前言

大学计算机基础课程是计算机学院承担教学任务，在北航学院托管开设的信息大类基础课程之一，是全体理科大类及强基计划，拔尖计划相关专业的学生在大一时应当修习的一门基础课程。课程主要内容涵盖**计算思维与模型建立**，**Python 基础语法**，**基本数据结构**，**程序控制结构**，**算法基础**，**图表绘制**，**数据分析**，**GUI 设计**等内容。为适应大语言模型等先进人工智能应用成果的实际发展显示情况，在 2024 年春季课程改革后，课程还添加了**人工智能初步**的内容。课程内容丰富，基本涵盖了理科大类对口各专业在未来使用计算机作为工具，在实际科研或工作中解决问题的需求，或是提供了较好的入门指导工作，可以培养学习者的计算思维和运用计算机解决具体问题的能力。无论是否有计算机使用经验，是否有程序编写或脚本编写经历，在学习本课程后都将具有相应的能力。

具体的课程内容与任务方面。课程主要包括 1 次讨论课，10 次课内实验和 1 个大作业，所有题目都是助教同学利用寒假时间全新准备的，具体内容见下。

讨论课主要的内容是对图论（七桥问题等的理解，最短路径问题等），最优化问题（枚举法，匈牙利算法等）等特定，相对较为复杂的算法与模型的理解和应用，由每个人独立思考，而后在小组内讨论并向全班汇报解决思路与方案，课后每个人需要形成**独立的实验报告**。

课内实验是跟随理论课程授课内容，对理论课上所讲授的知识进行上机运用的过程环节。课内共有 10 次实验，课内实验难度不是很大，最最重要的是**独立完成**（包括提交到平台上的代码和实验报告）。

具体实验安排如下：

0. Python 编程环境的使用
1. Python 基本语法
2. 程序控制结构与列表
3. 问题的描述—基本数据结构（字符串，字典）

4. 问题的描述—自定义数据结构（栈，队列）
5. 基本算法设计与实现（枚举，递归）
6. 较复杂算法设计与实现（贪心法，动态规划）
7. Python 实现数据图形绘制
8. 插值，拟合计算
9. Pandas 数据分析与 GUI 设计

其中，前几次实验（实验 0-实验 2）的难点在于对 Python 最基础的语法的理解与运用，难在入门。这些内容并不是很难，甚至可以称得上非常简单。只是可能较为新鲜，此前没有接触过，才会带来一定的困扰。只要实际上手写过一两个程序，这部分内容就可以很轻松地掌握。

在基础数据结构和算法部分（实验 3-实验 6），对知识本身的理解与运用则会成为相对困难的内容，但本课程的目的在于培养运用与实用的思维，并不在于要求对这些算法有多么深刻的理解，也不会要求大家去 leetcode 等算法训练平台做大量的题来理解和巩固，当然，少量做一些题目，来提高和巩固自己也是欢迎的。事实上，贪心法和动态规划都是计算机科学发展到现在，称得上是最基本，最简单而常用的计算机算法了，所以其难度并不是那么大，还请大家多多尝试学习和理解。

最后的几次实验（实验 7-实验 9），我认为这些是应用性较强的实验。直白来说，在这部分实验中，你并不需要知道多少原理，你要做的只是拿来几个模板，照猫画虎，按部就班地搭建“积木”，完成任务就可以了，并没有多少难度。但鉴于 Python 是一门脚本语言，从笔者的实际经历来看，这些内容在未来的实际科研和应用中，反而是非常非常有用的。

大作业通常在学期中后段布置，是一组具有不同难度系数，面向不同实用场景的实际问题的，较为系统和工程化的解决方案，是对整个学期课程所学的一次总体性的应用。在完成大作业的过程中，往往需要跟随实验指导，自学一定量的知识。大作业题目具有一定的复杂性，所以也需要投入一定的时间去完成。作业最后有答辩环

节，

完成对课程内容和总体任务的介绍后，让我们回到编纂手册的目的上来。正如前面所说，本课程授课过程面临时间紧，任务重的困难，对于少部分没有计算机基础或使用经验的同学来说，学习这门课程的过程中可能会面临较大的困难。这往往体现在开展实验实操时进度不佳，或面对的困难较多而疲于应对。而苦于授课教师和助教精力有限，分身乏术，在提供实验指导方面可能会有覆盖不到的地方，对部分同学的学习带来一定的压力，甚至是阻力。同时，实际线下实验指导时我们通常以口述方式进行指导，难以在讲述过程中配合相应的代码，最好的情况也是配合同学当前所写的代码进行辅导。但这对描述问题和思路上时不利的。通常，配合代码示例的形式对问题和思路进行描述往往可以使读者（或者听众）对问题和思路形成更好的理解。因此，我选择编纂本手册，一方面，是可以让同学们在每次实际进行实验前，通过学习实际题目和代码的方式，能够能对这一部分的内容做大致了解，做到“手上有枪”；另一方面，我作为课程的修习者，以学生的视角编纂一份手册，也是让大家能有一个相较于教材来讲，方便速查思路的材料。教材上的内容非常全面，高屋建瓴，但对于速查还是带来一定的困难，与同学们学习的思路上也不一定完全一致，而同龄人之间总是更容易找到相同的思路，产生一些共鸣。

以此，希望本书的工作能为同学们修习本课程提供一些帮助。鉴于编者水平有限，加之时间紧张，书中难免有一些错漏，敬请读者批评指正。

编 者

2024 年 1 月 26 日

目录

本书中所有代码，在 Windows 11 23H2 x86_64 及 macOS 14.2.1 23C71 arm64 上以 Anaconda 23.11.0, Python 3.11 作为环境运行测试通过。具体第三方库见下：

1. scipy
2. numpy
3. matplotlib
4. pandas

第一章　Python 编辑及运行环境

在本课程中，我们选择安装 anaconda 这一 Python 发行版进行学习与开发。一方面是因为 anaconda 在安装时不仅仅能够安装一个 python 解释器，其也会同时安装 IPython kernel，Jupyter notebook，matplotlib，scipy，numpy 等多个常用的（也就是本课程中会用到的）第三方库。另一方面，anaconda 提供 Spyder 这一高集成度，便于初学者学习和调试的 IDE，可以提供给我们“开箱即用”的体验[1]。

如果你想要挑战自己，当然可以自行到 Python 项目官网[2]上自行下载和安装 Python 解释器，而后使用 pip 包管理工具安装我们所需的各类第三方库或者第三方组件（包括 Spyder，也可以通过 pip 来自动安装）。但在正式开始教程之前，需要强调，作为初学者，请不要同时安装纯净 Python 解释器和 anaconda，即不要让使用下载的 Python 安装包所安装的 python 解释器同 anaconda 在同一台电脑上并存，因为此时你的电脑里将同时有两个 python 运行环境，且这两个运行环境没有通过环境管理器进行管理，其激活次序，即运行 python 脚本文件时正在使用的 python 解释器，或是尝试使用 pip 安装第三方库所指向的环境不易明确。总之，作为课程的初学者，也

[1] anaconda 本身也提供很多别的功能，本章节后面会进行简要介绍

[2] https://python.org

是本手册后面的安排，我们将基于 anaconda（conda+python 解释器）进行介绍，包括环境配置，工具使用等，所有代码也将会在这个环境下进行运行。

本章节，我们将会介绍 Anaconda 的下载与安装过程，Spyder 的基本使用方法，并在最后简要介绍 conda 环境管理工具和 pip 包管理工具的使用。

1.1 Anaconda 的下载与安装

1.1.1 Anaconda 下载与安装

Anaconda 现已实现完善的公司化运营，可以访问其网站下载 Anaconda：

https://www.anaconda.com/download

但通常，出于下载速度的考量，我们选择使用校园网联合镜像站的方式下载 Anaconda，这不仅仅可以提供更快的安装包下载速度，同时其分发的安装包也会将其中的默认镜像站替换成自己的分发站点，从而提供更高的 pip 与 conda 下载与更新速度。提供镜像站的组织有很多，如清华大学 TUNA 协会，中国科学技术大学所等高校及高校学生组织所提供的下载源，此处以清华大学 TUNA 协会提供的下载源为例。

访问 https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/[3]，获取 anaconda 安装包列表。根据自己的操作系统版本和硬件架构，选择下载对应的安装包体（如 Windows-x86_64 对应 AMD，Intel 处理器的 Windows 设备，MacOSX-arm64 对应搭载 Apple Silicon M1、M2、M3 的设备）。在当前的时间节点，建议选择 Anaconda3-

[3]此为双栈站点，可以自动选择 IPv4 或 IPv6 解析，将 mirrors 替换为 mirrors6 则只解析 IPv6，替换为 mirrors4 则只解析 IPv4，在校园网环境下，使用 IPv6 下载的文件不计流量。

2023.09 开头的安装包。如图 1.1所示

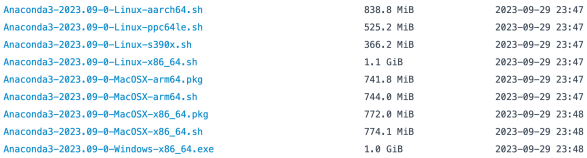


图 1.1: Anaconda3 安装包示例

对于不太熟悉命令行操作的同学来说，建议下载文件拓展名为 pkg 或 exe 的安装包，其可以提供图形化的安装界面和选项。具体安装过程在此不再赘述，但请注意，**不要**把 Anaconda 安装在含有中文的目录下，尤其注意你的 Windows 用户名是否带有中文。如果你不确定，可以直接将其安装在 C 盘外的其他盘符[4]，另外创建的文件夹中。

1.1.2 环境变量配置

配置环境变量的目的是令其他程序在运行 python 时能够直接找到 base 环境下的 python，而不需要手动指定，同时也方便我们直接打开命令行就可以直接使用 python 解释器运行和调试程序。anaconda 提供了方便快捷的命令行环境的环境变量配置方式，只需要切换到 conda.exe 的安装位置[5]，直接在命令行中运行 `conda init` 即可[6]。运行效果如图 1.2所示。

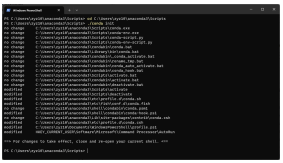
而后，重新打开命令行（Windows 10/11 中的终端[7]，macOS 中

[4]不要将 Anaconda 安装在 C:/Program Files 中

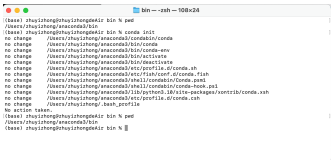
[5]windows 下的默认位置 C:\Users\用户名\anaconda3\Script，macOS 下的默认位置/Users/用户名/anaconda3/bin

[6]对于 powershell，需要使用`.\conda init`

[7]打开 powershell 时如果出现异常报错，则需要使用 Win+X 点选终端（管理员）打开 powershell，运行 Set-Execution Policy RemoteSigned 以使能脚本运行功能



Windows



macOS

图 1.2: 使用 conda init 自动配置环境变量的结果

的控制台），可以看到在命令前面出现了 (base) 字样，表明当前激活的 python 环境是 conda 管理的 base 环境。

至此，anaconda 的环境配置已经基本结束。对于 Windows 用户，如有需要，还可以在高级系统设置中手动将 anaconda 安装目录下的 bin 和 Script 目录添加到 Path 中，在此不再赘述。今后的课程学习中，用到的可能性很小。

1.2 Spyder 使用

Spyder 是今后我们在课程学习上将要主要使用的 IDE[8]，也是考试时要用的考试机上所具备的唯一的 Python 代码编辑工具[9]。Spyder 基于 IPython kernel 运行和调试 Python 代码，可以直观的将当前时刻各变量的值显示在窗口上，同时也具有命令行窗口，方便直接运行指令进行测试，或对变量值进行直接的修改。总之，对于初学者而言，Spyder 是非常简单，易用的 Python IDE。下面将会对 Spyder 的功能进行简单的介绍。

安装流程结束后，在开始菜单（macOS：启动台）中寻找到 Anaconda Navigater，点击运行。加载结束后，可以打开 Spyder，如图 1.3所示

[8]集成开发环境

[9]另一个是 Python 安装时会有的 IDLE，不具备联想提示和调试功能

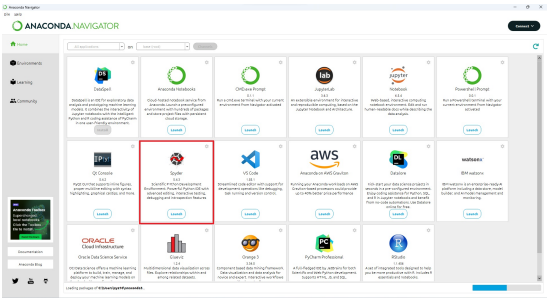
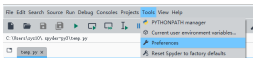
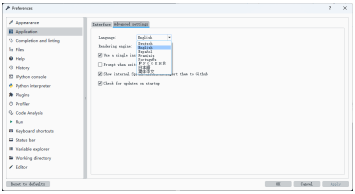


图 1.3: Anaconda Navigater 主界面

下面介绍 Spyder 主界面各面板的功能。但在开始之前，我们先来把 Spyder 的界面语言修改成中文。点选 Tools-Preferences 打开选项页面，而后在 Application-Advanced settings 中找到 Languages，选择简体中文即可。如下图所示。





下面介绍 Spyder 主界面各面板的功能与作用。设置中文并打开 Spyder 后，初始界面如图 1.4所示。

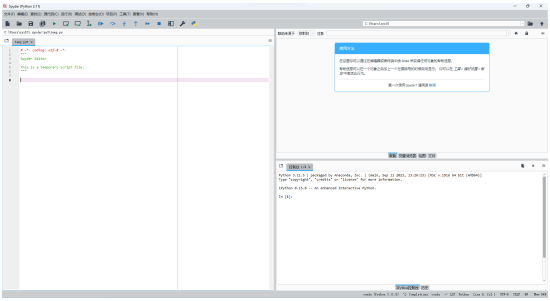


图 1.4: Spyder 主界面

左侧是代码编辑区域，可以在此修改和编辑 Python 程序的源代码。

可以看到，在源代码编辑框右侧，未到达编辑框边界处有一根纵向的线，这个位置代表 Python 源代码编写规范 PEP 8 中的要求，每行代码不超过 79 个字符。有兴趣的同学可以自行查找资料，深入了解 PEP 代码规范。简单来说，PEP 8 是对代码可读性提供的规范，用来保证程序员书写出的代码具有可读性和可维护性。还请大家尽量遵循代码规范，包括后面介绍基本语法时会提到的变量命名规范和注释规范等，方便各位同学后面再读懂自己的代码，也方便授课教师和我:-)。

下面让我们聚焦于界面上方的一排按钮。如图 1.5所示。我们使用的大部分操作都可以在这里找到。从左到右，按钮的功能依次为:

1. **新建文件**: 在当前目录新建一个 python 源文件，默认以「未命名 x.py」为命名，保存时可修改文件名
2. **打开文件**: 选择一个 Python 源文件并打开到工作区
3. **保存文件**: 保存当前打开的文件

4. **保存所有打开的文件**: 将所有打开的文件进行保存操作

5. **运行**: 从头开始运行当前的文件

6. **运行单元格**:

新建文件、打开文件、保存文件、保存所有打开的文件、运行（从头开始运行当前打开的文件）、运行单元格（需创建 cell，多用于调试）、运行当前单元格并跳到下一单元格、调试文件（开始调试）、运行当前行（在开始调试的情况下，运行暂停是点按生效）、步入当前函数或方法、执行完当前函数到返回、继续运行到下一个断点、停止调试。最后三个按钮分别对应用户界面的操作，打开设置和 Python Path 管理器，除打开设置外，在本课程中并不常用。

图 1.5: Spyder 快捷按钮

右侧上半部分有 Spyder 提示，变量浏览器，绘图和文件四个选项卡。Spyder 提示可以显示库函数的帮助提示信息，具体使用方法在后面详细的语法部分将会进行更加详细的介绍。变量浏览器可以显示当前内存中所存放的变量，包括类型、大小[10]和值。不同版本的 Spyder 在处理变量浏览器选项卡时可能有所不同，但总体上大同小异。我们只需要了解，双击值这一列对应的格子即可看到和修改变量的值，便足够了。

绘图和文件选项卡在前期使用的较少。绘图选项卡可以保存每一次使用 matplotlib 等库绘制出来的图片，形成一个历史记录，方便查阅和调用。具体使用方法将会在后面绘图的部分再行介绍。文件选项卡则会显示当前打开的 Python 文件所处的目录下面的文件，

[10]可以理解为规模，如 3x2 的数组会显示为 (3,2)；普通的整形、浮点会显示为 1

在后面进行文件操作或处理大作业时，观察和操作当前目录下有哪些文件[11]非常便利。

右侧下半部分提供了一个命令行窗口，在此可以直接与 IPython 内核交互。Spyder 运行 Python 文件是使用 IPython 内核进行解释运行的，如果真的直接使用 Python.exe 对文件进行解释运行，在每一次运行完程序之后，Python.exe 不会驻留在后台，相关内存会被释放，自然也不能看到内存中存储的变量的情况。而原始的 Python 解释器也不提供调试功能。

书归正题，在这个命令行窗口中，我们可以直接创建变量，运行命令，抑或是在调试过程中修改变量的值，都是可行的。具体的使用方法，在下一章将会结合具体代码进行概述。现在，你只需要了解到这个界面上各个块的功能即可。

再行补充一个知识，在命令行窗口右侧上部，可以看到一个三横线选项卡，将其打开，里面有对 IPython 内核的交互。如图所示。可以新建，中断或重启 IPython 内核。这对于程序陷入死循环或卡死时是十分有效的，可以在这里重启或直接开启一个新内核运行程序。当然，也可以直接关闭控制台选项卡，这也能起到关掉对应 IPython 内核的作用，当没有 IPython 内核在运行时，Spyder 会自动开启一个新的 IPython 内核[12]。

1.3 pip 和 conda 的使用 *

在这一部分，我们将要介绍 `pip` 和 `conda` 的使用方法。这两个工具都可以作为环境下的包管理工具，conda 同时还可以作为虚拟环境的管理工具。在课程中，我们使用这两个工具都不会特别深

[11]包括其他 Python 脚本或者数据文件

[12]IPython 中的 I 代表 Interact，即交互。我一直将其称之为内核，其实其本质上就是一个进程，使用其运行 Python 脚本，本质上就是要求这个进程读取这个文件并进行解释运行。如果想要深入了解 IPython，可以访问 ipython.org

入，因此这里只介绍一些基础的使用方法。

1.3.1 pip 包管理工具

Python 是一个具有系统性的包管理和共享体系的体系，pip 是 python 标准安装后会带有的包管理工具。如果想要下载或使用一个第三方库，如 matplotlib，scipy 等，可以先访问 https://pypi.org 查看包名，文档和快速入门（如有）。也可以在这个平台上搜索第三方库。当搜索到一个第三方库后，可以使用下列格式的命令安装：

```
pip install <package name> <-U> <-i> <mirror>
```

`package name` 参数是必选参数，是包名。`<-i mirror>` 参数是指定 `pip` 从哪个镜像站下载软件包，`mirror` 参数应当填写镜像站地址。比如下面这个例子

```
pip install scipy -U -i \
https://pypi.tuna.tsinghua.edu.cn/simple
```

其指定使用清华大学 TUNA 协会的镜像[13]，安装（以及升级）`scipy` 库。要补充说明的是，上面这个代码框内的“\”的目的只是为了换行

[13]另可参阅 TUNA 协会提供的 pypi 镜像使用帮助：
https://mirrors.tuna.tsinghua.edu.cn/help/pypi/

使用，无论是 bash 还是 python 中，“\”符号的含义均是将一行代码拆成两行书写，解释器会将第二行，甚至更多行（如果你连续使用“\”的话）的内容按照顺序拼接到一起来理解和执行。

如果希望指定安装的库的版本，可以参考下面的示例：

```
1    pip install scipy==1.11.3
```

1.3.2 conda 环境管理工具

anaconda 是 conda 管理工具的一个发行版，除此之外，还有 miniconda 等更加精简的发行版本，其更加适合于诸如 Raspberry Pi 等资源相对有限的计算平台，以及运行在无头模式下的服务器上使用。本小节将集中于 conda 工具的使用的介绍。

conda 创建和管理虚拟环境

在安装完毕之后，anaconda 会自动创建一个 base 环境，在本课程中，我们大部分情况下都可以在 base 环境内进行操作，这是因为我们在课内所使用的第三方库的数量不是很多，集中在一个环境内进行操作不会带来什么问题。下面介绍一个可能的情况，在完成大作业时，课程要求我们建立一个 GUI，假设你需要使用 PyQt6（或者新版本的 PyQt5 ）来建立你的 GUI，这是一个非常好用的 GUI 库，而 Spyder 运行依赖于特定版本的 PyQt5，二者会产生冲突。在这种情况下，创建一个新的虚拟环境便是必要的了。

下面介绍创建和管理虚拟环境的办法。conda 支持创建运行特定版本 python 的虚拟环境，如下所示：

```
1    conda create -n <name> python=<version>
```

`<name>` 参数是虚拟环境的名称，支持大小写，且对大小写敏感。`<version>` 是指定 python 解释器版本，支持到小版本如 3.10.3。可以观察下面的示例

```
conda create -n MyPythonSpace python=3.10.3
```

这里我们指定创建一个名为 MyPythonSpace 的虚拟环境，解释器版本为 3.10.3 。运行时会提示是否继续，按下回车即为继续[14]，运行输出如下：

```
    Channels:
 - defaults
Platform: osx-arm64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: \
    /Users/<username>/anaconda3/envs/MyPythonSpace

  added / updated specs:
    - python=3.10.3


The following packages will be downloaded:

## 此部分为Python解释器包信息，略去 ##

The following NEW packages will be INSTALLED:

## 此部分忽略 ##


```

[14]形如 [y]/n 这样的确认提示，在不输入任何字符的情况下直接回车，则默认值为 y，若为 y/[n]，则默认值为 n

```
Proceed ([y]/n)? y


Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate MyPythonSpace
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

而后，运行 `conda activate MyPythonSpace`，即可激活 `MyPythonSpace` 虚拟环境，如下：

```
(base) bin % conda activate MyPythonSpace
(MyPythonSpace) bin % python --version
Python 3.10.3
```

需要返回 `base` 环境时，只需要运行 `conda deactivate` 即可。

下面介绍使用 conda 对已有环境进行管理。首先，如果需要列出当前存在的所有由 conda 管理的环境，可以运行 `conda env list`。其会列出当前存在的所有环境，以及其位置，如下列所示：

```
# conda environments
#
base             /Users/username/anaconda3
MyPythonSpace    /Users/username/anaconda3/envs/MyPythonSpace
```

如需删除某环境，可以考虑运行:

```
conda env remove --name <env>
```

如删除 MyPythonSpace 这一环境:

```
conda env remove --name MyPythonSpace
```

这一命令可以删除这个 Python 环境下的所有软件包以及这个环境本身。

使用 conda 进行包管理 *

下面介绍使用 conda 安装第三方库的一个示例。在大部分情况下，我们使用 pip 安装第三方库已经足够了，但在这里还是介绍一下。同时，要注意，不要尝试使用 pip 管理 conda 安装的第三方库，反之亦然。

使用 conda 安装 PyTorch 的方法如下:

```
    # From https://pytorch.org
    conda install pytorch::pytorch torchvision \
            torchaudio -c pytorch
```

运行之后，conda 会在当前激活的虚拟环境下安装 pytorch, torchvision, torchaudio 三个第三方库。在这里，install 表明要求 conda 执行安装包的操作，-c pytorch 指定在 pytorch 这一发行频道上查找软件包。

如果需要卸载软件包，可以运行 conda remove --name <env> <package> 进行卸载软件包的操作。如，卸载 MyPythonSpace 中的 numpy 软件包:

```
conda remove --name MyPythonSpace numpy
```

下面介绍使用 conda 对环境中的第三方库进行升级的方法。但需要注意的是，如果你的环境当前可以满足需求，不需要用到新版

本的特性，当前运行的版本也没有需要通过更新来修复的安全漏洞，那么就不要更新某个第三方库。同样的，也不必对整个环境内的第三方库进行统一的更新，但在此仍然介绍这种方法。

```
conda update --all
```

这样，conda 便会更新当前激活的环境下的所有库。相似的，更新某个库的方法为：

```
conda update <package>
```

习题 1

- conda 的作用是什么，pip 的作用是什么，如何理解 Anaconda 这一整体?
- 创建名为 `PythonBais` 的环境，指定解释器版本为 3.10.2，并使用 pip 包管理工具在其中安装版本为 1.0.0 的 numpy；而后，删除此环境
- * 卸载 Anaconda 并安装 miniconda，与 Anaconda 进行比较。

第二章　Python 基本语法与程序控制结构

在讨论基本语法前，先让我们来简单了解一下本课程中，我们所主要涉及到的程序的基本结构：输入-处理-输出结构，即 IPO 结构。顾名思义，IPO 结构主要有三部分构成，先获取输入，而后对输入的数据进行处理，得到结果，按照要求输出。看起来非常的简单，不错，我们课程中所涉及到的绝大部分任务都可以用这样的控制结构来实现。

这样便引出了我们设计程序所需要了解的基本语法和控制结构。获取输入时，如何指定程序获取来自键盘的输入，亦或者是读取磁盘中的文件，进阶来看，获取其他进程发来的通信也可以看作一种输入[1]。获取输入后，如何将数据按照怎样的组织形式存入内存，而后又如何处理这些数据，这都涉及到基本语法中的基本数据类型（以及数据结构），基本运算和程序控制结构。最后，如何获得我们想要的输出，怎样控制输出等。解决了以上问题，便实现了一个 IPO 程序框架下的程序实例。

因此，本章将首先介绍 Python 中的变量、输入与输出，然后介绍基本数据类型及其操作，之后介绍程序控制结构、函数的声明与调用，最后介绍最基本的数据结构。在本章的末尾，将会有一段对

[1]但本课程不涉及多线程

Python 中类的简要介绍，这部分内容并不是课内所要求掌握的，留给有兴趣的同学尝试理解。

2.1 变量、输入与输出

2.1.1 变量

变量，以及常量[2]，是在编程中用于存储值以及对象的基本单元。因此，我们在这里首先介绍变量的概念。Python 是一门弱类型语言，具有自动内存管理的特点。这意味着，在 Python 中声明一个变量时，不必指定其变量类型，通俗来说，不用指定一个变量里要存储的值是什么类型的。直接对一个变量赋值，如果这个变量已经被声明了，那么其会改变变量的值，否则，解释器会为这个变量划分内存空间，创建这个变量，而后向这个变量赋值。如下所示：

```
1 a = 10
2 # 创建变量a，类型为整形(int)，并赋值为10
3 b = 3.14
4 # 创建变量b，类型为浮点型(float)，并赋值3.14
5 a = b
6 # 将b的值赋给a，并改变a的类型为b的类型
7 # 现在a是浮点型(float)
```

可以看到，在 Python 中，一个变量的类型在程序执行过程中是可变的。因此，在实际编程中，要特别注意这一点。否则会给 debug 带来困扰。

下面让我们来了解一下哪些变量名是合法的。一个合法的变量名可以由数字、字母、下划线以及汉字混合组成，但变量名的开头

[2]Python 中并没有声明常量的关键字，也就是没有办法让一个变量的值保持不变。但从规范的角度来说，当变量名全部为大写字母时，则认为这个变量是常量，其值不应当被改变，但解释器在执行程序的过程中，并不会强制要求这一点

不能是数字。同时，Python 对变量名中的英文大小写是敏感的，比如 `Movie` 和 `movie` 不会被认为是同一个变量。下划线可以作为变量名的开头，但这通常被用于声明库内变量来使用，一般书写程序时不必使用下划线作为开头。

基本数据类型

我们首先介绍 Python 中的整形 (int)，浮点型 (float) 和字符串 (str)。这三种常用内建数据类型最为基本，也与自然语言吻合的最好，因此首先介绍。

整形只能用于存放整数，其构造函数为 `int`。当然，在 Python 中，一般情况下我们不需要使用构造

变量命名规范

既然讲到了变量的声明，那么我们不妨在这里简单聊一下关于变量名的代码规范。对变量名的最基本要求很简单——见文知义，即自己或者他人看到这个变量名后，能够知道这个变量的值所代表的含义，比如 `age` 或者 `nianling` 代表年龄[3]，`movieList` 可以代表电影列表等等。尽量避免大量使用一个字母，或者一两个字母的简单变量名[4]。否则可能对后期代码维护，以及老师阅卷带来不必要的麻烦，甚至造成看不到得分点。

下面我们简单介绍一下 Python 中变量名的规范取法。通常，在 Python 中使用小驼峰命名法或蛇形命名法。下面我们来逐一介绍。

驼峰命名法是指，对于一个变量名，组成其含义的各个单词第一个首字母大写作为分割，比如：

```
1 RawData # 原始数据
```

[3]但尽量不要使用拼音

[4]当然，对于一些约定俗成的变量，比如 `i`、`j`、`k` 作为迭代器迭代次数，`f` 作为打开文件的句柄这类名称是可以被接受的。

```
2 ArticleList # 文章列表
```

驼峰法有一种变体，被称为小驼峰，即第一个单词的首字母不大写，后面出现的单词的首字母大写，如：

```
1 rawData = NULL
2 articleList = NULL
```

下面介绍蛇形命名法，蛇形命名法是将各个单词用下划线分割开来。在通常的蛇形命名法中，不需要对单词的首字母做大小写处理。下面是两个示例：

```
1 raw_data
2 article_list
```

而如果变量名以一个下划线起始，这通常代表这个变量是类的内部变量，不应该在类外访问，即这个变量是被保护的[5]。如下所示[6]：

```
1 _charsize_cache
2 _bbox_patch
```

也可以将驼峰法和蛇形命名法结合起来一同使用，在此不在赘述。

在本课程中，具体选用哪种变量的命名规范取决于你自己，甚至是否遵循 PEP 8 都并不是那么重要的。遵守变里命名规范，以及代码规范最终的目的是让自己的代码保持一定的可读性，一方面方便自己回看复习，另一方面方便我们进行实验指导以及阅卷。

[5]这并不意味着解释器会保护这个变量，但在类外随意修改这类变量的值可能带来程序运行异常

[6]摘自 matplotlib 源码