

项目 1

Python 概述与开发环境搭建



学习目标

- 了解 Python 发展历程和不同程序设计语言的代码形式；
- 掌握 Python 环境搭建方法。



素养目标

- 培养严谨规范的开发环境配置习惯；
- 提升跨平台软件适配与问题排查能力；
- 建立对开源生态的认知与社区协作意识；
- 形成代码可读性与简洁性的编程理念；
- 增强技术版本迭代的适应能力。



项目重难点

项目内容	工作任务	建议学时	技能点	重难点	重要程度
Python 概述 与开发环境 搭建	任务 1.1 Python 全景探秘——从语言本质到生态魅力	1	无	Python 语言特性	★
	任务 1.2 Python 开发环境搭建与测试	1	Python 开发环境搭建、测试	安装失败原因查找并解决	★★



任务 1.1 Python 全景探秘——从语言本质到生态魅力

实施任务单

任务编号	1.1	任务名称	Python 全景探秘——从语言本质到生态魅力
任务简介	初学者能够概述 Python 语言特点		
设备环境	Windows 系统、Python 3.8 及以上版本环境、IDLE 或任意代码编辑器		
任务难度	初级	实施日期：_____年_____月_____日	
任务要求	口述 Python 特点、Python 的生态魅力		

1.1.1 Python 程序语言概述

1. Python 程序语言简介

Python 是一种广泛使用的解释型、高级和通用的编程语言，支持多种编程范型，包括结构化、过程式、反射式、面向对象和函数式编程。它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，并且其本身拥有一个巨大而广泛的标准库。它的语言结构和面向对象的方法，旨在帮助程序员编写逻辑清晰的代码。

2. Python 发展简史

Python 由荷兰程序员吉多·范罗苏姆（Guido van Rossum）于 1989 年圣诞节期间创建。当时，他在荷兰国家数学和计算机科学研究所（CWI）工作，希望开发一种易读、简洁且功能强大的编程语言，改进 ABC 语言的复杂性。吉多以英国喜剧团体“Monty Python”命名该语言，体现其轻松的设计理念。1991 年，Python 0.9.0 发布，包含函数、类、异常处理等核心特性，奠定了语言基础。

2000 年，Python 2.0 发布，引入了列表推导式、垃圾回收和更好的 Unicode 支持，显著提升了语言的实用性。2008 年，Python 3.0 发布，旨在修复设计缺陷，如统一字符串编码为 Unicode，并优化语法。然而，Python 3 不完全向后兼容，导致社区分裂，Python 2 和 3 并行维护多年。Python 2 于 2020 年 1 月 1 日正式停止支持，标志着向 Python 3 的全面过渡。此后，Python 3 不断优化，发布多个版本，增强性能和功能。

Python 的成功离不开其开源社区。20 世纪 90 年代，Python 用户组和 PyCon 会议逐渐兴起，促进了技术交流。2001 年，Python 软件基金会（PSF）成立，负责语言的开发、维护和推广。Python 的标准库丰富且强大，涵盖文件处理、网络通信等功能。第三方库的繁荣进一步推动了 Python 的普及。例如，NumPy、Pandas 用于数据分析，Django、Flask 用于 Web 开发，TensorFlow、PyTorch 用于机器学习。这些工具使 Python 成为多领域首选语言。

Python 因其简洁的语法和易学性，在教育、科研和工业领域广受欢迎。它在数据科学、人工智能、Web 开发、自动化脚本等领域表现出色。例如，工具 Jupyter Notebook 简化了数据分析流程，而框架 Scikit-learn 和 Keras 加速了 AI 应用开发。根据该指数（该指数是衡量编程语言流行度的指标），Python 编程语言长期位居编程语言排行榜前三，尤其在 2020 年后多次登顶。该指数每月更新一次，评级基于全球熟练工程师的数量、课程和第三方供应商。图 1.1 所示为 2026 年 3 月 TIOBE 语言排行榜，图 1.2 所示为 Python 使用走势图。行业内习惯称 Python 为胶水语言，其“可读性优先”的设计原则，吸引了从初学者到专业开发者的广泛用户。

Mar 2026	Mar 2025	Change	Programming Language		Ratings	Change
1	1			Python	21.25%	-2.59%
2	4	▲		C	11.55%	+2.02%
3	2	▼		C++	8.18%	-2.90%
4	3	▼		Java	7.99%	-2.37%
5	5			C#	6.36%	+1.49%
6	6			JavaScript	3.45%	-0.01%
7	9	▲		Visual Basic	2.50%	-0.02%
8	8			SQL	2.00%	-0.57%
9	16	▲		R	1.88%	+0.94%
10	10			Delphi/Object Pascal	1.80%	-0.36%

图 1.1 TIOBE 语言排行榜（2026 年 3 月）

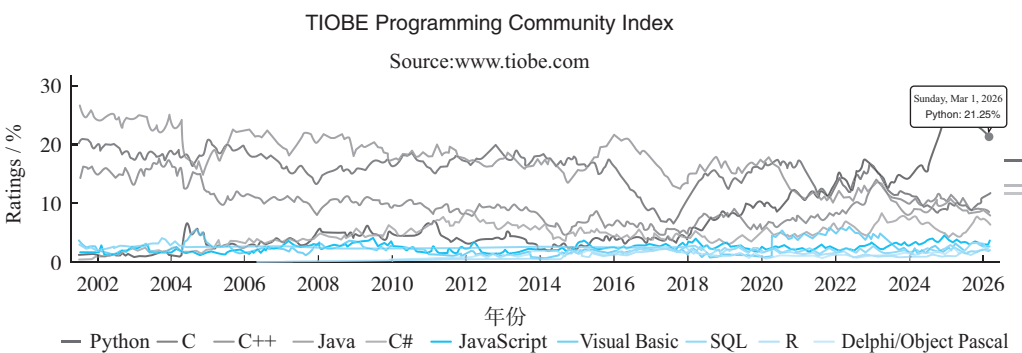


图 1.2 Python 使用走势图

如今，Python 仍在快速发展，版本更新注重性能优化（如 3.11 版本引入了更快的解释器）。2024 年 10 月，Python 3.13 引入了从 PyPy 引进的新交互式解释器，并实验性地支持了即时编译器。Python 3.14 正式支持了自由线程构建选项，其跨平台性和灵活性使其适应云计算、物联网等新兴领域。未来，Python 有望继续引领编程语言的发展，成为技术创新的重要基石。

3. Python 的特点

Python 的主要特点见表 1.1。

表 1.1 Python 的主要特点

特 点	描 述
简洁与可读性	Python 语法清晰简洁，强调代码可读性，使用缩进定义代码块，减少符号冗余
动态类型	无须声明变量类型，运行时自动推断，支持灵活的变量赋值和类型转换
解释型语言	代码逐行执行，无须编译，适合快速开发和调试，支持交互式编程（如 REPL）
跨平台性	可在 Windows、Linux、macOS 等多平台运行，代码高度可移植
丰富的标准库	提供大量内置模块和函数，覆盖文件处理、网络通信、数据处理等，减少重复开发
强大的第三方生态	支持 NumPy、Pandas、Django、TensorFlow 等库，广泛应用于数据科学、AI、Web 开发
面向对象与函数式	支持面向对象编程（类、继承）和函数式编程（lambda、列表推导式），灵活多变
异常处理	提供完善的异常处理机制，通过 try-except 捕获和处理错误，提升代码健壮性
社区支持与开源	拥有活跃的社区和 Python 软件基金会（PSF），开源生态推动持续更新和扩展

4. Python 技术特性

Python 有九项重要的技术特性，每项特性都包含描述和简短的代码示例，清晰展示其功能和应用。

1) 动态类型与弱类型

Python 无须显式声明变量类型，运行时自动推断变量的数据类型（如整数、字符串等），允许同一变量在不同时刻存储不同类型的数据。这种灵活性简化了代码编写，但要求开发者注意类型相关的潜在错误，特别在大型项目中。

```
a = 5          # 整数
a = "Hello"    # 动态转换为字符串
print(a)       # 输出: Hello
```

2) 解释型执行

Python 是一种解释型语言，代码由解释器逐行执行，无须预先编译为机器码，支持交互式编程环境（如 REPL）。这使开发者可以快速测试和调试代码，特别适合原型开发和教学，但执行速度相较编译型语言稍慢。

```
# 在 Python 解释器中直接运行
for i in range(3):
    print(i)    # 输出: 0, 1, 2
```

3) 垃圾回收机制

Python 通过引用计数和垃圾回收器自动管理内存，开发者无须手动分配或释放内存。这降低了内存泄漏的风险，但某些复杂场景可能需要手动调用垃圾回收函数以优化性能。



```
import gc
obj = [1, 2, 3]
del obj                                # 删除对象
gc.collect()                          # 强制垃圾回收
```

4) 多范式编程

Python 支持多种编程范式，包括面向对象（通过类和继承）、函数式（通过 lambda 函数和列表推导式）以及过程式编程。这种灵活性允许开发者根据项目需求选择合适的开发方式，提升代码的可维护性和表达力。

```
class Dog:
    def bark(self):
        return "Woof!"                # 函数式
square = lambda x: x**2
print(Dog().bark(), square(3))        # 输出: Woof! 9
```

5) 模块化与扩展性

Python 提供丰富的标准库，涵盖文件操作、网络通信等领域，同时支持模块化开发，便于代码组织和复用。此外，通过 C/C++ 扩展（如 CPython 的 C 模块），开发者可以优化性能或集成低级功能。

```
import math
print(math.sqrt(16))                  # 使用标准库模块，输出: 4.0
```

6) 异常处理机制

Python 提供强大的异常处理机制，通过 try-except-finally 结构捕获和处理运行时错误，支持细粒度的错误类型管理。这提高了代码的健壮性，尤其在处理文件、数据库或网络操作时能有效应对异常情况。

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")    # 输出: Cannot divide by zero!
```

7) 跨平台字节码

Python 代码通过 CPython 解释器编译为平台无关的字节码，可在 Windows、Linux、macOS 等系统上运行，无须修改代码。这种跨平台特性确保了代码的高度可移植性，适合开发跨系统应用。

```
# 同一代码在 Windows/Linux/macOS 系统运行
print("Hello, World!")                # 输出: Hello, World!
```

8) 并发与并行支持

Python 通过 threading（多线程）、multiprocessing（多进程）和 asyncio（异步 I/O）支持并发和并行编程，适合高并发场景，如网络爬虫或服务器开发。尽管全局解释器锁

(global interpreter lock, GIL) 限制了多线程的 CPU 并行性能, 但是 multiprocessing 和 asyncio 提供了替代方案。

```
import asyncio
async def say_hi():
    await asyncio.sleep(1)
    print("Hi")
asyncio.run(say_hi())           # 输出: Hi
```

9) Unicode 支持

Python 3 默认使用 Unicode 编码, 支持多语言字符处理 (如中文、阿拉伯文等), 适合开发国际化应用。相比 Python 2 的 ASCII 默认编码, Unicode 的支持显著提高了文本处理的灵活性和兼容性, 中文也可以作为变量名。

```
text = "你好, 世界!"
print(text)                     # 输出: 你好, 世界!
中='a'
print(中)                       # 输出: a
```

5. PEP20

PEP20 是编写 Python 程序的设计原则, 在 >>> 提示符后面输入 import this 就能输出该设计原则。

```
import this                     # 输出 Python 之禅 (设计原则)
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

翻译如下：

优美胜于丑陋（ Beautiful is better than ugly ）；
明确胜于隐晦（ Explicit is better than implicit ）；
简单胜于复杂（ Simple is better than complex ）；
可读性至上（ Readability counts ）；
.....

1.1.2 何以选择 Python

以输出“Hello World！”为例，理解不同程序语言所用的代码。

1. C 语言的 Hello World 程序样例

```
#include <stdio.h>
int main(void) {
    printf("Hello world!\n");
    return 0;
}
```

2. Java 语言的 Hello World 程序样例

```
public class HelloWorld {
    public static void main(String[] args){
        System.out.println("Hello World!");
    }
}
```

3. C# 语言的 Hello World 程序样例

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

4. C++ 语言的 Hello World 程序样例

```
#include <iostream>
int main()
{
    std::cout<<"Hello World!"<<endl;
    return 0;
}
```



5. Python 语言的 Hello World 程序样例

```
print("Hello World!")
```

从以上五种编程语言实现同一输出结果的代码示例来看，Python 语言在简洁性与可读性方面具有压倒性的优势。

（1）极简的语法结构。相较于 C、C++、Java 和 C# 需要定义复杂的程序框架（如包含头文件、声明类、编写主函数 `main()` 以及处理返回值或系统库调用），Python 仅需一行 `print()` 语句即可实现相同的输出结果。这充分印证了 Python “减少符号冗余”和“简洁”的特性。

（2）降低开发门槛。Python 这种“电池内置”且“易学”的特性，使开发者能够避开烦琐的底层语法约束，将精力集中在解决逻辑问题上。

（3）高效的开发体验。作为一种解释型语言，其简单的语法配合交互式执行能力，使其成为原型开发、教育科研以及数据科学等领域的首选工具。



Python 启
程——开
发环境与
首个程序

任务 1.2 Python 开发环境搭建与测试

实施任务单

任务编号	1.2	任务名称	Python 环境搭建与测试
任务简介	初学者能够自行安装 Python 开发环境并进行简单测试		
设备环境	Windows 系统、Python 3.8 及以上版本环境、IDLE 或任意代码编辑器		
任务难度	初级	实施日期：_____ 年 _____ 月 _____ 日	
任务要求	1. 下载安装包 2. 安装搭建环境 3. 环境测试		

1.2.1 Python 开发环境搭建

Python 自带的开发环境比较简单，适用于 Python 的集成开发环境（integrated development environment, IDE）软件。除了标准二进制发布包所附的 IDLE，还有许多其他选择。其中有些软件设计有语法着色、语法检查、运行调试、自动补全、智能感知等便利功能。由于 Python 的跨平台性，这些软件往往也具备适用各种操作系统的多个版本或一定的移植性。

可以在 Python 官方网站（图 1.3）直接下载 IDLE 安装包（图 1.4）。本书基础部分采用 Python 官方的 IDLE 3.12.0 版本（图 1.5），后面高阶部分根据具体应用采用其他编辑器。

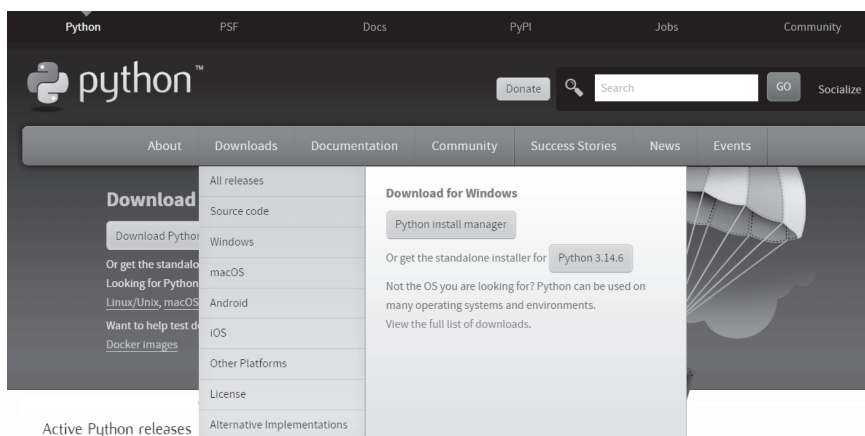


图 1.3 Python 官方网站



图 1.4 Python 安装界面

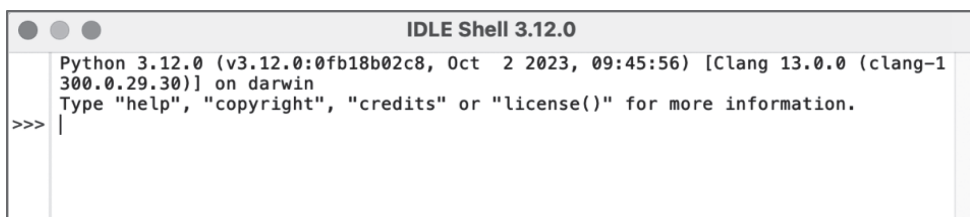


图 1.5 IDLE 界面

PyCharm（专业编辑器）是一个用于计算机编程的 IDE，主要用于 Python 语言开发，由捷克公司 JetBrains 开发，拥有 Windows、macOS 和 Linux 版本。社区版在 Apache 许可证下发布，提供代码分析、图形化调试器，集成测试器、集成版本控制系统，并支持使用 Django 进行网页开发。PyCharm 安装过程比较简单，下载安装包可以访问 PyCharm 官网，下载免费社区版。打开 PyCharm，单击新建项目，设置项目路径。PyCharm 的初始界面如图 1.6 所示。



图 1.6 PyCharm 的初始界面

1.2.2 Python 环境测试

【例 1.1】编写第一个 Python 程序。

任务描述：输出问候语“Hello World，欢迎开启 Python 之旅！”

实施步骤如下。

(1) 打开编辑器。打开 IDLE 后，出现一个增强的交互解释器窗口。它具有比基本的命令行 shell 更多的功能。

(2) 编写代码。在编辑器 >>> 提示符后，输入以下内容：

```
print("Hello World，欢迎开启 Python 之旅!")
```

运行结果：

```
Hello world，欢迎开启 Python 之旅！
```

测试结果如图 1.7 所示。

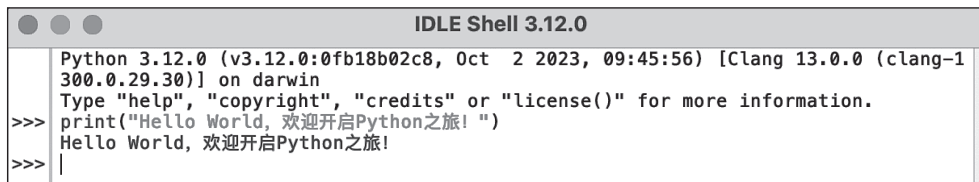


图 1.7 第一个 Python 程序测试结果

代码解释：print() 是输出函数，括号内文字需用英文引号括起来。

以上是利用 IDLE 进行简单的交互界面程序测试。另一种测试是文件形式的程序测试，