

<https://www.jianshu.com/p/eb606812765d>





# 单元小结

# Python开发环境配置

- Python语言的发展历史
- 选取一种系统平台构建Python开发环境
- 尝试编写与运行1个Python小程序



# Python语言程序设计

## 1.3 实例1: 温度转换

---







# "温度转换"问题分析

# 温度转换

## 温度刻画的两种不同体系

- **摄氏度：中国等世界大多数国家使用**

以1标准大气压下水的结冰点为0度，沸点为100度，将温度进行100等分刻画

- **华氏度：美国、英国等国家使用**

以1标准大气压下水的结冰点为32度，沸点为212度，将温度进行180等分刻画

# 需求分析

## 两种温度体系的转换

- 摄氏度转换为华氏度
- 华氏度转换为摄氏度

那么问题是：如何利用计算机进行温度转换呢？

# 问题分析

## 该问题中计算部分的理解和确定

- **理解1：直接将温度值进行转换**
- **理解2：将温度信息发布的语音或图像形式进行理解和转换**
- **理解3：监控温度信息发布渠道，实时获取并转换温度值**

分析问题：可以从很多不同角度来理解温度转换问题的计算部分

# 问题分析

## 分析问题

- 采用 理解1：直接将温度值进行转换

温度数值需要标明温度体系，即摄氏度或华氏度

转换后也需要给出温度体系

# 问题分析

## 划分边界

- **输入：带华氏或摄氏标志的温度值**
- **处理：根据温度标志选择适当的温度转换算法**
- **输出：带摄氏或华氏标志的温度值**

即明确问题的输入数据、输出数据和对数据处理的要求

# 问题分析

## 输入输出格式设计

**标识放在温度最后，F表示华氏度，C表示摄氏度**

**82F表示华氏82度，28C表示摄氏28度**

# 问题分析

## 设计算法

根据华氏和摄氏温度定义，利用转换公式如下：

$$C = (F - 32) / 1.8$$

$$F = C * 1.8 + 32$$

其中，C表示摄氏温度，F表示华氏温度



**问题分析清楚，可以开始编程啦！**



# "温度转换"实例编写

```
#TempConvert.py
```

```
TempStr = input("请输入带有符号的温度值: ")
```

```
if TempStr[-1] in ['F', 'f']:
```

```
    C = (eval(TempStr[0:-1]) - 32)/1.8
```

```
    print("转换后的温度是{:.2f}C".format(C))
```

```
elif TempStr[-1] in ['C', 'c']:
```

```
    F = 1.8*eval(TempStr[0:-1]) + 32
```

```
    print("转换后的温度是{:.2f}F".format(F))
```

```
else:
```

```
    print("输入格式错误")
```

根据IPO描述和算法设计编写上述代码，并保存为TempConvert.py文件

# 运行效果

**IDLE打开文件，按F5运行**

>>>

请输入带有符号的温度值：**82F**  
转换后的温度是**27.78C**

>>>

>>>

请输入带有符号的温度值：**28C**  
转换后的温度是**82.40F**

>>>



"温度转换"举一反三

# 举一反三

## 输入输出的改变

- 温度数值与温度标识之间关系的设计可以改变
- 标识改变放在温度数值之前：C82, F28
- 标识字符改变为多个字符：82Ce、28Fa

# 举一反三

## 计算问题的扩展

- 温度转换问题是各类转换问题的代表性问题
- 货币转换、长度转换、重量转换、面积转换...
- 问题不同，但程序代码相似

Python语言程序设计

## 1.4 Python程序语法元素分析

---





# Python程序语法元素分析



- 程序的格式框架
- 命名与保留字
- 数据类型
- 语句与函数
- Python程序的输入输出
- "温度转换"代码分析





# 程序的格式框架

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

Python采用严格的“缩进”来表明程序的格式框架

缩进是指每一行代码开始前的空白区域，用来表示代码之间的包含于层次关系，不需要缩进的代码顶行编写，不留空白代码编写中，缩进可以用Tab键实现，也可以采用多个空格（一般是4个）实现，但两者不能混用，建议用4个空格方式

代码高亮：编程的色彩辅助体系，不是语法要求

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**缩进**：一行代码开始前的空白区域，表达程序的格式框架  
严格的缩进可以约束程序结构，有利于维护代码结构的可读性  
当存在缩进，表明这行代码在逻辑上属于之前紧邻的无缩进代码行的所属范畴

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

## 单层缩进

除了单层缩进，程序的缩进还可以“嵌套”形成多层缩进，并且可以“无限制”嵌套使用

```
DARTS = 1000
hits = 0.0
clock()
for i in range(1, DARTS):
    x, y = random(), random()
    dist = sqrt(x**2 + y**2)
    if dist <= 1.0:
        hits = hits + 1
pi = 4 * (hits/DARTS)
print("Pi的值是 {:.2f}F".format(pi))
```

## 多层缩进

# 缩进

## 缩进表达程序的格式框架

- **严格明确**：缩进是语法的一部分，缩进不正确程序运行错误
- **所属关系**：表达代码间包含和层次关系的唯一手段
- **长度一致**：程序内一致即可，一般用4个空格或1个TAB

## #TempConvert.py

```
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**注释：**用于提高代码可读性的辅助性文字，不被执行  
是程序员在代码中加入的一行或多行信息，用来对语句、函数、数据结构或方法等进行说明

# 注释

## 不被程序执行的辅助性说明信息

- 单行注释：以#开头，其后内容为注释

# 这里是单行注释

- 多行注释：以'''开头和结尾（3个单引号）

''' 这是多行注释第一行

这是多行注释第二行 '''



## #TempConvert.py

```
TempStr = input("请输入带有符号的温度值: ")
```

```
if TempStr[-1] in ['F', 'f']:
```

```
    C = (eval(TempStr[0:-1]) - 32)/1.8  
    print("转换后的温度是{:.2f}C".format(C))
```

```
elif TempStr[-1] in ['C', 'c']:
```

```
    F = 1.8*eval(TempStr[0:-1]) + 32  
    print("转换后的温度是{:.2f}F".format(F))
```

```
else:
```

```
    print("输入格式错误")
```

**缩进 注释**



命名与保留字



```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

与数学概念类似，python程序中采用“变量”来保存和表示具体的数据值

# 变量

## 用来保存和表示数据的占位符号

- 变量采用标识符(名字) 来表示，关联标识符的过程叫命名

TempStr是变量名字

- 可以使用等号(=)向变量赋值或修改值，=被称为赋值符号

TempStr = "82F"    #向变量TempStr赋值"82F"

命名是为了保证程序元素的唯一性

# 命名

## 关联标识符的过程

- **命名规则: 大小写字母、数字、下划线和中文等字符及组合**

如: TempStr, Python\_Great, 这是门Python好课

- **注意事项: 大小写敏感、首字符不能是数字、不与保留字相同**

Python和python是不同变量, 123Python是不合法的

# 保留字

被编程语言内部定义并保留使用的标识符

- Python语言（3.8版本）有35个保留字(也叫关键字)

*if, elif, else, in*

- 保留字是编程语言的基本单词，大小写敏感

*if* 是保留字，If 是变量

# 保留字

(26/35)

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

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**变量 命名 保留字**





# 数据类型

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**数据类型**：字符串、整数、浮点数、列表

# 数据类型

**10,011,101 该如何解释呢？**

- **这是一个二进制数字 或者 十进制数字**

**作为二进制数字，10,011,101的值是十进制157**

- **这是一段文本 或者 用逗号,分隔的3个数字**

**作为一段文本，逗号是文本中的一部分，一共包含10个字符**

# 数据类型

## 供计算机程序理解的数据形式

- 程序设计语言不允许存在语法歧义，需要定义数据的形式

需要给10,011,101关联一种计算机可以理解的形式

- 程序设计语言通过一定方式向计算机表达数据的形式

"123"表示文本字符串123，123则表示数字123

# 数据类型

**10,011,101**

- **整数类型：** 10011101
- **字符串类型：** "10,011,101"
- **列表类型：** [10, 011, 101]

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**字符串**：由0个或多个字符组成的有序字符序列

# 字符串

由0个或多个字符组成的有序字符序列

- 字符串由一对单引号或一对双引号表示

"请输入带有符号的温度值:" 或者 'C'

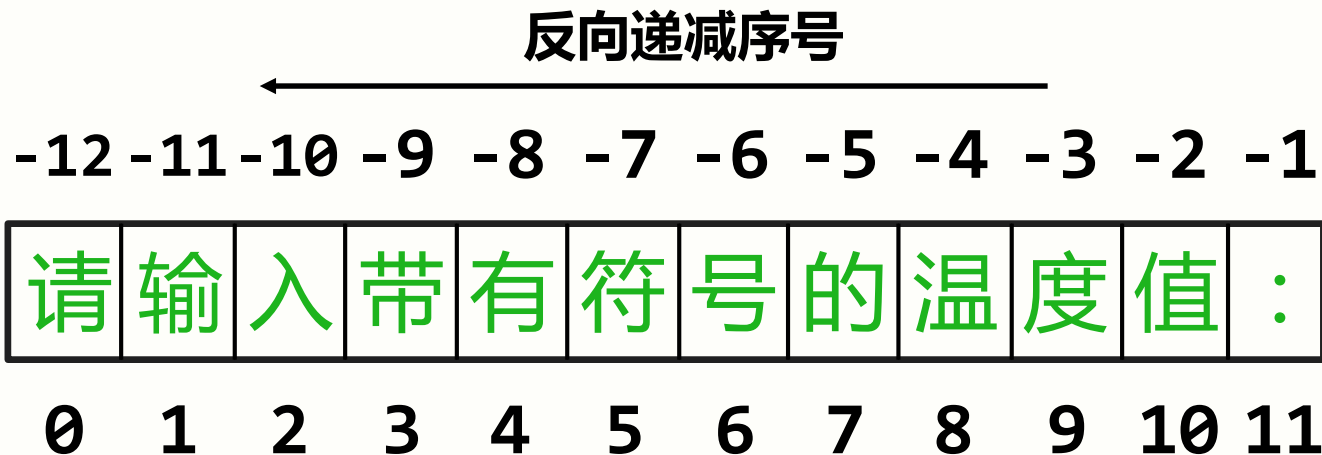
- 字符串是字符的有序序列，可以对其中的字符进行索引

"请" 是 "请输入带有符号的温度值:" 的第0个字符

字符串包括两种序号体系：正向递增序号和反向递减序号

# 字符串的序号

## 正向递增序号 和 反向递减序号



正向递增序号以最左侧字符序号为0，向右依次递增，最右侧字符序号为L-1

反向递减序号以最右侧字符序号为-1，向左依次递减，最左侧字符序号为-L



# 字符串的使用

## 使用[ ]获取字符串中一个或多个字符

- 索引：返回字符串中单个字符      <字符串>[M]

"请输入带有符号的温度值: "[0]    或者    TempStr[-1]

- 切片：返回字符串中一段字符串      <字符串>[M: N]

"请输入带有符号的温度值: "[1:3]    或者    TempStr[0:-1]

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**数字类型：整数和浮点数**

# 数字类型

整数和浮点数都是数字类型

- **整数：数学中的整数**

32    或者    -89

- **浮点数：数学中的实数，带有小数部分**

1.8    或者    -1.8    或者    -1.0

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**列表类型**：由0个或多个数据组成的有序序列

# 列表类型

由0个或多个数据组成的有序序列

- 列表使用[ ]表示，采用逗号(,)分隔各元素


[ 'F', 'f' ]表示两个元素'F'和'f'

- 使用保留字 in 判断一个元素是否在列表中

TempStr[-1] in [ 'C', 'c' ]判断前者是否与列表中某个元素相同

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**字符串 整数 浮点数 列表**



# 语句与函数

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

## 赋值语句：由赋值符号构成的一行代码

即将右侧的计算结果赋给左侧变量



# 赋值语句

## 由赋值符号构成的一行代码

- 赋值语句用来给变量赋予新的数据值

`C=(eval(TempStr[0:-1])-32)/1.8` #右侧运算结果赋给变量C

- 赋值语句右侧的数据类型同时作用于变量

`TempStr=input("")` #input()返回一个字符串，TempStr也是字符串

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**分支语句**：由判断条件决定程序运行方向的语句

是控制程序运行的一类重要语句

# 分支语句

## 由判断条件决定程序运行方向的语句

- 使用保留字 *if elif else* 构成条件判断的分支结构

*if* TempStr[-1] in ['F','f'] : #如果条件为True则执行冒号后语句

- 每个保留字所在行最后存在一个冒号(:)，语法的一部分

冒号及后续缩进用来表示后续语句与条件的所属关系

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**函数**：根据输入参数产生不同输出的功能过程

# 函数

根据输入参数产生不同输出的功能过程

- 类似数学中的函数， $y = f(x)$

```
print("输入格式错误") #打印输出 "输入格式错误"
```

- 函数采用 <函数名>(<参数>) 方式使用

```
eval(TempStr[0:-1]) # TempStr[0:-1]是参数
```

eval()函数能够以python表达式的方式解析并执行字符串，并将返回结果输出

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**赋值语句   分支语句   函数**



# Python程序的输入输出

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值: ")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**input() : 从控制台获得用户输入的函数**



# 输入函数 input()

## 从控制台获得用户输入的函数

- input()函数的使用格式：

<变量> = input(<提示信息字符串>)

- 用户输入的信息以字符串类型保存在<变量>中

```
TempStr = input("请输入") # TempStr保存用户输入的信息
```

无论用户在控制台输入什么内容，input函数都以字符串类型返回结果

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**print() : 以字符形式向控制台输出结果的函数**

# 输出函数 print()

## 以字符形式向控制台输出结果的函数

- print()函数的基本使用格式：

`print(<拟输出字符串或字符串变量>)`

- 字符串类型的一对引号仅在程序内部使用，输出无引号

`print("输入格式错误")` # 向控制台输出 输入格式错误

当输出纯字符信息时，可以直接将待输出内容传递给print()函数

# 输出函数 print()

## 以字符形式向控制台输出结果的函数

### - print()函数的格式化：

```
print("转换后的温度是{:.2f}C".format(C))
```



**{ }**表示槽，后续变量填充到槽中

**{:.2f}**表示将变量C填充到这个位置时取小数点后2位

当输出变量值时，需要采用格式化方式输出，通过format()方法将待输出变量整理成期望输出的格式

# 输出函数 print()

以字符形式向控制台输出结果的函数

```
print("转换后的温度是{:.2f}C".format(C))
```

如果C的值是 123.456789 , 则输出结果为 :

转换后的温度是123.45C

```
#TempConvert.py
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32)/1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误")
```

**eval()** : 去掉参数最外侧引号并执行余下语句的函数

# 评估函数 eval()

去掉参数最外侧引号并执行余下语句的函数

- eval()函数的基本使用格式：

`eval(<字符串或字符串变量>)`

```
>>> eval("1")
```

```
1
```

```
>>> eval("1+2")
```

```
3
```

```
>>> eval('"1+2"')
```

```
'1+2'
```

```
>>> eval('print("Hello")')
```

```
Hello
```

# 评估函数 eval()

去掉参数最外侧引号并执行余下语句的函数

```
eval(TempStr[0:-1])
```

如果TempStr[0:-1]值是"12.3"，输出是：

12.3



```
#TempConvert.py
```

```
TempStr = input("请输入带有符号的温度值: ")
```

```
if TempStr[-1] in ['F', 'f']:
```

```
    C = (eval(TempStr[0:-1]) - 32)/1.8
```

```
    print("转换后的温度是{:.2f}C".format(C))
```

```
elif TempStr[-1] in ['C', 'c']:
```

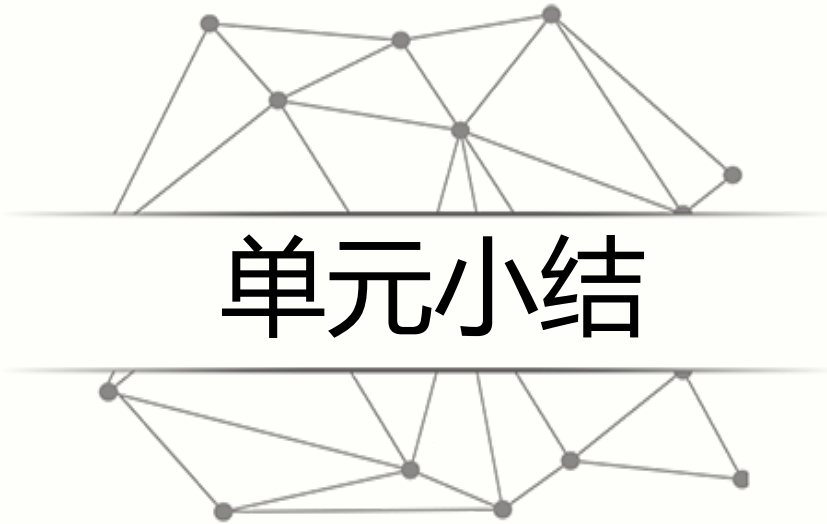
```
    F = 1.8*eval(TempStr[0:-1]) + 32
```

```
    print("转换后的温度是{:.2f}F".format(F))
```

```
else:
```

```
    print("输入格式错误")
```

**input()   print()   eval()**



# 单元小结

# Python程序语法元素分析

- 缩进、注释、命名、变量、保留字
- 数据类型、字符串、整数、浮点数、列表
- 赋值语句、分支语句、函数
- input()、print()、eval()、 print()格式化



