概述

"Python 库"中包含了几种不同的组件。

它包含通常被视为语言“核心”中的一部分的数据类型，例如数字和列表。对于这些类型，Python语言核心定义了文字的形式，并对它们的语义设置了一些约束，但没有完全定义语义。（另一方面，语言核心确实定义了语法属性，如操作符的拼写和优先级。）

这个库也包含了内置函数和异常 --- 不需要 `import` 语句就可以在所有Python代码中使用的对象。有一些是由语言核心定义的，但是许多对于核心语义不是必需的，并且仅在这里描述。

不过这个库主要是由一系列的模块组成。这些模块集可以不同方式分类。有些模块是用 C 编写并内置于 Python 解释器中；另一些模块则是用 Python 编写并以源码形式导入。有些模块提供专用于 Python 的接口，例如打印栈追踪信息；有些模块提供专用于特定操作系统的接口，例如操作特定的硬件；另一些模块则提供针对特定应用领域的接口，例如万维网。有些模块在所有更新和移植版本的 Python 中可用；另一些模块仅在底层系统支持或要求时可用；还有些模块则仅当编译和安装 Python 时选择了特定配置选项时才可用。

本手册以 "从内到外" 的顺序组织：首先描述内置函数、数据类型和异常，最后是根据相关性进行分组的各种模块。

这意味着如果你从头开始阅读本手册，并在感到厌烦时跳到下一章，你仍能对 Python 库的可用模块和所支持的应用领域有个大致了解。当然，你并非 *必须* 如同读小说一样从头读到尾 --- 你也可以先浏览内容目录 (在手册开头)，或在索引 (在手册末尾) 中查找某个特定函数、模块或条目。最后，如果你喜欢随意学习某个主题，你可以选择一个随机页码 (参见 `random` 模块) 并读上一两小节。无论你想以怎样的顺序阅读本手册，还是建议先从 内置函数 这一章开始，因为本手册的其余内容都需要你熟悉其中的基本概念。

让我们开始吧！

可用性注释

- 如果出现“适用：Unix”注释，意味着相应函数通常存在于 Unix 系统中。 但这并不保证其存在于某个特定的操作系统中。
- 如果没有单独注明，所有声称"可用性：Unix"的函数都支持 macOS、iOS 和 Android ，所有这些构建都基于 Unix 内核。
- 如果一条可用性注释同时包含最低 Kernel 版本和最低 libc 版本，则两个条件都必须满足。 例如当某个特性带有注释 *可用性: Linux >= 3.17 且 glibc >= 2.27* 则表示同时要求 Linux 3.17 以上版本和 glibc 2.27 以上版本。

WebAssembly 平台

WebAssembly 平台 `wasm32-emscripten` (Emscripten) 和 `wasm32-wasi` (WASI) 分别提供了 POSIX API 的一个子集。 WebAssembly 运行时和浏览器都处于沙盒模式中并具有对主机和外部资源的受限访问权。 任何使用了进程、线程、网络、信号或其他形式的进程间通信 (IPC) 的 Python 标准库模块都或者不可用，或者其作用方式与在其他类 Unix 系统上不同。 文件 I/O, 文件系统和 Unix 权限相关的函数也同样会受限。 Emscripten 不允许阻塞式 I/O。 其他阻塞式操作如 `sleep()` 则会阻塞浏览器的事件循环。

Python 在 WebAssembly 平台上的特性与行为依赖于 Emscripten-SDK 或 WASI-SDK 的版本, WASM 运行时 (浏览器, NodeJS, wasmtime) 以及 Python 编译时旗标。 WebAssembly, Emscripten 和 WASI 都是尚在不断演化中的标准；某些特性例如网络可能会在未来被支持。

对于在浏览器上运行的 Python，用户可以考虑 Pyodide 或 PyScript。 PyScript 是在 Pyodide 之上构建的，后者本身则是在 CPython 和 Emscripten 之上构建的。 Pyodide 提供了对浏览器的 JavaScript 和 DOM API 的访问并通过 JavaScript 的 `XMLHttpRequest` 和 `Fetch` API 提供了受限的网络功能。

- 进程相关的 API 或者不可用或者将始终报错失败。 这包括生成新进程 (`fork()`, `execve()`), 等待进程 (`waitpid()`), 发送信号 (`kill()`) 或者以其他方式与进程交互的 API。 `subprocess` 可以被导入但将没有任何作用。
- `socket` 模块可以使用，但将会受限而使其行为与在其他平台上不一致。 在 Emscripten 上，套接字将始终为非阻塞式的并且要求额外的 JavaScript 代码和服务器上的辅助工具来代理通过 WebSockets 的 TCP；请参阅 Emscripten Networking 了解详情。 WASI snapshot preview 1 只允许来自现有文件描述符的套接字。
- 某些函数是不执行任何操作的空壳或是始终返回硬编码的值。
- 有关文件描述符、文件访问权、文件所有权和链接的函数均受到限制并且不支持某些操作。 例如，WASI 不允许具有绝对文件名的符号链接。

移动平台

Android 和 iOS 在大多数方面都是 POSIX 操作系统。文件 I/O、套接字处理和线程的行为都与在任何 POSIX 操作系统上的行为相同。但是，有几个主要区别：

- 移动平台只能在"嵌入"模式下使用Python。没有Python REPL，也不能使用单独的可执行文件，如 **python** 或 **pip** 。要在移动应用程序中添加 Python 代码，必须使用 Python 嵌入式 API 。更多详情，请参阅 在Android上使用 Python 和 在 iOS 上使用 Python 。

- 子进程：

  - 在 Android 上，创建子进程是可能的，但"非官方支持 <https://issuetracker.google.com/issues/128554619#comment4>`__"。尤其是，Android 没有支持 System V IPC 的任何部分API ，因此 `multiprocessing` 不可用。
  - iOS 应用程序不能使用任何形式的子进程、多进程或进程间通信。如果 iOS 应用程序尝试创建子进程 ，创建子进程的进程要么会上锁 ，要么会崩溃。iOS 应用程序无法看到正在运行的其他应用 ，也无法与运行中的其他应用通信，除非使用为此目的而存在的 iOS 特定 API 。

- 移动应用程序修改系统资源（如系统时钟）的权限有限。这些资源通常*可读*，但试图修改这些资源通常会失败。

- 控制台输入与输出：
  - 在 Android 系统上，原生的 `stdout` 和 `stderr` 没有连接到任何设备，因此 Python 安装了自己的流，将信息重定向到系统日志。这些信息可分别在 `python.stdout` 和 `python.stderr` 标签下看到。
  - iOS 应用程序对控制台输出的概念有限。 `stdout` 和 `stderr` *存在*，在 Xcode 中运行时，写入 `stdout` 和 `stderr` 的内容在日志中可见，但这些内容 *不会* 记录在系统日志中。 如果安装了您的应用程序的用户提供其应用程序日志作为诊断辅助工具，这些日志将不包括写入 `stdout` 或 `stderr` 的任何细节。
  - 移动应用程序根本没有可用的 `stdin` 。虽然应用程序可以显示屏幕键盘，但这是软件特性，而不是连接在 `stdin` 上的。

    因此，涉及控制台操作的 Python 模块（如 `curses` 和 `readline` ）无法在移动平台上使用。