答应章小曼同学，简单列一下编程语言的学习经验，然而几个月来一直忙于工作，迟迟未动笔，感到十分愧疚，这几天陪我爸在医院看病，空闲一点，基于自己的记忆，零零散散整理内容如下：

1. **编程语言之基本概念篇**
2. 问题空间（面对业务逻辑）->解空间（面对机器）

程序员必须建立起在机器模型（位于解空间内，这是你对问题建模的地方，例如计算机）和实际待解问题的模型（位于“问题空间”内，这是问题存在的地方，例如一项业务）之间的关联（摘自”Java编程思想”）。

1. 数据类型，运算操作，逻辑操作，循环，控制流，函数，作用域，递归，调试
2. 类，访问权限，初始化，清理，闭包，OOP, 集合，异常处理…

具体内容可以参考书籍，建议国外书籍，很多语言发明者写的书，思想很深刻。

1. **编程语言之语言篇**

建议先google一下,搜”编程语言排行榜”，对前50名编程语言有个大概了解.然后根据自己的研究领域，选择合适的语言。以下是基于我的个人经历如何学习编程语言。计算机和软件学院的同学，可以忽略此文。

***Matlab***

有人称汇编语言为第一代语言（人类可阅读语言，你要是能阅读机器语言，我只有膜拜，不提倡），C/C++是第二代语言，matlab是第三代语言。为什么先介绍matlab呢？有几个理由：matlab脚本语言，自带IDE，易调试，丰富的包（我理解为算法层封装），含数据读取,处理和可视化，适合搞科研，缺点占用资源多，不适合开发GUI程序，deploy文件大.

笔者如何使用matlab?

首先是项目组织结构

Core

Data

Gui

Test

Core 一般存放代码，一个公有函数用一份文件来表示，函数名同文件名，函数体最好只有二三十行，我个人是倾向于按职能分解，越短越好，如果一个公有函数过长，可以在同一份文件里拆解成一个公有函数+几个私有函数，那么对外接口只有公有函数，代码函数名和变量名尽量有一定的语义，从而有文档的功能。接口注释文档涉及三点即可：实现功能，输入输出参数和负责人是谁, 写得详细点未尝不可，如使用范例，算法参考的文档等等。强调一下：如果更新了代码，也需要更新对应的文档。客户端仅需要了解API

当然，若用类来组织，而不是函数，代码和文件数会少很多，基于过程式编程比较容易理解，个人觉得够用就可以了，等将来C++/C#/Java类的思想深刻了，再提高matlab的编程能力

Data文件夹一般存放代码输入数据

Gui文件夹一般存放matlab gui应用程序，仅和界面有关的代码和逻辑放在这个文件夹里。如果该项目验证科研思想，结果很不错，要转化为prototype,那么会有R&D部门接手，将core文件夹代码翻译为C++动态库，Net程序集或java包，用WPF,javafx或html+CSS实现美观的界面。

Test文件夹一般存放自己的代码测试，我不太在matlab里用单元测试，但是每个公有函数最好有个范例脚本文件，一方面用来作为文档，另一方面用来测试功能

基本知识点：Matlab基本语法, 矩阵操作，struct操作，cell操作, 基于过程, 面向函数。重点强调一下面向函数这种编程方式，用得好matlab代码质量和性能会有质的提高。关键实现点是函数指针.如ds.(col) = cellfun(@convert2Num, varCell) ; varCell是一个数组，遍历数组，取每个元素作为输入参数传给函数convert2Num(),输出一个数组。

Matlab是矩阵实验室的简称，从单字上容易理解，易用来实现信号处理，图像处理。详细内容请参考matlab guide。

Matlab验证科研思想很容易，然而封装太彻底，改动很困难，而且机器运行效率很低。若要深入了解底层工作原理，需要进一步学习C语言

Matlab 自带的帮助文档十分详细，有很多范例，英文版写得通俗易懂，网络上有中文版，比任何一本书籍都好。这里不推荐matlab任何语言类书籍，偏业务逻辑的书籍很赞，这里推荐两本

1. 《数字图像处理matlab版》, 作者:冈萨雷斯
2. 《Matlab数值计算》, 作者：[Cleve B.Moler](http://www.youlu.net/search/result/?author=Cleve+B.Moler" \t "_blank) （注意此人是matlab之父,好像还有另一本新书，《编程实战》，没看过，不敢推荐）

***C***

个人一直认为C语言是必学的，C语言有很多的书籍可以参考，指针概念一定要深入理解，指针就是内存地址，不论什么指针，如void\*, int\*, float\*都占用4个字节，但通过指针引用的内存大小是根据指针变量类型和开辟的内存大小的乘积=sizeof(type)\*n，指针运算也是基于指针变量类型来计算的，如

int\* pInt = new int[10] ;

++ pInt ; //指针移动4个字节

double\* pDouble = new double[10] ;

++ pDouble; //指针移动8个字节

注意，释放内存后，指针悬空，需要置空

C语言一般用来实现操作系统，假定你和我一样，偏业务逻辑，很少负责实现操作系统，不予讨论。

另一方面C语言用来提升性能，完全可以用更优美的C++来替代

但是有些硬件专家只能用C语言来实现业务逻辑，尽管我这方面没有涉猎，大概了解到也不需要自己从头开始，可以利用前人的成果库代码，如信号处理库，数值分析库，图像处理库，机器学习库等

C语言有很多很优秀的书籍，推荐如下:

1. **《C语言程序设计》(C语言之父)**
2. **《C接口与实现》**
3. **《C陷阱与缺陷》**
4. C语言的科学与艺术
5. C和指针
6. C编程专家

有时间，建议把这些书籍都过一遍，毕竟都是大牛们的呕心之作啊,前三本一定要精读

***C++***

IT界流利一句话：真正的程序员用C++。很明显，共识C++很复杂，分三大部分内容:C语言兼容(面向过程)+类(面向对象)+模板(泛型编程)。个人认为有C语言基础，学一下面向对象编程即可，至于泛型编程，议题过于复杂，不求深入，略懂概念，会用STL即可。

重点概念：作用域，接口和实现，类的三个特性：封装（信息隐藏），继承（代码复用）和多态（接口契约，无限子类化）,…

C/C++的动态库实现要熟练掌握，参考模板即可。

程序员的效率优先于机器效率，人类可读的代码优先于机器效率。因此常常用更纯粹更抽象的语言来实现业务逻辑，比如matlab,C#或python来实现系统。而系统耗时最多一般仅在20%的代码中，特别是大量的for循环，仅对这些代码块进行C/C++性能优化且并行加速，就会有很显著的提升。然后将这些加速的并行化代码写成动态库，主程序采用策略设计模式，可以切换先前的未优化算法为封装后的优化算法，封装层代码调用动态库(这里涉及到混合编程，如果语言没有提供混合编程，可能要在更抽象的一层进行调用，比如com,进程甚至web service)。关于性能优化可以参考我的性能优化PPT。

同样利用前人的劳动成果，借助类库来实现自己的动态库。STL, boost, 数值分析库，图像处理库(ITK), 可视化库(VTK)等。

C++的好书籍非常多，推荐如下：

1. **《c++ primer》**
2. **《C++语言程序设计》（c++语言之父）**
3. **《C++编程思想》（上下卷，第二卷绝对是重点）**
4. **《Effective C++ & More Effective C++》（感觉有点像考点，把重点挑出来了）**
5. **《深度探索C++对象模型》　（从C的观点，C++编译器在底层上为你实现了什么）**
6. Exceptional C++ & More Exceptional C++(感觉也有点像考点，把难点挑出来了）
7. C++编程规范
8. 高质量C/C++编程指南
9. C++模板完全导引
10. C++模板元编程
11. c标准库
12. c++ STL
13. C++标准程序库
14. Boost入门导引
15. Modern C++ Design

C++确实很复杂，好书也特别多，有时间把前5本看了吧

***C#***

移除了C++不容易使用的知识点，更纯粹的面向对象, String增加很多功能，interface,单继承，用C#主要是用来实现桌面端应用程序，下面简要列一下个人实现界面的一点经历和看法:

记得刚开始学MFC，C++的界面框架，学起来不易，用起来更不易，bug数不清，代码依赖性很强，相当痛苦。MFC无法讲清事件原理，我是通过圣经才搞懂的，强烈推荐Charles Petzold的圣经《windows程序设计》

后来用Qt，不得不说，Qt很人性化。容易上手，也容易使用，但华丽的界面还是通过贴图的方式。

学C#时，当时主流还是Winform,用起来和Qt很像，或许模仿Qt也说不定。

最喜欢的还是用WPF实现界面，相当人性化，界面逻辑用XAML语言，仿XML格式，流式布局，双向绑定，动画，GPU渲染等，超级赞。

我的建议（我常常这么做），桌面端应用程序，界面用WPF，业务逻辑用C#,持久化层用ADO.NET。耗时算法用C++实现的动态库。

C#是微软实现的语言，有丰富的.NET程序集。当然也可以通过互操作，调用C++实现的动态库。

C#书很多，好书不多，建议深入阅读C/C++书籍后，仅需阅读C#如下两本书籍：

1. **《C#高级程序设计》**
2. **《WPF编程宝典》 作 者：（美）麦克唐纳**

***Java***

Java和C#风格很像，如果C/C++/C#熟练的话，Java只要一周就可以上手了，桌面应用程序，界面可以用javafx实现，用FAML语言。Java作为服务端语言比较多。

Java推荐书籍如下：

1. **《Java编程思想》**
2. **《Java语言程序设计》作者：梁勇**

***Python***

Python,我最喜欢的代码，Python是由一位数学家发明的，语法很严谨，不提倡多种实现方法，图像处理，文本处理，网络处理，数据挖掘有很多的工具包，快速开发。

我也仅花一周时间就上手了，Python推荐书籍如下：

1. **《Python基础教程》**
2. **《Python标准库》**

***PHP***

语言类C风格，有很多的函数集可以直接使用，最好使用框架，比如thinkphp，mvc模型，极大地提升程序员效率和代码质量.

仅看一看书，号称php+mysql圣经，推荐如下：

1. **《PHP和MySQL Web开发》**

***Html5+CSS3+Javascript***

不得不说web前端设计很乱，其发展史乱，而且浏览器也很乱。语法不严谨，语义不充分，我看了很多相关的书，代码简直惨不忍睹，维护太困难了。基于个人的经验用html5+css3+javascript实现前端，就像手工制造汽车。

传统的方法：一般是借鉴别人实现的网站，做为模板，进行修改，实现分栏导航主体页脚等等。

实现动态web，服务端语言如php,java,c#,python直接实现html页面，然后发送到前端。前后端耦合让人很抓狂

这几年提倡RIA（rich internet application），每个页面后端实现，前端渲染，实时交互对网络带宽要求很高。实际上页面一般只更新部分面板，意味着只需传输部分内容。而这个部门内容往前端通过ajax请求，后端处理业务逻辑，然后传送回json数据给前端。

实际上前后端可以完全分开来实现，后端不论什么语言，仅需传送前端json数据，前端可以有自己的mvc框架，javascript类库，来实现界面和行为。前端比较好用的框架有angularjs, bootstrap, jquery, extjs等，注意这些框架或类库可以配合一起使用

正如很乱的前端设计,相关的书籍多如牛毛，尽管我阅读不下十多本，就没觉得哪本是好书，免得误人子弟，这里不列了。

***R***

适合于统计学和数据挖掘，有丰富的工具包和可视化包，比matlab好用多了。

推荐书籍如下：

1. **《R语言实战》**
2. **《数据挖掘与R语言》**
3. **编程语言之平台篇**

写干净的代码，除了学习语法知识，最重要的是实践和思想。课本上的范例一般是玩具式的代码，最好是模仿优秀的代码，大量实践，同时阅读大量相关书籍，久而久之，转化为自己的思想。

代码经验不多的同学，希望能边敲代码边看结果，对于脚本语言，比如matlab,python,php可以用控制台命令进行交互，对于编译型语言，比如C/C++/C#/Java,需要编译链接执行。个人提倡安装IDE，编辑器皮肤修改为赏心悦目一点，配合调试器，可以提高程序员工作效率。

1. Matlab IDE最适合初学编程的同学，有控制台交互，有编辑器，自带调试器。安装完整版，几乎各个研究领域都有相关的包，直接可用. matlab guide英文版或中文版文档都很好，非常详细，而且有很多的范例可以参考。因为matlab主要是用来验证想法，所以代码一般是便贴式的，很少要求详细的单元测试
2. C/C++/C#,最好用最新版visual studio，solution=project + project + …， 最好对生成的各个文件有个大概了解，比如.sln, .proj, .obj, .pch等等，用版本控制时，最好仅上传需要的文件，否则易引起冲突。若想进一步提高程序员效率，提升智能体验，可以安装visual assist。很多项目文件一起合作时，项目之间会有依赖性，即动态库的依赖性，可以用dependency walker来解析.对大的项目特别有用.
3. Java, python, php，最好用eclipse + pydev + phpDev. 具体安装和使用，可以参考网上的文章，和visual studio使用很像.
4. Html5+css3+javascript,建议使用webstorm,调试时可以直接在chrome里，按F12，打开开发者平台。
5. R，建议使用R studio,和matlab IDE很像。
6. **编程语言之提高篇**

要想写出干净整洁的代码，同时考虑运行的性能效率等等，需要付出很多的努力。

对解空间十分熟悉，处理器和操作系统工作原理

熟悉语言惯用法，语言禁忌，使用准则等等（这些都是大牛几十年的经验啊）

熟悉模式，框架。

有很多的好书，推荐书籍如下：

1. **代码大全**
2. **敏捷软件开发**
3. **重构**
4. **重构与模式**
5. **单元测试（CppUnit,NUnit, JUnit）**
6. **设计模式**
7. **算法导论**
8. **数据结构c**
9. **操作系统概念**
10. **程序设计实践**
11. **深入理解计算机系统**