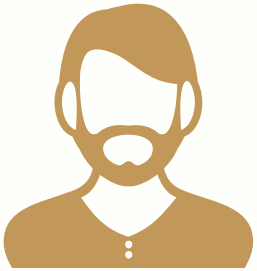
Python语言程序设计

第1章 程序设计基本方法

本课概要

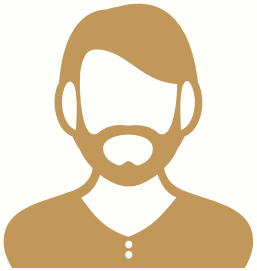
第1章 Python基本语法元素



- 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

python
语言程序设计

第1章 Python基本语法元素



方法论

- 程序的基本编写方法：IPO

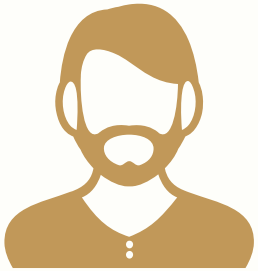
实践能力

- 看懂10行左右简单Python代码

练习与作业

第1章 Python基本语法元素

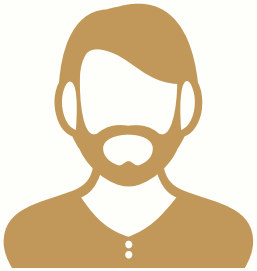
练习



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

计算机与程序设计

计算机的概念

计算机是根据指令操作数据的设备

- **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

- **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

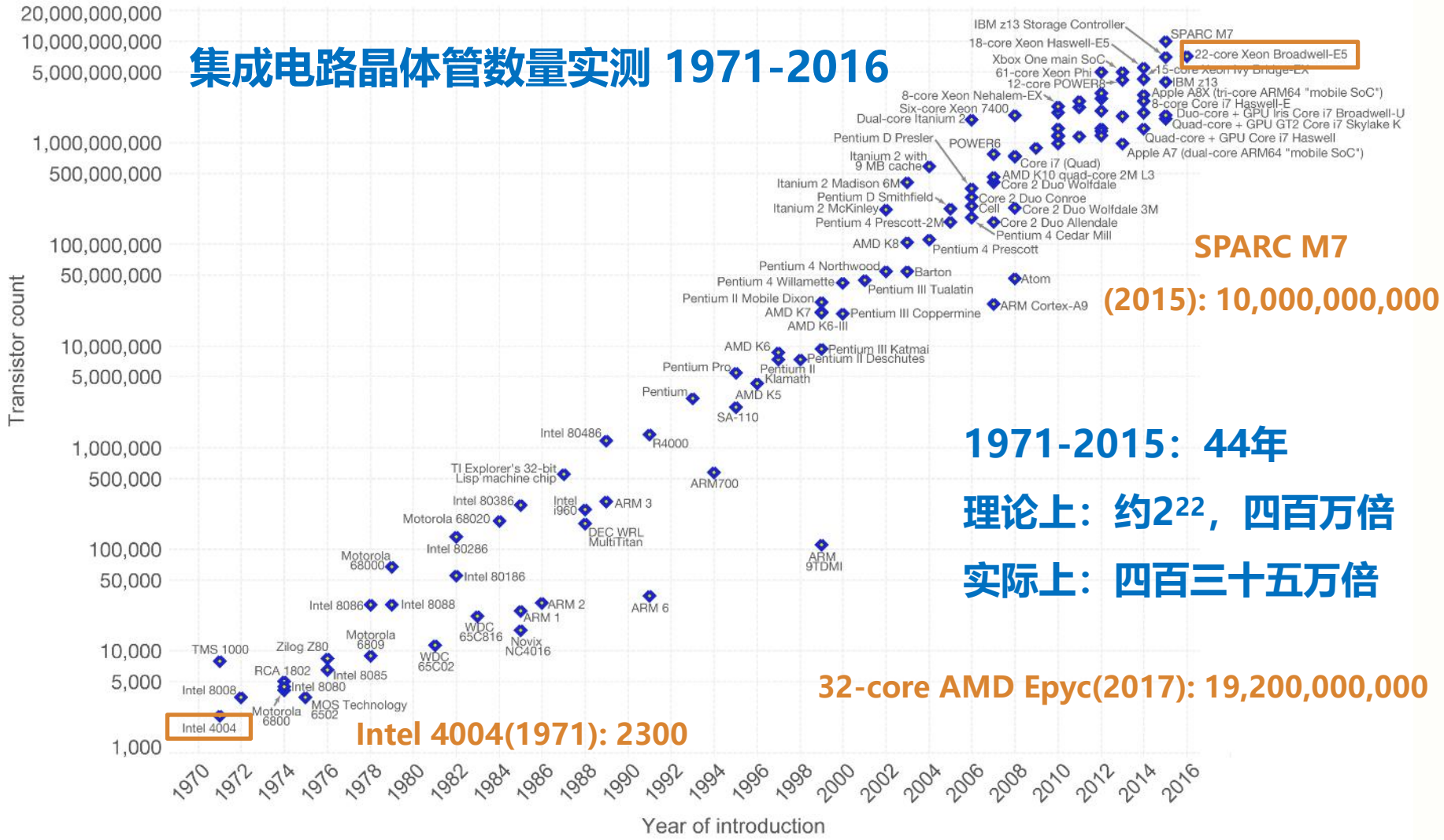
计算机的发展参照摩尔定律，表现为指数方式

- **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**

摩尔定律 Moore's Law

计算机发展历史上最重要的预测法则

- Intel公司创始人之一戈登•摩尔在1965年提出
- 单位面积集成电路上可容纳晶体管的数量约每两年翻一番
- CPU/GPU、内存、硬盘、电子产品价格等都遵循摩尔定律


集成电路晶体管数量实测 1971-2016
Transistor count
Year of introduction
20,000,000,000
10,000,000,000
5,000,000,000
1,000,000,000
500,000,000
100,000,000
50,000,000
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
IBM z13 Storage Controller
SPARC M7
18-core Xeon Haswell-E5
22-core Xeon Broadwell-E5
Xbox One main SoC
61-core Xeon Phi
15-core Xeon Ivy Bridge-EX
12-core POWER8
IBM z13
Apple A8X (tri-core ARM64 "mobile SoC")
8-core Xeon Nehalem-EX
8-core Core i7 Haswell-E
Six-core Xeon 7400
Duo-core + GPU Iris Core i7 Broadwell-U
Dual-core Itanium 2
Quad-core + GPU GT2 Core i7 Skylake K
Pentium D Presler
POWER6
Quad-core + GPU Core i7 Haswell
Itanium 2 with 9 MB cache
Core i7 (Quad)
Apple A7 (dual-core ARM64 "mobile SoC")
AMD K10 quad-core 2M L3
Itanium 2 Madison 6M
Core 2 Duo Wolfdale
Pentium D Smithfield
Core 2 Duo Conroe
Itanium 2 McKinley
Cell
Core 2 Duo Wolfdale 3M
Pentium 4 Prescott-2M
Core 2 Duo Allendale
Pentium 4 Cedar Mill
AMD K8
Pentium 4 Prescott
Pentium 4 Northwood
Barton
Atom
Pentium 4 Willamette
Pentium III Tualatin
Pentium II Mobile Dixon
ARM Cortex-A9
AMD K7
Pentium III Coppermine
AMD K6-III
AMD K6
Pentium III Katmai
Pentium II Deschutes
Pentium Pro
Pentium II Klamath
Pentium
AMD K5
SA-110
Intel 80486
R4000
TI Explorer's 32-bit Lisp machine chip
ARM700
Intel 80386
Intel i960
ARM 3
Motorola 68020
DEC WRL MultiTitan
Intel 80286
Motorola 68000
ARM 9TDMI
Intel 80186
Intel 8086
Intel 8088
ARM 2
ARM 1
ARM 6
WDC 65C816
Motorola 6809
Novix NC4016
TMS 1000
Zilog Z80
WDC 65C02
RCA 1802
Intel 8085
Intel 8080
Intel 8008
MOS Technology 6502
Motorola 6800
Intel 4004
SPARC M7
(2015): 10,000,000,000
1971-2015：44年
理论上：约2^22，四百万倍
实际上：四百三十五万倍
32-core AMD Epyc(2017): 19,200,000,000
Intel 4004(1971): 2300

计算机的发展

计算机的发展参照摩尔定律，表现为指数方式

- **当今世界，唯一长达50年有效且按照指数发展的技术领域**
- **计算机深刻改变人类社会，甚至可能改变人类本身**
- **可预见的未来30年，摩尔定律还将持续有效**

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

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

编程语言种类很多，但生命力强劲的却不多

- **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，它是最流行最好用的编程语言**

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**

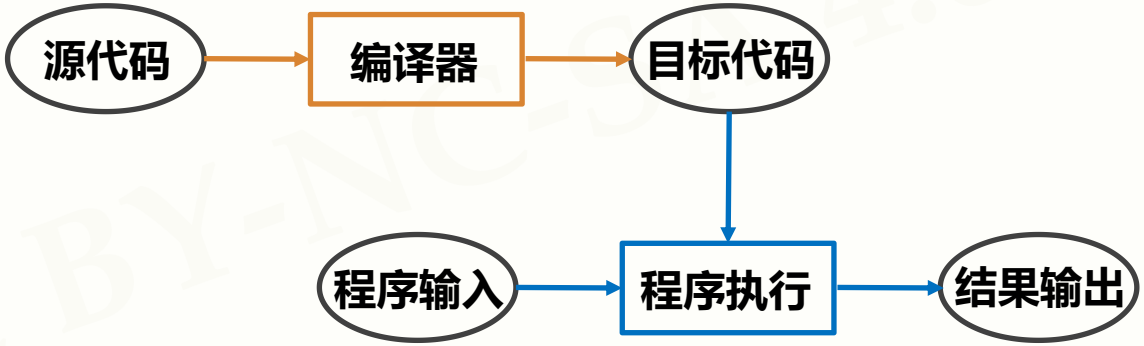
  例如：result = 2 + 3

- **目标代码：计算机可直接执行，人类不可读 (专家除外)**

  例如：11010010 00111011

编译

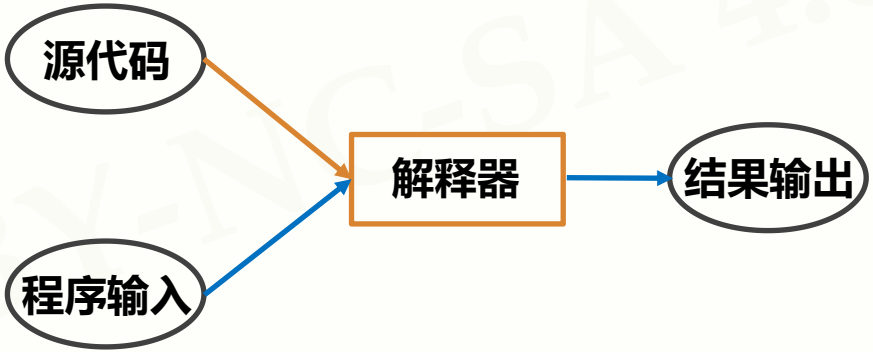
将源代码一次性转换成目标代码的过程



执行编译过程的程序叫作编译器

解释

将源代码逐条转换成目标代码同时逐条运行的过程



执行解释过程的程序叫作解释器

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

- **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

- **脚本语言：执行程序时需要源代码，维护更灵活**

  源代码在维护灵活、跨多个操作系统平台

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

理解IPO

输入

- **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

- **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

- 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

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
- 划分边界：划分问题的功能边界，规划IPO
- 设计算法：设计问题的求解算法，关注算法

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**
- **调试测试：调试程序使正确运行，运行调试**
- **升级维护：适应问题的升级维护，更新完善**

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：确保程序按照正确逻辑能够正确运行**

计算机编程

Q：为什么要学习计算机编程?

A：因为“编程是件很有趣的事儿”！

计算机编程

编程能够训练思维

- 编程体现了一种抽象交互关系、自动化执行的思维模式
- 计算思维：区别逻辑思维和实证思维的第三种思维模式
- 能够促进人类思考，增进观察力和深化对交互关系的理解

计算机编程

编程能够增进认识

- 编程不单纯是求解计算问题
- 不仅要思考解决方法，还要思考用户体验、执行效率等
- 能够帮助程序员加深用户行为以及社会和文化认识

计算机编程

编程能够带来乐趣

- 编程能够提供展示自身思想和能力的舞台
- 让世界增加新的颜色、让自己变得更酷、提升心理满足感
- 在信息空间里思考创新、将创新变为现实

计算机编程

编程能够提高效率

- 能够更好地利用计算机解决问题
- 显著提高工作、生活和学习效率
- 为个人理想实现提供一种借助计算机的高效手段

计算机编程

编程带来就业机会

- 程序员是信息时代最重要的工作岗位之一
- 国内外对程序员岗位的缺口都在百万以上规模
- 计算机已经渗透于各个行业， 就业前景非常广阔

学习编程的误区

Q：编程很难学吗？ A：掌握方法就很容易！

- **首先，掌握编程语言的语法，熟悉基本概念和逻辑**
- **其次，结合计算问题思考程序结构，会使用编程套路**
- **最后，参照案例多练习多实践，学会举一反三次**

编程辣么好，还等什么？努力学习吧！

单元小结

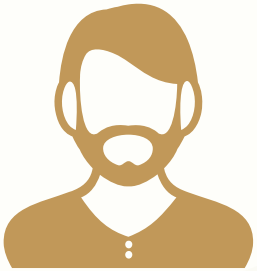
程序设计基本方法

- 计算机的功能性和可编程性
- 编译和解释、静态语言和脚本语言
- 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

Python语言概述

Python [`paiθən]，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

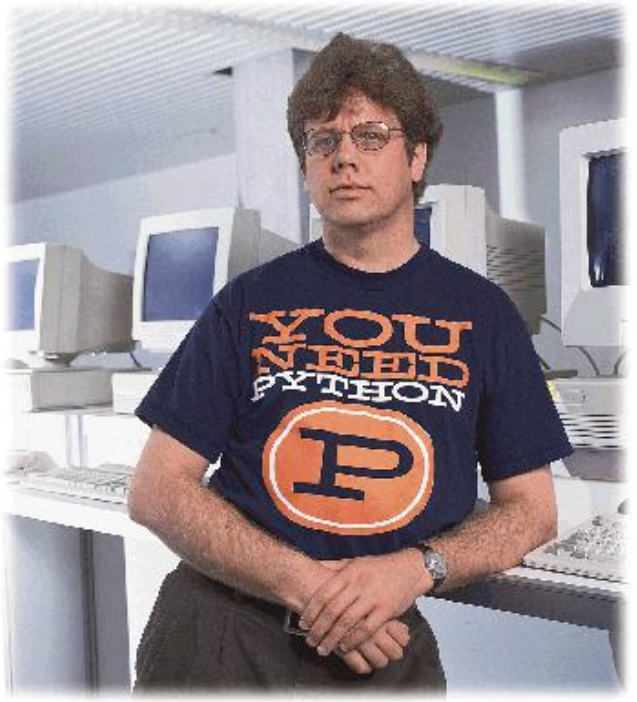
Python语言的诞生

Guido van Rossum

Python语言创立者

2002年，Python 2.x

2008年，Python 3.x

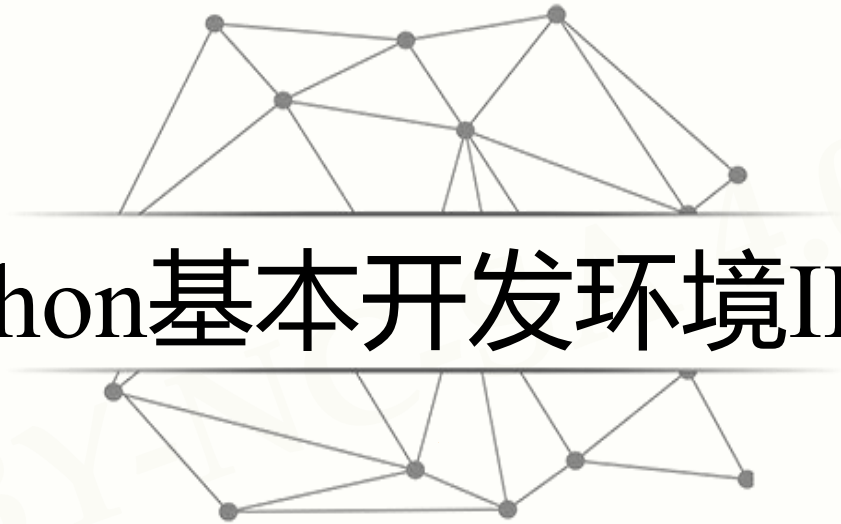


Monty Python组合

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**

Python程序编写与运行

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**

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)
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
```

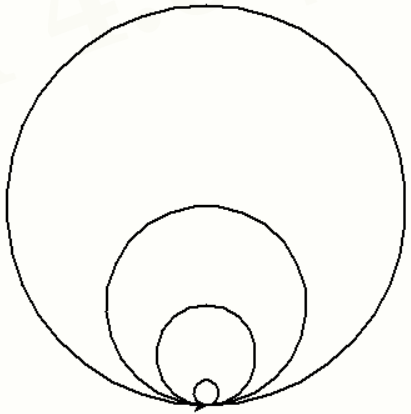
保存为CalCircle.py文件并运行

文件式

实例2: 同切圆绘制

绘制多个同切圆

```
import turtle
turtle.pensize(2)
turtle.circle(10)
turtle.circle(40)
turtle.circle(80)
turtle.circle(160)
```



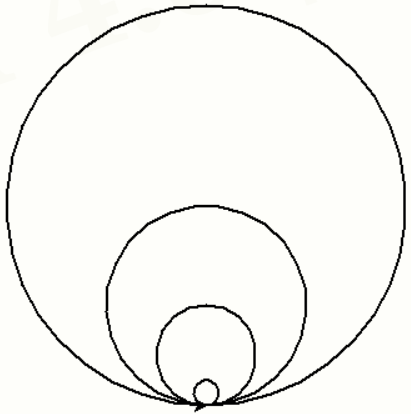
保存为TangentCirclesDraw.py文件并运行

文件式

实例2: 同切圆绘制

绘制多个同切圆

```
>>> import turtle
>>> turtle.pensize(2)
>>> turtle.circle(10)
>>> turtle.circle(40)
>>> turtle.circle(80)
>>> turtle.circle(160)
```



交互式

实例3: 五角星绘制

绘制一个五角星

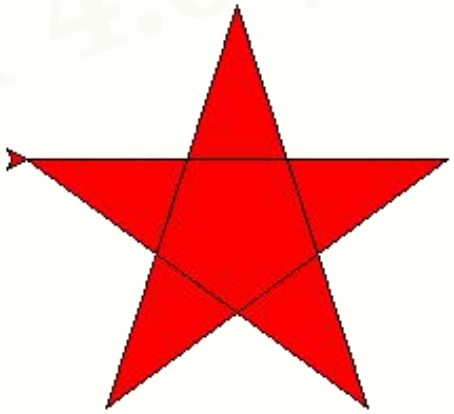
```
>>> from turtle import *
>>> color('red', 'red')
>>> begin_fill()
>>> for i in range(5):
        fd(200)
        rt(144)

>>> end_fill()
>>>
```

交互式

实例3: 五角星绘制

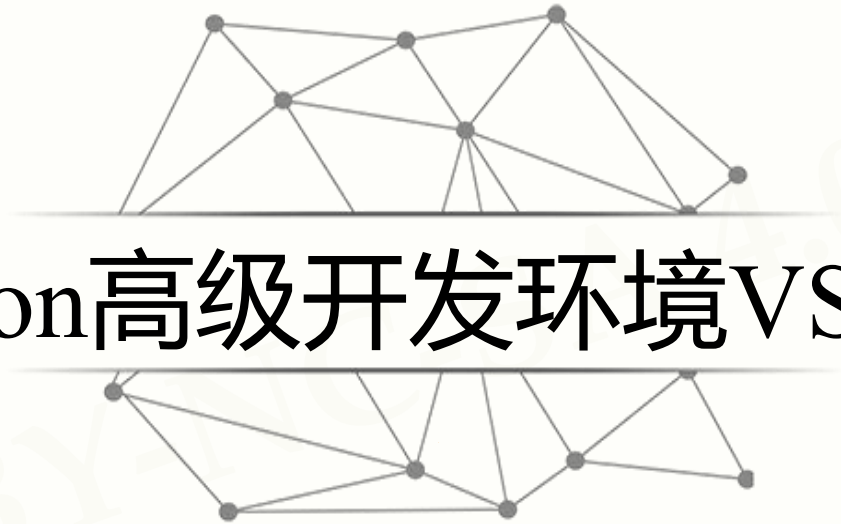
绘制一个五角星

```
from turtle import *
color('red', 'red')
begin_fill()
for i in range(5):
    fd(200)
    rt(144)
end_fill()
done()
```



保存为StarDraw.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**

Visual Studio Code Docs Updates Blog API Extensions FAQ Search Docs Download

Version 1.31 is now available! Read about the new features and fixes from January.

Download Visual Studio Code

Free and open source. Integrated Git, debugging and extensions.

Windows
Windows 7, 8, 10

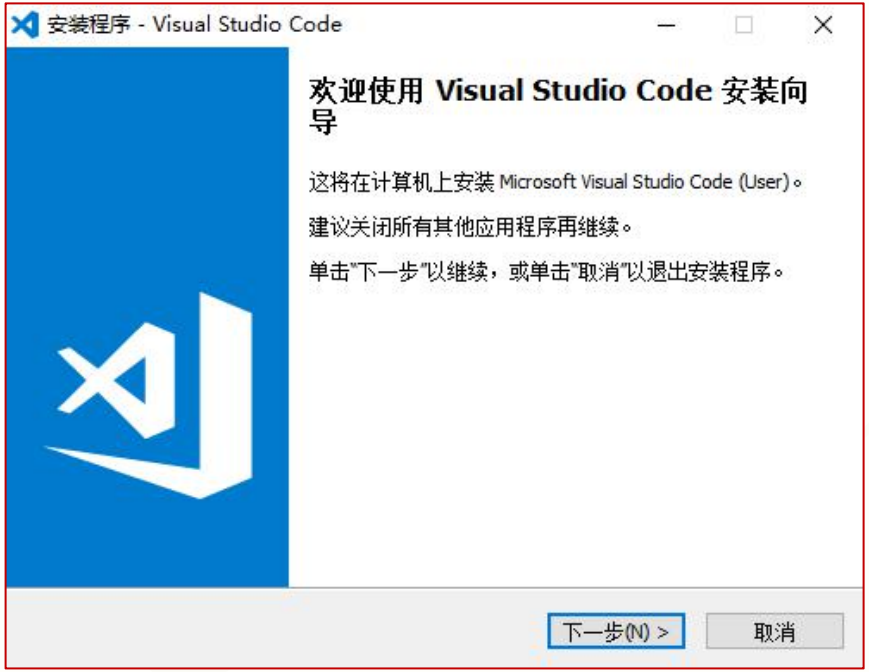
User Installer	64 bit	32 bit
System Installer	64 bit	32 bit
.zip	64 bit	32 bit

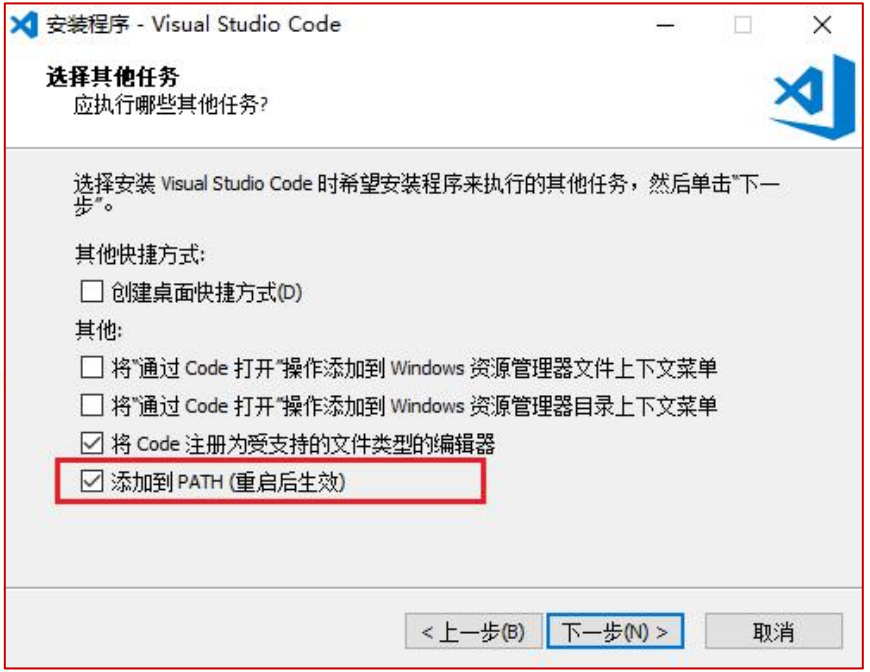
.deb
Debian, Ubuntu

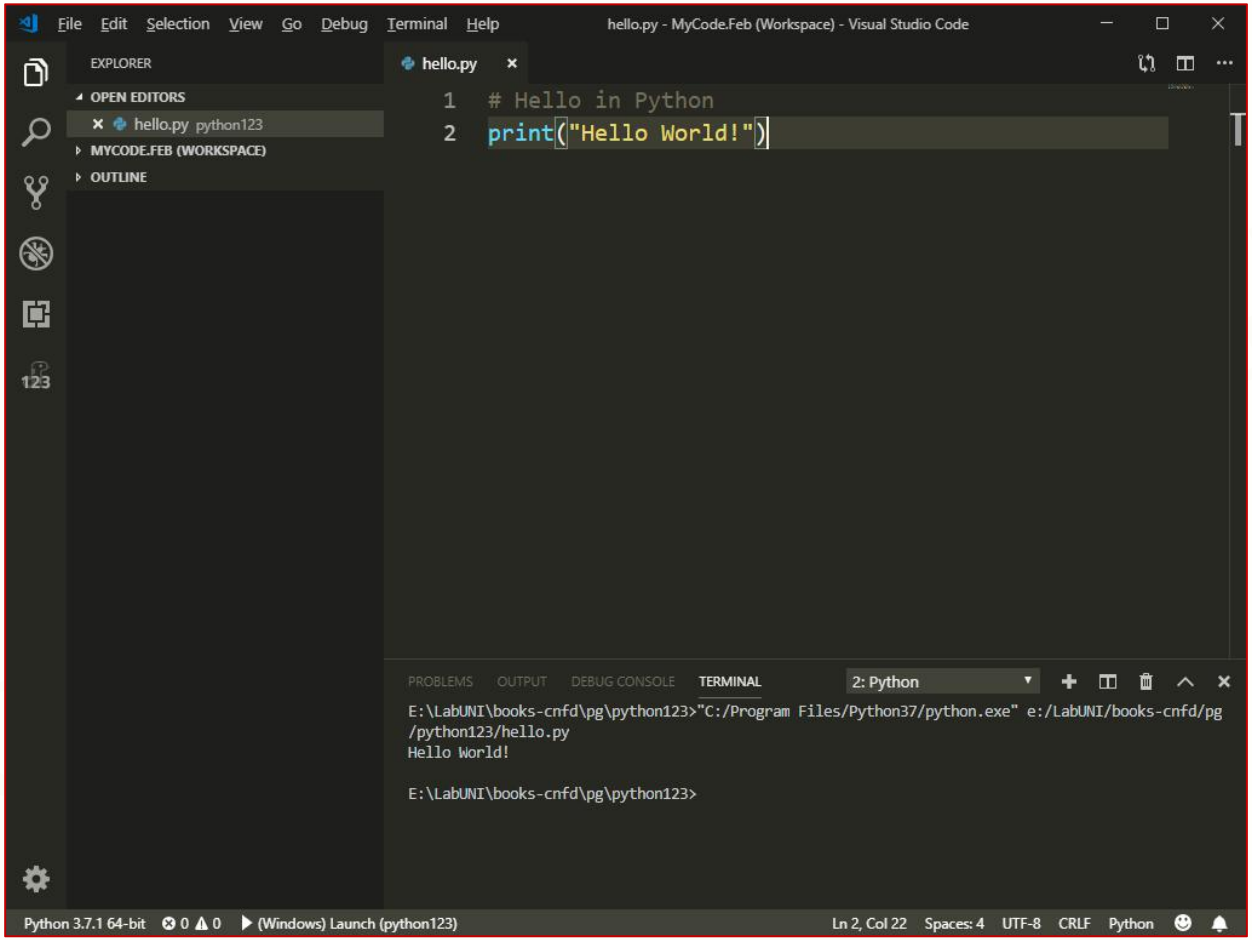
.rpm
Red Hat, Fedora, SUSE

.deb	64 bit	32 bit
.rpm	64 bit	32 bit
.tar.gz	64 bit	32 bit

Mac
macOS 10.9+


安装程序 - Visual Studio Code
欢迎使用 Visual Studio Code 安装向导
这将在计算机上安装 Microsoft Visual Studio Code (User)。
建议关闭所有其他应用程序再继续。
单击"下一步"以继续，或单击"取消"以退出安装程序。
下一步(N) >
取消


安装程序 - Visual Studio Code
选择其他任务
应执行哪些其他任务?
选择安装 Visual Studio Code 时希望安装程序来执行的其他任务，然后单击"下一步"。
其他快捷方式:
创建桌面快捷方式(D)
其他:
将"通过 Code 打开"操作添加到 Windows 资源管理器文件上下文菜单
将"通过 Code 打开"操作添加到 Windows 资源管理器目录上下文菜单
将 Code 注册为受支持的文件类型的编辑器
添加到 PATH (重启后生效)
< 上一步(B)
下一步(N) >
取消


File Edit Selection View Go Debug Terminal Help
hello.py - MyCode.Feb (Workspace) - Visual Studio Code
EXPLORER
OPEN EDITORS
hello.py python123
MYCODE.FEB (WORKSPACE)
OUTLINE
hello.py
1 # Hello in Python
2 print("Hello World!")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python
E:\LabUNI\books-cnfd\pg\python123>"C:/Program Files/Python37/python.exe" e:/LabUNI/books-cnfd/pg/python123/hello.py
Hello World!
E:\LabUNI\books-cnfd\pg\python123>
Python 3.7.1 64-bit
0 0
(Windows) Launch (python123)
Ln 2, Col 22 Spaces: 4 UTF-8 CRLF Python

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3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

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：将温度信息发布的声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

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

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")
```

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

```
#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")
```

举一反三

Python语法元素理解

- 温度转换程序共10行代码，但包含很多语法元素
- 清楚理解这10行代码能够快速入门Python语言
- 参考框架结构、逐行分析、逐词理解

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

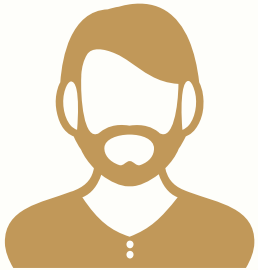
计算问题的扩展

- 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

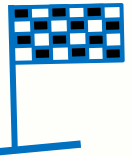
Python语言程序设计

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



python
语言程序设计

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")
```

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

```
#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

```
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

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**

```
'''     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")
```

变量：程序中用于保存和表示数据的占位符号

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

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**

字符串的序号

正向递增序号 和 反向递减序号

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子串　　<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

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

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
```

```
#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
```

```
#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
```

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

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
```

```python
#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()

"温度转换"代码分析

```
#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")
```

"温度转换"实例代码逐行分析

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