

第一章 Python 入门

不要重复发明轮子

1.1 为什么是 Python?

计算机系的同学一般都学过不止一门编程语言，比如 C/C++、Java 这些。那么学习这门课程，我们为什么还要再学习一门新的语言？这门新语言为什么是 Python？

回答这些问题，笔者认为可以从**开发效率**和**应用生态**两方面来考虑。首先，“智能”是应用，应用首要考虑的是开发效率。其次，既然“智能”是应用，就需要形成一个生态系统，大家分工合作、相互启发、相互促进。Python 语言符合这两方面的基本要求，面向对象、高级的数据结构（比如容器对象、列表、字典）能够有效提高开发效率；建立在开源基础之上的成熟生态系统（比如丰富的各种库和应用、完备的文档、活跃的全球开发社区）能够形成产业标准、集众智解难题。Google 的 Android 是一个很好的可类比的例子，应用层采用 Java（解释型、面向对象）作为开发语言，有效提高开发效率；同时基于开源构建了成熟的 Android 生态系统（涵盖硬件、操作系统、支撑软件、应用编程框架、丰富的应用），形成行业和产业标准，借助全球范围内的强大开发群体和社区不断向前发展。

Python 还具有直观的语法、方便和强大的文本处理、解释型语言便于调试这些优点，便于非计算机专业的广大的开发人员学习和使用。尤其是，近年来，随着机器学习（特别是深度学习）在各行各业的广泛和深入应用，Python 的热度逐年上升，我记得 Github 上的年度排名冲到了前 3，由此可见一斑，也从侧面说

明了 Python 的优势和对于大众的吸引力。

话说回来，任何事情都有两面性。Python 也有其**不足**之处，主要的一点是执行效率不够，对于很多对实时性有要求或者对成本敏感的实际应用不太适合。因此，为了提高执行效率，可以将性能瓶颈模块用 C/C++ 重写，如果这样做还不够，还可以整个用 C/C++ 重写。软件工程有两个基本思想：一个是增量式开发、一个是先跑起来再打磨。具体到这里，我们可以先用 Python 快速开发出应用系统（利用 Python 开发效率高的优势），然后对其性能进行评测，进而决定是局部优化还是整体重构，也可以基于增量式开发的思想，先局部优化，增量式进行，直到满足要求为止。

另外，对于执行效率没有过高要求的应用，采用强类型的 Python（比如 Cython、PyPy）也是一个选择。

Python 还有一个关键不足之处要特别引起注意：**用户级多线程**。操作系统课程里我们详细讲过内核级和用户级线程的区别，简单说来就是，用户级多线程可能只对应内核级的单线程，这意味着 Python 实际上可能是一个单线程应用，无法发挥出系统的并发和并行能力，从而严重制约执行效率和系统的资源利用率。幸运的是，这个问题可以通过采用内核级线程库来解决，比如 PyQT。Python 的生态优势在此得到了体现。

Python 有句名言：不要重复发明轮子。这是其设计理念的重要方面——复用，有现成的库先抓来用，有好的库何必自己又去低水平重复呢。本课程由于是本科层次的机器学习基础课程，因此有必要要求不能调用机器学习库（比如著名的 Scikit-learn），而要求自己从头写，这是一条**红线**。当然，其它的库尽管去用，也可以将自己的实现和机器学习库的实现进行比较，看看自己的实现是好是坏，

好在哪里，差在哪里，如何改进，我们鼓励这样做。