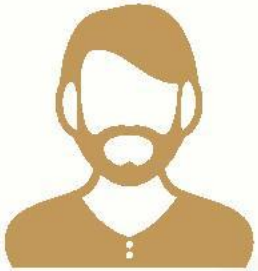
Python 程序设计

第 1 章 Python基础

Python语言程序设计

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、输出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

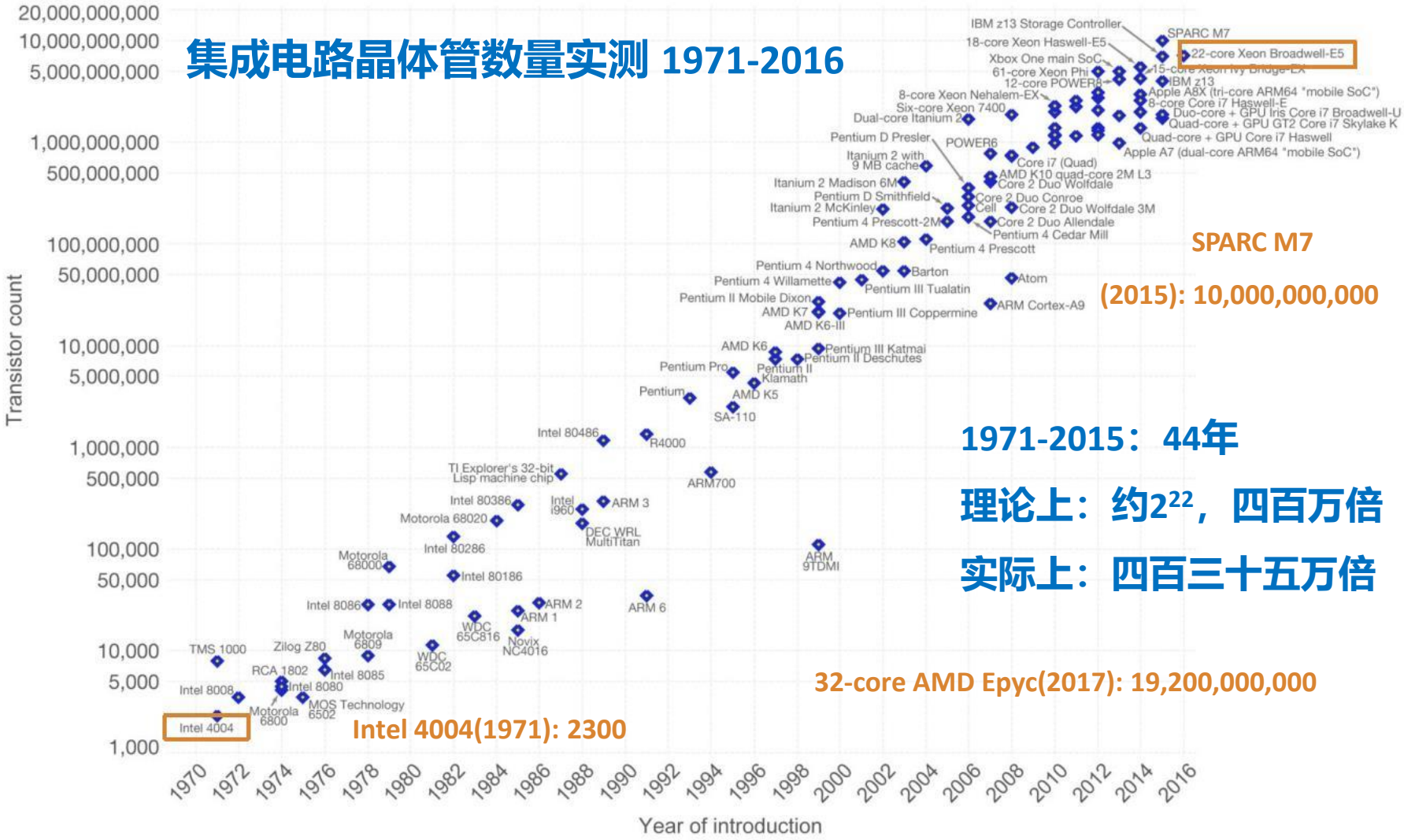
计算机的发展参照摩尔定律，表现为指数方式

- **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
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
ARM 9TDMI
Motorola 68000
Intel 80186
Intel 8086
Intel 8088
ARM 2
ARM 1
ARM 6
WDC 65C816
Novix NC4016
Motorola 6809
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

计算机技术发展的时代性总结为4个阶段。

第一阶段:1946-1981年，“计算机系统结构阶段”。这个阶段始于1946年，以全球首台数字计算机 ENIAC诞生为标志。在这个阶段，计算机技术主要围绕计算机系统结构设计开展，服务于科学计算和商业数值类计算，产生了超级计算机、高性能计算机、工作站、个人计算机等不同类型的计算机系统。这个阶段的计算需求催生了执行高效的C语言(1972年)C语言的可编程性体现在通过指针优化底层内存的使用，进而使程序在有限计算资源下高速运行。计算机技术的第一个阶段持续了35年，随着以BMPC为代表的个人计算机的诞生(1981年)，计算机技术进入了面向大众的新阶段。

第二阶段:1982-2007年，“计算机网络和视窗阶段”。这个阶段始于1982年，以面向全球子网间组网的TCP/P网络协议的标准化为标志，互联网(Internet，最初含义是连接子网的网络)时代到来了。在这个阶段，计算机技术主要围绕网络技术、视窗技术、多媒体技术发展，以个人计算机和服务器为主要计算平台，由此诞生了具备跨平台功能的Java语言(1995年)。与此同时，由于微软 Windows操作系统在个人计算机领域的高度普及，视窗应用“所见即所得”的开发需求催生了 Visual C+(vC) Visual Basic(vB)(1991年)等视窗编程语言。计算机技术的第一个阶段持续了25年，随着美国苹果公司正home智能手机的推出(2001年)和广泛普及，计算机技术进入了面向移动网络应用的新阶段。

第三阶段:2008年至今，“复杂信息系统阶段”这个阶段始于2008年，以安卓(Andriod开源移动操作系统的发布为起点，一批新的计算概念和技术几乎同时提出并显著推动了计算技术的升级换代，这些概念包括移动互联网、多核众核、云计算、可信计算、大数据、可穿戴计算、物联网、互联网+等。这些概念的提出反映了计算平台和应用的多样性，也带来了更复杂的安全问题。虽然概念很多，但没有以哪个概念为主引领技术发展，这说明计算机技术的发展已经进入了复杂信息系统阶段，这个阶段很难有任何一个技术领域独领风骚，任何系统都需要不间断地完善才能够提供更加安全可靠及更佳用户体验的功能，系统之间通过网络、开源项目和社交关系等高度关联，人类将会逐渐认识到计算机系统的复杂性会到达人类所能掌控的边界。面对复杂的功能性和紧迫的迭代周期，计算机需要更高抽象级别的程序设计语言来表达可编程性， Python语言(2008年3.0版本)已经成为这个阶段计算机系统的主流编程语言。

第四阶段：约20年后某个时期开始，“人工智能阶段”随着深度学习、开源硬件、智能机器人、在线搜索引擎、量子计算等技术的发展，未来某个时期将会出现人工智能主导计算的技术阶段，计算机技术将结合智能技术展示更加友好的交互方式和用户体验。此时计算机或许已经没有了独立的载体，它将通过网络、数据和机器整合切可用自然资源，逐步接管人类所有非创造性工作，计算机技术将进入个未知的新阶段。

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**

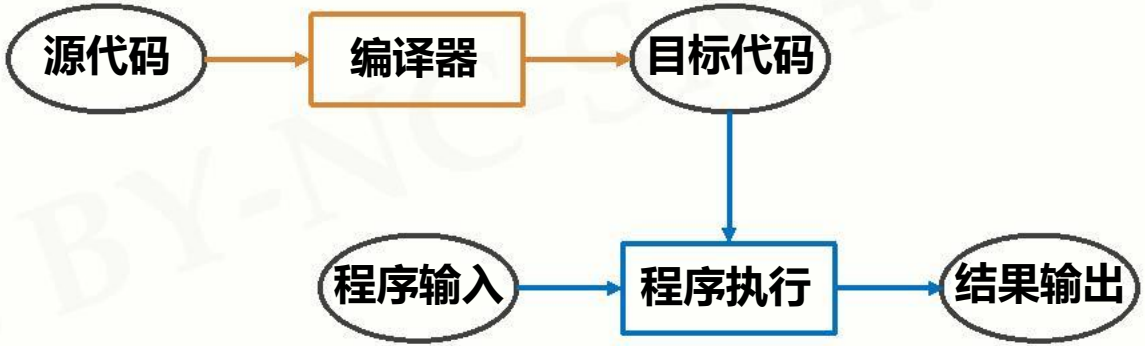
  例如：result = 2 + 3

- **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



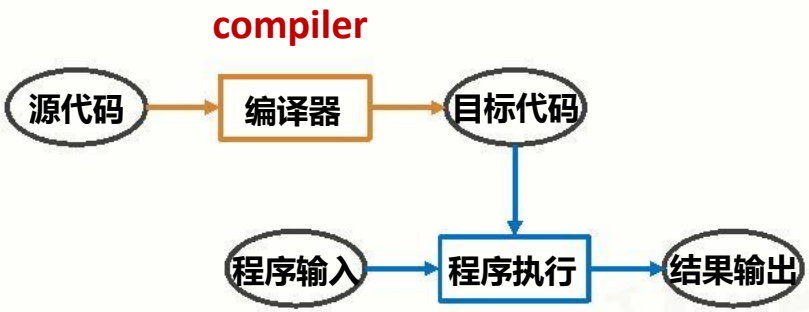
执行编译过程的程序叫作编译器

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，在同类操作系统上使用灵活。**

  C/C++语言、Java语言

- **脚本语言：使用解释执行的编程语言，可在任意操作系统中运行。可移植性好。**

  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**

程序设计基本方法

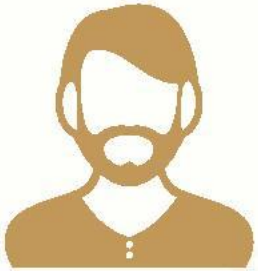
- 计算机的功能性和可编程性
- 编译和解释、静态语言和脚本语言
- IPO、理解问题的计算部分
- 掌握计算机编程的价值

Python语言程序设计

Python开发环境配置

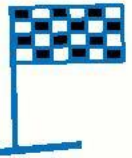
单元开篇

Python开发环境配置



- Python语言概述
- Python语言Windows系统开发环境
- Python语言Mac系统开发环境
- Python语言Linux系统开发环境
- Python语言Web开发环境
- Python程序编写与运行

三选一



python
语言程序设计

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

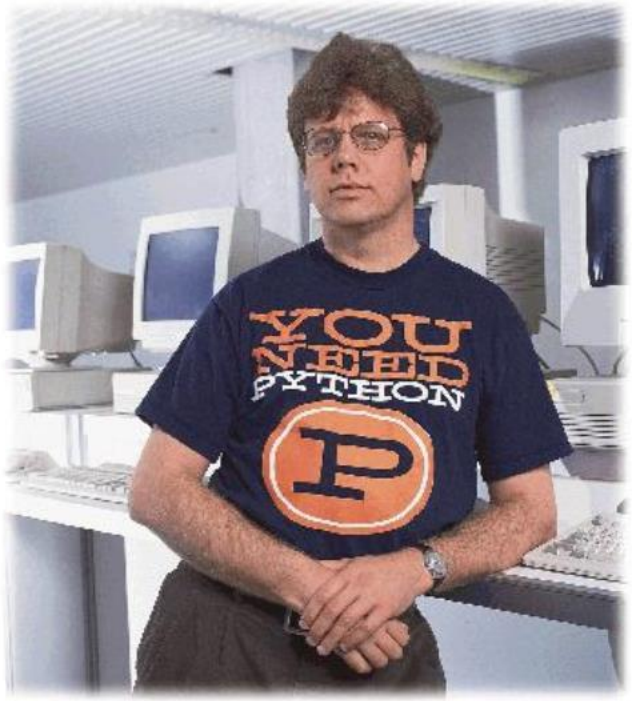
Python语言的诞生

Guido van Rossum

Python语言创立者

2002年，Python 2.x

2008年，Python 3.x



Python，由荷兰人Guido van Rossum于1989年发明，第一个公开发行版发行于1991年。

作为python的作者，是什么促使他设计了这个语言呢？ 在80年代，个人电脑的配置低，程序员不得不努力思考如何最大化利用空间，让guido感到苦恼。他认为这样编写程序实在是太过于耗费时间，于是他想到了shell。shell可以像胶水一样，将UNIX下的许多功能连接在一起。许多C语言下上百行的程序，在shell下只用几行就可以完成。然而，shell的本质是调用命令，它并不是一个真正的语言，shell不能全面的调动计算机的功能。 后来他进入CWI（数学和计算机研究所）工作，并参加了ABC语言的开发。1989年圣诞节期间，在阿姆斯特丹，Guido为了打发圣诞节的无趣，决心开发一个新的脚本解释程序，做为ABC 语言的一种继承。而取名python，是取自他挚爱的一部电视剧Monty Python’s Flying Circus 。1991年，第一个Python编译器诞生，它是用C语言实现的，guido为防止重蹈ABC的覆辙，着重注意python的可扩展性，并且也沿用了C中的大部分语法习惯，而这，使python得到guido同事的欢迎。他们迅速的反馈使用意见，并参与到Python的改进。

1990年代初，计算机的性能大大提高。许多程序员以及资深计算机用户频繁使用Internet进行交流，这使得python没有了硬件上的束缚与传播上的困难，再加上python易于使用的特点，使python得到了一定程度上的传播。python相当的开放，任何人可对python进行拓展或改造。由Guido决定是否将新的特征加入到python或者标准库中。后来的python2.0，转为完全开源的开发方式，python的数据库的扩展速度与传播速度也由此更进一步。 到今天，Python的框架已经确立。Python语言以对象为核心组织代码(Everything is object)，支持多种编程范式(multi-paradigm)，采用动态类型(dynamic typing)，自动进行内存回收(garbage collection)。Python支持解释运行(interpret)，并能调用C库进行拓展。Python有强大的标准库 (battery included)。这也是python相较于C，java一类语言的优势。

Monty Python组合

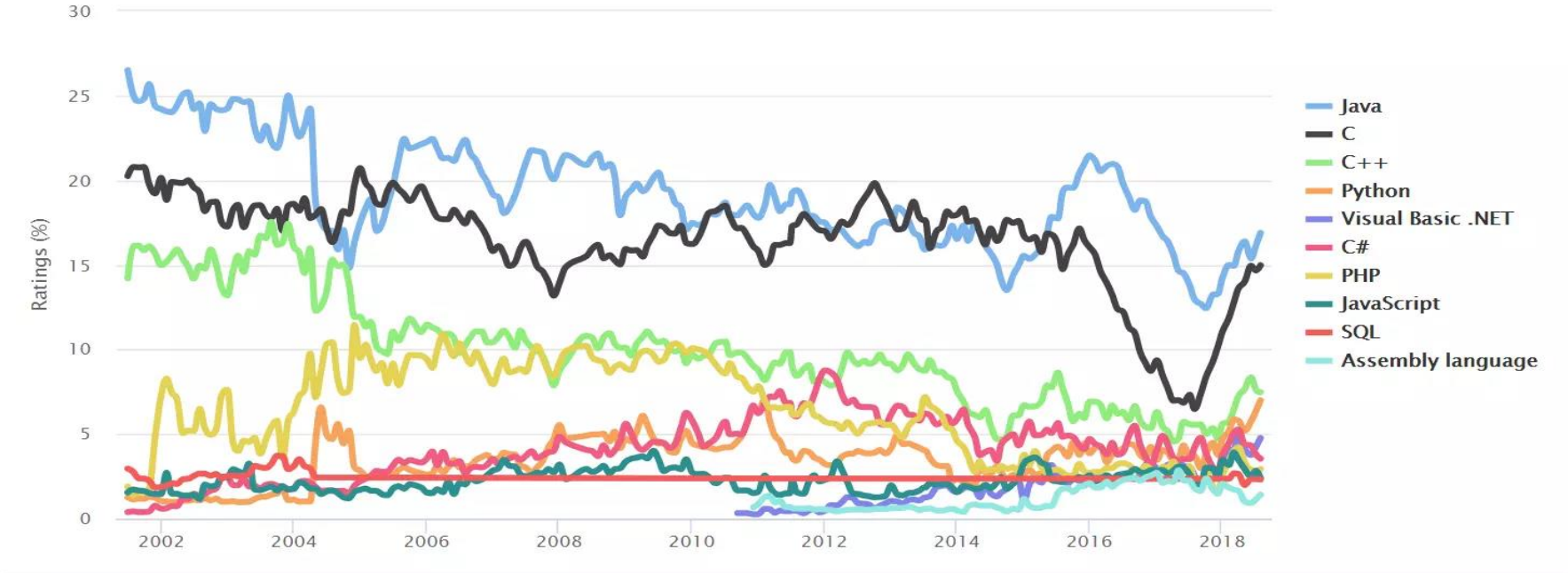

It's ...
MONTY PYTHON'S
FLYING CIRCUS
SPAM

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

TIOBE INDEX:编程语言流行程度排行榜



Python语言的特点

1、语法简洁
2、与平台无关
3、粘性扩展
4、开源理念
5、通用灵活
6、强制可读
7、支持中文
8、模式多样
9、类库丰富

Python语言的优点

优点一：优雅、简单、明确

（减少花哨、晦涩或以“炫技”为目的的代码）

让数据分析师们摆脱了程序本身语法规则的泥潭，更快的进行数据分析

C语言

```c
#include<stdio.h>
main()
{
    printf("Hello World");
}
```

Python语言

```
print "Hello World"
```

```
Hello World
```

优点二：强大的标准库

完善的基础代码库，覆盖了网络通信、文件处理、数据库接口、图形系统、XML处理等大量内容，被形象地称为“内置电池”（batteries included）

Python使用者——“调包侠”

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

优点三：良好的可扩展性

大量的第三方模块，覆盖了科学计算、Web开发、数据接口、图形系统等众多领域，开发的代码通过很好的封装，也可以作为第三方模块给别人使用。如Pandas、Numpy、Seaborn、Scikit-learn等等

优点四：免费、开源

Python语言的缺点

缺点一：运行速度慢

缺点二：加密难

缺点三：缩进规则

缺点四：多线程灾难

```
if True:
print "right"
```

```
  File "<ipython-input-1-ab7d6f176ce6>", line 2
    print "right"
        ^
IndentationError: expected an indented block
```

```
if True:
    print "right"
```

```
right
```

```
junyilu ··· misc > python > gil
junyilu ··· misc > python > gil
junyilu ··· misc > python > gil  python2.7 ./single_thread.py
Total time: 11.4724829197
junyilu ··· misc > python > gil  python2.7 ./multi_thread.py
Total time: 16.1935360432
junyilu ··· misc > python > gil
```

Python语言与Java

动态类型和静态类型

Python中一切皆对象

括号与缩进

应用领域

Python语言与R语言

机器学习的一把利器

可读性强，便于上手

灵活性强：可与其他如Web应用程序进行整合

- 以统计推断为导向
- 数据分析之外的领域有所限制
- 包凌乱且一致性较差

Python语言与R语言的应用场景对比

网络爬虫

连接数据库

内容管理系统

API构建

- 统计分析
- 互动式图标/面板

Python语言Windows系统开发环境

Python语言Mac系统开发环境

Python语言Linux系统开发环境

Python语言Web开发环境

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}F".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}F".format(area))
```

输出结果如下:

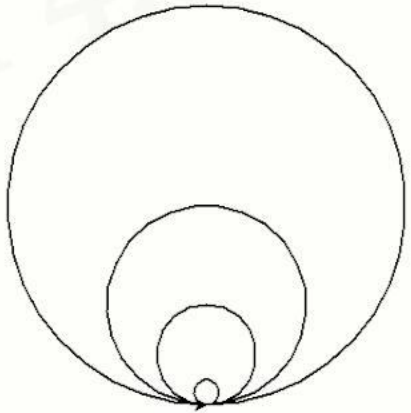
```
1963.4375000000002
1963.44
```

保存为CalCircle.py文件并运行

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)
```



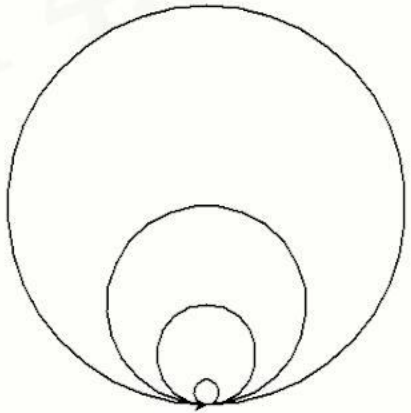
保存为TangentCirclesDraw.py文件并运行

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)
```



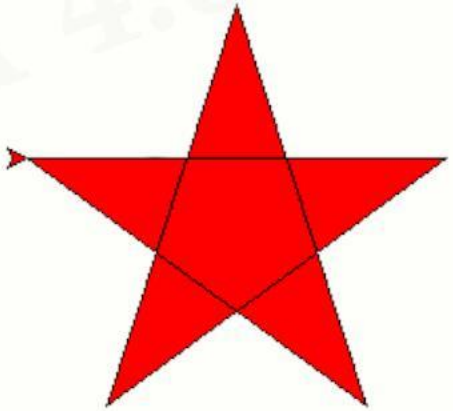
交互式

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

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**

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

"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

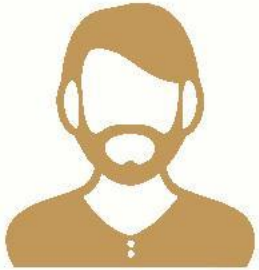
- 温度转换问题是各类转换问题的代表性问题
- 货币转换、长度转换、重量转换、面积转换...
- 问题不同，但程序代码相似

Python语言程序设计

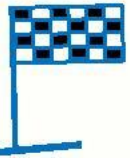
Python程序语法元素分析

单元开篇

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

```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")
```

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

当程序变的更大更复杂时，读起来也更困难。程序的各部分之间紧密衔接，想依靠部分的代码来了解整个程序要做的，是困难的。在现实中，经常会遇到一段代码，很难弄清楚它在做什么、为什么那么做。

因此，在程序中加入自然语言的笔记来解释程序在做什么，是个不错的主意。这种笔记称为注释（comments），注释必须以符号“#”开始。

注释可以单独占一行，也可以放在语句行的末尾。

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

比如如下的代码中注释与代码重复，毫无用处：

```
>>> r=10 #将10赋值给r
```

而下面这段代码注释则包含了代码中隐藏的信息，如果不加注释，很难让人看懂这个是什么意思（虽然在实际中可能可以根据上下文判定，但需要浪费不必要的思考时间）。

```
>>> r=10 #半径，单位是米
```

选择好的变量名，可以减少注释的需要，但长名字也会让复杂表达式更难阅读，所以这两者之间需要衡量舍取。

```
#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"
```

在Python中，变量的使用环境非常宽松。没有

在Python中，等号（ = ）是赋值语句，可以把任意数据类型赋值给变量。

如下为定义一个名为xiaohong的变量：

```
>>> xiaohong='XiaoHong‘
```

此操作解释：xiaohong是我们创建的变量，=是赋值语句，XiaoHong是变量值，需要用引号标记。整句话的意思为：创建变量xiaohong并赋值为XiaoHong。（注意：字符串必须以引号标记开始，并以引号标记结束。）

打印变量结果：

```
>>> print(xiaohong)
XiaoHong
```

在使用变量前，需要对其赋值，没有值的变量是没有意义的，编译器也不会编译通过。例如我定义一个变量为abc，不赋任何值，输入及结果如下：

```
>>> abc
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    abc
NameError: name 'abc' is not defined
```

同一个变量可以反复赋值，而且可以是不同类型的变量，输入如下：

```
>>> a = 123
123
>>> a='ABC'
>>> print(a)
ABC
```

这种变量本身类型不固定的语言称之为动态语言，与之对应的是静态语言。静态语言在定义变量时必须指定变量类型，如果赋值的时候类型不匹配，就会报错。和静态语言相比，动态语言更灵活，就是这个原因。

当不能确定变量或数据的类型时，可以借助解释器内置的函数type进行确认。在交互模式下可以如下输入：

```
>>> type('Hello,world!')
<class 'str'>
>>> type(100)
<class 'int'>
>>> type(3.0)
<class 'float'>
>>> a='test type'
>>> type(a)
<class 'str'>
```

请不要把赋值语句的等号等同于数学的等号。比如下面的代码：

```
a = 100
a = a + 200
```

在编程语言中，赋值语句先计算右侧的表达式a + 200，得到结果300，再赋给变量a。由于a之前的值是100，重新赋值后，a的值变成300。我们通过交互模式验证，输入如下：

```
>>> a=100
>>> a=a+200
>>> print(a)
300
```

理解变量在计算机内存中的表示也非常重要。当我们写：

```
>>> a='ABC'
```

时，Python解释器干了两件事情：

（1）在内存中创建了一个'ABC'的字符串；

（2）在内存中创建了一个名为a的变量，并把它指向'ABC'。

也可以把一个变量a赋值给另一个变量b，这个操作实际上是把变量b指向变量a所指向的数据，例如下面的代码：

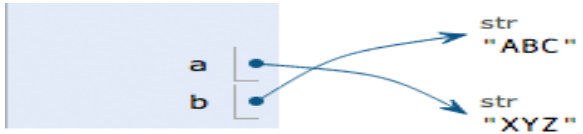
```
>>> a='ABC'
>>> b=a
>>> a='XYZ'
>>> print(b)
```

最后一行打印出变量b的内容到底是'ABC'呢还是'XYZ'？

执行a = ‘ABC’，解释器创建了字符串‘ABC’和变量a，并把a指向‘ABC’，如下图所示：

执行b = a，解释器创建了变量b，并把b指向a指向的字符串'ABC'，如图所示：

执行a = 'XYZ'，解释器创建了字符串'XYZ'，并把a的指向改为'XYZ'，但b并没有更改，如图所示：



所以，最后打印变量b的结果自然是'ABC'了。

命名与保留字

命名

关联标识符的过程

- **命名规则: 大小写字母、数字、下划线和汉字等字符及组合**

  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

下划线“_”可以出现在变量名中。它经常用于连接多个词组。比如happy_study, do_it_with_more_practice。交互模式输入如下：

```
>>> happy_study='stay hungry stay foolish'
>>> print(happy_study)
stay hungry stay foolish
```

如果给变量取非法的名称，解释器显示语法错误。请看下面的示例：

```
>>> 2wrongtest='just for test'
SyntaxError: invalid syntax
```

该示例提示语法错误，错误信息为无效的语法，原因是它不是以字母开头。

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33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

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	
def	if	pass	del	

```
#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

```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")
```

数据类型：整型、字符串、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]`

数 据 类 型

Python3 中有六个标准的数据类型：**Number（数字）、String（字符串）、List（列表）、Tuple（元组）、Sets（集合）、Dictionary（字典）**。本节将讲解Number（数字）**String（字符串）**数据类型，其它数据类型由后续相关章节进行介绍。

Python3支持三种不同的数值类型：

整型（int）、浮点型（float）、复数（complex）。

1 整型

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。
例如交互模式下输入如下：

```
>>> 51
51
```

这里使用的就是整型。
整型加法如下：

```
>>> 25+25
50
```

整型减法：

```
>>> 51-50
1
```

整型乘法：

```
>>> 51*2
102
```

整型除法：

```
>>> 153/51
3.0
>>> 155/51
3.0392156862745097
```

此处出现除不尽的情况了

在整数除法中，除法（/）计算结果是浮点数，即使是两个整数恰好整除，结果也是浮点数，如果只想得到整数的结果，丢弃可能的分数部分，可以使用地板除（//），整数的**地板除（//）永远是整数**，即使除不尽。

改成如下写法：

```
>>> 153//51
3
>>> 155//51
3
```

地板除（//）只取结果的整数部分，Python还提供一个余数运算（%），可以得到两个整数相除的余数。如下：

```
>>> 153%51
0
>>> 155%51
2
```

2 浮点型

浮点型(float)，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。

先看示例：

```
>>> 3.3*102
336.59999999999997
```

按预计应该一位小数，但输出结果却有这么多位小数。是因为整数和浮点数在计算机内部存储的方式是不同的，整数运算永远是精确的，而浮点数运算则可能会有四舍五入的误差。

如下输入：

```
>>> 3.3*102+15.5
352.09999999999997
```

浮点除法：

```
>>> 153/51.0
3.0
>>> 155/51.0
3.0392156862745097
```

浮点地板除：

```
>>> 155//51.0
3.0
>>> 155%51.0
2.0
```

3 复数

复数((complex))，复数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支持复数，Python的复数我们当前阶段使用或接触的比较少，此处就不做具体的讲解，读者有一个概念即可，有兴趣可以自行查阅相关资料。

4 数据类型转换

有时候，我们需要对数据内置的类型进行转换，**数据类型的转换**，你只需要将**数据类型作为函数名即可**。详细将在后期介绍。

关于数据的类型转换，有如下几个函数可以使用：

int(x) 将x转换为一个整数。

float(x) 将x转换到一个浮点数。

complex(x) 将x转换到一个复数，实数部分为 x，虚数部分为 0。

complex(x, y) 将 x 和 y 转换到一个复数，实数部分为 x，虚数部分为 y。x 和 y 是数字表达式。

5 常量

所谓常量就是不能变的变量，比如常用的数学常数π就是一个常量。在Python中，通常用全部大写的变量名表示常量。

Python中有两个比较常见的常量，分别为：PI和E。

PI：数学常量 pi（圆周率，一般以π来表示）。

E：数学常量 e，e即自然常数（自然常数）。

这两个常量将会在后续章节中被使用，具体的用法在使用中进行体现。

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**

```
#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]

字符串可以使用操作符+，但其功能和数学中的不一样，它会进行拼接（concatenation）操作，即将前后两个字符首尾连接起来。

如：

```
>>> string1='hello'
>>> string2='world'
>>> print(string1+string2)
helloworld
```

如果想让字符串之间有空格，可以建一个空字符变量，插在相应的字符串之间让它们隔开，或是在字符串中加入相应的空格。交互模式下输入如下：

```
>>> string1='hello'
>>> string2='world'
>>> space=' '
>>> print(string1+space+string2)
hello world
```

或者

```
>>> string1='hello'
>>> string2=' world'
>>> print(string1+string2)
hello world
```

```
#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

```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")
```

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