1. Python + 财经 -> 最佳实践

- **内容服务的对象：财经人员**，本书分上下两册，上册为通用教程，可服务于所有入门python的读者；下册为专业教程，主要服务从事金融、会计等相关工作的人。
- **内容安排的原则：知识屏蔽**，即对教学内容进行最小分割，力求使学习过程中所接触到的每一个知识点都是当前唯一要去理解的东西。
-

1.1. CH01 - The Very Intro to Python

1.1.1. Overview

本章节主要讲解学习Python的一般途径与职业洞见，以提供读者对Python多方位、立体化的认知，并指导搭建好读者的程序开发环境，为后续的学习、生产建立基础。

1.1.2. What's python and what's python for?

1.1.2.1. 为什么要学Python？

我们先把这个问题放在首位，这个问题其实很重要，因为这并不不偶然。

有人会说：“因为大家都在学Python，所以我不能落下”，这或许可以称之为一个理由。

但你真地明白大家为啥要学Python吗？或者，你去问问那些学的比较深入的人，他真地喜欢Python吗？我觉得，在你学习Python之前，如果可以，问一下那些人：“嘿，老哥，你为啥学Python？学习Python给你生活带来了什么改变吗？”作为一门编程语言，按照直觉，这应该不会有什么令人出乎意料的答案。比如你可能听到：“啊，学习Python啊，我感觉它很容易学“，或者”谁知道呢，也许可以用Python减轻我原本的工作吧”。

这些答案都是可以接受的，但如果你问我，我会回答：“我可以用Python完成非常多原本不可能完成的任务；在学习过程中，我更懂计算机了；我从未感觉学习一门编程语言能像Python一样如此富有魅力，以至于更像是一种享受，以至于我现在全身心地投入到了Python的推广教学当中去了，以至于我必须把Python的首字母大写以表示这是Python而非其他种python”。

"Easy, Powerful, Prevailing and then Enjoyable, Beautiful, Philosophical"，这就是我为什么要学Python的答案：

1. 因为简单，所以我很快地掌握了它的使用
2. 因为有用，所以我很快地做出了一些东西
3. 因为流行，所以我很快地找到了一些工作（准确地说是“实习”）
4. 并且它是美丽而富有哲学意味的，由此开发Python的体验是享受的
5. 最后，它一定程度上改变了我的生活，并带给了我一种新的审视世界的角度与态度

感性到此结束，我们来看看这三个要素究竟是怎么把我引向学习Python的热潮中去的：

- C、C++、JAVA等语言非常Powerful并且Prevailing，但是它们并不easy，比如C中的指针让人望而生畏，知乎的轮子哥曾经专门教一个人C语言，安排一周讲解所有基本概念，但却安排了两周给他讲授指针，可想而知
- Matlab、R、JavaScript等语言并不那么难，但都只是特定领域的专用语言，如果你要走学术那你可要好好学学matlab或R等，如果你要做前端可要好好学习JavaScript等，总之它们都不适用于所有人，当然类似的还有一些大数据时代专用的语言比如Go、Scala等也可归于这一类
- 再举个不是很流行的例子，比如lua之类的语言
- 但有一种算不上编程语言的语言，我必须突出一下，那就是Markdown，它是一种标记语言，广泛地用于互联网的博客写作与Github项目的首页展示，比如本系列教程就是完全基于Markdown的。By the way，我的英文名就叫Mark
- 因此，当你不知道要学哪门编程语言以应对未来就业市场的冲击的时候，选Python准没错；如果你知道自己要学哪门编程语言，又正好读到我们这份教程，那我想说：“加油，请坚定地走下去”。此时或许可以再来波鸡汤：“未来你一定会感谢现在如此明智而又努力的你的！”

1.1.2.2. Python到底是什么？

在写这份系列教程之前，我有承诺给我的读者，要在书中体现程序员的思维，Now，这是第一个：**证伪往往比证明更容易**。即我无法给你定义Python是什么，但可以准确地回答一个伪命题，比如：“Python是最好的编程语言”。这显然是不成立的，毕竟Python的平均运行效率在Java的十分之一左右，在C的百分之一左右。

我们可以再深入一些，什么是证伪比证明更容易，我们可以借用软件工程中单元测试的概念来试图说明这一点。比如你写了一个验证用户输入的手机号码是否合法的程序，作为用户如果他不信任你的程序的话，他会怎么测试你的程序呢？他可以先输入一个空值，看看会不会报错；如果会，再输入一个字符串看看会不会；如果依旧会报错，再输入一个负数看看会不会报错；如果还是会，再输入一个小数看看会不会报错；如果仍然会报错，再输入一个十位数号码看看会不会报错……

严格意义上，到上一步，测试已经结束了，如果所有的以上都报错，然后用户输入了一个11位正整数提示通过，那么你的程序就已经合格了。所以你看前面的所有步骤都是在证伪，是一步步去伪存真，最后才能能够勉强证明你的程序是合格的。为啥这还只能算合格？因为你还没考虑首字为0的情况，以及没有考虑该电话号码是否符合其他更鲜为人知的中国电话号码规范，furthermore，我们还没讨论该程序对国外电话号码的检验与适用情况。所以你看，证明一个程序是正确的，或者说robust（稳健的），是多么地不容易。这便是我要花这么长的篇幅拒绝一上来就回答Python是什么的原因。

此外，我们还有第二个程序员思维：**封装是一把双刃剑**。同样是这个测试手机号码输入合法的程序，一些对Python了解较为深入的同学会嘲笑到，直接用正则表达式匹配11个数字不就解决了初步的校验问题了吗？没错，是这样的。正则表达式就是对一些字符串处理逻辑、模式的封装，并对上层提供一个更加简单、容易理解、容易调用的接口，以使上层的开发人员能够用更少的代码完成更强大、更复杂的逻辑处理。但这是一把双刃剑，因为封装让所有的一切变得更加抽象，让你无法理解你的错误，为啥会这样呢？什么又是封装呢？为了更好地理解这一点，我们再选取一个通俗易懂的例子：金融危机。

对于程序员来说，整个世界的金融体系就是一个庞大的系统工程，它是由无数的政客、商人、普通人等，以及一整套货币体系、商品流动等构成的。当我不是一个金融研究者的时候，整个金融体系对于我来说就是一个封装地很完整的工程，我并不了解其金融的运转细节，但当我要去取钱的时候，我知道我要做的就是我去这个金融体系里的所有银行中的我可以取钱的那家银行取钱。用程序的逻辑描述就是：

```
金融体系.所有银行.返回我开户的那家银行().返回这家银行旗下离我最近的网点().前往取钱(取钱金额=?, 其他参数)
```

是吧，其实就是这个意思，这个世界的封装让世界变得有序，让做某件事情的逻辑变得很清晰，但一旦哪里出了问题，由于你不知道每一步的逻辑是怎样的，也不知道有没有隐藏的bug，因此出了问题就会很棘手。比如就这个取钱的函数，你因为下雨没取到钱，这是一件很严重的事情，因为你会想不应该啊，这个金融体系应该能够支持我即使下雨天也能取到钱的，毕竟也没让我输入是否下雨这个参数啊，也就是说下雨不应该成为最终能不能取到钱的影响因素。但为啥最后没取到呢？

你最后仔细检查了一下程序逻辑，才发现，在这个`前往取钱(取钱金额=?, 其他参数)`的函数里虽然有考虑在下雨天的时候带伞以能够前往网点的函数调用，但这个调用没考虑你下雨天因为不小心滑倒摔伤被送进医院从而导致这个调用无法正常返回一个`已经抵达网点`的较低层输入，最后自然就没取取到钱，甚至会直接报错。你这么去想的话，就能明白为啥金融危机的争论这么久了，因为整套ABS体系过于复杂让人蒙圈，但比ABS更复杂的是整个世界的国际霸权体系，而比这个更复杂的是，根植在这个世界下的人性的复杂，这导致了推演、排查整个金融危机的原因，并优化这个金融体系的工作变得相当困难。甚至直到现在，我们都不能确定这个金融体系能像1+1保证等于2一样稳定的运转，这便是封装带来的不利之处。

而Python正是基于C语言之上封装起来的更高级的程序，它的easy正是来自于对很多c语言较为直观的逻辑的复杂的组合、包装与简化。

但直到现在，我都无法给出Python是什么，虽然你可以在维基百科、百度百科、Python官网、Guido van Rossum（Python创始人）口中得到无数种定义。相比于在这些囿于世俗、教科书般死板的解释，我更愿意简单地称Pyhton就是一门***Easy, Powerful, Prevailing***的语言，比如以下。

```
>>> 1+1
2
```

好吧，不开玩笑，我们用直观的例子，解释什么是Python。你打开你的计算机，在没有开机的时候它就只是一个硬件，是没有任何灵魂可言的。现在你开机了，随着屏幕的不断闪烁，你进入了你习以为常的欢迎界面，这就是构建在计算机底层芯片、内存、主板之上的操作系统，可能是Windows，可能是Mac，也可能是Linux等。接着，你终于进入了你的桌面，然后打开了你的某个开发Python程序的软件，可能是PyCharm，可能是Jupyter Notebook，可能是IDLE，也可能是Sublime，甚至可能就是一个记事本，然后输入了一些读起来很像英文诗的语句：

```
from datetime import date

class Person:
    def love_or_not(self, the_guy, the_time):
        """
        To love or not to love, that's a question.
        """
```

```
        return True

if __name__ == "__main__":
    me = Person()
    she = Person()

    if she.love_or_not(the_guy=me, the_time=date.today()):
        print("I should be a better man!")
    else:
        print("Please give me a cigar!")

    print("After all, tomorrow is another day!")
```

作为对Python的尊敬，我们还是提供一份维基百科的解释如下：

> **Python**（英国发音：/ˈpaɪθən/ 美国发音：/ˈpaɪθɑːn/）是一种广泛使用的解释型、高级编程、通用型编程语言，由吉多·范罗苏姆创造，第一版发布于1991年。可以视之为一种改良（加入一些其他编程语言的优点，如面向对象）的LISP。[来源请求]Python的设计哲学强调代码的可读性和简洁的语法（尤其是使用空格缩进划分代码块，而非使用大括号或者关键词）。相比于C++或Java，Python让开发者能够用更少的代码表达想法。不管是小型还是大型程序，该语言都试图让程序的结构清晰明了。
>
> 与Scheme、Ruby、Perl、Tcl等动态类型编程语言一样，Python拥有动态类型系统和垃圾回收功能，能够自动管理内存使用，并且支持多种编程范式，包括面向对象、命令式、函数式和过程式编程。其本身拥有一个巨大而广泛的标准库。
>
> Python 解释器本身几乎可以在所有的操作系统中运行。Python的其中一个解释器CPython是用C语言编写的、是一个由社群驱动的自由软件，当前由Python软件基金会管理。

1.1.2.3. 各大编程语言基本信息

（待更新）

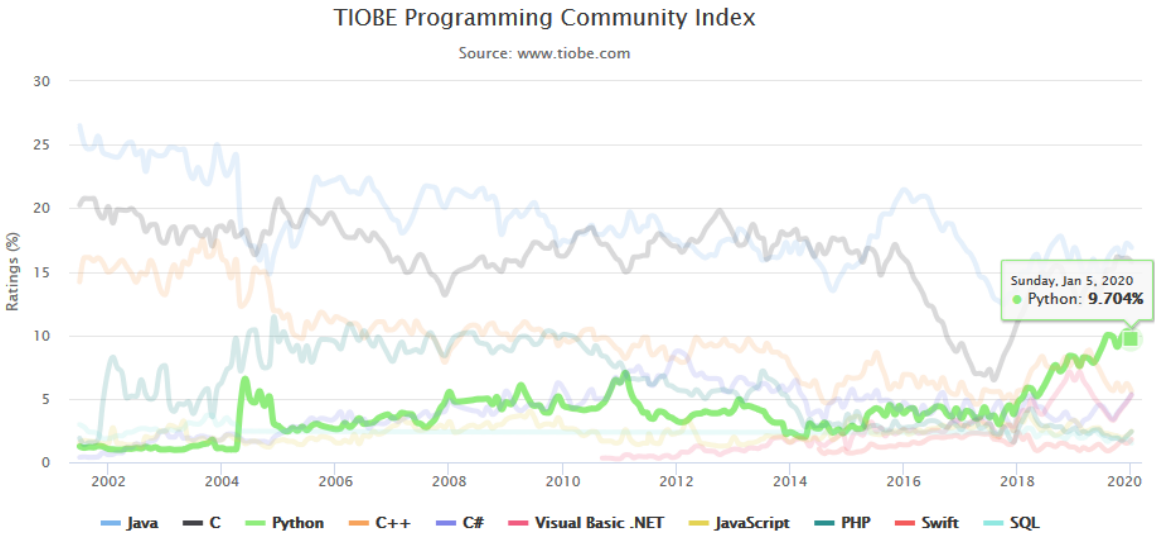
	创始人	时间	国家	主要应用
Python2				
Python3				
C				
C++				
C#				
Java				
JavaScript				
SQL				
PHP				
Ruby				

1.1.2.4. 各大编程语言排名信息

排行榜这种东西，总会具有争议性，但总体来说还是具有相当地参考价值。就好比关于国内高校的学术排名历来就有“Top1有两个，Top3有五个，Top5有十个“的梗一样，在IT界也是很容易一言不合就开架的。Anyway，我们就根据TIOBE的排名来看一下各大编程语言的排行情况，对此有意见或感兴趣的可以访问TIOBE官网：index | TIOBE - The Software Quality Company以及其它相关网站。

> TIOBE编程社区索引是编程语言受欢迎程度的指标。索引每月更新一次。评级基于全球熟练的工程师，课程和第三方供应商的数量。诸如Google，Bing，Yahoo！，Wikipedia，Amazon，YouTube和Baidu等流行的搜索引擎用于计算评分。重要的是要注意，TIOBE索引与*最佳*编程语言或*大多数代码行*所用的语言无关。

根据TIOBE 2020年1月份的摘要显示，每个人都认为Python将连续第二年成为TIOBE的年度编程语言。但这一次却是C语言凭借2.4%的年增长率获得了该奖项。为什么C语言仍然很热门？这种趋势背后的主要驱动力是物联网（IoT）和当今发布的大量小型智能设备。当将C应用于对性能至关重要的小型设备时，C会很出色；并且C相对来说还算容易学习，每个处理器都有一个C编译器。以下是目前TIOBE编程社区索引指数对比图。



此外，比起趋势图，用表格展示或许更直观一些，且看：

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
Java	**1**	2	**1**	2	3	-	-	-
C	2	**1**	2	**1**	**1**	2	**1**	**1**
Python	**3**	7	6	6	**23**	21	-	-
C++	4	4	3	3	2	**1**	2	12
C#	5	5	5	8	8	-	-	-
Visual Basic .NET	6	10	-	-	-	-	-	-
JavaScript	7	8	8	9	6	-	-	-
PHP	8	6	4	5	29	-	-	-
SQL	9	-	-	97	-	-	-	-
Objective-C	10	3	22	37	-	-	-	-
Lisp	31	18	16	12	14	5	3	2
Ada	35	29	25	15	15	6	4	3

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
Ada	55	29	25	15	15	6	4	5
Pascal	219	18	13	73	12	3	20	5

整体来看，1895年起C是第一名的位置，十年后C++可能由于STL的原因开始登顶，并开始与C语言并驾其驱。直到2010年JAVA开始后来居上。而Python则默默发力，一举从2000年排名20几直到现在稳居第三。我们可以简单理一下，编程语言的发展经历了机器语言、汇编语言和高级语言的过程，其中C语言是高级语言的集大成者与基石，在那个年代单机控制系统是编程的刚需，所以火热是正常的。后来开始进入了桌面时代，因此C++开始大显身手，并编写出了很多复杂的程序。接着进入了移动互联时代，JAVA因为其一次编译随处运行的特性迅速占领了整个嵌入式市场。而我们现在是什么时代呢？数据时代！对于海量数据的处理需求与人工智能的火热，催生了基于数据与应用的交互需求，因此Python这种脚本语言得以大显身手，并在其发展中不断演化，吸收其他编程语言的优秀思想与设计，实现了阶级跃升。

大胆预测一种较大概率存在的模型：底层用C，设备用JAVA，桌面用C++，之间的粘合与原型开发用Python，网页用JavaScript，苹果推自己的Swift，微软推自己的C#，等等。所以从适用性来看，C/C++、JAVA、Python的组合无意是强大的，当然了，背后还有一个语言与它们交互，那就是SQL/NoSQL。这就是我觉得较为稳定、主流的框架了。其他各个领域会有各自的主流语言，比如统计语言R/SAS，人工智能领域Scala/Go等。

1.1.3. How does python work and why is python designed like this?

1.1.4. How to choose, install and configure your IDE?

1.1.4.1. To Start: IDLE

1.1.4.2. For Data Science: Anaconda

1.1.4.3. For Professional Projects: PyCharm

1.1.4.4. For a faster coding experience: Sublime

1.1.5. Practice: programming using both CMD and GUI

1.2. CH02 - Python Basic Programming

1.2.1. Overview

1.2.2. Mutable Datatype

1.2.3. Changeable Datatype

1.2.4. Function and Structured Programming

1.2.5. Class and Object Oriented Programming

1.2.6. Module and Project Level Programming

1.2.7. Practice: recursive function application

1.3. CHO3 - Python Core Programming

1.3.1. JSON: re-recognize data structure

1.3.2. Practice: search for your favorite gitub repository