

Python 语言程序设计

陈 峦 副教授

13880209111, chluan@uestc.edu.cn

研究院大楼316#

课程考核说明

- 1. 平时成绩：50%

- 平时作业：单项选择题(在本课程QQ群中发布)

- 2. 期末课程设计：50%

- 编写一个Python应用程序。例如：

- (1) 自动组卷评卷考试系统；

- (2) 绘图工具的实现；

- (3) K-means算法的实现；

- (4) 记事本工具的实现；

在IEEE发布的2018年编程语言
排行榜中，Python排名第一。



**进入火热的AI人工智能时代后，
它逐渐取代Java，成为编程界的
头牌语言！**



Python已经进入小学教材， 从小学生学起！

许多著名大学都采用Python来教授程序设计课程。
例如：卡耐基梅隆大学的编程基础、麻省理工学院的
计算机科学及编程导论就使用Python语言讲授。

**目前Python人才缺口巨大，因此，
学习Python就业形势非常乐观！**

- **Python在编程语言学习和就业方面具有很大的优势，再加上人工智能、大数据和云计算的发展，就业前景会越来越好！**

- 在过去的几年间，Python一路高歌猛进，成功窜上“最火编程语言”的宝座。
- 更可怕的是，这把火不仅仅是在程序员的圈子里越烧越旺，甚至还烧到了程序员的圈子外，从小学生到职场老司机，都在学习这一门语言！！！！

全国计算机等级考试二级教程

Python语言程序设计(2023年版)

教育部考试中心 编

高等教育出版社

ISBN: 9787040580549



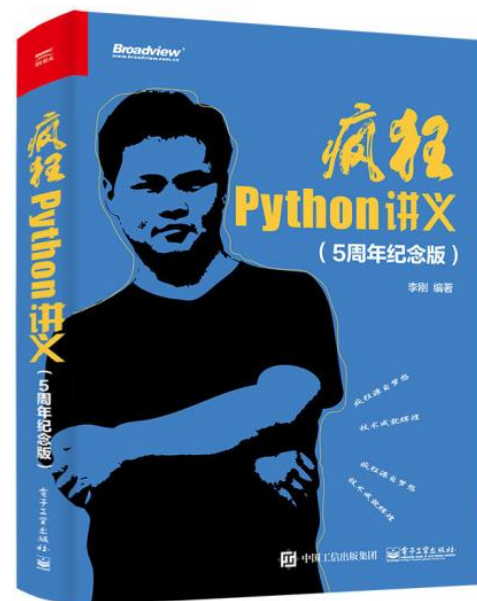
课程参考书

疯狂Python讲义（5周年纪念版）

李刚 编著

电子工业出版社

出版时间：2023年01月

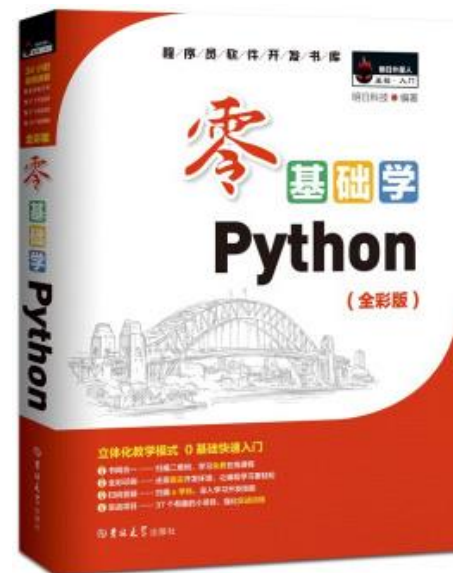


零基础学Python（Python3.9全彩版）

明日科技（MingRi Soft） 著

吉林大学出版社

ISBN: 9787569222258



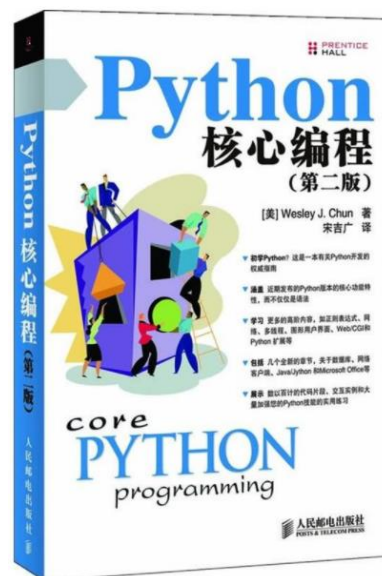
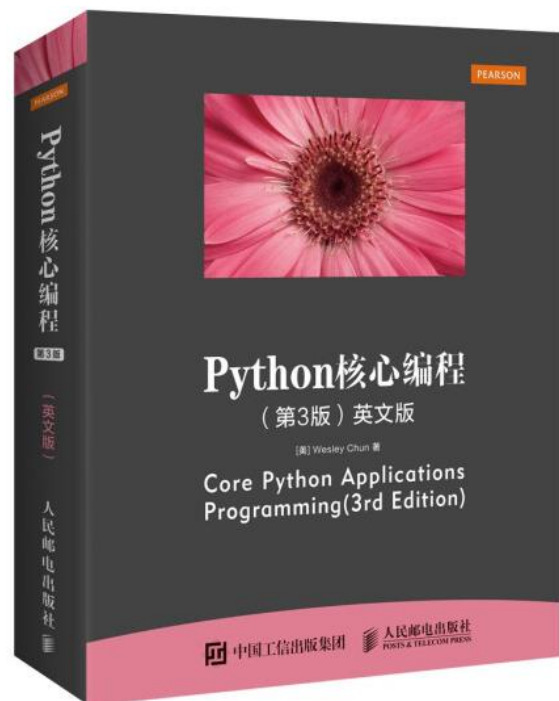
课程参考书

Python核心编程(第2版) (第3版)

[美]Wesley Chun

人民邮电出版社

出版时间：2021年07月



第1章 Python 语言基础

- Python语言是一种语法简洁、跨平台、可扩展的、开源、自由、通用的脚本语言。



python: 蟒蛇, ['paɪθən]

- Python是一种**面向对象**、**解释型**、**动态数据类型**的高级程序设计语言，简单易学，功能强大，能满足大多数应用领域的开发需求。
- Python 语言**优雅**，阅读Python代码就像阅读一篇优美的文章。

- Python支持命令式编程、函数式编程，支持面向对象程序设计。

- **Python**可以说是全能的，**Python**被广泛用于人工智能、机器学习、大数据科学、云计算、**Web**开发、数据库编程、多媒体应用、网络爬虫、数据分析、科学计算、金融、系统运维、自动化测试、网站开发、数据可视化和图形处理等领域。
- 从来没有哪一种语言可以同时在这这么多领域扎根。

- Python常被称为“**胶水语言**”（glue language），
能够把用其他语言制作的各种模块（尤其是
C/C++/Java）很轻松地连接在一起。

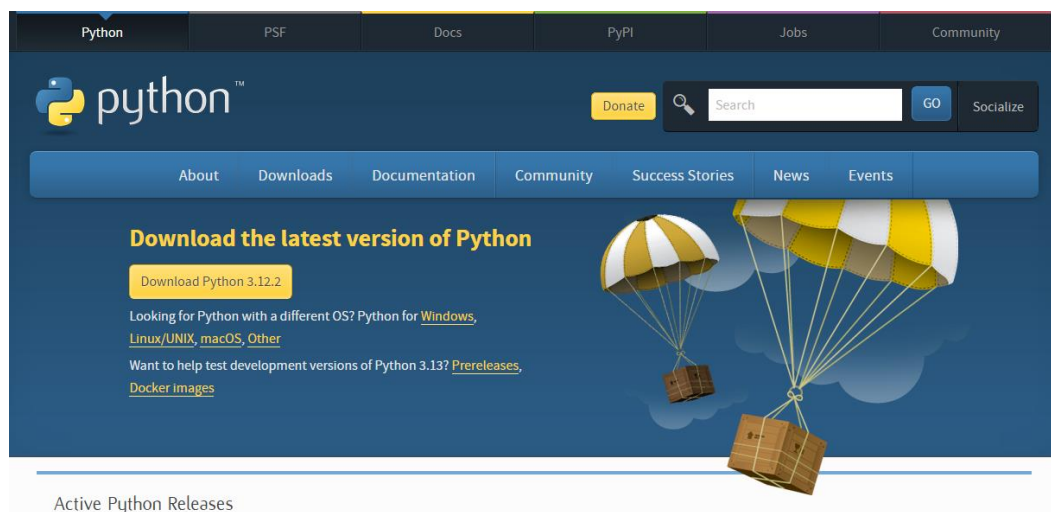
- **Python** 语言具有大量优秀的第三方函数模块。
- **Python**拥有很多面向不同应用的开源扩展库，你能想到的功能基本上都已经有人替你开发了，你只需把想要的程序代码拿来来进行组装便可构建个性化的应用。

- 对于同样的功能，其代码量只有其他语言的
1/10~1/5。
- 用Python写的**代码更短**、更简洁。



1.1 Python 语言概述

- Python语言是开源项目的优秀代表，其解释器的全部代码都是开源的，可以在Python语言的主网站(<https://www.python.org/>)自由下载。
- 下载网址：<https://www.python.org/downloads/>



Python 语言的发展历史

- Python语言诞生于1990 年，由Guido van Rossum（荷兰国家数学与计算机科学研究所的研究员，吉多·范罗苏姆）设计并领导开发。

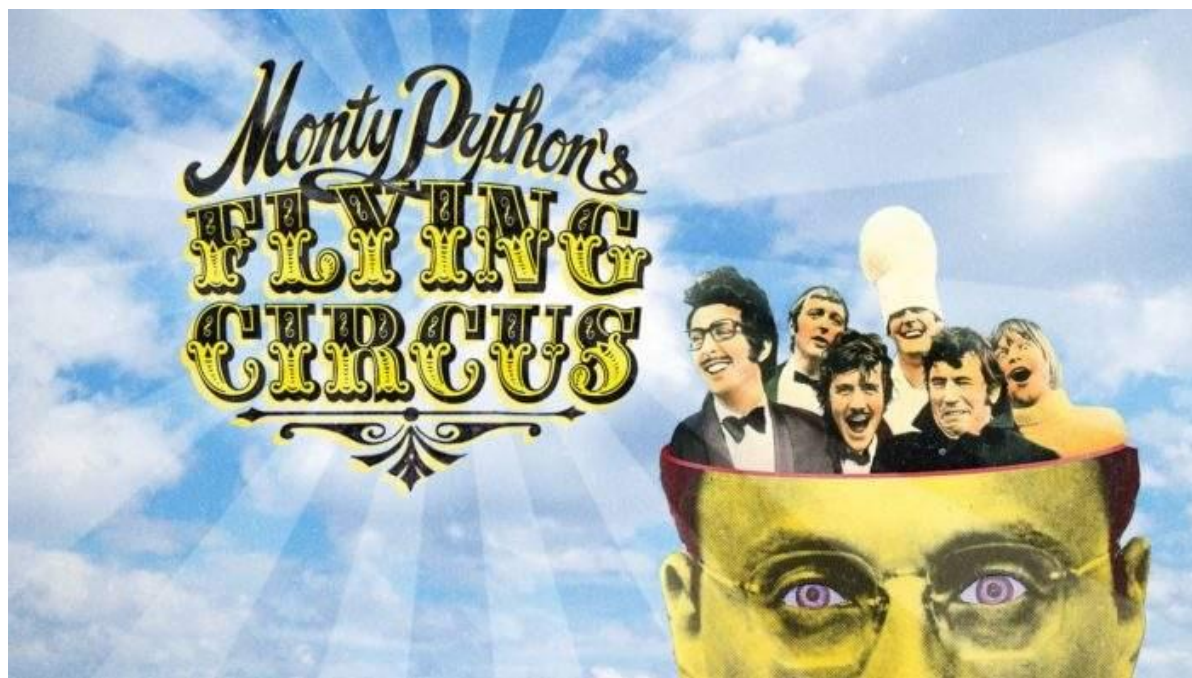
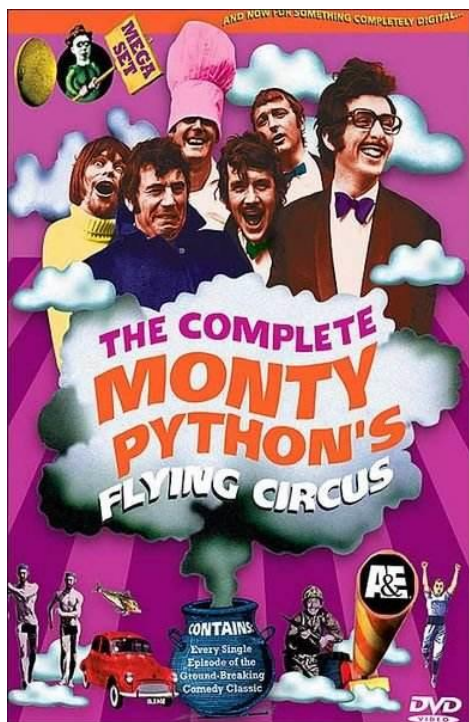


- **Python** 语言起源于**1989**年年末。
- 当时，荷兰国家数学与计算机科学研究所（**CWI**）的研究员吉多·范罗苏姆（**Guido van Rossum**）需要一种高级脚本编程语言，为其研究小组的**Amoeba** 分布式操作系统执行管理任务。
- 为创建新语言，吉多从高级教学语言**ABC**（**All Basic Code**）汲取了大量语法，并从系统编程语言**Modula-3** 借鉴了错误处理机制。



ZEUX Project
zeux.org
free software, free socie

- 吉多把这种新的语言命名为Python，是因为他是BBC 电视剧——蒙提·派森的飞行马戏团（Monty Python's Flying Circus）的爱好者。



- **ABC** 是由吉多参加设计的一种教学语言。
- 就吉多本人看来，**ABC** 这种语言非常优美和强大，是专门为非专业程序员设计的。
- 但是，**ABC** 语言并没有成功，究其原因，吉多认为是非开放造成的。
- 吉多决心在**Python** 中避免这一缺陷，并取得了非常好的效果。

- **Python** 语言的第一个版本于**1991**年年初公开发行。
- 由于功能强大和采用开源方式发行，**Python**发展很快，用户越来越多，形成了一个庞大的语言社区。

- **Python 2.0** 于**2000 年10 月**发布，增加了许多新的语言特性。
- 同时，整个开发过程更加透明，社区对开发进度的影响逐渐扩大。
- **Python 3.0** 于**2008 年12 月**发布，此版本不完全兼容之前的**Python** 版本，导致用早期**Python** 版本设计的程序无法在**Python 3.0** 上运行。

- 在Python 发展过程中，形成了Python 2.x 和Python 3.x 两个版本，目前正朝着Python 3.x 进化。
- Python 2.x 和Python 3.x 两个版本是不兼容的，由于历史原因，原有的大量第三方函数模块是用2.x 版实现的。
- 随着Python的普及与发展，近年来Python 3.x下的第三方函数模块日渐增多。
- 本课程选择Windows操作系统下的Python 3.x 版本作为程序实现环境（下载安装版本是Python 3.12.2）。



python-3.12.2-amd64

Python 语言的特点

- Python语言是一种被广泛使用的高级通用脚本编程语言。
- （1）语法简洁：实现相同功能，Python语言的代码行数仅相当于其他语言的 $1/10 \sim 1/5$ 。
- （2）与平台无关：作为脚本语言，Python程序可以在任何安装解释器的计算机环境中执行，因此，用该语言编写的程序可以不经修改地实现跨平台运行。

- (3) **粘性扩展**: **Python**语言具有优异的扩展性, 体现在它可以集成**C**、**C++**、**Java**等语言编写的代码, 通过接口和函数库等方式将它们“粘起来”(整合在一起)。此外, **Python**语言本身提供了良好的语法和执行扩展接口, 能够整合各类程序代码。
- (4) **开源理念**: 对于高级程序员, **Python**语言开源的解释器和函数库具有强大的吸引力, **Python**语言倡导的开源软件理念为该语言发展奠定了坚实的群众基础。

- **（5）通用灵活：**Python语言是一个通用编程语言，可用于编写各领域的应用程序，这为该语法提供了广阔的应用空间。几乎各类应用，从科学计算、数据处理到人工智能、机器人，Python语言都能够发挥重要作用。
- **（6）强制可读：**Python语言通过强制缩进（类似文章段落的首行空格）来体现语句间的逻辑关系，显著提高了程序的可读性，进而增加了Python程序的可维护性。
- **（7）支持中文：**Python 3.0解释器采用UTF-8编码表达所有字符信息。UTF-8编码可以表达英文、中文、韩文、法文等各类语言，因此，Python程序在处理中文时更加灵活且高效。

- (8) **模式多样**: 尽管Python 3.0解释器内部采用面向对象方式实现, 但Python语法层面却同时支持面向过程和面向对象两种编程方式, 这为使用者提供了灵活的编程模式。
- (9) **类库丰富**: Python解释器提供了几百个内置类和函数库, 此外, 世界各地程序员通过开源社区贡献了十几万个第三方函数库, 几乎覆盖了计算机技术的各个领域, 编写Python程序可以大量利用已有的内置或第三方代码, 具备良好的编程生态。

Python 语言的优势

- (1) 简单易学。
- Python 语言语法结构简单，组成一个Python 程序没有太多的语法细节和规则要求，“信手拈来”就可以组成一个程序。
- 一个良好的Python 程序就像一篇英语文章一样，代表问题求解过程的描述。

- (2) 程序可读性好。
- **Python** 语言和其他高级语言相比，一个重要的区别就是，一个语句块的界限完全是由每行的首字符在这一行的位置来决定的。
- (C 语言用一对大括号 “{}” 来明确界定语句块的边界，与字符的位置毫无关系)。
- 通过强制程序缩进，**Python** 语言确实使得程序具有很好的可读性，同时**Python** 的缩进规则也有利于程序员养成良好的程序设计习惯。

- (3) 丰富的数据类型。
- 除了基本的数值类型外，**Python** 语言还提供了字符串、列表、元组、字典和集合等丰富的复合数据类型，利用这些数据类型，可以更方便地解决许多实际问题，如文本处理、数据分析等。

- (4) 开源的语言。
- **Python** 语言是一种开源的语言，可移植到多种操作系统，只要避免使用依赖于特定操作系统的特性，**Python** 程序不需修改就可以在各种平台上运行。**Python** 的开源特性使得有很多的开放社区对用户快速的技术支持，学习和使用**Python** 技术不再是孤军奋战。
- 如今，各种社区提供了成千上万不同功能的开源函数模块，而且还在不断地发展，这为基于**Python** 语言的快速开发提供了强大支持。

- (5) 解释型的语言。
- 用**Python** 语言编写的程序不需要编译成二进制代码，而可以直接运行源代码。
- 在计算机内部，**Python** 解释器把.py 文件中的源代码转换成**Python** 的字节码（**Byte code**），然后再由**Python** 虚拟机（**Virtual machine**）一条一条地执行字节码指令，从而完成程序的执行。

- 对于**Python** 的解释语言特性，要一分为二地看待。
- 一方面，每次运行时都要将源文件转换成字节码，然后再由虚拟机执行字节码。较之于编译型语言，每次运行都会多出两道工序，所以程序的执行性能会受到影响。
- 另一方面，由于不用关心程序的编译以及库的连接等问题，所以程序调试和维护会变得更加轻松方便，同时虚拟机距离物理机器更远了，所以**Python** 程序更加易于移植，实际上不需改动就能在多种平台上运行。

- (6) 面向对象的语言。
- **Python** 语言既可以面向过程，也可以面向对象，支持灵活的程序设计方式。

Python 语言的局限性

- 虽然 **Python** 语言是一个非常成功的语言，但也有它的局限性。
- 相比其他一些语言（如**C**、**C++**语言），**Python** 程序的运行速度比较慢，对于速度有着较高的要求的应用，就要考虑**Python** 是否能满足需要。
- 不过这一点可以通过使用**C** 语言编写关键模块，然后由**Python** 调用的方式加以解决。
- 而且现在计算机的硬件配置在不断提高，对于一般的开发来说，速度已经不成问题。

- 此外，**Python** 用代码缩进来区分语法逻辑的方式还是给很多初学者带来了困惑，即便是很有经验的**Python** 程序员，也可能陷入陷阱当中。
- 最常见的情况是**Tab** 和空格的混用会导致错误，而这是用肉眼无法分辨的。

缩 进

- Python开发者有意让违反了缩进规则的程序不能通过编译，以此来强制程序员养成良好的编程习惯。
- 并且Python语言利用缩进表示语句块的开始和退出（**Off-side**规则），而非使用花括号或者某种关键字。
- 增加缩进表示语句块的开始，而减少缩进则表示语句块的退出。
- 缩进成为了语法的一部分。

例：if语句

python3

```
1  if age<21:  
2      print("你不能买酒。")  
3      print("不过你能买口香糖。")  
4  print("这句话在if语句块的外面。")
```

- 根据规定，必须使用**4**个空格来表示每级缩进（可以自定义空格数，但是要满足每级缩进间空格数相等）。
- 使用**Tab**字符和其它数目的空格虽然都可以编译通过，但不符合编码规范。
- 支持**Tab**字符和其它数目的空格仅仅是为兼容很旧的**Python**程序和某些有问题的编辑程序。

Python 语言的应用领域

- 1. Windows 系统编程
- Python 是跨平台的程序设计语言，在Windows 系统下，通过使用pywin32 模块提供的Windows API函数接口，就可以编写与Windows 系统底层功能相关的Python 程序，包括访问注册表、调用ActiveX控件以及各种COM 组件等工作。
- 还有许多其他的日常系统维护和管理工作也可以交给Python 来实现。
- 利用py2exe 模块可以将Python 程序转换为.exe 可执行程序，使得Python 程序可以脱离Python 系统环境来运行。

2. 科学计算与数据可视化

- 科学计算也称数值计算，是研究工程问题的近似求解方法，并在计算机上进行程序实现的一门科学，既有数学理论上的抽象性和严谨性，又有程序设计技术上的实用性和实验性的特征。
- 随着科学计算与数据可视化**Python** 模块的不断产生，使得**Python** 语言可以在科学计算与数据可视化领域发挥独特的作用。

- **Python** 中用于科学计算与数据可视化的模块有很多，例如**NumPy**、**SciPy**、**SymPy**、**Matplotlib**、**Traits**、**TraitsUI**、**Chaco**、**TVTK**、**Mayavi**、**VPython**、**OpenCV** 等，涉及的应用领域包括数值计算、符号计算、二维图表、三维数据可视化、三维动画演示、图像处理以及界面设计等。

- **NumPy** 模块提供了一个在**Python** 中做科学计算的基础库，主要用于矩阵处理与运算；**SciPy** 模块是在**NumPy** 模块的基础上开发的，提供了一个在**Python** 中做科学计算的工具集。
- 例如，统计工具（**statistics**）、最优化工具（**optimization**）、数值积分工具（**numerical integration**）、线性代数工具（**linear algebra**）、傅里叶变换工具（**Fourier transforms**）、信号处理工具（**signal processing**）、图像处理工具（**image processing**）、常微分方程求解工具（**ODE solvers**）等；**Matplotlib** 是比较常用的绘图模块，可以快速地将计算结果以不同类型的图形展示出来。

3. 数据库应用

- 在数据库应用方面，Python 语言提供了对所有主流关系数据库管理系统的接口，包括SQLite、Access、MySQL、SQL Server、Oracle 等。
- Python 数据库模块有很多，例如，可以通过内置的sqlite3模块访问SQLite 数据库，使用pywin32 模块访问Access 数据库，使用pymysql 模块访问MySQL 数据库，使用pywin32 和pymssql 模块来访问SQL Sever 数据库。

4. 多媒体应用

- **Python** 多媒体应用开发可以为图形、图像、声音、视频等多媒体数据处理提供强有力的支持。
- **PyMedia** 模块是一个用于多媒体操作的**Python** 模块，可以对**WAV**、**MP3**、**AVI** 等多媒体格式文件进行编码、解码和播放；**PyOpenGL** 模块封装了**OpenGL** 应用程序编程接口，通过该模块可在**Python**程序中集成二维或三维图形；**PIL**（**Python Imaging Library**，**Python** 图形库）为**Python** 提供了强大的图像处理功能，并提供广泛的图像文件格式支持。
- 该模块能进行图像格式的转换、打印和显示，还能进行一些图像效果的处理，如图形的放大、缩小和旋转等，是**Python** 进行图像处理的重要工具。

5. 网络应用

- **Python** 语言为众多的网络应用提供了解决方案，利用有关模块可方便地定制出所需要的网络服务。
- **Python** 语言提供了 **socket** 模块，对 **Socket** 接口进行了二次封装，支持 **Socket** 接口的访问，简化了程序的开发步骤，提高了开发效率；
- **Python** 语言还提供了 **urllib**、**cookielib**、**httplib**、**scrapy** 等大量模块，用于对网页内容进行读取和处理，并结合多线程编程以及其他有关模块可以快速开发网页爬虫之类的应用程序；
- 可以使用 **Python** 语言编写 **CGI** 程序，也可以把 **Python** 程序嵌入到网页中运行；
- **Python** 语言还支持 **Web** 网站开发，比较流行的开发框架有 **web2py**、**django** 等。

6. 电子游戏应用

- **Python** 在很早的时候就是一种电子游戏编程工具。
- 目前，在电子游戏开发领域也得到越来越广泛的应用。
- **Pygame** 就是用来开发电子游戏软件的**Python** 模块，在**SDL** 库的基础上开发，可以支持多个操作系统。
- 使用**Pygame** 模块，可以在**Python** 程序中创建功能丰富的游戏和多媒体程序。

1.2 Python 语言的开发环境

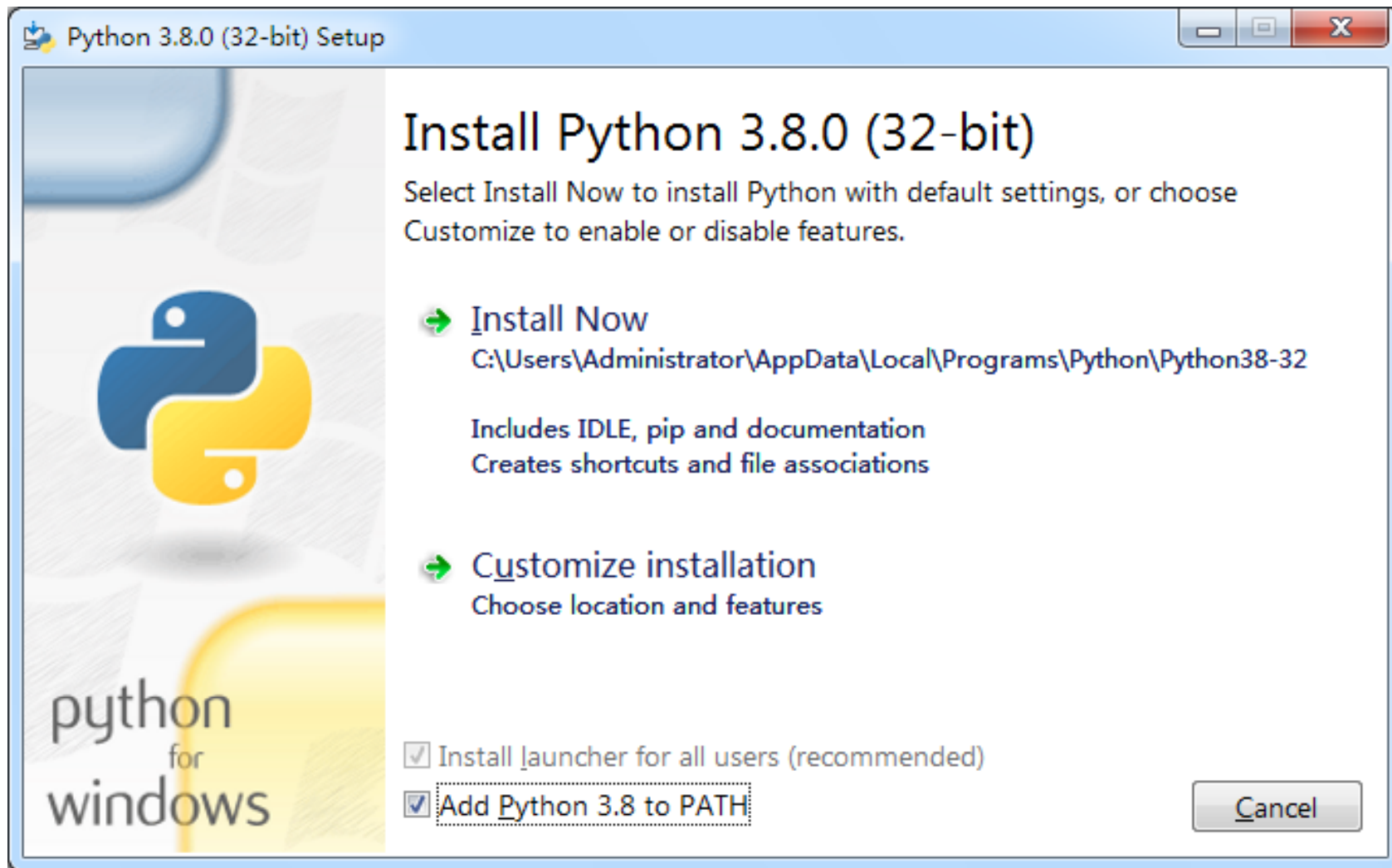
- 运行 **Python** 程序需要相应开发环境的支持。**Python** 内置的命令解释器（称为**Python Shell**，**Shell**有操作的接口或外壳之意）提供了**Python** 的开发环境，能方便地进行交互式操作，即输入一行语句，就可以立刻执行该语句，并看到执行结果。
- 此外，还可以利用第三方的**Python** 集成开发环境（**IDE**）进行**Python** 程序开发。
- 本课程基于**Windows** 操作系统，使用**Python** 内置的命令解释器。

Python 系统的下载与安装

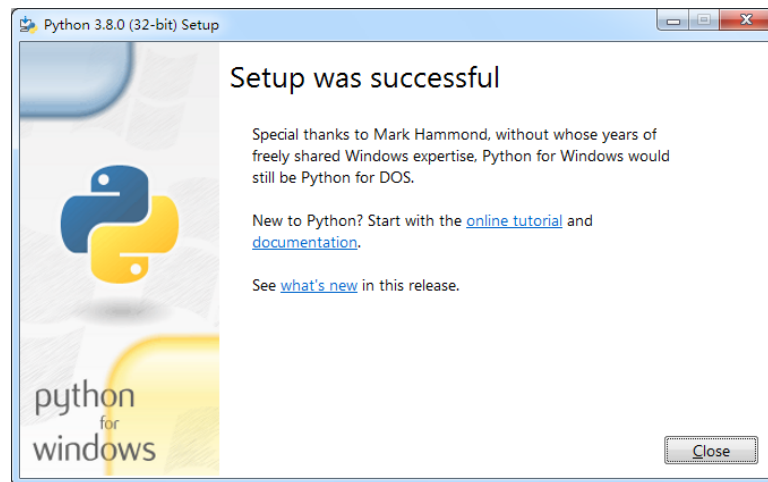
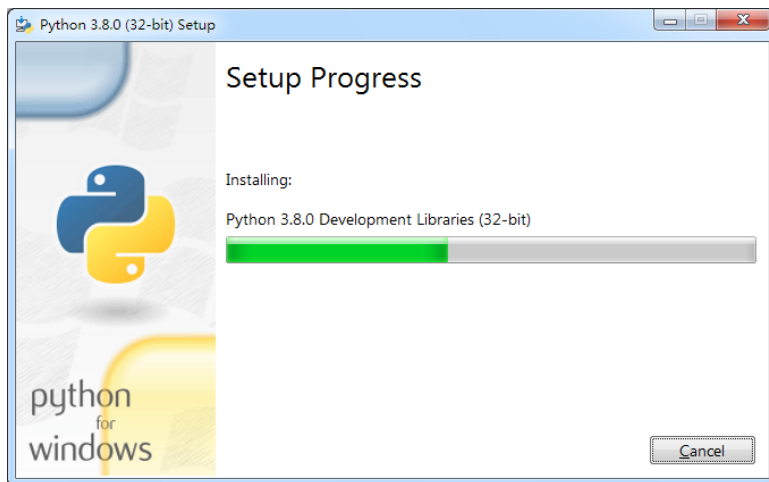
- 要使用 **Python** 语言进行程序开发，必须安装其开发环境，即**Python** 解释器。
- 安装前先要从**Python**官网下载**Python** 系统文件，下载地址为<https://www.python.org/downloads/>。
- 选择下载基于**Windows** 操作系统的当前最新版本 **Python 3.12.2**。

python-3.12.2-amd64

- 例：下载完成后，运行系统文件python-3.8.0.exe，进入Python 系统安装界面。



- 选中 “**Add Python 3.8 to PATH**”复选框，并使用默认的安装路径，单击 “**Install Now**”选项，这时进入系统安装过程，安装完成后单击 “**Close**”按钮即可。
- 如果要设置安装路径和其他特性，可以选择 “**Customize installation**”选项。



系统环境变量的设置

- 在 **Python** 的默认安装路径下包含**Python** 的启动文件**python.exe**、**Python** 库文件和其他文件。
- 为了能在**Windows** 命令提示符窗口自动寻找安装路径下的文件，需要在安装完成后将**Python** 安装文件夹添加到环境变量**Path** 中。

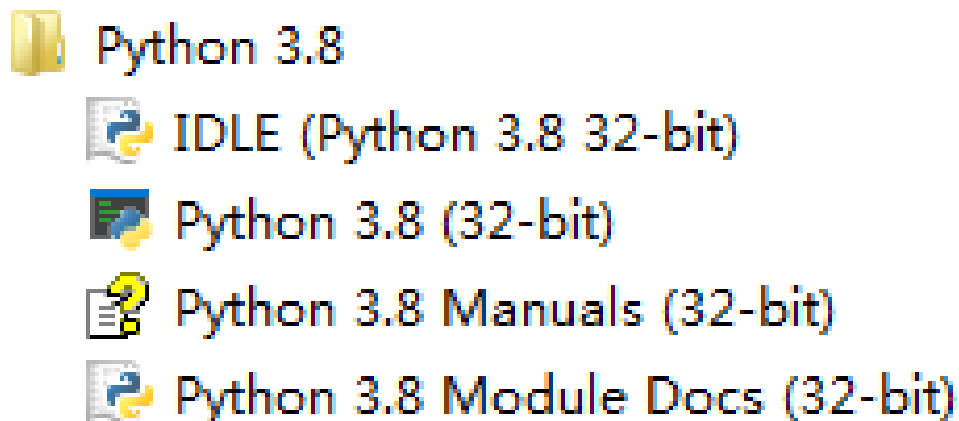
- 如果在安装时选中了“**Add Python 3.8 to PATH**”复选框，则会自动将安装路径添加到环境变量**Path**中，否则可以在安装完成后添加。
- 其方法为：在**Windows** 桌面右击“计算机”图标，在弹出的快捷菜单中选择“属性”命令，然后在打开的对话框中选择“高级系统设置”选项，在打开的“系统属性”对话框中选择“高级”选项卡，单击“环境变量”按钮，打开“环境变量”对话框，在“系统变量”区域选择“**Path**”选项，单击“编辑”按钮，将安装路径添加到**Path** 中，最后单击“确定”按钮逐级返回。

Python 程序的运行

- 在 Python 系统安装完成后，可以启动Python 解释器（Python Shell），它有命令行（Command Line）和图形用户界面（Graphical User Interface）两种操作界面。
- 在不同的操作界面下，Python 语句既可以采用交互式的命令执行方式，又可以采用程序执行方式。

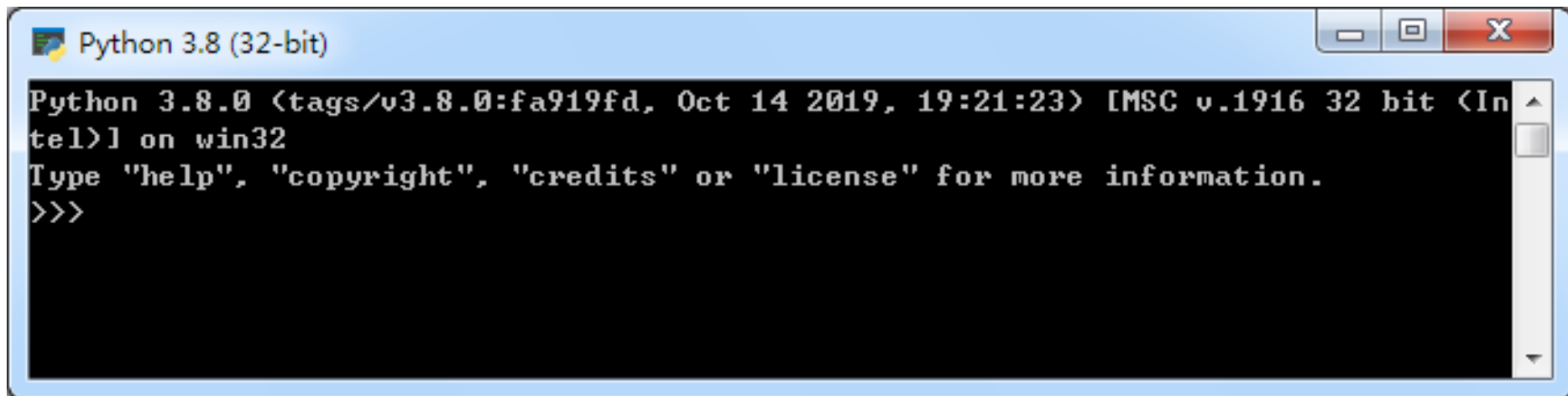
1. 命令行形式的Python解释器

- 启动命令行形式的Python 解释器，有以下方法。
- （1）在Windows 系统的桌面，选择“开始”→“所有程序”→“Python 3.8”→“Python 3.8（32-bit）”来启动命令行形式的Python 解释器。



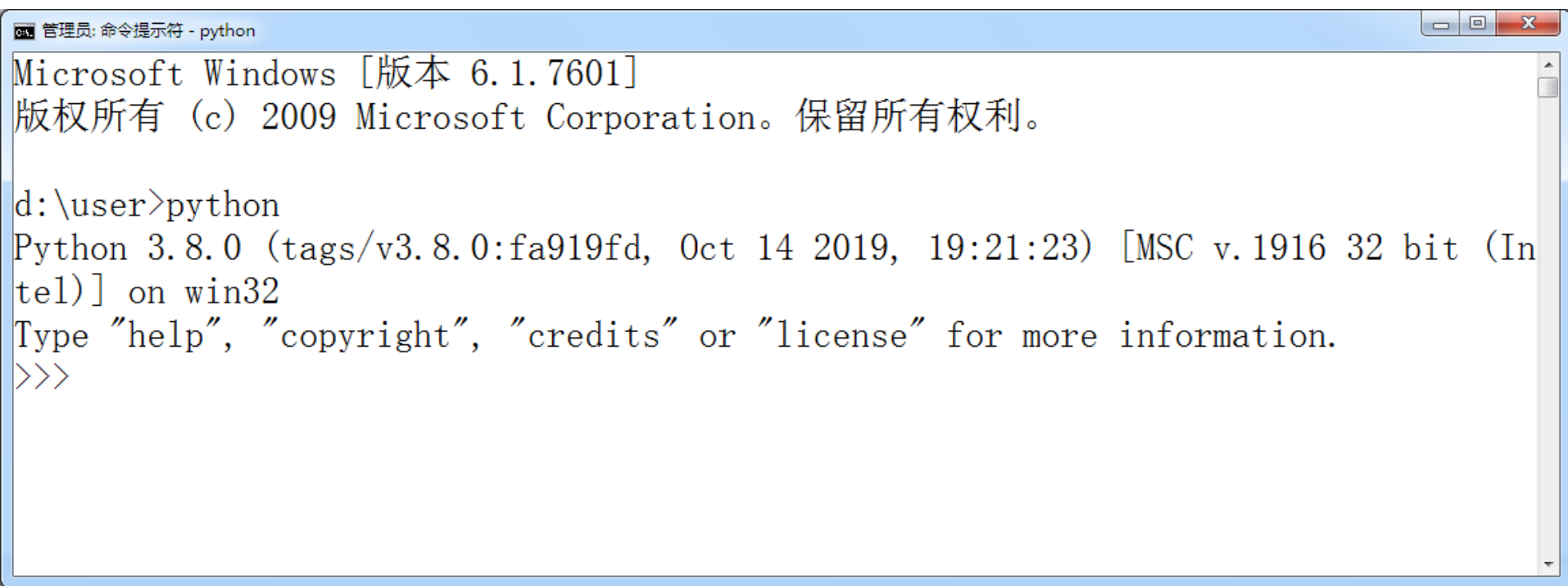
- (2) 在Windows 系统的桌面单击“开始”按钮，选择“运行”选项，在弹出的“运行”对话框中选择Python 启动文件的路径和文件名python.exe，单击“确定”按钮。
- (3) 可以到Python 的安装文件夹下，通过双击运行python.exe 文件来启动命令行的Python 解释器，或在Python 图标上单击右键，在弹出的菜单中选择“创建快捷方式”命令，然后把建立的快捷方式图标复制到桌面上，之后通过快捷方式来启动命令行的Python 解释器。

- 启动命令行形式的Python 解释器后出现相应的程序窗口。
- 其中 “>>>”是Python解释器的提示符，在提示符后面输入语句，Python 解释器将解释执行。



```
Python 3.8 (32-bit)
Python 3.8.0 <tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23> [MSC v.1916 32 bit <Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- 还可以在Windows 命令提示符（即DOS 操作界面）下直接运行python.exe 文件，来启动命令行的Python 解释器。



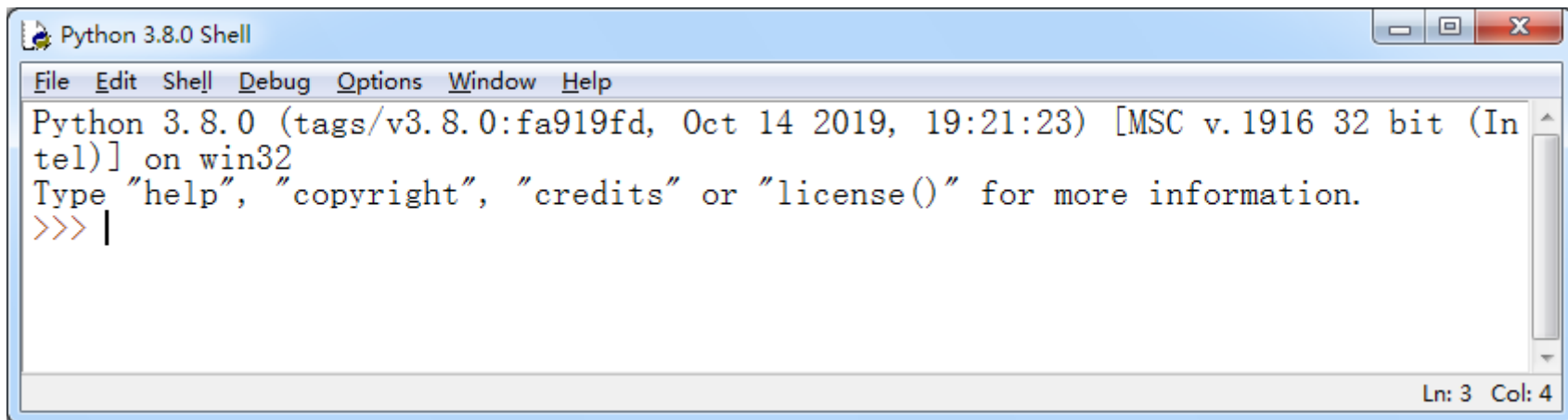
```
管理员: 命令提示符 - python
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

d:\user>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- 因为设置了Windows 系统的环境变量Path，其中包含Python的安装路径，所以在运行python.exe 时，Windows 系统会自动寻找到python.exe 文件，否则需要在python文件名前面加上安装路径。
- 先按Ctrl+Z 快捷键再按Enter 键，或输入quit()命令，或单击Python 命令行窗口的“关闭”按钮，均可退出Python 解释器。

2. 图形用户界面形式的Python解释器

- 在Windows 系统的桌面，选择“开始” → “所有程序” → Python 3.8 → IDLE (Python 3.8 32-bit)
- 来启动图形用户界面形式的Python 解释器（简称Python 解释器图形用户界面窗口）。



- 图形用户界面形式的**Python** 解释器集程序编辑、解释执行于一体，是一个集成开发环境，可以提高程序设计的效率。
- 在**Python** 解释器图形用户界面窗口，选择"**File→Exit**"命令，或按**Ctrl+Q** 快捷键，或输入**quit()**命令，或单击**Python** 解释器图形用户界面窗口的“关闭”按钮，均可退出**Python** 解释器图形用户界面窗口。

3. Python的命令执行方式

- 启动 Python 解释器后，可以直接在其提示符（>>>）后输入语句。
- 例：先在提示符>>>后输入一个输出语句，下一行将接着输出结果。
- >>> print("Hello,World!")
- Hello,World!
- 让Python 系统在屏幕上显示 “Hello,World!” 。

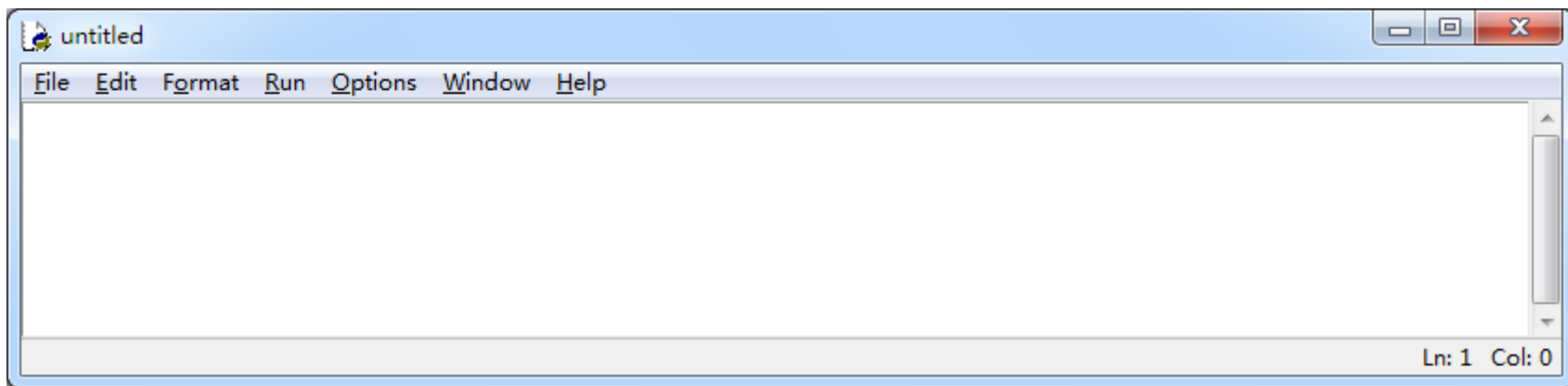
- **Python** 解释器用起来的确有点像是计算器，利用输出语句可以输出一个表达式的值。
- 例：在提示符>>>后输入下列语句将得到结果**1.75**。
- **>>> print(1+3/4)**
- **1.75**

4. Python 的程序执行方式

- Python 的命令执行方式又称交互式执行方式，这对执行单个语句来说是合适的。
- 但是，如果要执行多个语句，就显得麻烦。
- 通常的做法是将语句写成程序，再把程序存放到一个文件中，然后再批量执行程序文件中的全部语句，这称为程序执行方式。

(1) 在Python程序编辑窗口执行 Python源程序

- 在Python 解释器图形用户界面窗口，选择“File→New File”命令，或按Ctrl+N 快捷键，打开Python 程序编辑窗口。

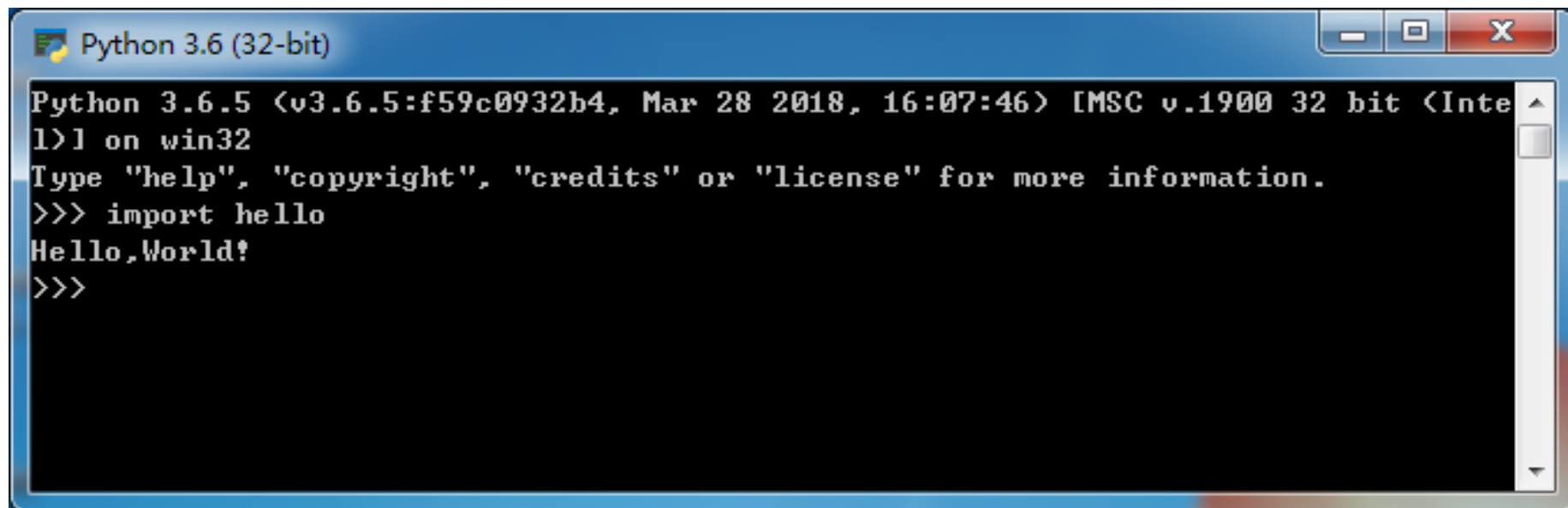


- 在Python 程序编辑窗口输入程序的全部语句。
- 例：输入语句：
- `print("Hello,World!")`
- 语句输入完成后，在Python 程序编辑窗口选择“File”→“Save”命令，确定保存文件位置和文件名，
例：e:\user\hello.py
- 在Python 程序编辑窗口选择“Run→Run Module”命令，或按F5 键，运行程序并在Python解释器图形用户界面窗口中输出运行结果。

(2) 在Python解释器提示符下运行 Python源程序

- 利用Python 程序编辑窗口或其他编辑程序（如Windows 记事本）建立一个Python 源程序文件后，可以在Python 解释器（命令行或图形用户界面）的提示符下执行import 语句来导入程序文件。
- import语句的作用是将Python 程序文件从磁盘加载到内存，在加载的同时执行程序。

- 注意：模块文件名不加扩展名 “.py”，因为系统自动假设模块具有 “.py” 扩展名。
- 例：运行 `hello.py`
- 可以使用下面的语句：
- `>>> import hello`



```
Python 3.6 (32-bit)
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import hello
Hello, World!
>>>
```

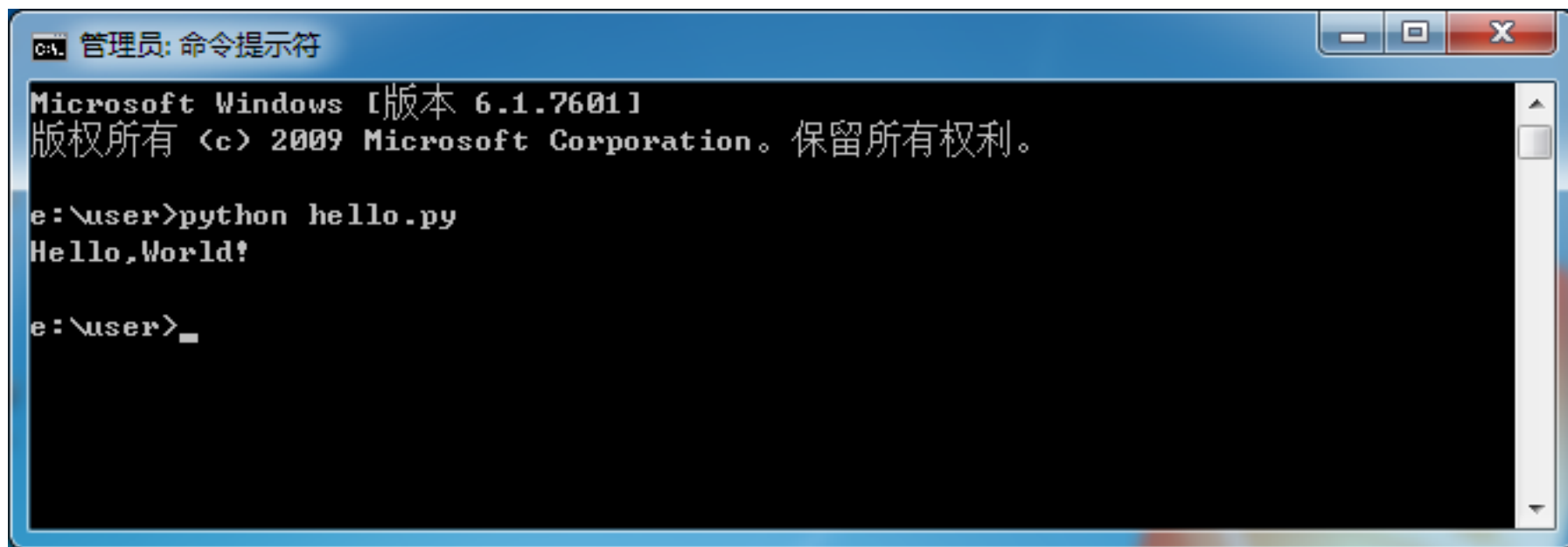
- 注意：
- ① 如果将程序文件保存在**Python** 的安装文件夹下，则使用 **import** 语句时可以搜索到相应程序文件并执行它。
- 但一般将系统文件和用户文件分开存放，以便于管理。
- 这时，如果在**import** 语句中直接使用文件名就会找不到指定文件。
- 为此，可以修改系统环境变量**PythonPath**，为**Python** 系统添加默认文件搜索路径。
- 假定程序文件存放在**e:\user**，则使用**PythonPath** 环境变量，在这个环境变量中输入路径**e:\user**。
- 如果**PythonPath** 变量不存在，可以创建它。
- 设定**PythonPath** 环境变量后，就可以在**import** 语句中直接使用程序文件名来运行该程序了。

- ② 在**Python** 中，每一个以**.py** 结尾的**Python** 文件都是一个模块，可以通过导入一个模块来读取该模块的内容。
- 从本质上来讲，导入就是载入一个文件，并能够读取那个文件的内容。
- 模块导入是一种运行**Python** 程序的方法。
- 但是对于一个文件，**import** 语句只能在第一次导入时运行文件，如果要再次运行文件，就需要调用**imp** 标准库模块中的**reload** 函数。
- **imp** 标准库模块需要导入才能使用。
- 如果要再次运行**hello.py**，可以使用以下语句：
- **>>> import imp**
- **>>> imp.reload(hello)**

(3) 在Windows命令提示符下运行 Python源程序

- 要想运行Python 程序，可以在Windows 命令提示符下切换到Python 程序文件所在文件夹，因为程序文件位于e:\user 文件夹下，所以可以先选择e: 盘并设置其当前文件夹为 e:\user。
- 然后，在Windows 命令提示符下输入python，后跟要执行的程序文件名即可。

- 例：要运行hello.py，可以使用以下命令：
- **python hello.py**



A screenshot of a Windows Command Prompt window titled "管理员: 命令提示符". The window shows the following text:

```
Microsoft Windows [版本 6.1.7601]  
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。  
  
e:\user>python hello.py  
Hello,World!  
  
e:\user>_
```

在python交互命令行中如何清屏？ 方法：

```
>>>import os
```

```
>>>_ = os.system("cls")
```

演示程序

```
from turtle import *  
pencolor("blue") #定义画笔的颜色  
pensize(2)       #定义画笔的线条宽度  
speed(10)        #定义绘图的速度  
for i in range(10):  
    up()          #提起画笔  
    goto(0,-10-i*10) #确定画笔的起点  
    down()        #放下画笔  
    circle(10+i*10) #画圆
```

```
from turtle import *  
  
pencolor("blue") #定义画笔的颜色  
  
pensize(2)       #定义画笔的线条宽度  
  
speed(10)        #定义绘图的速度  
  
for i in range(50): #绘出正n边形的n条边  
    forward(4+i*4)  
    right(360//4)
```

```
from turtle import *
pencolor("blue") #定义画笔的颜色
pensize(2)       #定义画笔的线条宽度
speed(10)        #定义绘图的速度
n=4
for i in range(20): #绘出正n边形的n条边
    up()           #提起画笔
    goto(-i*n*2,i*n*2) #确定画笔的起点
    down()         #放下画笔
    for j in range(n):
        forward(4+i*n*4)
        right(360//n)
```

```
import turtle
colors=['red','orange','yellow','green','blue','indigo','
purple']
for i in range(7):
    c=colors[i]
    turtle.color(c,c)
    turtle.begin_fill()
    turtle.rt(360/7)
    turtle.circle(50)
    turtle.end_fill()
turtle.done()
```

```
from turtle import *
```

```
color('red','red')
```

```
begin_fill()
```

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

```
    fd(200)
```

```
    rt(144)
```

```
end_fill()
```

```
done()
```


- import turtle
- turtle.speed(10)
- def qimian(x, y, color):
- turtle.up()
- turtle.goto(x, y)
- turtle.down()
- turtle.color(color)
- turtle.begin_fill()
- for i in range(1, 5):
- if i % 2 == 0: turtle.forward(205)
- else: turtle.forward(285)
- turtle.left(90)
- turtle.end_fill()
- def wujiaoxing(x, y, color):
- turtle.up()
- turtle.goto(x, y)
- turtle.down()
- turtle.color(color)
- turtle.begin_fill()
- for i in range(5):
- turtle.forward(30)
- turtle.right(144)
- turtle.end_fill()
- def xiaowujiao(x, y, single):
- turtle.up()
- turtle.goto(x, y)
- turtle.left(single)
- turtle.down()
- turtle.color("yellow")
- turtle.begin_fill()
- for i in range(5):
- turtle.forward(12)
- turtle.right(144)
- turtle.end_fill()
- qimian(-110, -90, "red")
- wujiaoxing(-85, 55, "yellow")
- xiaowujiao(-35, 80, 30)
- xiaowujiao(-14, 55, 60)
- xiaowujiao(-12, 28, 30)
- xiaowujiao(-22, 10, 60)
- turtle.hideturtle()
- turtle.done()

```

● from turtle import *
● # 旗帜
● width(4)
● color('red', 'red')
● begin_fill()
● pencolor('red')
● pu()
● goto(0, -300)
● pd()
● lt(90)
● forward(500)
● right(90)
● forward(300)
● right(90)
● forward(180)
● right(90)
● forward(300)
● right(90)
● end_fill()
● # 五角星
● # x,y 起始位置; z 大小; a 角度
● def drawStar(x, y, z, a = 0):
●     pencolor('yellow')
●     pu()
●     goto(x, y)
●     pd()
●     seth(a)
●     for i in range(5):
●         fd(z)
●         rt(144)
●     color('yellow', 'yellow')
●     begin_fill()
●     drawStar(25, 160, 50) #2
●     end_fill()
●     begin_fill()
●     # drawStar(110, 170, 20) #3
●     drawStar(95, 180, 20, 340) #3
●     end_fill()
●     begin_fill()
●     drawStar(130, 160, 20, 250) #4
●     end_fill()
●     begin_fill()
●     drawStar(110, 125, 20) #5
●     end_fill()
●     begin_fill()
●     drawStar(85, 110, 20, 345) #6
●     end_fill()
●     pu()
●     goto(1000, 1000)
●     # 调用done()使得窗口等待被关闭, 否则将立刻关闭窗口
●     done()

```

```
for i in range(1,10):  
    for j in range(1,i+1):  
        print("{}*{}={:2} ".format(j,i,i*j),end="")  
print("")
```

r=25

area=3.1415*r*r

print(area)

print("{:.2f}".format(area))

1.3 常量与变量

1. 常量

- 常量按其值的表示形式区分它的类型。
- 例：
- 0、435、-78 是整型常量；
- -5.8、3.14159、1.0 是实型常量（也称为浮点型常量）；
- '410083'、'Python'是字符串常量。

2. Python 变量

(1) 变量的数据类型

- 一般而言，变量需要先定义，后使用，变量的数据类型决定了变量占用多少个字节的内存单元。
- 例：C 语言中的变量，需要在程序编译时确定数据类型并分配相应的内存单元。这种在使用变量之前定义其数据类型的语言称为静态类型语言。
- 但Python 语言不同，它是一种动态类型语言。

- Python 是一种动态类型语言。
- 这种动态类型语言确定变量的数据类型是在给变量赋值的时候，对变量的每一次赋值，都可能改变变量的类型。
- 在Python 中使用变量时不用像C 语言那样先定义数据类型，而可直接使用。

- 例:

- `>>> x=12` #给变量x 赋值

- `>>> y=12.34` #给变量y 赋值

- `>>> z='Hello World'` #给变量z 赋值

- `>>> print(x,y,z)`

- `12 12.34 Hello World`

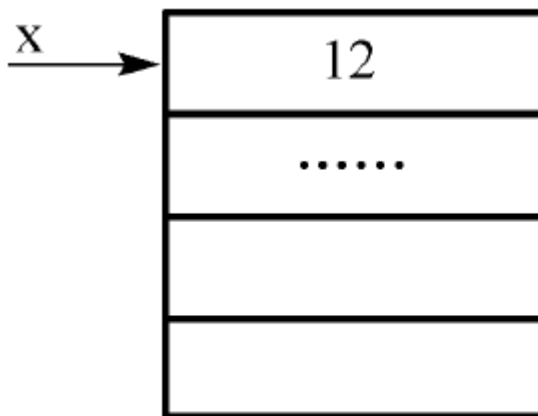
- 可以使用**Python** 内置函数**type()**来查询变量的类型。
- 例：
- `>>> type(x)` #查看x 的数据类型， x 是整型（**int**）变量
- `(class 'int')`
- `>>> type(y)` #查看y 的数据类型， y 是浮点型（**float**）
变量
- `(class 'float')`
- `>>> type(z)` #查看z 的数据类型， z 是字符串型（**string**）
变量
- `(class 'str')`

(2) 变量与地址的关系

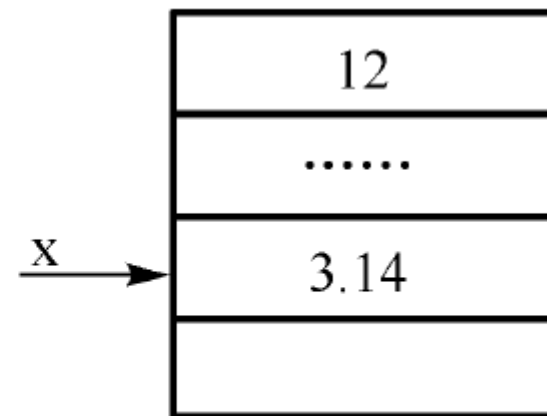
- 在C语言中，编译系统要为变量分配内存空间，当改变变量的值时，改变的是该内存空间的内容，在程序运行过程中，变量的地址是不再发生改变的，即变量所对应的内存空间是固定的。
- Python语言则不同，它采用的是基于值的内存管理方式，不同的值分配不同的内存空间。

- 当给变量赋值时，**Python**解释器为该值分配一个内存空间，而变量则指向这个空间，当变量的值被改变时，改变的并不是该内存空间的内容，而是改变了变量的指向关系，使变量指向另一内存空间。
- 这可理解为，**Python** 变量并不是某一个固定内存单元的标识，而是对内存中存储的某个数据的引用（**reference**），这个引用是可以动态改变的。

- 例：执行下面的赋值语句后，**Python** 在内存中创建数据**12**，并使变量**x** 指向这个整型数据，因此可以说变量**x** 现在是整型数据。
- `>>> x=12`
- `>>> print(x)`
- `12`
- 如果进一步执行下面的赋值语句，则**Python** 又在内存中创建数据**3.14**，并使变量**x** 改为指向这个浮点型（实型）数据，因此变量**x** 的数据类型现在变成了浮点型。
- `>>> x=3.14`
- `>>> print(x)`
- `3.14`



(a) x指向12



(b) x指向3.14

- **Python** 具有自动内存管理功能，对于没有任何变量指向的值（称为垃圾数据），**Python** 系统会自动将其删除。
- 例：当**x** 从指向**12** 转而指向**3.14** 后，数据**12** 就变成了没有被变量引用的垃圾数据，**Python**会回收垃圾数据的内存单元，以便提供给别的数据使用，这称为**垃圾回收**（**garbage collection**）。

- 也可以使用**del** 语句删除一些对象引用。
- 例：
- **del x**
- 删除**x** 变量后，如果再使用它，将出现变量未定义错（**name 'x' is not defined**）。

- **Python 的id()函数可以返回对象的内存地址。**

- **>>> a=2.0**

- **>>> b=2.0**

- **>>> id(a)**

- **35433824**

- **>>> id(b)**

- **35853488**

- **>>> a=2**

- **>>> b=2**

- **>>> id(a)**

- **1600796448**

- **>>> id(b)**

- **1600796448**

- **Python** 解释器会为每个出现的对象分配内存单元，即使它们的值相等，也会这样。
- 例如，执行 **a=2.0**, **b=2.0** 这两个语句时，会先后为 **2.0** 这个 **float** 类型对象分配内存单元，然后将 **a** 与 **b** 分别指向这两个对象。所以 **a** 与 **b** 指向的不是同一对象。

- 为了提高内存利用效率，对于一些简单的对象，如一些数值较小（-256~256）的整型（**int**）对象，**Python** 采取重用对象内存的办法。
- 例：执行**a=2**，**b=2** 时，由于**2** 作为简单的**int** 类型且数值小，**Python** 不会两次为其分配内存单元，而是只分配一次，然后将**a** 与**b** 同时指向已分配的对象。

- 如果赋值是较大的数值，情况就跟前面的一样了。

- 例：

- **>>> a=2222**

- **>>> b=2222**

- **>>> id(a)**

- **35853872**

- **>>> id(b)**

- **35852800**

4. Python 标识符

- 标识符 (**identifier**) 主要用来表示常量、变量、函数和类型等程序要素的名字，是只起标识作用的一类符号。
- 在**Python** 中，标识符由字母、数字和下画线 (**_**) 组成，但不能以数字开头，标识符中的字母是区分大小写的。
- 例：
- **abc**、**a_b_c**、**Student_ID** 都是合法的标识符；
- **sum**、**Sum**、**SUM** 代表不同的标识符。

- 单独的下画线（`_`）是一个特殊变量，用于表示上一次运算的结果。

- 例：

- `>>> 55`

- `55`

- `>>> _+100`

- `155`

5. Python 关键字

- 所谓关键字（**key word**），就是Python 语言中事先定义的、具有特定含义的标识符，有时又称保留字。
- 关键字不允许另作他用，否则执行时会出现语法错误。
- 可以在使用**import** 语句导入**keyword** 模块后使用**print(keyword.kwlist)**语句查看所有Python 关键字。

- **>>> import keyword**
- **>>> print(keyword.kwlist)**
- **['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']**

1.4 Python 数据类型

- **Python** 提供了一些内置的数据类型，它们由系统预定义好，在程序中可以直接使用。
- **Python** 数据类型包括数值型、字符串型、布尔型等基本数据类型，这是一般程序设计语言都有的数据类型。
- 为了使程序能描述现实世界中各种复杂数据，**Python** 还有列表、元组、字典和集合等复合数据类型，这是 **Python** 中具有特色的数据类型。

1.4.1 数值类型

- Python 支持3 种不同的数值数据类型:
- 整型 (`int`)
- 浮点型 (`float`)
- 复数型 (`complex`)

1. 整型数据 (int)

- 在Python 3.x 中，整型数据的值在计算机内的表示不是固定长度的，只要内存许可，整数可以扩展到任意长度，整数的取值范围几乎包括了全部整数（无限大），这给大数据的计算带来便利。

- **Python 的整型常量有以下4种表示形式。**
- **(1) 十进制整数**
- **如120、0、-374等。**
- **(2) 二进制整数**
- **它以0b或0B（数字0加字母b或B）开头，后接数字0,1的整数。**
- **例：**
- **>>> 0b1111**
- **15**
- **0b1111表示一个二进制整数，其值等于十进制数15。**

- (3) 八进制整数
- 它是以0o 或0O (数字0 加小写字母o 或大写字母O) 开头, 后接数字0~7的整数。
- 例:
- >>> 0o127
- 87
- 0o127 表示一个八进制整数, 其值等于十进制数87。

- (4) 十六进制整数
- 它是以0x 或0X 开头，后接0~9 和A~F（或用小写字母）字符的整数。
- 例：
- >>> 0xabc
- 2748
- 0xabc 表示一个十六进制整数，其值等于十进制数2748。

- **bin()**: 将其他进制的数转换为二进制数;
- **int()**: 将其他进制的数转换为十进制数;
- **oct()**: 将其他进制的数转换为八进制数;
- **hex()**: 将其他进制的数转换为十六进制数;
- 例:
- **>>> bin(10)**
- **'0b1010'**
- **>>> int(0b1010)**
- **10**
- **>>> oct(10)**
- **'0o12'**
- **>>> hex(30)**
- **'0x1e'**

- 为了提高数值型（包括**float**型）数据的可读性，**Python**允许为数据增加下划线作为分隔符（类似生活中的逗号），这些下划线不会影响数值本身。
- 例如：
- `>>> x=1_000_000`
- `>>> print(x)`
- `1000000`
- `>>> y=123_456.789_123`
- `>>> print(y)`
- `123456.789123`

2. 浮点型数据 (float)

- 浮点型数据表示一个实数，有以下两种表示形式。
- (1) 十进制小数形式
- 它由数字和小数点组成，如**3.23**、**34.0**、**0.0** 等。
- 浮点型数据允许小数点后面没有任何数字，表示小数部分为0，如**34.**表示**34.0**。

- (2) 指数形式

- 指数形式即用科学计数法表示的浮点数，用字母e（或E）表示以10为底的指数，e之前为数字部分，之后为指数部分，且两部分必须同时出现，指数必须为整数。

- 例：

- >>> 45e-5

- 0.00045

- >>> 45e-6

- 4.5e-05

- >>> 9.34e2

- 934.0

- 45e-5、45e-6、9.34e2 是合法的浮点型常量，分别代表 45×10^{-5} 、 45×10^{-6} 、 9.34×10^2 。

- 字母e（或E）前必有数， e（或E）后必为整数。
- 例： e4、 3.4e4.5、 34e 等是非法的浮点型常量。

- 对于浮点数，**Python 3.x** 默认提供**17** 位有效数字的精度，相当于C 语言中的双精度浮点数。
- 例：
- **>>> 1234567890123456.0**
- **1234567890123456.0**
- **>>> 1234567890123456789.0**
- **1.2345678901234568e+18**
- **>>> 1234567890123456789.0+1**
- **1.2345678901234568e+18**
- **>>> 1234567890123456789.0+1-1234567890123456789.0**
- **0.0**
- **>>> 1234567890123456789.0-1234567890123456789.0+1**
- **1.0**

- 在Python 中，为什么 $1234567890123456789.0+1-1234567890123456789.0$ 的结果为 0.0 ，而 $1234567890123456789.0-1234567890123456789.0+1$ 的结果为 1.0 ，这就需要了解Python 浮点数的表示方法。
- 数学上 $1234567890123456789.0+1$ 等于 1234567890123456790.0 ，但由于浮点数受17 位有效数字的限制，Python 中 $1234567890123456789.0+1$ 的结果等于 $1.2345678901234568e+18$ ，其中加1的结果被忽略了， $1234567890123456789.0+1$ 再减去 1234567890123456789.0 的结果为 0 。

- **1234567890123456789.0-**

1234567890123456789.0+1 先执行的是减法运算，得到0，然后再加上1，结果为1.0，所以计算机中的计算与数学上的计算是不同的，其原因是计算机中的计算必须依赖于计算机的计算能力。

- 在进行问题求解时必须注意这种差别，这就是计算思维的思想。

- 例:
- `>>> 1.001*10`
- `10.00999999999999998`
- 为什么Python 中`1.001*10` 结果是
`10.00999999999999998`，而不是`10.01`，其原因在于
十进制小数转换为二进制小数时可能出现无限小
数问题，而Python 在存储小数时使用的是双精度
浮点数，这种数只可以保存一定位数的有效数字，
所以当遇到无限小数时就会出现损失精度的问题。

● 例:

● $\ggg 1.001-2$

● -0.999000000000000001

● $\ggg 2.01-3$

● -0.9900000000000000002

● $\ggg 3.01-4$

● -0.9900000000000000002

● $\ggg 3.01*10$

● 30.09999999999999998

3. 复数型数据 (complex)

- 在科学计算问题中常会遇到复数运算问题。
- 例：数学中求方程的复根、电工学中交流电路的计算、自动控制系统中传递函数的计算等都要用到复数运算。
- **Python** 提供了复数类型，这使得有关复数运算问题变得方便容易。

- 复数类型数据的形式为： $a+bJ$ 或 $a+bj$
- 其中， a 是复数的实部， b 是复数的虚部， J 表示 -1 的平方根（虚数单位）。
- 虚数单位既可以用大写字母 J 也可以写成小写字母 j ，注意不是数学上的 i 。
- 例：
- `>>> x=12+34J`
- `>>> print(x)`
- `(12+34j)`

- 可以通过`x.real` 和`x.imag` 来分别获取复数`x` 的实部和虚部，
可以用内置函数`abs()` 求复数的模，结果都是浮点型。

- 例：

- `>>> x=2+3j`

- `>>> x.real`

- `2.0`

- `>>> x.imag`

- `3.0`

- `>>> abs(3+4j)`

- `5.0`

1.4.2 字符串类型 (str)

1. Python 标准字符串

- 在 **Python** 中定义一个标准字符串可以使用单引号、双引号和三引号（三个单引号或三个双引号），这使得**Python** 输入文本更方便。
- 例如，当字符串的内容中包含双引号时，就可以用单引号定义，反之亦然。

- 例:
- `>>> s='uestc'`
- `>>> print(s)`
- `uestc`
- `>>> print(s[0])` #输出字符串的第1 个字符
- `u`
- `>>> print(s[2:4])` #输出字符串的第3~4 个字符
- `st`
- `>>> t="I am 'Python'"`
- `>>> print(t)`
- `I am 'Python'`

- 用单引号或双引号括起来的字符串必须在一行内表示，这是最常见的表示字符串的方法，而用三引号括起来的字符串可以是多行的（文本块）。
- 用三引号括起来的字符串可以用作程序的多行注释（块注释，其本质是一个没有被使用的字符串表达式），“#”为单行注释。例：
 - '''
 - 这部分是多行注释，也称块注释
 - 这一行还是注释
 - '''
 - `print("Goodbye!")` #这是单行注释，到本行末尾结束

- **>>> s="""I'm "Python"!"""**

- **>>> print(s)**

- **I'm "Python"!**

- **>>> s="""**

- **1.AAAA**

- **2.BBBB**

- **3.CCCC**

- **请选择: """**

- **>>> print(s)**

- **1.AAAA**

- **2.BBBB**

- **3.CCCC**

- **请选择:**

- **Python 字符串中的字符元素不能被改变，给串中某个位置元素赋值会导致错误。**

- **例：**

- **>>> s="ABCDEFGH"**

- **>>> s[1]="8" #试图改变第2 个字符导致出错**

- **>>> s="abcdefgh"**

- **>>> t=s[0:3]+"X"+s[4::]**

- **>>> t**

- **'abcXefg'**

- 在**Python** 中，修改字符串只能重新赋值，每修改一次字符串就生成一个新的字符串对象，这看起来好像会造成处理效率下降。
- 其实，**Python** 系统会自动对不再使用的字符串进行垃圾回收，所以，新的对象重用了前面已有字符串的空间。

2. 转义字符

Python 常用的转义字符及其含义

转 义 字 符	十进制 ASCII 代码值	说 明
\0	0	空字符
\a	7	产生响铃声
\b	8	退格符 (Backspace)
\n	10	换行符
\r	13	回车符
\t	9	水平制表符 (Tab)
\\	92	反斜杠
\'	44	单引号
\"	34	双引号
\ddd		1~3 位八进制数表示的 ASCII 码所代表的字符
\xhh		1~2 位十六进制数表示的 ASCII 码所代表的字符

- 例：转义字符的用法示例。
- `print("**ab*c\t*de***\ttg**\n")`
- `print("h\nn***k")`
- 程序运行结果如下：
- `**ab*c` □ □ `*de***` □ □ `tg**`
- （空一行）
- `h`
- `n***k`
- 其中，□表示一个空格。

- 如果不想让反斜杠发生转义，可以在字符串前面添加一个r或R，表示原始字符串。

- 例：

- `>>> print('C:\some\name')` #“\n”当转义字符

- `C:\some`

- `ame`

- `>>> print(r'C:\some\name')` #“\n”不发生转义

- `C:\some\name`

3. 基本的字符串函数

(1) eval()函数

- 与字符串有关的一个重要函数是**eval**，其调用格式为：
- **eval(字符串)**
- **eval()**函数的作用就是删去字符串最外面那层引号。

- 例:
- `>>> c='23+45'`
- `>>> eval(c)`
- `68`
- 将一个给定对象转换为字符串: `repr()`或`str()`
- 例:
- `>>> repr(123)`
- `'123'`
- `>>> str(True)`
- `'True'`

(2) len()函数

- len()函数返回字符串的长度，即字符串中所包含的字符个数，其调用格式为：
- len(字符串)
- 例：
- >>> s='abcd'
- >>> len(s)
- 4

1.4.3 布尔类型 (bool)

- 在Python 中，布尔型数据有True 和False，分别代表逻辑真和逻辑假。
- 例：
- `>>> x=10`
- `>>> x>x+1`
- False
- `>>> x-1<x`
- True

- 在Python 中，逻辑值True 和False 实际上是分别用整型值1 和0 参与运算。
- 例：
- `>>> x=False`
- `>>> x+(5>4)`
- `1`

1.4.4 复合数据类型

- 数值类型、布尔类型数据不可再分解为其他类型，而列表、元组、字典和集合类型的数据包含多个相互关联的数据元素，所以称它们为复合数据类型。
- 字符串其实也是一种复合数据，其元素是单个字符。

- 列表、元组和字符串是有顺序的数据元素的集合体，称为**序列**（sequence）。
- 序列具有顺序存取的特征，可以通过各数据元素在序列中的**位置编号（索引）**来访问数据元素。
- 字典和集合属于无顺序的数据集合体，数据元素没有特定的排列顺序，因此不能像序列那样通过位置编号来访问数据元素。

1. 列表 (list)

- 列表 (list) 是Python 中使用较多的复合数据类型，可以完成大多数复合数据结构的操作。
- 列表是写在中括号之间、用逗号分隔的元素序列，元素的类型可以不相同，可以是数字、单个字符、字符串甚至可以包含列表（所谓嵌套）。

- 例:
- `>>> m=['brenden',45.3,911,'john',32]`
- `>>> print(m)` #输出完整列表
- `['brenden', 45.3, 911, 'john', 32]`
- `>>> print(m[0])` #输出列表的第1 个元素
- `brenden`

- 与Python 字符串不同的是，列表中的元素是可以改变的。

- 例：

- `>>> a=[1,2,3,4,5,6]`

- `>>> a[0]=9`

- `>>> a`

- `[9, 2, 3, 4, 5, 6]`

2. 元组 (tuple)

- 元组 (tuple) 是写在小括号之间、用逗号隔开的元素序列。
- 元组中的元素类型也可以不相同。
- 元组与列表类似，不同之处在于元组的元素不能修改，相当于只读列表。

- 例:
- `>>> m=('brenden',45.3,911,'john',32)`
- `>>> print(m)` #输出完整元组
- `('brenden', 45.3, 911, 'john', 32)`
- `>>> print(m[0])` #输出元组的第1 个元素
- `brenden`

- 注意：
- 空的圆括号表示空元组。
- 当元组只有一个元素时，必须以逗号结尾。
- 例：
- `>>> ()` #空元组
- `()`
- `>>> (9,)` #含有一个元素的元组
- `(9,)`
- `>>> (9)` #整数9
- `9`

- 任何一组以逗号分隔的对象，当省略标识序列的括号时，默认为元组。

- 例：

- `>>> 2,3,4`

- `(2, 3, 4)`

- `>>> s=2,3,4`

- `>>> s`

- `(2, 3, 4)`

- 元组与字符串类似，元素不能二次赋值。
- 可以把字符串看成一种特殊的元组。
- 以下给元组赋值是无效的，因为元组是不允许更新的，而列表允许更新。
- `>>> tup=(1,2,3,4,5,6)`
- `>>> list=[1,2,3,4,5,6]`
- `>>> tup[2]=1000` #在元组中是非法应用
- `>>> list[2]=1000` #在列表中是合法应用

- 元组和列表的区别：
- 列表元素用中括号[]括起来，且元素的个数及元素的值可以改变。
- 元组元素用小括号()括起来，且不可以更改。
- 元组可以看成只读的列表。

3. 字典 (dict)

- 字典 (**dictionary**) 是写在大括号之间、用逗号分隔的元素集合，其元素由关键字 (**key**，也称为键) 和关键字对应的值 (**value**) 组成，通过关键字来存取字典中的元素。

- 列表和元组是有序的对象结合，字典是无序的对象集合。
- 字典是一种映射类型（**mapping type**），它是一个无序的“**关键字:值**”对集合。
- 关键字必须使用不可变类型，也就是说列表和包含可变类型的元组不能做索引关键字。
- 在同一个字典中，关键字还必须互不相同。

- 例:
- `>>> dict={'name':'brenden','code':410012,'dept':'sales'}`
- `>>> print(dict)` #输出完整的字典
- `{'name': 'brenden', 'dept': 'sales', 'code': 410012}`
- `>>> print(dict['code'])` #输出关键字为 “code” 的值
- `410012`
- `>>> dict['payment']=4500` #在字典中添加一个 “关键字:值” 对
- `>>> print(dict)` #输出完整的字典
- `{'name': 'brenden', 'dept': 'sales', 'code': 410012, 'payment': 4500}`

4. 集合 (set)

- 集合 (**set**) 是一个无序且包含不重复元素的数据类型。
- 基本功能是进行成员关系测试和消除重复元素。
- 可以使用大括号或者**set()**函数创建集合类型。
- 注意：创建一个空集合必须用**set()**而不是**{}**，因为**{}**是用来创建一个空字典。

- 例:
- `>>> student={'Tom','Jim','Mary','Tom','Jack','Rose'}`
- `>>> print(student)` #重复的元素被自动去掉
- `{'Jim', 'Tom', 'Jack', 'Rose', 'Mary'}`
- `>>> s={}`
- `>>> type(s)`
- `<class 'dict'>`
- `>>> s=set()`
- `>>> type(s)`
- `<class 'set'>`

1.5 常用系统函数

- **Python** 的标准库包含很多模块，每个模块中定义了很多有用的函数，这些函数称为系统函数。
- 任何**Python** 程序都可直接或间接地调用这些函数。

- 例：
- 数学库模块（**math**）提供了很多数学运算函数；
- 复数模块（**cmath**）提供了用于复数运算的函数；
- 随机数模块（**random**）提供了用来生成随机数的函数；
- 时间（**time**）和日历（**calendar**）模块提供了能处理日期和时间的函数。

- 在调用系统函数之前，先要使用**import** 语句导入相应的模块，格式如下：
- **import 模块名**
- 该语句将模块中定义的函数代码复制到自己的程序中，然后就可以访问模块中的任何函数，其方法是在函数名前面加上“**模块名.**”。

- 例：调用数学模块`math`中的平方根函数`sqrt()`，语句如下：

- `>>>import math` #导入`math`模块

- `>>>math.sqrt(2)` #调用`sqrt()`函数

- `1.4142135623730951`

- `>>> dir(math)`

- `>>> help(math.sqrt)`

- 另一种导入模块的方法，格式如下：
- **from 模块名 import 函数名**
- 该语句从指定模块中导入指定函数的定义，这样调用模块中的函数时，不需要在前面加上“模块名。”。
- 例：
- **>>> from math import sqrt**
- **>>> sqrt(2)**
- **1.4142135623730951**

- 如果希望导入模块中的所有函数定义，则函数名用“*”。
- 格式如下：
- **from 模块名 import ***
- 这样调用指定模块中的任意函数时，都不需要在前面加“模块名.”。
- 使用这种方法固然省事方便，但当多个模块有同名的函数时，会引起混乱，使用时要注意。

- 导入模块的多个对象:
- **from 模块名 import 对象1,对象2,.....**
- 在导入模块的同时，给**模块**取别名:
- **import 模块名 as 别名**
- 在导入模块的单个对象时，给该**对象**取别名:
- **from 模块名 import 对象 as 别名**

1.5.1 常用模块函数

1. math模块函数

- **math**模块主要处理数学相关的运算。

(1) 数学常量

- **e**: 返回自然常数 e （自然对数的底）。
- **pi**: 返回圆周率 π 的值。

(2) 绝对值和平方根函数

- **fabs(x)**: 返回 x 的绝对值（返回值为浮点数）。
- 例: **fabs(-10)**返回10.0。
- **sqrt(x)**: 返回 x 的平方根（ $x>0$ ）。
- 例: **sqrt(4)**返回2.0。

(3) 幂函数和对数函数

- **pow(x,y):** 返回x的y次幂。
- 例: **pow(2,3)**返回8.0
- **exp(x):** 返回自然常数e的x次幂。
- 例: **exp(1)**返回2.718281828459045。
- **log(x[,base]):** 返回x的自然对数。
- 例: **log(e)**返回1.0。
- 可以使用**base**参数来改变对数的底。
- 例: **log(100,10)**返回2.0。
- **log10(x):** 返回x的常用对数。
- 例: **log10(100)**返回2.0。

(4) 取整和求余函数

- **ceil(x)**: 对x向上取整。
- 例: **ceil(4.1)**返回5。
- **floor(x)**: 对x向下取整。
- 例: **floor(4.9)**返回4。
- **fmod(x,y)**: 返回求x/y的余数（返回值为浮点数）。
- 例: **fmod(7,4)**返回3.0。

(5) 弧度角度转换函数

- **degrees(x):** 将弧度转换为角度。
- 例: **degrees(pi)**返回**180.0**。
- **radians(x):** 将角度转换为弧度。
- 例: **radians(90)**返回**1.5707963267948966**。

(6) 三角函数和反三角函数

- **$\sin(x)$** : 返回 x 的正弦值 (x 为弧度) 。
- 例: **$\sin(\pi/2)$** 返回1.0。
- **$\cos(x)$** : 返回 x 的余弦值 (x 为弧度) 。
- 例: **$\cos(\pi)$** 返回-1.0。
- **$\tan(x)$** : 返回 x 的正切值 (x 为弧度) 。
- 例: **$\tan(\pi/4)$** 返回0.99999999999999999999 (数学上为1) 。

- **asin(x)**: 返回x的反正弦值（返回值为弧度）。
- 例: **degrees(asin(1))**返回90.0。
- **acos(x)**: 返回x的反余弦值（返回值为弧度）。
- 例: **degrees(acos(-1))**返回180.0。
- **atan(x)**: 返回x的反正切值（返回值为弧度）。
- 例: **degrees(atan(1))**返回45.0。

2. cmath模块函数

- cmath模块函数与math模块函数基本一致。
- 包括圆周率 π 、自然常数 e 等常量。
- 复数的幂指数、对数函数、平方根函数、三角函数等。
- cmath模块函数名和math模块函数名相同。
- 只是math模块对实数运算，cmath模块对复数运算。

- 例:
- `>>> import cmath`
- `>>> cmath.pi`
- `3.141592653589793`
- `>>> cmath.sqrt(-1)`
- `1j`
- `>>> cmath.sin(1)`
- `(0.8414709848078965+0j)`
- `>>> cmath.log10(100)`
- `(2+0j)`
- `>>> cmath.exp(100+10j)`
- `(-2.255522560520288e+43-1.4623924736915717e+43j)`

- **cmath**模块包括复数运算特有的函数。
- 复数 $x=a+bi$, **phase(x)**函数返回复数 x 的幅角, 即 **atan(b/a)**。
- 例:
- **>>> from cmath import ***
- **>>> phase(1+1j)**
- **0.7853981633974483**
- **>>> phase(1+2j)**
- **1.1071487177940904**

- **cmath**模块的**polar()**函数和**rect()**函数可以对复数进行极坐标表示和笛卡儿表示方法的转换。
- **polar(x)**函数将复数的笛卡儿坐标表示转换为极坐标表示，输出为一个二元组(r,p)，复数的模 $r=\text{abs}(x)$ ，幅角 $p=\text{phase}(x)$ 。
- **rect(r, p)**函数将复数的极坐标表示转换为笛卡儿坐标表示，输出为 $r*\cos(p)+r*\sin(p)*1j$ 。

- 例:
- `>>> c=3+4j`
- `>>> r,p=polar(c)`
- `>>> print(r,p)`
- `5.0 0.9272952180016122`
- `>>> rect(r,p)`
- `(3.0000000000000004+3.9999999999999996j)`

3. random模块函数

(1) 随机数种子

- 使用**seed(x)**函数可以设置随机数生成器的种子，通常在调用其他随机模块函数之前调用此函数。
- 对于相同的种子，每次调用随机函数生成的随机数是相同的。
- 默认将系统时间作为种子值，使得每次产生的随机数都不一样。

(2) 随机挑选和排序

- **choice(seq):** 从序列的元素中随机挑选一个元素。
- **sample(seq,k):** 从序列中随机挑选k个元素。
- **shuffle(seq):** 将序列的所有元素随机排序。

- 例: `choice([0,1,2,3,4,5,6,7,8,9])`, 从0到9中随机挑选一个整数。
- `>>> from random import *`
- `>>> choice([0,1,2,3,4,5,6,7,8,9])`
- 6
- `>>> choice([0,1,2,3,4,5,6,7,8,9])`
- 0

(3) 生成随机数

- 下面生成的随机数符合均匀分布 (**uniform distribution**)，即范围内每个数字出现的概率相等。
- **random()**: 随机生成一个 $[0,1)$ 范围内的实数。
- **uniform(a,b)**: 随机生成一个 $[a,b]$ 范围内的实数。
- **randrange(a,b,c)**: 随机生成一个 $[a,b)$ 范围内以 c 递增的整数，省略 c 时以1递增，省略 a 时初值为0。
- **randint(a,b)**: 随机生成一个 $[a,b]$ 范围内的整数，相当于**randrange(a,b+1)**。

4. time模块函数

- **time()**: 返回当前时间的时间戳[chuō]。时间戳是从Epoch（1970年1月1日00:00:00 UTC）开始所经过的秒数，不考虑闰秒。
- **localtime([secs])**: 接收从Epoch开始的秒数，并返回一个时间元组。时间元组包含9个元素，相当于struct_time结构。省略秒数时，返回当前时间戳对应的的时间元组。

- **>>> from time import ***
- **>>> localtime()**
- **time.struct_time(tm_year=2019, tm_mon=3,
tm_mday=10, tm_hour=23, tm_min=33, tm_sec=43,
tm_wday=6, tm_yday=69, tm_isdst=0)**

- **asctime([tupletime]):** 接收一个时间元组，并返回一个日期时间字符串。时间元组省略时，返回当前系统日期和时间。
- 例：
- **>>> asctime()**
- **'Sun Mar 10 23:37:06 2019'**
- **>>> asctime(localtime(time()))**
- **'Sun Mar 10 23:37:43 2019'**

- **ctime([secs]):** 类似于**asctime(localtime([secs]))**,
不带参数时与**asctime()**功能相同。
- 例:
- **>>> ctime(time())**
- **'Sun Mar 10 23:39:22 2019'**

- **strftime(日期格式):** 按指定的日期格式返回当前日期。
- 例:
- **>>> strftime("%Y-%m-%d %H:%M:%S")**
- **'2019-03-10 23:41:13'**

Python时间日期格式化符号有：

- **%y**：表示两位数的年份（**00~99**）；
- **%Y**：表示4位数的年份（**000~9999**）；
- **%m**：表示月份（**01~12**）；
- **%d**：表示月中的一天（**0-31**）；
- **%H**：表示24小时制小时数（**0-23**）；
- **%I**：表示12小时制小时数（**01~12**）；
- **%M**：表示分钟数（**00~59**）；
- **%S**：表示秒（**00-59**）。

5. calendar模块函数

- 日历（**calendar**）模块提供与日历相关的功能。在默认情况下，日历把星期一作为一周的第一天，星期日为最后一天。要改变这种设置，可以调用**setfirstweekday()**函数。
- **setfirstweekday(weekday)**: 设置每个星期的开始工作日代码。星期代码是0~6，代表星期一~星期日。
- **firstweekday()**: 返回当前设置的每个星期开始工作日。默认是0，即星期一。
- **isleap(year)**: 如果指定年份是闰年返回**True**，否则为**False**。
- **leapdays(y1,y2)**: 返回在[y1,y2)范围内的闰年数。
- **calendar(year)**: 返回指定年份的日历。

● 例：

● `>>> from calendar import *`

● `>>> c=calendar(2021)`

● `>>> print(c)`

● `#将输出2021年的日历（略）。`

● `month(year,month)`： 返回指定年份和月份的日历。

● 例：

● `>>> c=month(2021,3)`

● `>>> print(c)`

● `#将输出2021年3月的日历（略）。`

- **monthcalendar(year,month):** 返回整数列表，每个子列表表示一个星期（从星期一到星期日）。
- 例：
- **>>> c=monthcalendar(2019,3)**
- **>>> print(c)**
- **[[0, 0, 0, 0, 1, 2, 3], [4, 5, 6, 7, 8, 9, 10], [11, 12, 13, 14, 15, 16, 17], [18, 19, 20, 21, 22, 23, 24], [25, 26, 27, 28, 29, 30, 31]]**

- **monthrange(year,month):** 返回两个整数，第1个数代表指定年和月的第一天是星期几，第二个数代表所指定月份的天数。
- 例:
- **>>>monthrange(2016,1)**
- **(4,31)**
- 表明**2016年1月**的第1天是星期五，该月有**31**天。
- **weekday(year,month,day):** 返回给定日期的星期代码。

1.5.2 常用内置函数

- Python还有一类函数叫内置函数(built-in function)。
- Python内置函数包含在模块__builtins__中，该模块在启动Python解释器时自动装入内存，而其他的模块函数都要等使用import语句导入时才会装入内存。
- 例：查看Python的全部内置函数
- >>> dir(__builtins__)
- >>> help(id)

- 内置函数随着Python解释器的运行而创建，在程序中可以随时调用这些函数。
- 例： `print()` ； `type()`； `id()` 。
- `>>> help(type)`
- `>>> help(print)`
- `print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`
- `sep`: 输出项之间的分隔符，默认值为空格；
- `end`: 输出末尾的结束符，默认为回车换行符；
- `flush`: 用于控制输出缓存，保持默认值False可以获得较好的性能。

- `print("锄禾日当午","汗滴禾下土",sep="**",end="##")`
- `print("谁知盘中餐","粒粒皆辛苦")`
- `print("悯农")`

- `print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`
- **file:** 输出的目标位置，默认值为系统的标准输出，即屏幕；
- 例：
- `f=open(r"d:\t.txt","w")` #打开文件以便写入，详见后面第10章PPT
- `print("锄禾日当午",file=f)`
- `print("汗滴禾下土",file=f)`
- `f.close()`

1. range() 函数

迭代器 (iterator) 和生成器 (generator) :

- range()函数返回的是可迭代对象，迭代时产生指定范围的数字序列。
- 迭代器可以看成是一个特殊的对象，每次调用该对象时会返回自身的下一个元素。
- 生成器是能够返回一个迭代器的函数。
- 迭代器不要求事先准备好整个迭代过程中所有的元素。
- 迭代器仅仅在迭代到某个元素时才计算该元素，而在这之前或之后，元素可以不存在。
- 这个特点使得迭代器能节省内存空间，特别适合用于遍历一些很大的或无限的集合。

1. range() 函数

迭代器 (iterator) 和生成器 (generator) :

- range()函数返回的是可迭代对象，迭代时产生指定范围的数字序列。
- 迭代器可以看成是一个特殊的对象，每次调用该对象时会返回自身的下一个元素。
- 生成器是能够返回一个迭代器的函数。
- 迭代器不要求事先准备好整个迭代过程中所有的元素。
- 迭代器仅仅在迭代到某个元素时才计算该元素，而在这之前或之后，元素可以不存在。
- 这个特点使得迭代器能节省内存空间，特别适合用于遍历一些很大的或无限的集合。

- **>>> help(range)**
- **class range(object)**
- **| range(stop) -> range object**
- **| range(start, stop[, step]) -> range object**

- **range()**函数的调用格式:
- **range([start,]end[,step])**
- **range()**函数产生的数字序列从**start**开始，默认是从**0**开始；序列到**end**结束，但不包含**end**；如果指定了可选的步长**step**，则序列按步长增加，默认为**1**。
- 例:
- **>>> range(2) #产生可迭代对象**
- **range(0,2)**

- 使用内置函数“**iter(可迭代对象)**”可以获取可迭代对象的迭代器。
- 使用内置函数“**next(迭代器对象)**”可以得到迭代器对象的下一个元素，如果迭代器对象没有新的元素，则抛出**StopIteration**异常。

- 例:
- `>>> range(2)`
- `range(0, 2)`
- `>>> s=range(2)` #产生可迭代对象
- `>>> t=iter(s)` #产生迭代器对象
- `>>> t`
- `<range_iterator object at 0x02178968>`
- `>>> next(t)` #产生迭代器对象的下一个元素
- `0`
- `>>> next(t)`
- `1`
- `>>> next(t)`
- #迭代器对象没有新的元素时导致StopIteration异常

- 可以利用**range()**函数和**list()**函数产生一个列表。
- 例：
- **>>> list(range(2,15,3))**
- **[2, 5, 8, 11, 14]**
- **>>> list(range(5))**
- **[0, 1, 2, 3, 4]**

- 可以利用**range()**函数和**tuple()**函数产生一个元组。
- 例：
- **>>> tuple(range(2,15,3))**
- **(2, 5, 8, 11, 14)**
- **>>> tuple(range(5))**
- **(0, 1, 2, 3, 4)**

2. 数值运算函数

- Python有些内置函数用于数值运算。
- **abs(x)**: 返回x的绝对值，结果保持x的类型。x为复数时返回复数的模。
- 例:
- **>>> abs(-10)**
- **10**
- **>>> abs(3+4j)**
- **5.0**

内置的数值运算函数

函数	描述
$\text{abs}(x)$	x 的绝对值
$\text{divmod}(x, y)$	$(x//y, x\%y)$, 输出为二元组形式(也称为元组类型)
$\text{pow}(x, y)$ 或 $\text{pow}(x, y, z)$	$x ** y$ 或 $(x ** y) \% z$, 幂运算
$\text{round}(x)$ 或 $\text{round}(x, d)$	对 x 四舍五入, 保留 d 位小数, 无参数 d 则返回四舍五入的整数值
$\text{max}(x_1, x_2, \cdots, x_n)$	x_1, x_2, \cdots, x_n 的最大值, n 没有限定, 可以任意数量
$\text{min}(x_1, x_2, \cdots, x_n)$	x_1, x_2, \cdots, x_n 的最小值, n 没有限定, 可以任意数量

- **pow(x,y[,z]):** 其中的**x**、**y**是必选参数，**z**是可选参数。省略**z**时，返回**x**的**y**次幂，结果保持**x**或**y**的类型。如果使用了参数**z**，其结果是**x**的**y**次方再对**z**求余数。

- 例：

- **>>> pow(2,3)**

- **8**

- **>>> pow(2,3,3)**

- **2**

- **round(x[,n]):** 用于对浮点数进行四舍五入运算，返回值为浮点数。它有一个可选的小数位数参数。如果不提供小数位参数，它返回与第一个参数最接近的整数，但仍然是浮点类型。第二个参数表示将结果精确到小数点后指定位数。

- 例：

- **>>> round(3.46)**

- **3**

- **>>> round(3.14159,3)**

- **3.142**

- **divmod(x,y):** 把除法和取余运算结合起来，返回一个包含商和余数的元组。对整数来说，它的返回值就是 x/y 取商和 x/y 取余数的结果。
- 例:
- **>>> divmod(7,4)**
- **(1,3)**

3. Python系统的帮助信息

- 查看Python帮助信息可以使用内置函数**dir()**和**help()**。
- **dir()**函数的调用方法很简单，只需把想要查询的对象加到括号中就可以了，它返回一个列表，其中包含要查询对象的所有属性和方法。
- 例：
- `>>> import math`
- `>>> dir(math)`
- `.....`
- 用**dir()**函数查看数学模块**math**的属性和方法。

- 查看某个对象的帮助信息可以用**help()**函数。
- 例:
- **>>>help(str)**
- 查看**str**类型的详细帮助文档。
- 例:
- **>>> help(math.sqrt)**
- 显示**sqrt()**函数的帮助信息。

- 在Python解释器提示符下输入“**help()**”命令，可以进入联机帮助环境。
- **>>>help()**
- **.....**
- **help>**
- “**help>**”是帮助系统提示符，在该提示符下输入想了解的主题，Python就会给出有关主题的信息。

- 例：输入**modules**可以得到所有模块的信息。
- **help>modules**
-
- 输入某个模块的名字可以得到该模块的信息。
- 例：输入**math**可以得到**math**模块中所有函数的意义和用法。
- **help>math**
-

- 输入quit命令返回Python解释器提示符。
- help>quit
-
- >>>

- 编写实用的**Python**应用程序，可以充分利用丰富的系统资源，而不需要自己从原始的算法开始，从而显著提高程序设计的效率。
- 根据需要随时查阅有关**Python**的标准模块资料，可以事半功倍。

1.6 基本运算

- Python的运算符非常丰富，包括算术运算符、位运算符、关系运算符、逻辑运算符、成员运算符、身份运算符等。
- 运算符的优先级基本与C语言一致。

运算符说明	Python运算符	优先级	结合性	优先级顺序
小括号	()	19	无	高 ^ 低
索引运算符	x[i] 或 x[i1: i2 [:i3]]	18	左	
属性访问	x.attribute	17	左	
乘方	**	16	右	
按位取反	~	15	右	
符号运算符	+ (正号)、- (负号)	14	右	
乘除	*, /, //, %	13	左	
加减	+, -	12	左	
位移	>>, <<	11	左	
按位与	&	10	右	
按位异或	^	9	左	
按位或		8	左	
比较运算符	==, !=, >, >=, <, <=	7	左	
is 运算符	is, is not	6	左	
in 运算符	in, not in	5	左	
逻辑非	not	4	右	
逻辑与	and	3	左	
逻辑或	or	2	左	
逗号运算符	exp1, exp2	1	左	

1.6.1 算术运算

算术运算符：

- +(加)、-(减)、*(乘)、/(除)、//(整除)、%(求余)、
**(乘方)
- “/”：浮点数除法，其运算结果是一个浮点数，
即使被除数和除数都是整型，也返回一个浮点数。

- “//”：整数除，除法运算后返回商的整数部分。
如果结果为正数，可将其视为朝向小数位取整
(注意：不是四舍五入)。
- 当整数除以负数，“//”运算符将结果朝着最近的
整数“向上”四舍五入。
- “//”运算符并非总是返回整数结果。如果分子
或者分母是浮点型，它返回的值将会是浮点类型。

● >>> 5/3

● 1.6666666666666667

● >>> 5/3.0

● 1.6666666666666667

● >>> 5//3

● 1

● >>> 5//3.0

● 1.0

● >>> -5//3

● -2

● >>> 5%3

● 2

● >>> 5%3.0

● 2.0

- “**”：乘方运算符。实现乘方运算，其优先级高于乘除运算。

- 例：

- $>>>2^{**}10$

- 1024

- $>>>4*5/2^{**}3$

- 2.5

书写Python语言表达式应遵循的规则:

- (1) 表达式中的所有字符都必须写在一行, 特别是分式、乘方、带有下标的变量等。
- (2) 表达式中常量的表示、变量的命名以及函数的调用要符合规定。
- (3) 要根据运算符的优先顺序, 合理地加括号, 以保证运算顺序的正确性。特别是分式中的分子分母有加减运算时, 或分母有乘法运算, 要加括号表示分子分母的起始范围。

例：

数学式

$$\sin 45^{\circ} + 10^{-5} |a - b|$$

$$\frac{\sqrt[3]{c}}{a+b}$$

$$\frac{e^2 + \ln 10}{\sqrt{xy}}$$

Python表达式

```
(math.sin(45*math.pi/180))+1e-5*abs(a-b)
```

```
c**(1/3)/(a+b)
```

```
(math.exp(2)+math.log(10))/math.sqrt(x*y)
```

2. 浮点数的计算误差

- Python中能表示浮点数的有效数字是有限的，而在实际应用中数据的有效位数并无限制，这种矛盾，势必带来计算时的微小误差。
- 例：
- `>>>x=2.2`
- `>>>x-1.2`
- `1.0000000000000000002`

- 尽管在很多情况下这种误差不至于影响数值计算结果的实际应用，但对浮点数进行“等于”判断时就会得到截然不同的结果。
- 例：
- `>>>x=2.2`
- `>>>x-1.2==1`
- `False`
- 结论：对浮点数判断是否相等要慎用“`==`”运算符。

解决方案:

- 判断它们是否“约等于”，只要在允许的误差范围内，这种判断仍是有意义的。
- 所谓“约等于”是指两个浮点数非常接近，即它们的差足够小（具体误差可以根据实际情况进行调整）。
- 例：
- `>>>x=2.2`
- `>>>abs((x-1.2)-1)<1e-6`
- `True`

3. 数据类型的转换

- 例:
- `>>>2+3.0`
- `5.0`
- `>>>2.0+3//5`
- `2.0`

- 当算术表达式中需要违反自动类型转换规则，或者说自动类型转换规则达不到目的时，可以使用类型转换函数，将数据从一种类型强制转换到另一个类型，以满足运算要求。

常用类型转换函数：

- **int(x)**: 将x转换为整型。**int()**取整不是四舍五入，是“向零取整”，是真正的“截尾取整”。与**ceil()**函数向上取整、**floor()**函数向下取整不同。
- **float(x)**: 将x为转换浮点型。
- **complex(x)**: 将x转换为复数，其中复数的实部为x和虚部为0。
- **complex(x,y)**: 将x和y转换成一个复数，其中实部为x和虚部y。

- `>>> float(2+3.0)`
- `5.0`
- `>>> int(2+3.0)`
- `5`
- `>>> int(-2.546)`
- `-2`
- `>>> complex(3)`
- `(3+0j)`
- `>>> complex(3.5,5.0)`
- `(3.5+5j)`
- `>>> 5-int(5/3)*3`
- `2`
- `>>> m=1234`
- `>>> (m//10)%10`
- `3`

提示：
使用取整、求余等运算可以进行整除的判断，可以分离整数的各位数字。

- 可以用**int()**函数将**整数字符串**转换成对应的整数；
- 可以用**float()**函数将**浮点数字符串**转换成对应的浮点数；
- 可以用**complex()**函数将**复数字符串**转换成对应的复数；
- 可以用**str()**函数将数值型数据转换为字符串。
- **bool()**：将其他类型的数据转换为布尔类型。
- **list()**， **tuple()**， **set()**， **dict()**：将其他类型的数据分别转换为列表、元组、集合、字典类型。

- **>>> int("7")+9**
- **16**
- **>>> str(9)**
- **'9'**
- **>>> float(str(8.9))+7**
- **15.9**
- **>>> complex('3+4j')**
- **(3+4j)**
- **>>> str(3+4j)**
- **'(3+4j)'**

1.6.2 位运算

- 位运算就是直接对整数按二进制位进行操作，其运算符主要有：&，|，~，^，>>和<<。
- 在计算机内部常用补码来表示数：运算量是补码，运算结果也是补码，人机界面使用原码。

1. 按位与运算

- 按位与运算符是**&**。

- 例：

- **>>>-5&3**

- **3**

2. 按位或运算

- 按位或运算符是**|**。

- 例：

- **>>>-5|3**

- **-5**

3. 按位异或运算

- 按位异或运算符是 \wedge 。
- $0\wedge 0=0$, $0\wedge 1=1$, $1\wedge 0=1$, $1\wedge 1=0$
- 例:
- $>>>-5\wedge 3$
- -8

4. 按位取反运算

- 按位取反运算符是 \sim 。
- $\sim 0=1$, $\sim 1=0$
- 例:
- ~ 7
- -8

5. 左移运算

- 左移运算符是 \ll 。（左移1位等效为乘2）
- 例:
- $3 \ll 2$
- 12
- 将3左移2位，右边（最低位）补0。

6. 右移运算

- 右移运算符是 \gg 。（右移1位等效为除2取商）
- 最高位用符号位填充：
 - 移动对象为正数时，高位补0。
 - 移动对象为负数时，高位补1。
- 例：
 - $\gg \gg -3 \gg \gg 2$
 - -1

- 例:
- $x=2^{**}10$
- $y=pow(2,10)$
- $z=2<<9$
- $a=3/5$
- $b=3//5$
- $c=3\%5$
- $print(x,y,z)$
- $print(a,b,c)$
- 程序输出结果为:
- 1024 1024 1024
- 0.6 0 3

习 题

一、选择题

- 1. Python语言属于（ **C** ）。
- A. 机器语言 B. 汇编语言
- C. 高级语言 D. 科学计算语言
- 2. 下列选项中，不属于Python特点的是（ **B** ）。
- A. 面向对象 B. 运行效率高
- C. 可读性好 D. 开源
- 3. Python源程序文件的扩展名是（ **D** ）。
- A. .python B. .pyt C. .pyc D. .py

- 4. 以下叙述中正确的是（ **C** ）。
- A. Python 3.x与Python 2.x兼容
- B. Python语句只能以程序方式执行
- C. Python是解释型语言
- D. Python语言出现得晚，具有其他高级语言的一切优点
- 5. 下列选项中合法的用户标识符是（ **A** ）。
- A. `_7a_b` B. `break` C. `_a#b` D. `7ab`
- 6. 下列标识符中合法的是（ **B** ）。
- A. `i'm` B. `_` C. `3Q` D. `for`

- 7. Python不支持的数据类型有（ A ）。
- A. char B. int C. float D. list
- 8. 关于Python中的复数，下列说法错误的是（ B ）。
- A. 表示复数的语法形式是a+bj
- B. 实部和虚部都必须是浮点数
- C. 虚部必须加后缀j，且可大写也可小写
- D. 函数abs()可以求复数的模
- 9. 函数type(1+0xf*3.14)的返回结果是（ D ）。
- A. <class 'int'> B. <class 'long'>
- C. <class 'str'> D. <class 'float'>

- 10. 字符串s='a\nb\tc', 则len(s)的值是 (C)。
- A. 7 B. 6 C. 5 D. 4
- 11. Python语句print(0xA+0xB)的输出结果是 (D)。
- A. 0xA+0xB B. A+B
- C. 0xA0xB D. 21
- 12. 下列属于math库中的数学函数的是 (C)。
- A. time() B. round()
- C. sqrt() D. random()

- 13. Python表达式中，可以使用下面哪个符号来控制运算的优先顺序？（ **A** ）
 - A. 圆括号()
 - B. 方括号[]
 - C. 大括号{} D. 尖括号<>
- 14. 下列表达式中，值不是1的是（ **D** ）。
 - A. $4//3$ B. $15 \% 2$ C. 1^0 D. ~ 1
- 15. Python语句`print(r"\nGood")`的运行结果是（ **C** ）。
 - A. 新行和字符串Good B. `r"\nGood"`
 - C. `\nGood` D. 字符r、新行和字符串Good

- 16. 语句`eval('2+4/5')`执行后的输出结果是（**A**）。
- A. 2.8 B. 2 C. 2+4/5 D. '2+4/5'
- 17. 整型变量x中存放了一个两位数，要将这个两位数的个位数字和十位数字交换位置，例如，13变成31，正确的Python表达式是（**A**）。
- A. $(x \% 10) * 10 + x // 10$ B. $(x \% 10) // 10 + x // 10$
- C. $(x / 10) \% 10 + x // 10$ D. $(x \% 10) * 10 + x \% 10$

● 18. 与数学表达式 $\frac{cd}{2ab}$, 对应的Python表达

式中, 不正确的是 (C) 。

- A. $c*d/(2*a*b)$
- B. $c/2*d/a/b$
- C. $c*d/2*a*b$
- D. $c*d/2/a/b$

● 二、填空题

- 1. Python语句既可以采用交互式的(____)执行方式，又可以采用(____)执行方式。命令，程序
- 2. 在Python集成开发环境中，可使用快捷键(____)运行程序。F5
- 3. 使用math模块库中的函数时，必须要使用(____)语句导入该模块。import math
- 4. Python表达式1/2的值为(____)，
1//3+1//3+1//3的值为(____)，5%3的值为(____)。
0.5, 0, 2

- 5. Python表达式`0x66 & 0o66`的值为(____)。 **38**
- 6. 设`m`, `n`为整型, 则与`m%n`等价的表达式为(____)。 **`m-m//n*n`**
- 7. 计算 $2^{31}-1$ 的Python表达式是(____)。 **`2**31-1` 或 `(1<<31)-1`**

8. 数学表达式 $\frac{e^{|x-y|}}{3^x + \sqrt{6} \sin y}$ 的 Python 表达式为_____。

`math.exp(abs(x-y))/(pow(3, x)+math.sqrt(6)*math.sin(y))`

三、问答题

1. 写出下列数学式对应的 Python 表达式。

$$(1) \frac{\sin \alpha + \sin \beta}{\alpha + \beta}$$

$$(2) \frac{1}{3} \sqrt[3]{a^3 + b^3 + c^3}$$

2. 按要求写出 Python 表达式。

(1) 将整数 k 转换成实数。

(2) 求实数 x 的小数部分。

(3) 求正整数 m 的百位数字。

(4) 随机产生一个 8 位数，每位数字可以是 1 到 6 中的任意一个整数。

- 3. 下列语句的执行结果是False，分析为什么？
- `>>> from math import sqrt`
- `>>> print(sqrt(3)*sqrt(3)==3)`
- False

测试题

一、选择题

1. B 2. C 3. C 4. D 5. B

一、选择题

1. Python 是一种_____类型的编程语言。

A. 机器语言 B. 解释 C. 编译 D. 汇编语言

2. Python 语句 `print("世界,你好")` 的输出是_____。

A. ("世界,你好") B. "世界,你好" C. 世界,你好 D. 运行结果出错

3. Python 语言通过 缩进 来体现语句之间的逻辑关系。

A. {} B. () C. 缩进 D. 自动识别逻辑

4. Python 解释器在语法上不支持 编程方式。

A. 面向过程 B. 面向对象 C. 语句 D. 自然语言

5. 以下哪项不属于 Python 语言的特点?

A. 语法简洁 B. 依赖平台 C. 支持中文 D. 类库丰富

6. 以下关于 Python 版本的说法中,哪个是正确的?

- A. Python 3.x 是 Python 2.x 的扩充,语法层无明显改进
- B. Python 3.x 代码无法向下兼容 Python 2.x 的既有语法
- C. Python 2.x 和 Python 3.x 一样,依旧不断发展和完善
- D. 以上说法都正确

7. Python 的输入来源包括_____。

- A. 文件输入
- B. 控制台输入
- C. 网络输入
- D. 以上都是

8. 下面哪个不是 IPO 模式的一部分?

- A. input
- B. program
- C. process
- D. output

9. 采用 IDLE 进行交互式编程,其中“>>>”符号是_____。

- A. 运算操作符
- B. 程序控制符
- C. 命令提示符
- D. 文件输入符

6. B 7. D 8. B 9. C

10. 关于 Python 语言,哪个说法是不正确的?

A. Python 语言由 Guido van Rossum 设计并领导开发

B. Python 语言由 PSF 组织所有,这是一个商业组织

C. Python 语言提倡开放开源理念

D. Python 语言的使用不需要付费,不存在商业风险

10. B

二、填空题

1. 运行 Python 程序有两种方式：_____和_____。
2. IDLE 是一个轻量级 Python 语言开发环境，
可以支持_____和_____两种编程方式。
3. Python 3.0 解释器采用_____编码表达所有字符信息。
4. IPO 运算模式的输入来源有_____、_____和_____。
5. 静态语言采用_____方式执行，如 C 语言、Java 语言；
脚本语言采用_____方式执行，如 Python 语言、JavaScript 语言。

二、填空题

1. 交互式 文件式
2. 交互式 文件式
3. Unicode
4. 文件输入、网络输入、控制台输入、交互界面输出、
随机数据输入、内部参数输入（任选 3 个即可）
5. 编译 解释

6. IPO 程序编写方法包含_____、_____以及_____三部分。
7. Python 内置的集成开发工具是_____。
8. Python 提供了丰富的 API 和工具,以便程序员能够轻松使用 C、C++ 语言来编写扩充模块,体现了 Python 的_____性。
9. Python 解释器命令行采用比较简单方便的_____方式及时响应用户代码输出结果。
10. Python 最常用的编程方式是_____,它将 Python 程序写在一个或多个文件中,启动 Python 解释器批量执行文件中的代码。

6. 输入 处理 输出
7. IDLE
8. 易扩展性 或 胶水性
9. 交互
10. 文件式

