

# 如何写好一个数据处理程序

## (Python 入门简介)

李洋 (Yang Li)<sup>1</sup>

<sup>1</sup>Department of Physics  
Tsinghua University

May. 2021

# 前置声明

## 内容声明

以下所有内容都是笔者参考自身编程经验总结得到，或许某些论点会出现认知或表述上的错误，还请大家批判性地接受。

## 版权声明

本 Slides 内容发布在 GitHub 上，并受 **GPL-3.0** 开源许可证保护。您可以自由的下载、复用、修改其中内容，并进一步作为它用。他人修改再发布版本与本作者无关。

<https://github.com/kYangLi/PublicPresentation/blob/master/AcaCode/latex-src/main.pdf>

# 目录

- ① 为什么需要编程?
- ② 学习编程前应具备的认识
- ③ 学习 Python 前的准备工作
- ④ Python 基础语法
- ⑤ 使用 Python 进行数据处理
- ⑥ 总结

# 目录

- ① 为什么需要编程?
- ② 学习编程前应具备的认识
- ③ 学习 Python 前的准备工作
- ④ Python 基础语法
- ⑤ 使用 Python 进行数据处理
- ⑥ 总结

# 编程的本质

## 编程的定义

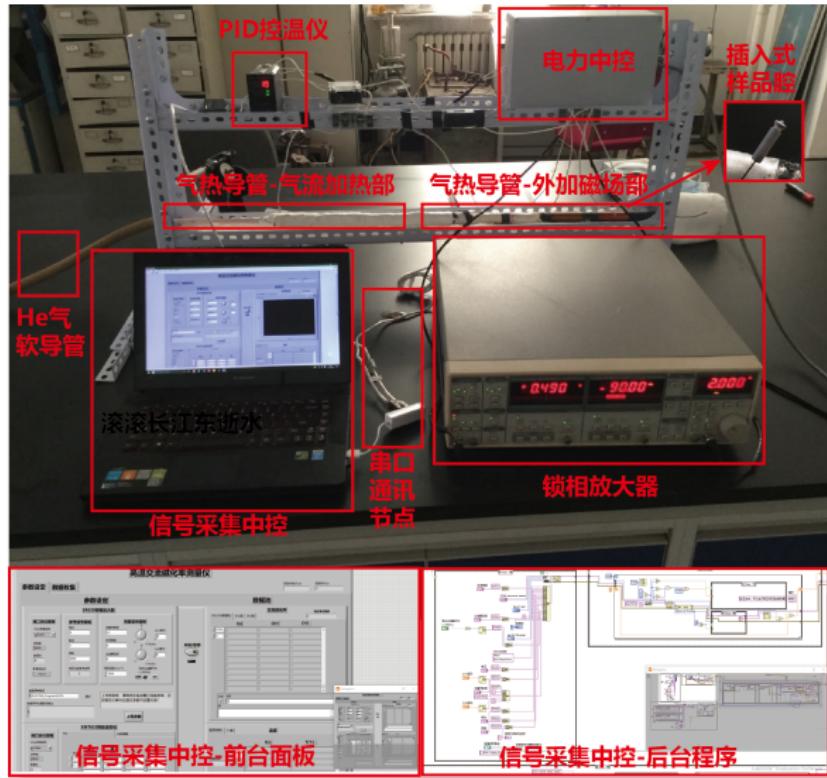
所谓编程，其实就是使用计算机语言，实现人与计算机间有效交流的手段，是让计算机理解并执行“人想进行的复杂操作”的媒介。计算机语言之所以称之为“语言”，是因为他确实有一般语言的特性，即，他是一种约定俗成的，可以被两个独立个体分别无信息损失地理解的记号。在有些时候，编程语言也成为了人与人之间交流算法和感情的基本语言。

## 高级计算机语言的特点

- 语法规则非常严密，有时写错一个符号就会导致整体代码无法运行。
- 结构十分规整，一段优秀的代码天然会具有层次分明的结构美感。
- 起源于英语，并与英语表达有相似之处。一段优秀的代码会使阅读代码者产生阅读英文概述的错觉。
- 种类繁多，特性各异。

真正控制了世界的，其实是程序员们。

# 编程应用 1: 实验仪器的数据采集 (串口通讯)



各种各样的传感器和信号采集设备规范化了实验的进行。信号采集和数据处理完全需要依靠相关语言，如 LabVIEW 中的 G 语言，编程实现。

# 编程应用 2: 实验数据峰值拟合

## Fitting the Phonons

```
def gaussian(x,*params):
    y = params[0] * np.exp(-((x-params[1])/(np.sqrt(2)*params[2]))**2) / \
        np.sqrt(2*np.pi*params[2]*params[2])
    return y

def lorentz(x,*params):
    y = (params[0] / (1+((x-params[1])/params[2])**2)) / (np.pi*params[2])
    return y

def alorentz(x,*params):
    y = params[0] * \
        (params[2]*x / ((x**2-params[1]**2)**2 + (2*params[2]*x)**2)) \
        * np.heaviside(x, 0)
    return y

def final_func(x,*params):
    y = gaussian(x, *params[1:3]) + \
        gaussian(x, *params[3:6]) + \
        alorentz(x, *params[6:9]) + \
        gaussian(x, *params[9:12])
    return y

params, params_covariance = optimize.curve_fit(final_func, fitx, fity,
                                                p0=guess,
                                                bounds=(lowbound, highbound))
```

实验的后续  
数据处理，  
如寻找实验  
信号中的峰  
值和模式匹  
配时一般也  
需要编程实  
现。

# 编程应用 3: 流程化 VASP 第一原理计算

The screenshot shows a GitHub repository page for 'KYangLi / vasprun'. The repository has 1 branch and 1 tag. The code tab is selected. A terminal window is displayed on the right side of the repository page, showing a VASP calculation process. The terminal output includes:

```
(base) liyang1@w001:Graphene$ ls
1-RELAX.pbs.submit    INCAR.DOS      POSCAR
2-SSC.vasp.inp         INCAR.RELAX   POSCAR.RELAXED
3-BAND                INCAR.SSC     POTCAR
4-DOS                 KILLJOB.sh   RESULT
CLEAN.sh              KPOINTS.BAND  vasp_submit.pbs.sh
cores-list            KPOINTS.DOS   vr.allpara.json
Graphene.o3800        KPOINTS.RELAX vr.input.json
INCAR.BAND            KPOINTS.SSC

(base) liyang1@w001:Graphene$ vasprun
+-----+
====| Welcome to vasprun | ====
+-----+
Press <Enter> to start the vasprun process... [ ]
```

将计算任务写成流程化脚本，可以极大的缩短在数据文件处理上的时间。  
将重复劳动都交给计算机来做，让我们有精力去思考更物理的问题。

# 编程应用 4: 进行 TB 模型计算

The screenshot shows the PythTB website. The left sidebar has links for About, Installation, Examples, Formalism, Usage, and Resources, with a 'Quick search' input field and a 'Go' button. The main content area has a title 'Python Tight Binding (PythTB)' and a brief description of the software. Below is a navigation menu with links to About, Installation, Examples, Formalism, Usage, and Resources. The 'Quick installation' section contains instructions to type 'pip install pythtb --upgrade' in the terminal. The 'Quick example' section provides a simple example for defining a graphene tight-binding model.

PythTB

- About
- Installation
- Examples
- Formalism
- Usage
- Resources

Quick search

Go

## Python Tight Binding (PythTB)

PythTB is a software package providing a Python implementation of the tight-binding approximation. It can be used to construct and solve tight-binding models of the electronic structure of systems of arbitrary dimensionality (crystals, slabs, ribbons, clusters, etc.), and is rich with features for computing Berry phases and related properties.

- About
- Installation
- Examples
- Formalism
- Usage
- Resources

### Quick installation

Type in terminal:

```
pip install pythtb --upgrade
```

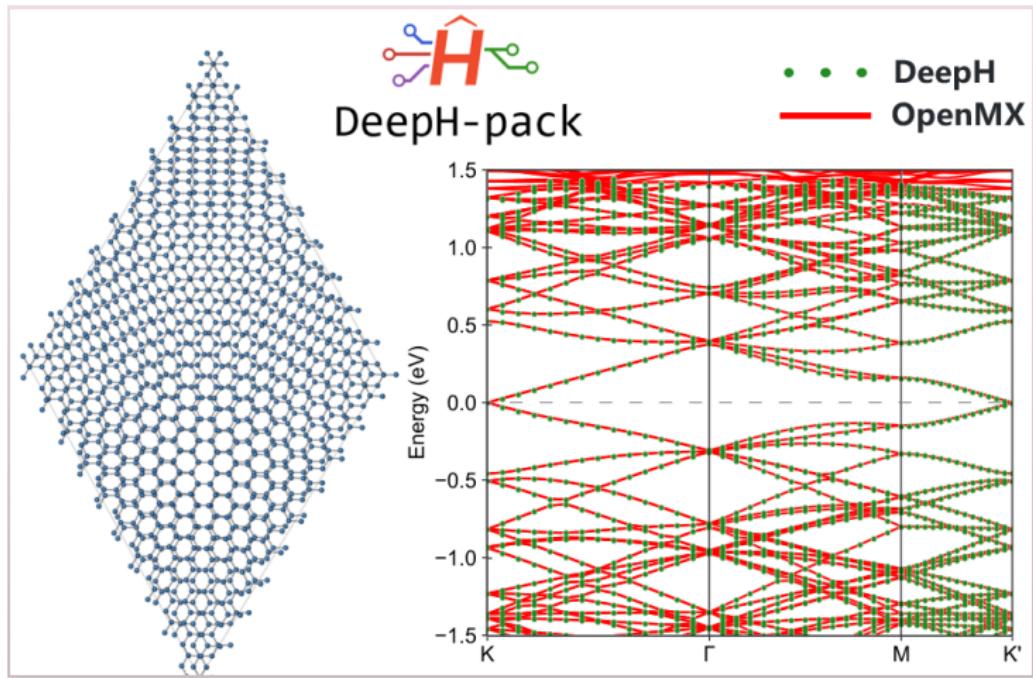
or if you need more assistance follow these [instructions](#).

### Quick example

This is a simple example showing how to define graphene tight-binding model with first neighbour hopping only. Below is the source code and plot of the resulting band structure. Here you can find [more examples](#).

使用别人造好的“轮子”，站在前人的肩膀上，才能更快更好地取得下一步进展。科学的进步固然需要瞬间的灵感，但更需要持久的积累。而积累则是需要媒介的。

# 编程应用 5: 机器学习与大数据



自从计算机算力大幅提升后, 机器学习和大数据无疑成为了计算机领域最为火爆的项目.

# 数据处理的基本内涵

一般来说，数据处理无非就是依靠计算机完成以下三步：**获取数据，处理数据，输出数据。**

- 数据源可能有多种来源，例如，存放在硬盘中，由实验仪器串口通讯获得，位于计算机内存中，由人手动输入，等。下面我们将讨论大家最容易接触到的一类情况：以文件的形式存放在硬盘中的数据处理。
- 如何处理数据一般是跟具体的科研过程有关，一般要做好这一步首先要非常明白：自己有什么、自己要什么。而后心思细密地函数式地逐步完成中间数据处理工作。
- 数据输出的方向也是多种多样的，例如可以直接打印在屏幕上，存储在硬盘文件中，存储在内存中供下一级程序调用，通过 UDP 或 TCP 协议通过网卡传送到另一台机器或 CPU 上进行协同处理，或者输入回实验仪器使数据转换为具体的实验操作。下面我们也只讨论存储进硬盘这一种情况。

从硬盘某个文件读取数据，处理他们，并保存进新的文件。

# 目录

- ① 为什么需要编程?
- ② 学习编程前应具备的认识
- ③ 学习 Python 前的准备工作
- ④ Python 基础语法
- ⑤ 使用 Python 进行数据处理
- ⑥ 总结

# 如何选择第一门语言

根据个人不同的需求，可以选择不同的路径。

## 想系统地学习计算机编程

如果真的想非常系统的学习计算机编程知识，在笔者看来，应该首先学习计算机硬件知识。了解计算机内部基本的组件和简要工作原理。对计算机硬件系统有了大概了解之后，学习的**第一门高级语言**应该是离计算机底层较近的语言，如 C/C++。而后在学会了函数式编程，地址，指针，栈，传参，等概念之后，可以开始着手学习用于快速开发且全平台适配性好的语言，如 Java 或 Python.

## 想快速实现某个大规模数值计算或模拟程序（如蒙卡）

一般需要选择一门静态语言进行学习。如 C/C++ 等。

## 想快速做一下基于硬盘文件的数据处理

需要学习一个能快速开发并且便于和文件管理系统交互的语言，如 Python.

# 一个程序的基本组分

## 个人对程序组分的解读

计算机程序其实就是一个复杂动作的实现流程，这个流程一般由两部分组成，**变量和算法**。

- 变量是存储在计算机内存中的不同数据的代指，每个变量都有一个变量名指定。变量本身可以有不同的内部结构，如浮点数，整数，字符串，数组，结构体等。
- 而算法则是操作这些变量相互作用的动作，当然算法也可以更广义的理解为高度总结泛化后的程序流程。算法靠顺序排列的流程控制语句和运算符实现。
- 一般来说，一个程序所实现的复杂流程可以被拆分成不同子流程。我们将这些子流程单独写出，形成所谓的“函数”，而后在主程序中通过函数调用将不同模块组合在一起，完成指定动作。因此也有人说，程序的基本组分就是一个个“**函数**”。

# 学会书写一个‘好’程序

## 编程时应尽量遵循的规则

- 决定开始编写程序之前，强烈建议您先大概浏览一下已有的编程规范，例如“[谷歌编程规范](#)”，以防止学习过程中不良习惯的养成。这些编程规范是在总结了海量的程序 Bug 出现情景后提出的。遵循这些规范在让你的代码变得优雅易懂的同时，也会极大地降低错误率。
- 一定要习惯并乐于书写[注释](#)。注释应该至少出现在那些你认为比较难懂不能一眼看出代码含义的地方。别怕麻烦，因为有了这些注释，你的代码才能发扬广大。俗话说的好，己所不欲，勿施于人，即使你不认同这一点，至少为了三个月后的自己着想一下。而且，合理的注释也能帮助您理清思路。
- [变量命名](#)非常有讲究，如果感觉使用 `a b c i j k m n x y` 式的变量命名非常自然，那大概率您还没有经历过较大规模的程序开发。
- 如何界定一个[函数](#)所管辖的范围是整个程序优雅性的精髓所在。您的程序可不可以称为“好”程序，很大一部分就取决于您如何界定和链接每个函数的功能。
- [合理的换行](#)也是需要注意的。没有人期望看到长达几百个字符的一行。一般来说，一行代码包括行首缩进不能超过 80 个字符。

# '好' 程序内涵

## 一句话定义什么是好程序

一个好的程序，就是会让你在阅读这个程序时会由衷地感谢作者的贴心，会不住地赞叹作者是真的在为读者着想。

程序写出来就是给人和机器看的。机器比较死板，如果它看不懂，就会报错，此时你就得到了一个无法正常运行的程序。可以说，让机器能看懂是一个程序的下限；而让其他人能看懂，则是一个优秀程序的下限。

# 如何书写注释

- 注释不应该是复述某一步操作的表层意义是什么，因为对于任何懂得这门语言的人，都可以自发解读出来，我们更应该用注释标明自己做此**操作的动机**。
- 有人喜欢边写代码边注释，有人喜欢写完代码后慢慢注释。这都没有关系，重要的是注释应当尽量在“你还记得此处详细思路，并且不打断后续编程灵感”的条件下进行。例如在你感觉需要放松一下或者要长舒一口气的时候，可以写一写注释；需要集中精神的时候，可以用一个注释符标明这里应该有注释，但现在没法停下来写他。
- 要善用 **TODO注释**。**TODO** 标明了此处还需要继续开发，但作者真的该回家陪老婆孩子了。在你开始新的一天，或从假期中返回时，搜索一下 **TODO** 的位置就可以快速进入开发状态。
- 刚开始写注释，你可能会非常不习惯。有时可能会絮絮叨叨，有时又会过于简略。但没有关系，多练几次，慢慢习惯这种行为后，你的语言会逐渐变得干练。你将逐步学会如何清晰精准恰当地表述自己的意思。

# 如何命名变量

- 变量名就是以下划线或字母开头的，一堆字母数字下划线的组合。如何在近乎无穷的组合中取到一个**最优雅得体**的名字，是非常值得学习的。
- 变量命名是一个非常需要讲究的东西。应该时刻牢记，我们取的每一个名字在未来会至少出现在某个师弟师妹的工作当中。任何一个名字的形成，经过了一段时间的混沌效应后，都可能在未来诱导巨变。变量名应该**简洁明了地表达其本意**。如果做不到简洁，至少应该做到明了。毕竟相较于代码的可读性，程序文件是 25K 还是 30K 是不太重要的。同时，30 个字符的变量名基本就是人类阅读舒适的极限。
- 如果你实在是喜欢 `a = b` 这种表述，那至少也应该在这一行**加入注释**，解释一下 `a` `b` 分别代表什么。
- 如果你在写一个科学计算软件，并已写好了相关的公式算法的 Notes。这时候，在程序中不遵循编程规范中的约束，并**严格复用 Notes 中的符号**(如 `S`, `x`, `H_ij`, `K_q`) 是可取的，前提是你可以确保看到这个程序的人同时一定可以看到你写的 Notes。在程序一开始声明“该程序中所有符号均与 xx 网址上的 Notes 相同”将是十分友善的做法。

# 如何定义函数

- 函数的定义是非常非常值得注意的。一个好的函数的定义是：该函数不能被继续拆分成若干子函数了。当然对于初学者来说，要做到这一点非常困难。因为对于我们这些不太熟悉编程的人来说，光是想清楚该如何从程序层面上实现一整套操作就十分不容易了，做到把问题拆解更是难上加难。大多数时候都是走一步看一步。但随着编程经历的增加，我们会越来越体会到前人如此要求的良苦用心。一个不良好定义的函数会让我们脑子一团浆糊，并令人很容易就陷入到某些细枝末节的实现的愁苦情绪中，失去对整个程序的把控。
- 同变量名一样，函数名也应该言简意赅地描述该函数的作用，如 `get_magnetic_J()` 就是一个不错的名字，他告诉读者，这个函数的作用就是得到磁性交换相互作用系数  $J$ 。而 `gmj()` 就相对拉垮一些。
- 每个函数都应该有详细描述该函数功能的注释。注释应至少包括对该函数实现的功能的描述，最好有对输入输出项的简要解释。注意注释不应该以函数为第一人称，而应该在第三人称视角描述函数作用。
- 充分细化的函数不但不会让程序复杂，恰恰相反，他会让程序看起来分外清爽。

# 定义函数示例

## 良好定义的函数们

```
1 def 举起(身体部位):  
2     pass #TODO  
3  
4 def 展开(身体部位):  
5     pass #TODO  
6  
7 def 对准(身体部位, 物体):  
8     pass #TODO  
9  
10 def 缩紧(身体部位):  
11    pass #TODO  
12  
13 def 拉回(身体部位):  
14    pass #TODO  
15  
16 def 握住(手, 物体):  
17    展开(手)  
18    对准(手, 物体)  
19    缩紧(手)  
20  
21 def 摘(手, 物体):  
22    举起(手)  
23    握住(手, 物体)  
24    拉回(手)  
25  
26 def main():  
27    摘(苹果)
```

## 不良好定义的函数们

```
1 def 对准苹果(手):  
2     """这里实现了对准苹果的操作"""  
3     ...<一堆非常冗长的操作>  
4  
5 def bulibula():  
6     """实现了拿住苹果的操作"""  
7     ...<一堆操作>  
8  
9 def 摘苹果():  
10    # 下面实现了举起手的操作  
11    ...<一堆非常冗长的操作>  
12    # 下面实现了展开手的操作  
13    ...<一堆非常冗长的操作>  
14    对准苹果(手)  
15    bulibula()  
16    # 下面实现了拉回苹果的操作  
17    ...<一堆非常冗长的操作>  
18  
19 def main():  
20    摘苹果()
```

## 什么是“好”函数

该函数不能被继续拆分成任何子函数。

## 过犹不及

- 以上种种规范，都是基于方便编程者和读者的角度提出的。无论对于程序开发者还是阅读者，他们无疑都是非常善意的提醒。但作为初学者的我们也不要被这些条条框框约束地不敢写代码，一步三回头。须知，**大胆尝试，小心验证**。只有代码水准到了更高的层次才能更好的践行这些准则。
- 遇到错误第一时间想到的应该是**先看报错信息**，分析无果后可以上网查证，也可以数值验证。具体该如何 Debug 我们会在稍后 Slides 中说明。

# 什么是面向对象的编程?

## 定义

面向对象区别于面向过程(面向过程就是按照问题描述,按部就班地完成一系列操作. 面向过程的核心是“过程”如何一步步实现), 在解决问题时, 它会**把“过程”中出现的事物抽象成对象 (Object)**, 然后给予对象一些属性和方法. 而后通过对象间属性、方法的相互作用实现程序目标.

## 特性

- 相较于面向过程, 面向对象**更贴近人类思考事物的思维模式**. 因此在面对一些复杂功能时, 面向对象往往更轻松就能实现.
- 面向对象有三大特点: **封装, 继承, 多态**.
- 面向对象编写的程序, 性能一般比面向过程低. 但其易维护、易复用、易扩展, 代码不同模块间耦合度很低, 整个代码系统灵活有序.
- 科学计算一般使用面向过程编程, 复杂过程一般使用面向对象编程.
- 从面向过程到面向对象, 类似于, 从牛顿力学到理论力学.

# 面向对象代码示例

## 面向过程编程

```
1 def 举起(身体部位):
2     pass #TODO
3
4 def 展开(身体部位):
5     pass #TODO
6
7 def 对准(身体部位, 物体):
8     pass #TODO
9
10 def 缩紧(身体部位):
11     pass #TODO
12
13 def 拉回(身体部位):
14     pass #TODO
15
16 def 握住(手, 物体):
17     展开(手)
18     对准(手, 物体)
19     缩紧(手)
20
21 def 摘(手, 物体):
22     举起(手)
23     握住(手, 物体)
24     拉回(手)
25
26 def main():
27     摘(苹果)
```

## 面向对象编程

```
1 class 身体部位():
2     def __init__(self):
3         pass # Nothing done
4     def 举起(self):
5         ...<实现了身体部位的举起>
6     def 展开(self):
7         ...<实现了身体部位的展开>
8     def 缩紧(self):
9         ...<实现了身体部位的缩紧>
10    def 对准(self, 物体):
11        ...<实现了身体部位对准某物体>
12    def 拉回(self):
13        ...<实现了身体部位的拉回>
14
15 class 手(身体部位):
16     def 握住(self, 物体):
17         self.展开(手)
18         self.对准(手, 物体)
19         self.缩紧(手)
20     def 摘(self, 物体):
21         self.举起(手)
22         self.握住(手, 物体)
23         self.拉回(手)
24
25 def main():
26     我的手 = 手()
27     我的手.摘(苹果)
```

# 游戏中的面向对象编程



设想编程实现如下功能

下雨天时，人物躲在树荫下，此时一道闪电劈过来，人物手中的火把连同树荫下的草地都被闪电点燃。人物走出树荫，火把又被雨水浇灭。

# 如何学习一门编程语言

如同绝大部分语言一样，学会并熟练掌握一门编程语言的最有效方法就是多看多练。学习编程时，一定要保证自己是可以访问互联网的。

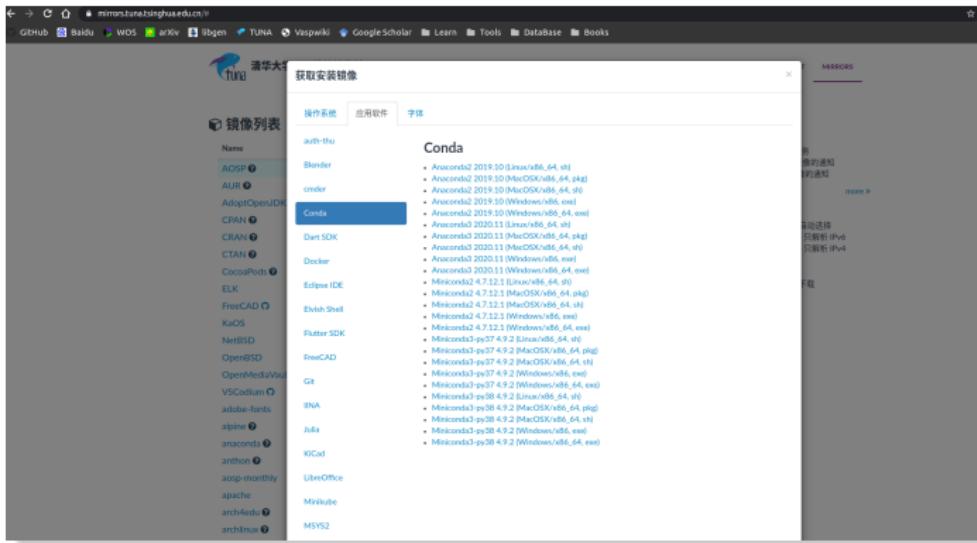
## 以笔者个人 Python 学习为例

- 首先笔者去“菜鸟教程”系统地学习关于 python 的所有基础内容。
- 而后花了两周时间，第一周先尝试编写一些小程序，如识别回文数，产生斐波那契数列，解一元二次方程组，寻找  $1E10$  以下的质数，等。
- 而后在后面一周，通过 python 尝试编写出了一个**地牢探险游戏**。那两周就是一直在上网查“python 中如何实现 xxx”“python 错误 xxx”，然后不断尝试网页上的提示，不断与各种 bug 斗智斗勇，在此过程中对 python 的理解也加深了不少，也乐在其中。
- 接着，等到笔者认为自己基本理解了 python 的使用之后，又去“菜鸟教程”浏览了一遍其内容，查缺补漏。
- 之后就是不断将自己学到的 python 编程知识应用到实践中，不断认识、否定认识、再认识。直到现在也在一直学习。

# 目录

- ① 为什么需要编程?
- ② 学习编程前应具备的认识
- ③ 学习 Python 前的准备工作
- ④ Python 基础语法
- ⑤ 使用 Python 进行数据处理
- ⑥ 总结

# 正确安装 Python (Conda)



## 安装 Anaconda3

前往[清华镜像站](#), 点击右侧边栏中的“获取下载链接”, 获取最新版 Anaconda3 下载包。conda 是 python 的包管理器。此软件可以高效便捷地管理 python 环境。可访问其[帮助文档](#), 学习如何使用 conda 指令。在 python 代码开发时, 使用环境管理器是至关重要的, 它可以省去很多麻烦。

# 高效使用编辑器或 IDE

工欲善其事，必先利其器。

能高效的使用一款编辑器或者 python IDE 是非常关键的。笔者曾见过有人使用 word 写代码，当时心中五味杂陈，也不免对其编写的程序多了几分忌惮。就好像是走在大街上迎面遇到了几个红绿头发说着火星文的花里胡哨的年轻男子。

言归正传，能正确使用一款高效的编辑器/IDE，才能最大限度地提升代码的舒适度和效率，也能在一定程度上提高 Debug 的精准度。高的编程效率和低的错误率总是我们所期望的。

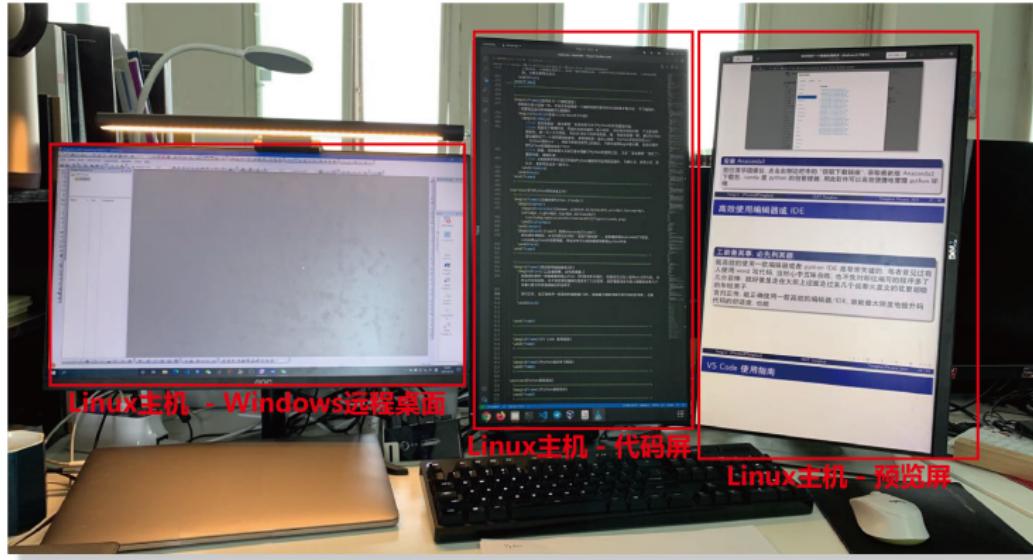
如果让笔者推荐一款编辑器，我大概会推荐由微软推出的 **VS Code**。如果说推荐一款 IDE，笔者就没有什么经验了，但好像大家都在用 **Spyder**。

# 提升工作环境

工欲善其事，必先利其器。

- 如果您希望能开始专业写代码工作，那么一个舒适高效的工作环境是必不可少的。试想，用一块手机屏幕学编程，和同时用两块屏幕，一块写代码，一块看教程看结果，其效率显然是不同的。
- 因此作为初学者，笔者建议至少配备一台笔记本，或者一个带显示屏的主机，之后根据个人工作需求添加额外配置即可。同时要保证，您的机器是可以访问网络的。
- 另外，根据笔者的经验，竖屏对编程和浏览文献是更加友好的；将 Linux 作为生产主机会比 Windows 高效一些。
- 同时要杜绝消费主义，我们的配套设施满足需求即可，不必追求过于奢华的配置。

# 笔者的工作环境



## 目前的工作环境

- 一台装载**Ubuntu 系统**、配置中游、与三块屏幕相连的主机，作为**主要生产工具**，主要用于代码开发、文献调研、数据分析、**LATEX** 文档书写、三维图片渲染等工作。
- 一台装载**Windows 系统**、配置下游、通过远程桌面访问的主机，作为**辅助生产工具**，主要用于 Origin 精修数据图、Office 全家桶文档书写、微信交流等。

# VS Code 使用指南

- 毫不夸张地说，自 VS Code 出现之后，他基本碾压了其他所有的编辑器。现在的 VS Code 就如同当年的 vim 和 emacs 一样风靡。
- VS Code 的火热很大程度上是因为其优雅的外观和丰富的插件。
- VS Code 给广大开发者提供了插件开发的接口，并由此诞生了数目众多的外接扩展库，如 C/C++，LaTeX Workshop，Remote-SSH，Python，Jupyter，Prettier 等等。这些扩展库有些是广受欢迎的，拥有千万级的订阅量。例如 LaTeX Workshop 插件的 LATEX 保存即编译功能就会让你爱不释手。已被 python 官方插件 Python 吸收 Pylance 插件，会让代码看起来非常舒适，也极大地降低了语法错误率。

让我们简单演示几个 Python 在 VS Code 中使用的例子。

# Python 语法学习网站

计算机语言学习不同于其他知识，他在互联网上的分布非常广泛，只要你愿意，在搜索引擎中输入‘Python 教程’，‘python tutorial’等关键字，就可以搜索出众多高质量的教程。

**RUNOOB.COM**

搜索.....

首页 HTML CSS JAVASCRIPT JQUERY VUE2 VUE3 BOOTSTRAP PYTHON3 PYTHON2 JAVA C C++ C# GO SQL 本地

Python 3 教程



Python3 教程

Python3 简介

Python3 环境搭建

Python3 基础语法

Python3 基本数据类型

Python3 解释器

Python3 注释

Python3 运算符

Python3 数字(Number)

## Python 3 教程



print("Hello, world!")

Python 的 3.0 版本，常被称为 Python 3000，或简称 Py3k。相对于 Python 的早期版本，这是一个较大的升级。为了不带入过多的累赘，Python 3.0 在设计的时候没有考虑向下兼容。

Python 介绍及安装教程我们在[Python 2.X 版本的教程](#)中已有介绍，这里就不再赘述。

[你也可以点击 Python2.x 与 3.x 版本区别](#) 来查看两者的不同。

本教程主要针对 Python 3.x 版本的学习，如果你使用的是 Python 2.x 版本请移步至[Python 2.X 版本的教程](#)。

官方宣布，2020 年 1 月 1 日，停止 Python 2 的更新。

<https://www.runoob.com/python3/python3-tutorial.html>

# 目录

- ① 为什么需要编程?
- ② 学习编程前应具备的认识
- ③ 学习 Python 前的准备工作
- ④ Python 基础语法
- ⑤ 使用 Python 进行数据处理
- ⑥ 总结

# Python 基础语法

- Python 对大小写敏感, 所有名称或标志符必须以字母或下划线开头, 并由字母、数字、下划线构成.
- Python 中单行注释以井号 `#` 开头. 可以使用三引号 `''' ... '''` 或 `''' ... '''` 对应的字符串块进行多行注释.
- 与大多数语言不同, python 不依靠首末起止符 (如左右大括号 `{...}`) 标定代码块, 而是通过行首缩进格数区分代码结构. 这就引出了 python 编程规范里一条非常重要的要求: 在任何时候都不要使用 `TAB` 键自动缩进.
- 如果感觉一行写不开, 可以用反斜线 `\` 来实现代码换行. 同时某些位置 (例如被圆括号括 `(...)` 起来的两行) 的换行是不需要反斜线标定的.
- 使用 `print()` 函数可以将目标参数输出至标准输出 (终端显示屏).
- Python 是解释型语言. 也即他的代码是逐行单独运行的.

请注意, 由于 PDF 编码问题, 请勿复制本 Slides 中的任何符号或代码到程序中.

# 变量类型和运算符

## 变量

- True, False, None
- 整数, 浮点数
- 字符串
- 列表, 元组, 字典, 集合

## 运算符

<code>+ - * /</code>	加, 减, 乘, 除
<code>// % **</code>	整除 (向下取整), 模, 幂
<code>== != &gt; &gt;= &lt; &lt;=</code>	逻辑等于, 不等于, 大于, 大于等于, 小于, 小于等于
<code>=</code>	赋值等于
<code>+= -= *= /= //=% *= **=</code>	赋值加等, 减等, 乘等, 除等, 整除等, 模等, 幂等
<code>&amp;   ~ ^ &lt;&lt; &gt;&gt;</code>	按位与, 或, 非, 异或, 左移, 右移
<code>and or not in</code>	并且, 或者, 不是, 包含

# 流程控制语句

## 概述

- `if (condition): ... elif ... else ...`
- `for (condition): ...`
- `while (condition): ...`
- `break, continue, pass`

## if 判断

```
1 if iamyourfather(me.name):  
2     me.get_money(1E9)  
3 else:  
4     print("No, I believe in Yoda.")
```

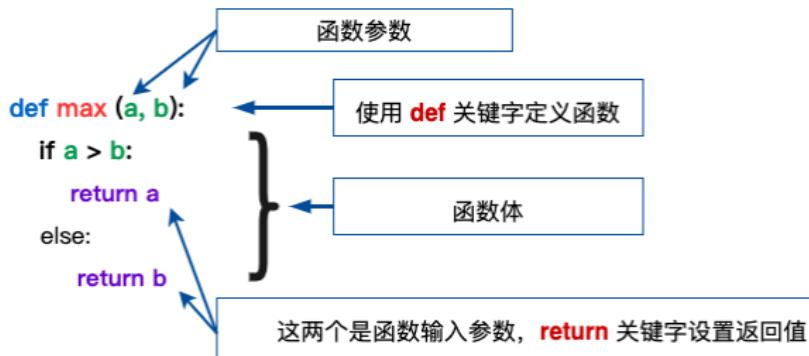
## for 循环

```
1 for index in range(10):  
2     print('Yoo! %d' %(index,))  
3 print("I finally find you!")
```

# 函数声明与定义

## 函数

函数可以输入参数，并设置返回值



www.runoob.com

Cite from www.runoob.com

```
1 def iamyourfather(a_name):  
2     if a_name == 'father':  
3         return True  
4     return False
```

# 如何安装和导入一个外部模块

## 安装

您可以使用 `conda install [pkg]` 指令安装指定包.

您可能也听说过可以使用 `pip` 来安装 python 包. 但随着对 python 使用的深入, 您会越来越体会到 `conda` 环境管理器的便捷之处.

## 导入

使用 `import` 指令导入被安装好的模块.

```
1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy import optimize
5 from math import *
```

## 导入规范

- 一般 `import` 语句仅会出现在文件最开头.
- 单个 `import` 中不建议导入多个库的内容.
- 某些库的导入是有建议顺序的.

# 面向对象编程

构造对象包括构建对象的属性和方法. 对象的属性就是在对象所对应的类中定义的变量, 对象的方法就是类中定义的函数.

```
1 class ClassName():
2     ...<some_statement>
```

```
1 # 声明类
2 class person():
3     # 类的初始化函数, 在该类实例化时自动执行
4     def __init__(self, name):
5         person.name = name # 类/对象的属性: 名字
6         person.money = 0 # 类/对象的属性: 金币数
7
8     # 类/对象的方法: 获得金币
9     def get_money(self, money)
10        person.money += money
11
12 # 类的实例化产生对象
13 me = person('father')
```

# 报错的解析和处理

```
1 # 类的定义
2 class person():
3     def __init__(self, name):
4         person.name = name # 类的属性
5         person.money = 0 # 类的属性
6     # 类的方法
7     def get_money(self, money):
8         person.money += money
9
10
11 # 函数定义
12 def iamyourfather(a_name):
13     if a_name == 'father':
14         return True
15     return False
16
17
18 # 实例化类 -> 对象
19 me = person('father')
20 # If 语句
21 if iamyourfather(me.name):
22     me.get_money(1E9)
23     print(me.name, me.money)
24     # For 循环
25     for index in range(10):
26         print('Yoo! %d' %(index,))
27         print("I finally find you!")
28 else:
29     print("No, I believe in Yoda.")
```

## 错误信息

```
1 Traceback (most recent call last):
2   File "/home/1.py", line 22, in <module>
3     me.get_money(1E9)
4   File "/home/1.py", line 8, in get_money
5     person.money += money1
6 NameError: name 'money1' is not defined
```

# 目录

- ① 为什么需要编程?
- ② 学习编程前应具备的认识
- ③ 学习 Python 前的准备工作
- ④ Python 基础语法
- ⑤ 使用 Python 进行数据处理
- ⑥ 总结

# 若干必要的函数和方法

## with, open, readlines, writelines

```
1 with open('test.txt', 'r') as frp:  
2     lines = frp.readlines()  
3  
4 with open('test-copy.txt', 'w') as fwp:  
5     fwp.writelines(lines)
```

## str.split(), str.replace(pattern, repl)

```
1 str = '123, viva la vida !'  
2 str = str.replace('123, v', '456! V')  
3 print(str)  
4 str = str.split()  
5 print(str)
```

# 若干必要的 Python 模块

导入命令	描述
<code>import sys</code>	操作系统交互模块
<code>import os, shutil, glob</code>	文件系统管理模块
<code>import pandas as pd</code>	数据表格化处理模块
<code>import argparse</code>	文件参数封装模块
<code>import re</code>	正则表达式匹配模块
<code>import numpy as np</code>	数值计算模块
<code>import scipy</code>	科学计算模块
<code>import json, h5py</code>	数据文件封装模块
<code>import matplotlib.pyplot as plt</code>	绘图模块

# 数据处理程序的基本流程

## 流程

读入参数 → 读取数据 → 处理数据 → 输出数据

- 让程序暴露出一些可变化的参数，是泛化程序功能的重要一步。毕竟谁都不希望自己的程序只能用一回。
- 读入数据一般来源于 ASCII 码编码的文件（\*.txt, \*.json 等），或者压缩的二进制数据文件（\*.hdf, \*.pkl 等）。
- 书写处理数据算法时，一般要考虑该算法的泛化能力。往往功能覆盖范围越广的代码需要越长的开发时间，因为要处理多种多样的不同情况。一个非常明智的做法是，使得不同情况之间的代码尽量脱耦，便于后人继续在我们的代码上发展。
- Python 的数据输出过程一般非常直接。例如，输出到 \*.json, \*.hdf, \*.pkl 等文件时，往往只需要一行命令就能完成。而将数据绘制成图，或者输出到类 \*.txt 文件时，则需要一番精心设计。

# Python 程序的书写格式

Python 数据处理程序可以有以下几种格式：

- **写成独立的可自运行的文件.** 这是最一般的形式, 只要写好了配套的说明, 就能让用户, 在保持一定的灵活性的前提下, 非常傻瓜式的使用该文件.
- **写成库 (模组) 文件.** 将你的程序写成一个库文件 (类似于 `numpy`), 使得用户可以直接 `import` 调用他. 这种数据处理程序使用门槛相对较高, 用户应至少会写 `python` 程序. 但这也是最灵活轻便的一种模式. 用户根据自身需求, 可以将我们的程序灵活地内嵌在其他程序中.
- **写成 Jupyter Notebook.** `Jupyter` 其实一开始是用于演示教学的. 它运行的过程就是将代码写在一个个代码块当中, 而后逐块运行, 并可以立即查看每块的运行结果. 正如其名 `Notebook`, 书写 `Jupyter` 的过程就像是在记笔记的过程, 因此很多人非常喜欢使用这种风格的代码, 认为他看起来更清晰明了. 其实仔细想想, 是 `Jupyter` 这种格式让人油然而生了一种**写注释**的想法. 正如之前提到的, 他非常适合教学, 您可以通过他来向别人说明传授数据处理的具体细节过程. 但如果希望书写一个高效、灵活、复用性高的代码, `Jupyter` 应该不是一个好的选择.

# 自运行程序的参数读入

读入参数的方式直接决定了代码的学习代价，一般要在学习代价和代码灵活性之间权衡。读入参数的方式有多种：

- **直接写死在程序中**. 于是每次有人想处理新的数据时，就要去改代码. 这其实是最“偷懒”的一种做法，但却也是被最广泛应用的.
- **通过 argparse 等库，将传参分配到终端**. 在参数较少时，使用这种方法是非常明智的. 配合清晰书写的 `-h` `--help` 说明，就能形成一套使用非常清爽的代码.
- **将程序做成交互式的，一问一答**. 这种程序输入借鉴了 Windows 下 GUI 界面的思想，比较人性化. 只适合复杂过程的单次运行. 这种设计的好处是学习代价很低，基本不需要写帮助文件，直接跟随程序引导作答即可.
- **将参数写在配置文件中**. 这种传参方式适合参数较多（比如有数百个可调参数）的程序. 通过一个标准的配置文件，如 `input.ini`, `input.json`, `input.txt` 等，实现参数的传入. 这种传参方式的好处是一套参数保存下来之后就可以快速复用. 但学习成本较高，一般需要书写专门的用户手册，指导用户如何使用不同的关键字.

实际程序读入参数的方式，可以是多种方法的结合。

# 目录

- ① 为什么需要编程?
- ② 学习编程前应具备的认识
- ③ 学习 Python 前的准备工作
- ④ Python 基础语法
- ⑤ 使用 Python 进行数据处理
- ⑥ 总结

# 总结

- 编程作用体现在科研的方方面面.
- 学习编程前, 应该意识到编程是有最基本的规范的, 遵循这些规范, 利己利人.
- 面向对象编程是一个非常有力的编程方式, 如果您需要实现一些非常复杂的功能, 应该考虑使用这种编程模式. 当然, 由于其性能问题, 开发大规模科学计算软件的包时除外.
- Python 一般使用 `conda` 管理, 一个好的编程工具和编程环境对高效编程是必不可少的.
- Python 基础语法十分明了, 如果您之前学习过其他编程语言, 应该很快就能上手.
- Python 最强大的是其丰富的外部拓展模块. 合理利用这些模块会让我们的工作事半功倍.
- 合理估计数据处理程序的目标, 在开始编程之前就确定一个合理的参数和文件处理流程, 将降低未来代码重构的工作量.