第一章

绪论

学习 Python 程序设计基础对商学院本科生培养计算思维能力至关重要，不仅能够增强学生的逻辑思维和问题解决能力，还能提升他们在未来商业世界中的竞争力和适应能力。

1.1 计算思维

计算思维（Computational Thinking, CT）是一种通过运用计算机科学的基本概念来解决问题、设计系统和理解人类行为的过程. 它不仅是编程技能的一部分，更是一种通用的思维方式，可以应用于各个领域。计算思维的核心包括以下几个方面：

1. **分解（Decomposition）：**将复杂问题分解为更小、更易于处理的部分，从而使问题更加可理解和可管理。

2. **模式识别（Pattern Recognition）：**识别问题中的模式和规律，找出相似性，从而简化解决方案的开发过程。

3. **抽象（Abstraction）：**提取问题的关键信息，忽略不必要的细节，以简化问题的表示和解决过程。

4. **算法思维（Algorithmic Thinking）：**设计一个明确的、有步骤的解决方案（算法），使其可以被计算机执行，或为解决问题提供一种系统化的方法。

这些技术和思维过程帮助人们用一种结构化的方式来分析问题、设计解决方案和进行预测。计算思维在教育中被广泛推广，因为它不仅有助于学生掌握编程和计算机科学的基本知识，还可以培养他们的逻辑思维和解决问题的能力。

案例:规划一次度假

假设你正在计划一次家庭度假,这个过程可以应用计算思维的四个核心方面:

1. **分解(Decomposition)**:首先,将度假计划分解为更小的任务。例如,你需要选择目的地、确定预算、预订住宿和安排交通。每一个任务都是一个独立的子问题,可以分别解决。
2. **模式识别(Pattern Recognition)**:在分解任务后,下一步是识别模式。例如,你可能注意到,每次度假都涉及类似的步骤,如选择住宿和交通工具。这些模式帮助你更有效地进行计划,因为你可以利用以前的经验来简化当前的任务。
3. **抽象(Abstraction)**:在规划过程中,你需要忽略不相关的细节,专注于关键因素。例如,在选择目的地时,你可能不需要考虑所有可能的天气条件,而只需要关注主要的气候类型和旅游季节。这种简化使得问题更加可控。
4. **算法思维(Algorithmic Thinking)**:最后,根据分解、模式识别和抽象的结果,制定一个明确的步骤来完成度假计划。这包括决定什么时候出发、如何预订机票和酒店,以及每天的活动安排。这些步骤应当详细到足以让任何人根据这些步骤顺利完成计划。

总结

通过应用计算思维的四个核心方面,度假规划变得系统化和高效。首先,通过分解,将复杂问题拆分为可管理的部分;然后,通过模式识别,利用已知的解决方法加速当前任务;通过抽象,聚焦于重要信息而忽略无关细节;最后,通过算法思维,制定一个可执行的计划。

培养计算思维对于商科专业的重要性

计算思维是一种系统性解决复杂问题的方法,通过分解问题、识别模式、抽象重要信息并设计算法解决方案来处理各类挑战。这种思维方式对商科专业的学习和未来职业发展具有重要意义。

- **提升问题解决能力**:在商业环境中,经常需要应对复杂的业务问题,如市场分析、财务建模和供应链管理等。通过培养计算思维,可以学会如何将复杂问题分解成更小的部分,找到问题中的模式,并设计出有效的解决方案。这种技能不仅适用于技术领域,还广泛应用于市场营销、会计、国际商务和大数据管理等多个领域。
- **数据分析和决策制定**:在大数据时代,商业决策越来越依赖于对大量数据的分析和解读。计算思维能够提升数据处理能力,使得在数据中识别模式、预测市场趋势或评估财务风险时更加高效和准确。这种能力对于进行数据驱动的决策至关重要。
- **创新与自动化**:计算思维不仅帮助解决现有问题,还能激发创新。通过理解和应用算法设计,可以开发新的工具和方法来自动化重复性的业务流程,提高工作效率和创造力。这在快速变化的商业环境中保持竞争力至关重要。
- **跨学科应用**:计算思维的核心方法,如分解、抽象和模式识别,能够在不同的学科和领域之间迁移和应

用。例如，可以将这些方法用于财务分析、物流优化以及市场营销策略的设计，从而在多种场景下提升分析能力和问题解决能力。

培养计算思维能力不仅有助于在学术领域取得更好的成绩，还能显著增强在职业生涯中的竞争力。掌握这种思维方式的人能够更好地应对复杂的商业挑战，并推动创新和效率的提升，在未来职场中取得成功。

1.2 计算机与计算机语言

1.2.1 计算机

计算机是能够被编程以自动执行一系列算术或逻辑操作的机器。现代数字电子计算机能够执行被称为程序的通用操作集，这些程序使计算机能够完成多种任务。计算机系统可以指包含完成全面操作所需的硬件、操作系统、软件和外部设备的完整计算机，或者指多个相互连接并协同工作的计算机，例如计算机网络或集群。

计算机的发展历史

早期计算设备：计算的历史可以追溯到古代的算盘，这是一种用于基本计算的手动工具。在 19 世纪初，一些机械装置被开发出来用于自动执行长时间、繁琐的任务。例如，查尔斯·巴贝奇在 1820 年代设计的差分机被认为是第一台机械计算机。

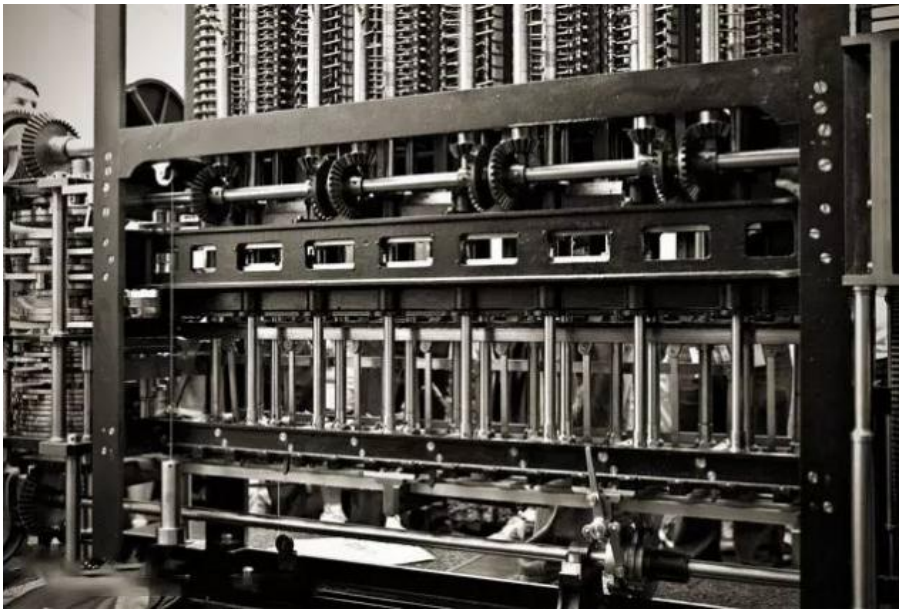

图 1.1: 差分机

电子计算机的出现：20 世纪早期，电子计算设备开始出现。1930 年代，艾伦·图灵提出了图灵机的概念，这为现代计算理论奠定了基础。1940 年代，电子数值积分计算机（ENIAC）作为第一台电子通用计算机问世，标志着电子计算机时代的开始。

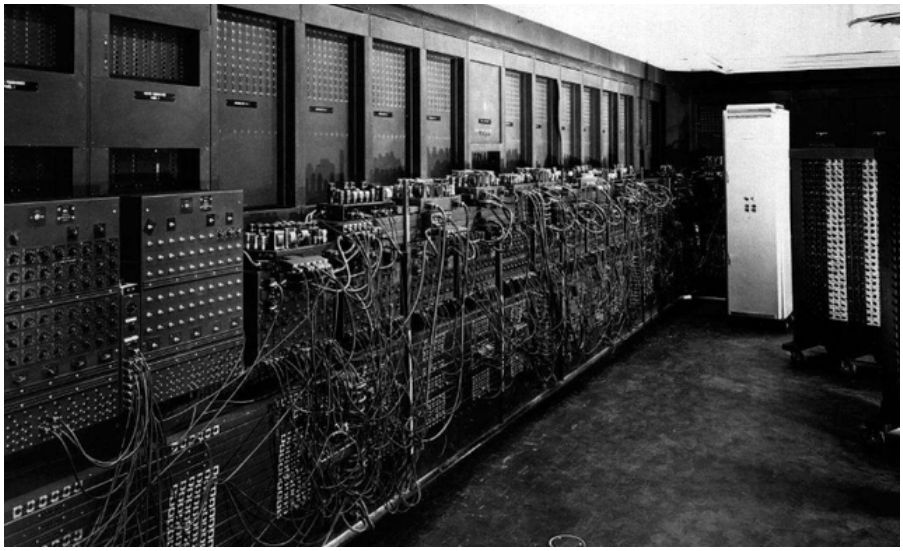

图 1.2: 电子数值积分计算机（ENIAC）

计算机的演变：20 世纪中期，计算机技术迅速发展。1947 年，贝尔实验室发明了第一款实用的晶体管，这项技术使得计算机变得更加紧凑和高效。1950 年代末和 1960 年代初，集成电路的发明使得微型计算机的出现成为可能，这些计算机在 1970 年代迎来了革命性的发展。

图 1.3: 微型计算机

个人计算机和互联网：1970 年代末至 1980 年代初，个人计算机（PC）开始进入市场，成为家庭和办公室的常见设备。20 世纪末和 21 世纪初，互联网的普及进一步推动了计算机的发展，使得全球范围内的信息交换和通信成为可能。

现代计算机：现代计算机具有强大的处理能力和多样化的功能，广泛应用于各个领域，从科学研究到日常生活。随着技术的进步，计算机正在不断向更高的速度、更大的存储容量和更高的集成度发展，甚

图 1.4: 个人计算机

至涉及到量子计算等前沿领域。

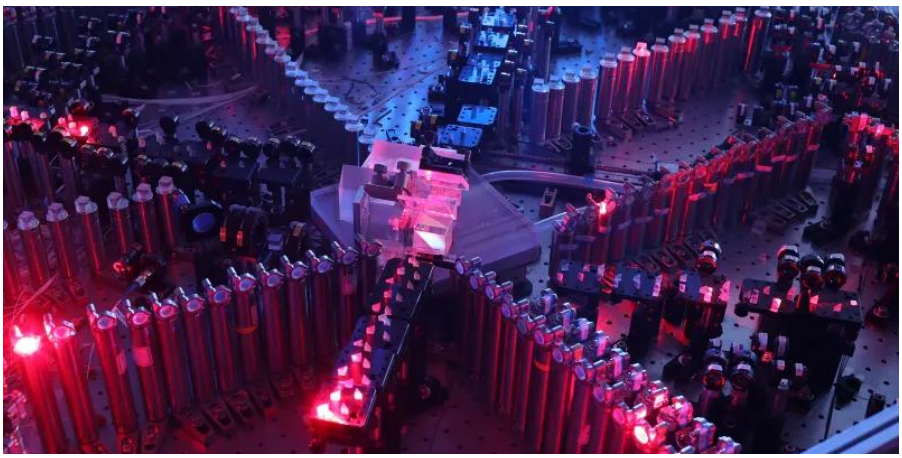

图 1.5: “九章”量子计算机

1.2.2 计算机语言

计算机语言是一种用于向计算机传达指令的符号系统。通过这些语言，人们可以编写计算机程序，这些程序告诉计算机如何执行特定的任务。计算机语言有许多种类，每种语言都有自己的语法和结构。

常用的计算机语言及其应用场景

Python：Python 是一种通用的编程语言，广泛用于数据分析、机器学习、人工智能、web 开发等。它的语法简单易学，适合初学者和专业开发者。

JavaScript：JavaScript 是一种主要用于前端开发的语言，常用于构建动态和交互式网页。它也是一种重要的全栈开发语言，因为它可以与 Node.js 结合用于后端开发。

图 1.6: 常用编程语言

Java：Java 是一种面向对象的编程语言，广泛用于企业级应用、安卓移动应用和大型系统开发。它以其稳定性和跨平台特性著称。

C++：C++ 是一种高性能的编程语言，常用于系统编程、游戏开发、机器人和科学计算。它提供了丰富的功能和灵活性，但学习曲线较陡。

R：R 是一种专门用于统计分析和数据可视化的语言，特别受数据科学家和统计学家的欢迎。它具有丰富的统计模型和图形库。

SQL：SQL（结构化查询语言）用于管理和操作关系数据库，是处理大规模数据集和进行数据分析的重要工具。

Swift：Swift 是苹果公司开发的一种编程语言，主要用于 iOS 和 macOS 应用开发。它语法简洁，易于学习，适合开发移动应用程序。

PHP：PHP 是一种服务器端脚本语言，常用于开发动态网页和 web 应用。它与 HTML 和数据库集成良好，广泛用于创建内容管理系统（如 WordPress）。

计算机语言的特点及与自然语言的不同之处

计算机语言和自然语言是两种用于交流的语言形式，但它们在本质、结构和使用上有显著的区别。

精确性和确定性：计算机语言是高度精确的，用于向计算机传达明确的指令。这些语言的语法和语义非常严格，必须完全按照规定的规则使用。如果代码中有任何语法错误，例如缺少一个引号，程序就会无法运行。而自然语言的语法规则较为灵活，即使语法不完全正确，人们通常仍能理解意思。

语法和语义：计算机语言的语法和语义固定不变，专门用于消除歧义并确保计算机能准确执行任务。自然语言的语法和语义则因上下文、文化背景等因素而异，具有多重解释的可能性，能够表达复杂的情感和抽象概念。

学习与习得：自然语言通常通过浸入式学习环境被人们从小习得，并不需要专门的语法学习。而计算机语言通常需要通过专门的教育和练习才能掌握，涉及对其语法、结构和编程环境的深入学习。

表达能力：自然语言具有丰富的表达能力，可以通过比喻、隐喻、情感等多种方式来传达复杂和抽象的思想。相比之下，计算机语言的表达能力有限，主要用于执行明确的计算任务和操作，缺乏对情感和复杂人类思想的表达能力。

结构和抽象：计算机语言通常使用固定的词汇和严格的语法规则来定义操作，而自然语言是动态的，词汇和用法会随时间演变和发展。计算机语言提供的抽象工具，旨在让程序员能够更好地控制计算机硬件，而自然语言的抽象通常用于传递高层次的概念和隐喻。

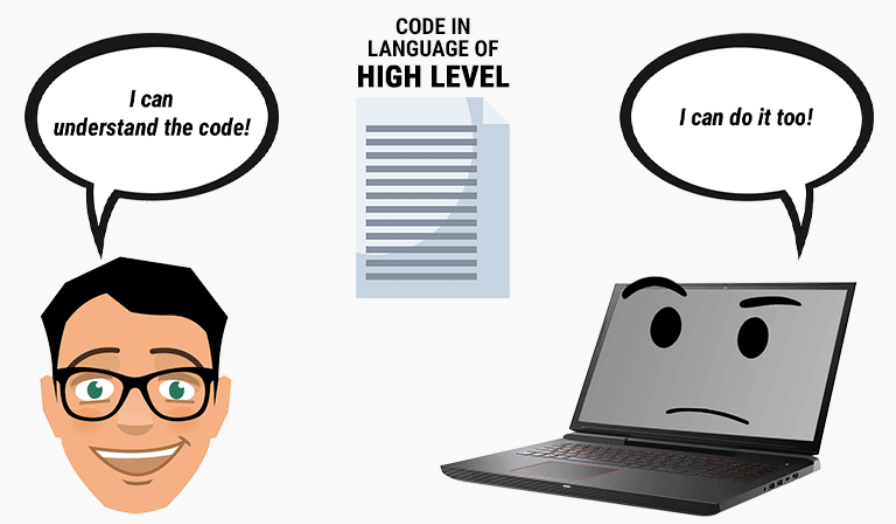

图 1.7: 人通过编程语言与计算机交互

静态语言与脚本语言

静态语言通常使用编译器并进行类型检查以确保代码的安全性和性能。而脚本语言往往使用解释器，具有动态类型，可以更方便地进行快速开发和迭代。

静态语言（Static Languages）：静态语言通常是指在编译时需要明确类型的编程语言。变量的类型在编写代码时就已经确定，并且在整个程序的生命周期中不会改变。这种类型的语言在编译阶段会进行类型检查，这有助于在代码运行之前捕获潜在的错误。例如，C、C++ 和 Java 都是典型的静态语言。

脚本语言（Scripting Languages）：脚本语言是一种解释执行的编程语言，通常用于自动化任务、处理数据或在 web 开发中生成动态内容。与静态语言不同，脚本语言通常具有动态类型，这意味着变量

的类型是在运行时确定的。脚本语言一般不需要预编译为机器码，而是通过解释器逐行执行。例如，Python、JavaScript 和 PHP 都是常见的脚本语言。

编译与解释

编译和解释是两种不同的程序执行方式。编译器在执行之前将整个源代码一次性转换为机器码，而解释器逐行执行代码。编译语言通常具有更好的性能，而解释语言则更具灵活性和开发速度。

编译（Compilation）：编译是将高级编程语言代码转换为机器码的过程。编译器会在程序运行之前将整个源代码翻译为机器可执行的文件。编译后的程序通常执行速度更快，因为它们直接运行在硬件上，没有解释的开销。然而，这也意味着每次代码更改后都需要重新编译。典型的编译语言包括 C、C++ 和 Rust。

解释（Interpretation）：解释是逐行读取和执行源代码的过程。解释器在代码运行时实时翻译和执行代码，这使得开发者能够快速测试和修改代码，因为不需要在每次修改后重新编译整个程序。尽管解释器灵活且便于开发，但解释执行通常比编译执行的效率低，因为每次执行时都需要重新翻译代码。常见的解释语言包括 Python、Ruby 和 JavaScript。

编程的基本原则

编写程序的主要原则是确保代码简洁、可读、易于维护，并能有效地执行任务。以下是一些核心编程原则，这些原则有助于开发人员编写高质量的代码，减少错误，提高代码的可读性和可维护性：

- **KISS（Keep It Simple, Stupid!）**：保持代码简单清晰，避免过度复杂化。简化代码有助于更容易地理解和维护。
- **DRY（Don't Repeat Yourself）**：避免代码重复。通过复用代码块，减少冗余，提高代码的可维护性和效率。
- **YAGNI（You Aren't Gonna Need It）**：只为当前需求编写代码，不为可能的将来功能编写代码，避免过度设计。
- **单一职责原则（Single Responsibility Principle）**：每个模块或函数应只负责一个特定功能，这样可以减少耦合，提高代码的可维护性。
- **文档化（Document Your Code）**：编写清晰的代码注释，帮助其他开发者理解代码的意图和功能，尤其在团队合作中尤为重要。
- **组合优于继承（Composition Over Inheritance）**：在面向对象编程中，优先使用组合而非继承来实现代码复用，以提高代码的灵活性和可维护性。

图 1.8: Python 之父-Guido van Rossum

1.2.3 Python 编程语言简介

Python 是一种高级、通用的编程语言，由 Guido van Rossum 于 1991 年首次发布。它以简洁、易读的语法和广泛的应用领域而闻名。

Python 的设计理念强调代码的可读性，使开发者能够用更少的代码表达复杂的概念，以减少错误并加快开发速度。Python 具有以下优点：

易于学习和使用：Python 的语法接近自然语言，非常直观，这使得它成为编程初学者的理想选择。它简洁的代码结构使得程序更容易编写和维护。Python 也是一种解释型语言，这意味着代码可以逐行执行，方便调试和即时测试。

广泛的应用领域：Python 是一种通用语言，适用于多种应用场景，包括数据科学、人工智能、机器学习、Web 开发、自动化脚本、数据分析等。其灵活性使其成为跨领域项目的理想选择。

丰富的库和社区支持：Python 拥有庞大的标准库和第三方库，这些库为图形处理、数据分析、机器学习、Web 开发等提供了强大的支持。这使得开发者可以快速构建复杂的功能，而不需要从零开始编写代码。此外，Python 拥有一个庞大且活跃的社区，这为学习者和开发者提供了丰富的资源和支持。

跨平台兼容性：Python 是平台独立的，可以在不同的操作系统（如 Windows、Linux、MacOS）上运行。这种兼容性使得 Python 非常适合开发需要跨平台部署的应用。

动态类型和自动内存管理：Python 是动态类型语言，变量类型在运行时确定。这种特性使得 Python 更灵活，能更快地开发和测试代码。Python 还具有自动内存管理功能，开发者不需要手动管理内存分配，这减少了编程中的常见错误。

Python 是一种功能强大且用途广泛的编程语言，其简洁的语法、强大的库支持以及跨平台的兼容性使得它成为商科学生学习编程的理想选择。通过学习 Python，商科学生能够掌握数据分析、自动化以及开发应用程序的技能，从而提升其在现代商业环境中的竞争力。

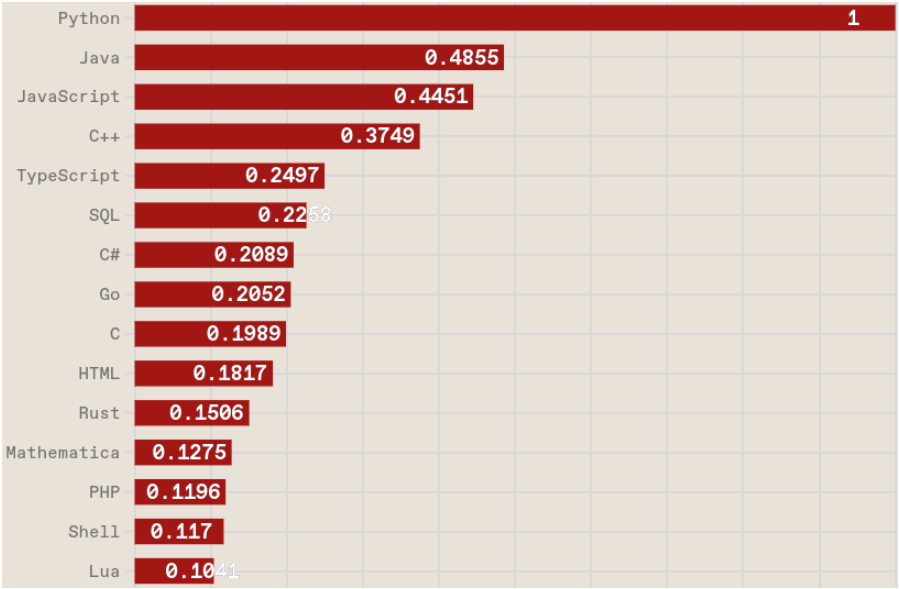

图 1.9: 2024 年度 IEEE Spectrum 编程语言排行榜

商科学生为什么要学习 Python

Python 是一种非常适合商科学生学习的编程语言，通过学习 Python，商科学生不仅能够提升数据分析能力和自动化水平，还能获得在现代商业环境中所需的技术技能，为未来的职业发展打下坚实的基础。

易于学习和使用：Python 以其简单易懂的语法著称，这使得它成为编程初学者的理想选择。Python 的语法接近自然语言，易于阅读和理解，减少了初学者的学习曲线。这对商科学生来说尤为重要，因为他们通常缺乏计算机科学的背景知识，因此使用 Python 可以更快上手编程。

广泛的应用领域：Python 是一种通用的编程语言，广泛应用于数据分析、机器学习、自动化和 web 开发等领域。这些应用对商科学生尤其有用，例如在数字经济和大数据管理中，Python 可以用来处理和分析大规模数据集，帮助做出数据驱动的决策；在会计学和国际商务中，Python 可以用于自动化财务报表生成和市场趋势分析。

强大的数据分析能力：Python 拥有丰富的库和工具，如 NumPy、Pandas、Matplotlib 和 Seaborn 等，可以用于数据清理、处理和可视化。这些工具帮助商科学生进行复杂的数据分析和建模，从而更好地理解商业数据和市场趋势，支持他们在商业决策中应用数据分析和统计方法。

自动化和效率提升：通过编写 Python 脚本，商科学生可以自动化许多日常任务，例如数据收集、报告生成和重复性的计算操作。这不仅提高了工作效率，还使他们能够专注于更具战略性的任务和分析。

在商业中的广泛应用：许多大型企业和机构，如阿里巴巴、腾讯和小红书，都在使用 Python 进行各种类型的开发和数据分析。这表明 Python 在商业环境中得到了广泛的认可和应用，掌握 Python 技能的商科学生在求职市场上更具竞争力，能够胜任多种技术和分析岗位。

图 1.10: Python 广泛应用于商业数据分析

Python 程序设计的基本原则

Python 编程语言的设计深受一组被称为“Python 之禅”（The Zen of Python）的哲学原则的影响。这些原则由 Python 社区的早期成员 Tim Peters 提出，它们总结了 Python 的设计理念，强调代码的可读性和简洁性。

- **美胜于丑（Beautiful is better than ugly）**：在 Python 中，代码的美观和清晰度非常重要。优美的代码不仅更容易理解和维护，而且有助于减少错误。
- **显式优于隐式（Explicit is better than implicit）**：Python 鼓励显式的表达方式，使代码的功能和行为更加清晰，不需要依赖额外的上下文理解。
- **简单优于复杂（Simple is better than complex）**：简单的解决方案通常比复杂的更好。Python 提倡使用简单的代码来解决问题，并鼓励开发者避免不必要的复杂性。
- **复杂优于晦涩（Complex is better than complicated）**：在必须处理复杂问题时，代码应该尽量保持清晰和易于理解，而不是让代码变得晦涩难懂。
- **扁平优于嵌套（Flat is better than nested）**：层次结构越少，代码就越清晰。Python 建议避免过度嵌套，因为嵌套的层次会增加代码的复杂性和难度。
- **稀疏优于密集（Sparse is better than dense）**：代码应该是稀疏而非密集的，以便于阅读和理解。紧凑的代码可能会让其他开发者难以解读。
- **可读性很重要（Readability counts）**：Python 非常重视代码的可读性。好的代码不仅是为了机器运行，也是为了让人类开发者能够轻松理解和维护。
- **不要为了特例而破坏规则（Special cases aren’t special enough to break the rules）**：遵循规则可以让代码更加一致和可预测，除非有很强的理由，否则不应该为特定情况破坏既定的规则。