目录

[VS Code 和 PyCharm 2](#_Toc204956363)

[一基础介绍 3](#_Toc204956364)

[## \*\*一、界面布局与核心区域\*\* 3](#_Toc204956365)

[### 1. \*\*启动与欢迎页\*\* 4](#_Toc204956366)

[### 2. \*\*主界面模块\*\* 4](#_Toc204956367)

[## \*\*二、基础操作：文件与工作区\*\* 5](#_Toc204956368)

[## \*\*三、编码效率：智能提示与扩展\*\* 7](#_Toc204956369)

[### 1. \*\*智能编码功能\*\* 7](#_Toc204956370)

[### 2. \*\*扩展市场（必用！）\*\* 7](#_Toc204956371)

[## \*\*四、运行与调试代码\*\* 8](#_Toc204956372)

[### 2. \*\*调试模式（排错神器）\*\* 9](#_Toc204956373)

[## \*\*五、个性化配置\*\* 10](#_Toc204956374)

[### 1. \*\*设置界面（图形化）\*\* 10](#_Toc204956375)

[### 2. \*\*settings.json（进阶配置）\*\* 10](#_Toc204956376)

[## \*\*六、高频场景：PowerShell 脚本运行（结合你的需求）\*\* 11](#_Toc204956377)

[打开多标签 14](#_Toc204956378)

[一、基础：让打开的文件默认保留为新标签（关闭 “预览模式”） 14](#_Toc204956379)

[多终端标签（同时开 CMD、PowerShell、Git Bash ） 15](#_Toc204956380)

[设置 15](#_Toc204956381)

[同步 15](#_Toc204956382)

[打开用户设置 16](#_Toc204956383)

[解释红框中运行自动检测到的任务 18](#_Toc204956384)

[原理层面 18](#_Toc204956385)

[作用层面 18](#_Toc204956386)

[Git 19](#_Toc204956387)

[git卡在这里是怎么回事,提交和更改 19](#_Toc204956388)

[提交方式 20](#_Toc204956389)

[配置信息 21](#_Toc204956390)

[git仓库和main主分支 21](#_Toc204956391)

[扩展 22](#_Toc204956392)

[1Cline 22](#_Toc204956393)

[安装 22](#_Toc204956394)

[2Multi-Cursor Tools 23](#_Toc204956395)

[命令 23](#_Toc204956396)

[3pyplance主要功能特性 24](#_Toc204956397)

[Python Debugger主要功能特性 25](#_Toc204956398)

[Copilot 27](#_Toc204956399)

[1添加上下文 27](#_Toc204956400)

[2设置 30](#_Toc204956401)

[2.1设置Python 类型检查模式 31](#_Toc204956402)

[快捷键 32](#_Toc204956403)

# VS Code 和 PyCharm

VS Code 和 PyCharm 是 \*\*两款独立的软件\*\* ，VS Code 里没法直接“安装”PyCharm，但可以分别安装它们，下面给你说清楚：

### 一、本质区别：各自独立，功能互补

- \*\*VS Code\*\*：是轻量级编辑器，靠装插件（比如 Python 插件）支持 Python 开发，本身不是“Python 专属”，也能写前端、Java 等。

- \*\*PyCharm\*\*：是专门给 Python 开发的 IDE（集成开发环境），功能更“重”、更专业，和 VS Code 是平行的软件，\*\*不能在 VS Code 里安装 PyCharm\*\* 。

### 二、正确操作：分别安装，按需切换

想同时用这俩工具？按下面步骤来：

#### 1. 安装 PyCharm（和 VS Code 共存）

- 去 [JetBrains 官网](https://www.jetbrains.com/pycharm/) 下载：

- 个人/学习用，选 \*\*社区版（Community）\*\* ，免费且功能够日常开发；

- 做专业项目（比如 Django/Flask 网站、AI 工程），可选 \*\*专业版（Professional）\*\* （付费，但有 30 天试用）。

- 下载后双击安装包，一路默认点“下一步”就行（想改安装路径也能自己选），装完桌面会有 PyCharm 图标。

#### 2. 怎么选着用？

- \*\*写小脚本、多语言混合开发\*\*：开 VS Code ，装个 Python 插件，顺手又轻量；

- \*\*做专业 Python 项目（比如 Django 网站、复杂算法工程）\*\*：开 PyCharm ，它的智能补全、调试器、数据库工具更“专业对口”；

- 完全不冲突，想用哪个开哪个，甚至能同时开着，VS Code 写前端、PyCharm 写 Python 后端，配合起来干活～

### 三、额外小技巧：联动使用

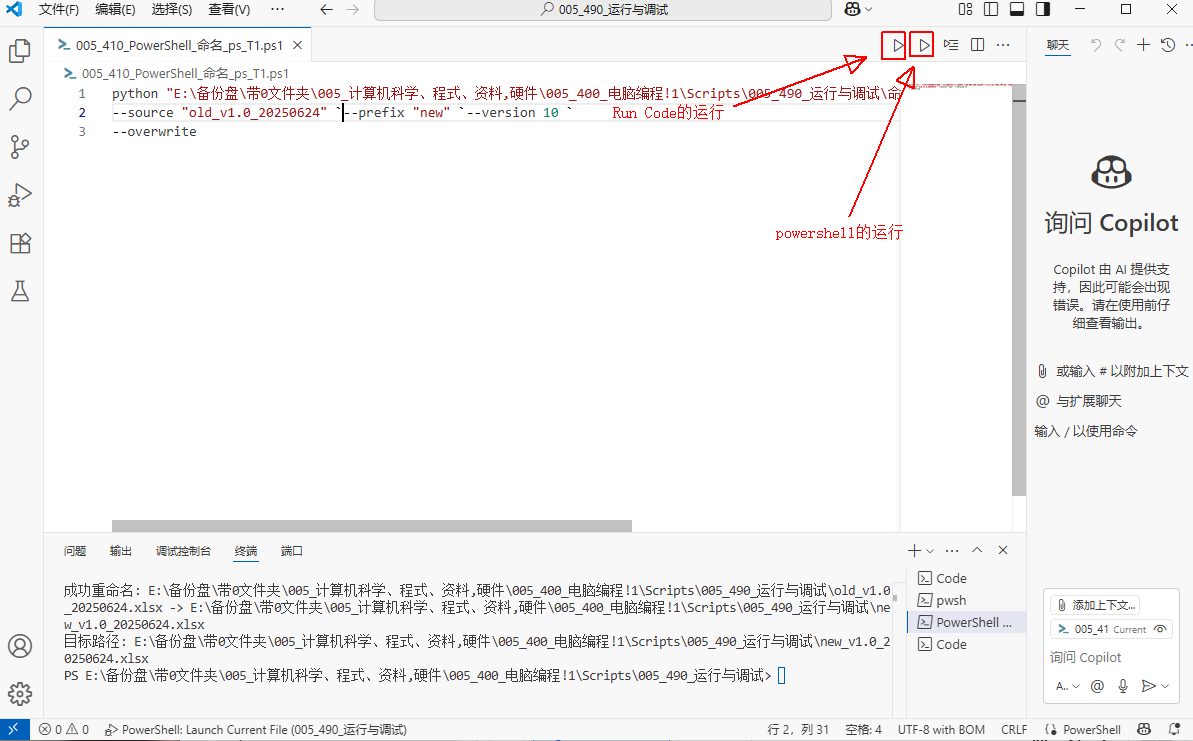
虽然不能“安装”，但可以让它们“配合”：

- \*\*从 VS Code 里打开 PyCharm 项目\*\*：在 VS Code 资源管理器，直接拖 PyCharm 的项目文件夹进去，就能编辑代码（不过专业功能还是得切 PyCharm 用）；

- \*\*共享配置\*\*：比如 VS Code 里装了代码格式化插件，PyCharm 里也能单独设置相同的格式化规则（比如都用 Black 格式化 Python 代码），保持编码习惯一致。

简单说：VS Code 和 PyCharm 是独立软件，分别安装就行，按开发需求选着用/一起用，互不影响～

在Visual Studio Code 使用powerShell扩展运行ps1文件方法



以下是结合图片说明的 \*\*Visual Studio Code（简称 VS Code ）使用手册\*\* ，从基础到进阶，帮你快速上手：

---

# 一基础介绍

## ## \*\*一、界面布局与核心区域\*\*

（以 Windows 系统为例，macOS 操作逻辑类似，快捷键替换 `Ctrl` 为 `Command` ）

### ### 1. \*\*启动与欢迎页\*\*

首次打开 VS Code ，会看到 \*\*欢迎界面\*\* ，包含快速入口：

- \*\*“新建文件”“打开文件夹”\*\*：创建/打开项目文件。

- \*\*“最近打开”\*\*：快速访问历史项目。

- \*\*“扩展”“设置”\*\*：配置个性化开发环境。

- \*\*“学习资源”\*\*：官方文档、教程链接（新手必看！）

![VS Code 欢迎页](https://www.runoob.com/wp-content/uploads/2023/03/vscode-start.png)

### ### 2. \*\*主界面模块\*\*

VS Code 界面分 \*\*5 大核心区域\*\* ，鼠标悬停活动栏可看功能说明：

| 区域 | 位置 | 功能说明 |

|---------------|------------|--------------------------------------------------------------------------|

|A \*\*活动栏\*\* | 左侧边栏 | 切换视图（文件、搜索、Git、调试、扩展等），点击图标展开/收起对应功能。 |

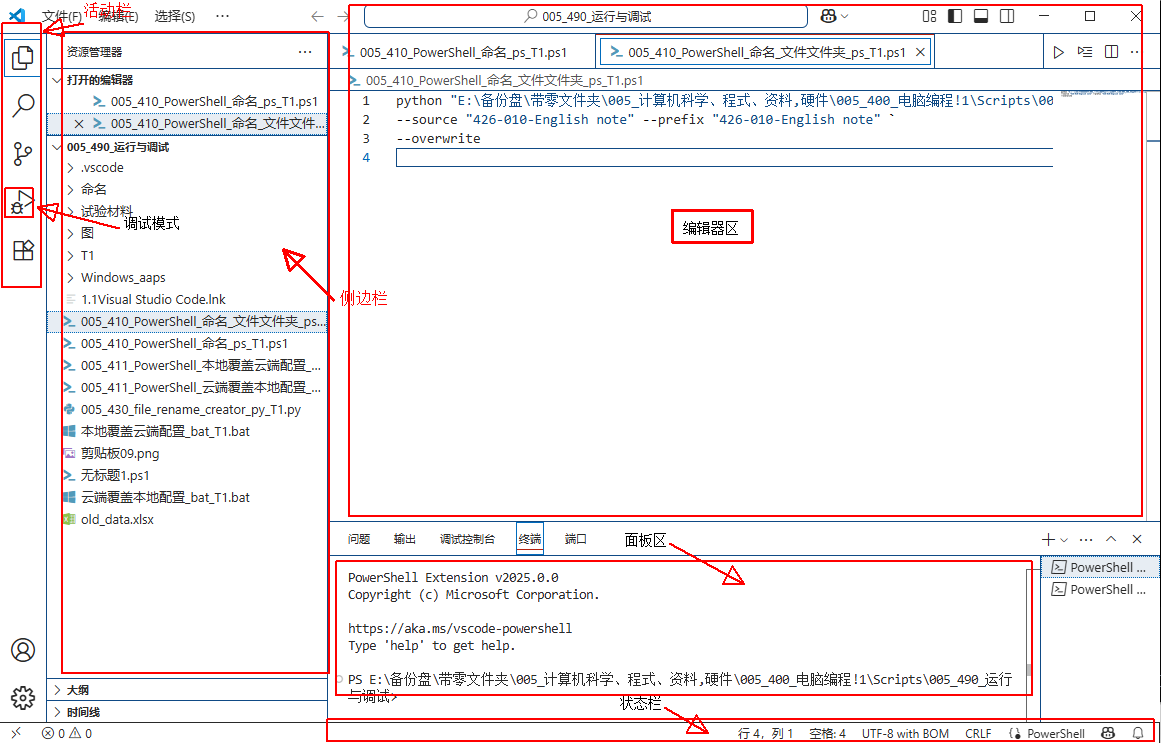
| B\*\*侧边栏\*\* | 活动栏右侧 | 显示当前视图的详细内容（如文件列表、搜索结果、Git 提交记录）。 |

| C\*\*编辑器区\*\* | 中间区域 | 代码编辑主窗口，支持多标签、分屏（`Ctrl + \` 垂直分屏 ）。 |

|D \*\*面板区\*\* | 底部区域 | 显示终端、输出、调试控制台等，按 `Ctrl + ·`（Esc 下方按键 ）快速调出终端。 |

|E \*\*状态栏\*\* | 最底部 | 显示文件编码、行号、列号、Git 分支、语言模式等信息。 |

![VS Code 界面布局]

---

## ## \*\*二、基础操作：文件与工作区\*\*

### 1. \*\*打开/新建项目\*\*

- \*\*打开文件夹（工作区）\*\*：

点击菜单栏 `文件 > 打开文件夹` ，选择项目根目录（如 `vscode-project` ），VS Code 会加载目录下所有文件。

- 首次打开陌生文件夹，会提示 \*\*“工作区信任”\*\* ，选 `信任` 启用完整功能（避免恶意代码执行）。

- \*\*新建文件\*\*：

点击侧边栏 `资源管理器 > 新建文件` ，或按 `Ctrl + N` ，保存时选路径、填文件名（如 `main.py` `script.ps1` ，后缀关联语言）。

### 2. \*\*文件管理快捷键\*\*

| 操作 | 快捷键 | 说明 |

|---------------------|-----------------|----------------------------------------|

| 保存文件 | `Ctrl + S` | 单文件保存；开启“自动保存”可在设置中配置。 |

| 关闭当前标签页 | `Ctrl + W` | 关闭编辑器标签。 |

| 快速切换文件 | `Ctrl + P` | 输入文件名/关键字，跳转已打开文件。 |

| 显示所有已打开文件 | `Ctrl + Shift + E` | 聚焦资源管理器，浏览文件列表。 |

---

## ## \*\*三、编码效率：智能提示与扩展\*\*

### ### 1. \*\*智能编码功能\*\*

VS Code 内置 \*\*IntelliSense（智能感知）\*\* ，输入代码时自动补全语法、函数、变量：

- 输入 `for` `if` 等关键字，按 `Tab` 生成代码模板。

- 悬停变量/函数，显示文档注释；按住 `Ctrl` 点击可跳转定义（`F12` 也可）。

\*\*代码片段\*\*：输入 `log`（JavaScript）、`print`（Python）等，快速插入常用代码块（可自定义片段，`文件 > 首选项 > 用户代码片段` ）。

### ### 2. \*\*扩展市场（必用！）\*\*

点击左侧活动栏 \*\*“扩展”（方块图标，或 `Ctrl + Shift + X` ）\*\* ，搜索安装插件增强功能：

- \*\*语言支持\*\*：

- Python、PowerShell、Java 等官方扩展（微软开发），提供语法高亮、调试、运行支持。

- 例：安装 `Python` 扩展后，`Ctrl + F5` 可直接运行 `.py` 文件。

- \*\*效率工具\*\*：

- `Prettier`：自动格式化代码（`Shift + Alt + F` 快捷键）。

- `GitLens`：显示代码提交历史、作者，增强 Git 管理。

- `Auto Rename Tag`：修改 HTML/XML 标签时，自动同步闭合标签。

- \*\*主题与外观\*\*：

- `Material Icon Theme`：美化文件图标，直观区分文件类型。

- `One Dark Pro`：热门暗色主题，保护眼睛。

---

## ## \*\*四、运行与调试代码\*\*

### 1. \*\*终端直接运行\*\*

按 `Ctrl + ·` 调出 \*\*集成终端\*\* ，直接执行命令：

终端”（Terminal）功能，简单说就是在编辑器内直接运行命令行操作的工具

- \*\*Python\*\*：`python script.py`（需安装 Python 扩展 + 配置环境）。

- \*\*PowerShell\*\*：`powershell .\script.ps1`（或右键 `Run PowerShell File` ，需官方扩展）。

- \*\*前端\*\*：`npm run dev`（运行 Node 项目）。

### ### 2. \*\*调试模式（排错神器）\*\*

点击左侧活动栏 \*\*“运行和调试”（虫子图标，或 `Ctrl + Shift + D` ）\*\* ，按步骤配置：

1. \*\*选择调试环境\*\*：

首次使用选 `创建 launch.json 文件` ，再选语言（如 Python、PowerShell、JavaScript ）。

2. \*\*设置断点\*\*：

点击代码行号左侧，标记小红点（断点），程序会在此暂停。

3. \*\*启动调试\*\*：

按 `F5` 运行，程序暂停后：

- `F10`：逐过程执行（跳过函数调用）。

- `F11`：逐语句执行（进入函数内部）。

- `Shift + F11`：跳出当前函数。

- 查看“变量”“调用栈”面板，分析运行时状态。

---

## ## \*\*五、个性化配置\*\*

### ### 1. \*\*设置界面（图形化）\*\*

按 `Ctrl + ,` 打开 \*\*设置面板\*\* ，搜索关键词自定义：

- \*\*自动保存\*\*：`文件 > 自动保存` 选 `afterDelay`（延迟 1 秒保存）或 `onFocusChange`（切换窗口时保存）。

- \*\*字体与主题\*\*：`外观 > 颜色主题` 换主题；`文本编辑器 > 字体` 改字号、字体（推荐 `Fira Code` ，支持连字）。

### ### 2. \*\*settings.json（进阶配置）\*\*

点击设置面板右上角 `打开设置 (JSON)` ，直接编辑配置文件，例：

```json

{

"editor.fontSize": 14, // 编辑器字号

"editor.tabSize": 2, // Tab 缩进 2 个空格

"files.autoSave": "afterDelay", // 延迟自动保存

"terminal.integrated.defaultProfile.windows": "PowerShell" // 默认终端

}

```

---

## ## \*\*六、高频场景：PowerShell 脚本运行（结合你的需求）\*\*

### 1. \*\*必备扩展\*\*

安装 \*\*微软官方 PowerShell 扩展\*\*（搜索 `PowerShell` ，微软开发的蓝色图标扩展），支持：

- 右键 `Run PowerShell File` 直接运行 `.ps1` 文件。

- 调试 PowerShell 脚本（断点、变量监控）。

### 2. \*\*运行步骤\*\*

- \*\*方法 1：右键运行\*\*

打开 `.ps1` 文件，右键编辑器空白处 → `Run PowerShell File` ，结果在终端显示。

- \*\*方法 2：终端执行\*\*

终端输入 `powershell .\脚本名.ps1` ，或切换终端为 PowerShell（状态栏右下角选 `PowerShell` ）。

- \*\*方法 3：调试模式\*\*

点击左侧 `运行和调试` → 选 `PowerShell` 环境 → 按 `F5` 启动调试，断点处暂停分析。

---

## \*\*七、常见问题与解决方案\*\*

### 1. \*\*“无法加载 PowerShell 脚本，因为禁止运行脚本”\*\*

原因：系统执行策略限制。

解决：

1. 以 \*\*管理员身份\*\* 打开 PowerShell 终端。

2. 执行命令：

```powershell

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned -Force

```

### 2. \*\*终端中文乱码\*\*

原因：编码不匹配。

解决：

设置 `settings.json` ，添加：

```json

"terminal.integrated.profiles.windows": {

"PowerShell": {

"args": ["-Command", "chcp 65001"], // 设置 UTF-8 编码

"source": "PowerShell"

}

}

```

### 3. \*\*扩展安装失败\*\*

原因：网络问题或权限不足。

解决：

- 切换网络（如手机热点），或配置 VS Code 代理（`文件 > 首选项 > 设置 > 代理` ）。

- 以管理员身份运行 VS Code（Windows 右键 `以管理员身份启动` ）。

---

## \*\*八、学习资源推荐\*\*

- \*\*官方文档\*\*：命令面板（`Ctrl + Shift + P` ）输入 `Open Documentation` ，或访问 [VS Code 官网文档](https://code.visualstudio.com/docs)。

- \*\*视频教程\*\*：B站/YouTube 搜索 “VS Code 入门”，推荐微软官方频道、“技术胖”“狂神说” 等 UP 主。

- \*\*快捷键 cheat sheet\*\*：`文件 > 首选项 > 键盘快捷方式` ，或下载 [官方快捷键 PDF](https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf)。

通过以上内容，你已掌握 VS Code 从基础操作到编码、运行、调试的核心流程！多实操（比如用它写个 PowerShell 脚本、Python 小程序 ），熟练后编码效率会飞涨～ 遇到问题，记得善用扩展市场和官方文档！

## 打开多标签

### 一、基础：让打开的文件默认保留为新标签（关闭 “预览模式”）

VS Code 默认是预览模式：单击文件时，新文件会覆盖当前标签（方便快速浏览，但不利于保留多标签）。若想让每个文件都作为独立标签保留，需修改设置关闭预览模式：

图形化设置（简单直观）

按 Ctrl + ,（Windows/Linux）或 Command + ,（macOS）打开设置界面，搜索 enablePreview：

找到 Workbench > Editor: Enable Preview，取消勾选（默认是勾选状态）。

同时确保 Workbench > Editor: Show Tabs 是勾选状态（控制是否显示标签栏）。



## 多终端标签（同时开 CMD、PowerShell、Git Bash ）

## 设置

### 同步

3. 手动触发同步（可选）

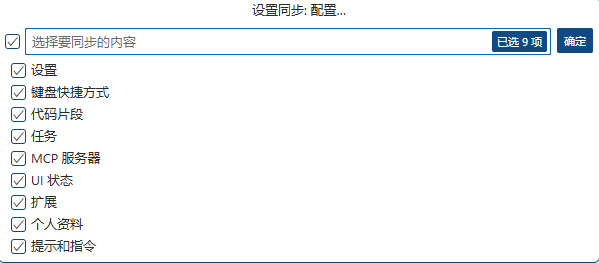
若想立即同步或拉取最新配置，按 Ctrl + Shift + P（Windows/Linux）或 Command + Shift + P（Mac）打开命令面板，输入 “Sync: Sync Now” 并执行，可手动触发一次同步，上传本地更改、下载云端最新配置。

4. 管理同步设置

通过命令面板输入 “Preferences: Open Settings (UI)” 打开设置界面，搜索 “Sync”，能细致控制同步内容（比如关闭扩展同步、仅同步特定类型设置等） ，根据需求调整哪些配置参与同步。

微软账户的同步选项

Configure...（配置）：点击后可进入更详细的同步配置界面，一般能设置同步的内容，根据需求自定义同步规则 。



Show Settings（显示设置）：会展示当前已设置好的同步相关参数，像同步的项目、频率、账户关联等信息，方便查看现有同步配置情况。

Show Synced Data（显示已同步数据）：用于查看当前已经成功同步到本地（或对应设备 / 应用）的数据，比如已同步的邮件列表、联系人详情、日历事件等，了解同步内容。

Sync Now（立即同步）：手动触发同步操作，不管当前是否到自动同步时间，点击后会立即尝试将远程服务器（Outlook 服务器）上的最新数据与本地（或对应端）进行同步更新，比如有新邮件、日历变动，执行后能及时获取。

Turn Off kd13128963060@outlook.com (Microsoft)：点击会关闭该 Outlook 账户（kd13128963060@outlook.com ）的同步功能，之后本地（或对应端）不会再从服务器同步新数据，也不会把本地对同步内容的修改（如新建日历事件等，依设置而定）同步回服务器 ，若想停止该账户同步，可执行此操作。

#### 确认同步成功

查看同步日志（高级方式）

如果上述方法无法明确判断同步是否成功，还可以查看同步日志：

按 Ctrl + Shift + P （Windows/Linux）或 Command + Shift + P （Mac）打开命令面板。

输入 “Developer: Toggle Developer Tools” 并执行，会打开开发者工具窗口。

在开发者工具窗口中，切换到 “Console” 标签页，然后再次执行 “Sync: Sync Now” 手动同步操作。此时，在 Console 中会输出同步相关的日志信息，你可以通过查看这些信息来确认同步过程中是否有错误或警告，进而判断同步是否成功。如果没有报错信息，并且有类似同步完成的相关提示，通常意味着同步成功。

### 打开用户设置

“用户” 和 “工作区” 是设置作用范围的不同维度，差别如下：

* **用户设置**：对当前用户生效，不管打开哪个项目（工作区），这些设置都会起作用。比如你设置了全局的代码格式化规则，换个项目打开，格式化规则还是这个。
* **工作区设置**：只对当前打开的工作区（项目文件夹）生效。不同工作区可单独配置，互不影响 。像某个项目需要特殊的 Python 解释器路径，就可在该工作区设置里指定，不影响其他项目 。

User settings （用户设置） Workspace settings （工作区设置）

方法一按 Ctrl + Shift + P（Win/Linux）/Command + Shift + P（Mac），输入 Open Settings (JSON) 打开完整配置文件，检查这两项：

// 终端的单词分隔符，确保 - 等字符已移除

"terminal.integrated.wordSeparators": "`~!@#$%^&\*()=+[{]}\|;:'\",.<>/?",

// 编辑器的单词分隔符，同样确认 - 被移除

"editor.wordSeparators": "`~!@#$%^&\*()=+[{]}\|;:'\",.<>/?"

方法二1. 修改 VS Code 的 “单词分隔符” 配置（核心步骤）

打开 VS Code 设置（按 Ctrl + , 或 Command + ,）。

搜索 files.wordSeparators，默认值里包含 - ! 等字符，这些字符会让 VS Code 把文本拆分成

多个单词。

##### 1. ****Code Action（代码操作）****

Code Action：VS Code 的自动化代码修复功能，通过 “灯泡” 图标触发。

**Code Action** 是 VS Code 的核心功能之一，指编辑器根据当前代码上下文提供的**自动化代码修复、优化或重构建议**。  
当你的代码存在语法错误、风格问题（如不符合 ESLint 规则）、可优化点（如未使用的变量）时，VS Code 会在代码下方显示波浪线（红色 / 黄色），此时：

* 鼠标悬停在波浪线上，会显示问题描述；
* 点击左侧的 “灯泡” 图标（或按快捷键 Ctrl+. / Cmd+.），会弹出**Code Action 列表**，包含可执行的修复方案（如 “自动导入模块”“删除未使用的变量”“格式化代码” 等）。

简单说，Code Action 是 VS Code 帮你 “自动改代码” 的工具，提高编码效率。

##### 2. ****Hello（通常指 “Hello World” 相关代码）****

“Hello” 本身没有特殊含义，大概率是你在代码中编写了类似 console.log("Hello") 或 print("Hello") 的语句（常见于入门级 “Hello World” 示例代码）。

当这段代码存在问题时（例如语法错误、未定义的变量等），VS Code 会通过**Code Action**提供修复建议。例如：

* 如果漏写了引号（如 console.log(Hello)），Code Action 会建议 “添加字符串引号”；
* 如果是 Python 代码漏写了括号（如 print "Hello"），Code Action 会建议 “添加括号”（符合 Python 3 语法）。

# 解释红框中运行自动检测到的任务

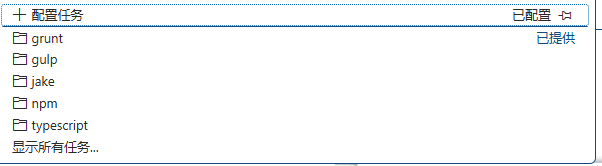
在 VS Code 里，“运行自动检测到的任务” 是借助其任务（Tasks）功能，自动识别项目里常见工作流（像构建、测试、启动开发服务等）相关任务并执行 。以下从原理和作用两方面展开说：

## 原理层面

VS Code 会扫描项目里的配置文件（比如 package.json 里的 scripts 脚本、tasks.json 自定义任务配置 ），识别出可执行的任务逻辑，像 npm start 这类启动命令，就是它能检测到的任务来源 。

## 作用层面

* **简化操作**：不用手动在终端敲命令，点一下就能执行项目构建、启动开发环境等流程，提升开发效率 。比如前端项目，点它可能直接启动 npm run dev 开启本地开发服务。
* **标准化流程**：让团队成员执行相同任务时，用统一的触发方式，减少因手动输入差异导致的问题，保障项目运行环境、流程一致 。



简单说，就是 VS Code 帮你自动找项目里能执行的流程任务，一键触发，让开发中跑脚本、启动服务等操作更便捷 。

grunt / gulp / jake：都是前端工程化工具，用于定义和执行项目的自动化流程（比如编译 Sass、压缩代码、监听文件变化）。VS Code 内置了对它们的支持，选这些可快速生成对应工具的任务配置。

npm：Node.js 包管理工具，项目 package.json 里的 scripts 脚本（如 npm run dev 启动开发、npm run build 打包），VS Code 能识别并通过 “npm” 选项配置任务。

typescript：TypeScript 语言的编译、校验等任务配置入口，选它可生成编译 TypeScript 代码的任务（比如自动把 .ts 转成 .js）。

显示所有任务…：展开查看当前项目里所有可识别的任务（包括自动检测、已配置的），方便筛选 / 执行。

3. 实际用途

你选某工具（如 npm），VS Code 会帮你生成 tasks.json 配置文件，定义任务的执行命令、触发条件等，后续点 “运行任务” 就能自动执行，不用手动在终端敲命令，提升开发效率。

简单说，这就是 VS Code 为不同工具预设的 “任务快捷配置入口”，让你更方便地自动化执行项目流程 ～

ylance 是微软开发的一款用于 Visual Studio Code（VS Code）的 Python 语言服务器扩展，基于微软的静态类型检查工具 Pyright 开发 ，旨在为 Python 开发者提供更高效、专业的开发体验，以下是对它的详细介绍：

# **Git**

## git卡在这里是怎么回事,提交和更改

从你提供的VS Code界面来看，这是在进行Git提交操作时，进入了提交信息编辑界面 ，之所以“卡住”，是因为Git要求必须填写符合规范的提交信息，才会完成本次提交操作。以下是具体说明和解决办法：

### 原因分析

在Git中执行提交（commit）操作时，会打开一个默认的文本编辑器（比如你这里在VS Code中显示的 `COMMIT\_EDITMSG` 文件），用于让用户输入本次提交的说明信息。以 `#` 开头的行是注释内容，在提交时会被忽略，而如果整个提交信息内容为空，Git会中止这次提交操作，所以看起来就像是“卡住”了。

### 解决办法

1. \*\*编写提交信息\*\*：在 `COMMIT\_EDITMSG` 文件中，删除以 `#` 开头的注释行（也可以保留注释行，不影响提交），然后在文件中添加一行或多行描述本次代码更改的信息。比如，如果是添加了新功能，可以写 `Add new feature for user authentication` ；如果是修复了Bug，可以写 `Fix bug that causes login failure` 。

2. \*\*保存并关闭编辑器\*\*：编写好提交信息后，保存文件（在VS Code中可以使用 `Ctrl + S` 快捷键保存 ），然后关闭这个 `COMMIT\_EDITMSG` 编辑窗口。

3. \*\*完成提交\*\*：关闭编辑窗口后，VS Code的Git插件会检测到提交信息已填写，会继续完成提交操作，这时可以在VS Code的“源代码管理”视图中看到提交成功的提示，或者在终端中看到提交相关的日志信息。

### 补充说明

如果你想修改默认的提交信息编辑器，或者希望简化提交信息填写流程，也可以通过以下方式操作：

- \*\*修改默认编辑器\*\*：可以使用 `git config --global core.editor` 命令来指定自己喜欢的编辑器。比如，指定VS Code为默认编辑器，命令为 `git config --global core.editor "code --wait"` 。

- \*\*使用命令行参数直接指定提交信息\*\*：在执行 `git commit` 命令时，使用 `-m` 参数直接在命令行中指定提交信息，例如 `git commit -m "Initial commit"` ，这样就不会打开编辑

器来编辑提交信息了。

提交和更改

一、提交文件到本地仓库

打开源代码管理面板

快捷键：Ctrl+Shift+G（Windows/Linux）或 Cmd+Shift+G（Mac）

或点击左侧活动栏的 源代码管理图标（类似分支的图标）。

暂存更改

在 “更改” 列表中，勾选需要提交的文件前的复选框（或右键点击文件选择 “暂存更改”），将文件加入暂存区。

若要暂存所有更改，可点击 “更改” 旁的 + 图标。

填写提交信息

在面板顶部的输入框中，输入本次提交的描述（如 “修复登录按钮样式”），简洁明了地说明更改内容。

提交到本地仓库

点击输入框下方的 √ 提交 按钮，或使用快捷键 Ctrl+Enter（Windows/Linux）/ Cmd+Enter（Mac）。

二、同步到远程仓库（推送 / 拉取）

提交到本地仓库后，需与远程仓库（如 GitHub、GitLab 等）同步，确保本地和远程版本一致。

1. 拉取远程最新代码（避免冲突）

在推送前，建议先拉取远程仓库的最新更改，防止冲突：

点击源代码管理面板右上角的 ⋯（更多操作），选择 拉取（Pull）。

或使用快捷键 Ctrl+Shift+P 打开命令面板，输入并执行 Git: 拉取。

2. 推送本地提交到远程

点击源代码管理面板右上角的 ⋯，选择 推送（Push）。

或使用快捷键 Ctrl+Shift+P，执行 Git: 推送。

首次推送时，可能需要关联远程仓库，按提示输入远程仓库地址（如 https://github.com/用户名/仓库名.git）。

## 提交方式

1. 提交（常规提交）

含义：仅把暂存的更改提交到本地 Git 仓库，不会推送到远程仓库（如 GitHub/GitLab 等）。适合本地多次修改、逐步整理提交记录，后续再统一推送的场景。

场景：比如你在本地开发新功能，想先把阶段性的代码保存到本地版本库，还没准备好同步到远程时，选它。

2. 提交（修改）

含义：专门用于修改上一次的提交。会打开提交信息编辑框，让你修改上一次提交的说明，且新改动会合并到上一次提交里（修改提交历史）。

场景：要是刚提交完，发现提交信息写错，或者漏加了文件，就用这个选项修正。

3. 提交和推送

含义：先执行本地提交，接着自动把提交的内容推送到远程仓库（需提前关联好远程仓库，且有权限推送）。

场景：本地开发完成、确认改动没问题，想直接同步到远程让团队其他成员获取，选这个一步到位。

4. 提交和同步

含义：逻辑类似 “提交和推送”，但更智能。会先拉取（pull ）远程仓库最新内容（避免冲突），再提交本地改动，最后推送（push ）。

场景：多人协作开发时，确保本地代码和远程同步，减少冲突概率，适合日常多人协同的提交流程。

简单总结：

想只存本地，选 提交；

改上次提交，选 提交（修改）；

本地改完直接推远程，选 提交和推送；

多人协作、怕冲突，选 提交和同步

## 配置信息

作者署名git config --global user.name "kid1997"

绑定邮箱git config --global user.email "kid1997@139.com"

远程仓库名称: my – first

## git仓库和main主分支

在 Git 版本控制系统中，Git 仓库和main 分支是两个核心概念，但含义和作用不同：

1. Git 仓库（Repository）

含义：仓库是存储项目所有文件、历史记录和版本信息的地方，是一个完整的项目容器。

作用：

保存项目的所有文件（代码、文档等）。

记录所有修改历史（谁、何时、修改了什么）。

支持分支管理、版本回溯、多人协作等核心功能。

形式：

本地仓库：保存在你的电脑上（.git 隐藏文件夹就是仓库的核心）。

远程仓库：托管在服务器上（如 GitHub、GitLab 等），用于多人共享和备份。

2. main 分支（Branch）

含义：分支是仓库中独立的代码开发线，main 是默认创建的主分支（早期多称为 master，现在逐步改为 main）。

作用：

main 通常用于存放稳定、可发布的代码，是项目的 "主线"。

开发者可以从 main 分支创建其他分支（如 dev、feature/login）进行开发，避免直接修改主分支代码。

开发完成后，再通过 "合并"（merge）将其他分支的代码整合回 main。

区别总结

Git 仓库 main 分支

是整个项目的容器，包含所有文件和历史 是仓库中的一条代码线，仅包含某一版本的代码

一个项目只有一个仓库（本地 + 远程对应） 一个仓库可以有多个分支（main 是默认主分支）

负责存储和管理所有版本信息 负责隔离不同阶段的开发工作（如功能开发、修复 bug）

简单说：仓库是整个项目的 "档案库"，而 main 分支是这个档案库中最核心的一份 "正式文件"，其他分支则是 "草稿文件"，最终可以合并到 "正式文件" 中。

# 扩展

## 1Cline

### 安装

下面给出 3 种在 VS Code 里“用上 Kimi”的主流方案，任选其一即可；它们都基于 Moonshot AI 最新开放的 kimi-k2-0711-preview 模型，免费额度 100 万 token/日，支持超长 128 K 上下文。

方案 A：用 Cline 扩展（官方推荐，Anthropic 协议）

准备工作

• VS Code 最新版

• 到 Moonshot 控制台 创建 API Key，名称随意，复制备用

安装与配置（3 分钟）

#### 获取API Key流程

按下面的 3 个步骤即可在 Moonshot AI 控制台 创建自己的 API Key，全程约 2 分钟。

打开控制台并登录

访问 https://platform.moonshot.cn

用 手机号/邮箱 注册或直接登录；首次登录会自动获得 15 元体验金 。

进入 API Key 管理页

登录后 → 右上角头像 → 控制台 → API Key 管理（或左侧菜单“API 密钥”）。

创建并保存 Key

• 点击 新建 → 输入名称（如 “vscode”）→ 确定

• 系统生成以 sk- 开头的字符串，仅在创建时展示一次 → 立即 复制并妥善保存（可用密码管理器或记事本）。

• 若遗失，只能 删除后重新创建。

创建完成后，就可以把这串 sk-xxx 填到 VS Code 插件、Python/Node 代码或其他工具里开始调用 Kimi 了。

以上是获取API Key流程

VS Code → 左侧扩展图标（Ctrl/Cmd+Shift+X）→ 搜索 Cline → Install

安装完左侧会出现“Cline”图标，点进去 → Settings

按下图一次填写

表格

项 值

Provider Anthropic

API Key 刚才复制的 Key

Base URL https://api.moonshot.cn/anthropic

Model 保持默认即可（会自动指向 kimi-k2）

Browser Tool 建议关闭（Disable）

保存 Done

体验

新建一个空文件，例如 index.html，在 Cline 对话框里输入：

Build a 3D first-person shooter using Three.js in a single file

30 秒内会生成完整可运行的游戏代码并附带调试脚本

## 2Multi-Cursor Tools

### 命令

在VS Code的`Multi-Cursor Tools`扩展中，红框处的命令及其使用方法如下：

### 1. `engineernick.multi-cursor-tools.expand\_to\_next\_occurrence`

- \*\*命令标题\*\*：Expand to Next Occurrence

- \*\*功能\*\*：将当前选中的内容扩展到下一个出现相同文本的位置，并为每个匹配的位置添加一个光标 。这在需要同时编辑多个相同文本内容时非常有用。

- \*\*使用方法\*\*：首先选中一段文本，然后通过快捷键`Ctrl + K`（在Windows和Linux系统上，macOS上可能是`Command + K` ，具体可在扩展设置中查看或修改）触发该命令，光标就会移动到下一个相同文本处并新增一个光标，重复操作可以继续扩展到更多相同文本处。

### 2. `engineernick.multi-cursor-tools.insert\_line\_above\_sticky`

- \*\*命令标题\*\*：Sticky Line Above

- \*\*功能\*\*：在当前光标所在行的上方插入一行，并且新插入的行处于选中状态，同时光标会自动移动到新插入行中，方便快速在多行上方批量插入相同内容。

- \*\*使用方法\*\*：将光标定位到需要插入行的位置（可以同时定位多个光标到不同行），按下对应的快捷键（默认未明确显示，可在扩展的快捷键设置中查看或自定义 ），就会在每一个光标所在行的上方插入一行，并且光标会进入新插入行等待输入。

### 3. `engineernick.multi-cursor-tools.left\_align\_cursors\_using\_spaces`

- \*\*命令标题\*\*：Align Left Selections by Adding Spaces

- \*\*功能\*\*：通过添加空格的方式，将多个光标所在行的文本进行左对齐。这在需要格式化多行文本，使其在左侧保持整齐时非常有效。

- \*\*使用方法\*\*：将多个光标定位到需要对齐的行（可以通过前面提到的扩展命令添加多个光标 ），按下对应的快捷键（可在扩展快捷键设置中查看或自定义 ），扩展会自动分析各行文本并添加适当数量的空格，实现左对齐。

你可以根据自己的操作习惯，在VS Code的“文件” -> “首选项” -> “键盘快捷方式”中，搜索上述命令ID，对相应的快捷键进行查看、修改或添加，以提高操作效率。

## 3pyplance主要功能特性

* **智能代码补全**：提供快速且精准的代码自动完成功能，当你输入代码时，它能根据上下文感知，智能排序并给出函数、方法、变量、关键字等的自动完成列表 。比如输入numpy.后，会弹出numpy模块中所有可用方法和属性的提示框，方便快速找到所需函数。
* **实时类型检查**：以 Pyright 为核心，提供强大的静态类型检查功能。能够在运行代码之前，高亮显示语法错误和潜在问题，还能推断 Python 代码中变量的类型，验证参数在传入函数执行前类型是否正确。例如定义函数def add\_numbers(a:int, b:int) -> int: ，如果调用时参数类型错误，如result = add\_numbers("1", "2")，Pylance 会自动给出警告 。不过类型检查默认是关闭的，需要在设置中，将python.analysis.typeCheckingMode设置为basic或者strict来启动。
* **自动导入建议**：可以自动为输入的模块和对象添加导入语句，节省手动查找和导入模块的时间和精力。
* **高级语法分析**：具备强大的语法分析能力，能实时检查代码中的语法错误并高亮提示，帮助开发者及时发现和修正问题。
* **代码导航**：支持诸如 “转到定义”“查找引用” 等功能，方便开发者在代码库中轻松导航，快速定位到符号的定义位置，提高代码阅读和理解的效率。
* **文档字符串与签名帮助**：将鼠标悬浮在对应的函数上方时，就可以显示对应的函数文档，同时还能提供带有类型信息的函数签名帮助，方便了解函数的参数和返回值等信息 。
* **代码格式化**：可按照 Python 的 PEP 8 风格指南帮助开发者自动格式化代码，保持代码风格的一致性。
* **多根工作区支持**：允许开发者在单个 VS Code 窗口中处理多个 Python 项目，方便进行项目间的切换和协作。
* **与其他工具集成**：能与静态类型工具（如 Mypy）深度集成，无缝协作，还兼容 Jupyter notebooks，在其中也能提供代码补全和智能提示等功能。

Python Debugger（Python 调试器）是 Visual Studio Code（VS Code）中的一个扩展，由微软开发，基于 debugpy 库，用于帮助 Python 开发者调试 Python 代码。它在软件开发流程中扮演着关键角色，能够有效提升代码的质量和开发效率，以下是对它的详细介绍：

### Python Debugger主要功能特性

* **设置断点**：开发者可以在代码中的特定行设置断点，当程序执行到该断点时，会暂停执行，方便开发者查看此时程序中变量的值、函数调用栈等信息，从而分析代码的执行逻辑是否正确。比如在一个计算斐波那契数列的函数中，在关键计算步骤处设置断点，就能观察每一次循环中数列各项数值的变化情况。
* **单步调试**：包括单步跳过（Step Over）、单步进入（Step Into）和单步跳出（Step Out）。单步跳过会执行当前行代码，并直接进入下一行；单步进入可以进入当前行调用的函数内部进行调试；单步跳出则是执行完当前函数的剩余代码，并返回到调用该函数的上一级代码处。通过这些操作，开发者可以细致地跟踪代码的执行流程。
* **变量查看**：在调试暂停时，能够实时查看当前作用域内所有变量的值，包括局部变量、全局变量等。可以直观地看到变量在程序执行过程中的变化，判断其是否符合预期，从而快速定位问题。
* **调用栈查看**：显示当前程序执行的函数调用顺序，方便开发者了解程序的执行路径，以及在嵌套函数调用的情况下，清晰地知道当前执行点处于哪个函数中，有助于分析函数之间的调用关系和数据传递情况。
* **条件断点**：除了普通断点，还支持设置条件断点，只有当满足特定条件时，程序才会在该断点处暂停。例如，在一个循环次数较多的循环中，可以设置条件断点，当循环变量达到某个特定值时才暂停，这样能更精准地定位问题，避免在不必要的循环步骤中反复暂停调试。
* **调试控制台**：在调试过程中，可以通过调试控制台执行 Python 表达式，动态修改变量的值，或者调用函数进行测试，方便快速验证一些想法，而无需修改代码并重新运行程序。

在 VS Code 扩展中，**Python Path** 是由 Mathias Gesbert 开发的一款用于管理 Python 导入相关功能的实用工具 ，以下是对它的详细介绍：

#### Python Path主要功能特性

* **导入路径管理**：Python 在导入模块时，会按照特定顺序在系统路径中查找模块。Python Path 扩展允许开发者自定义 Python 导入模块时的搜索路径。比如在大型项目中，代码可能分布在多个目录下，通过该扩展可以方便地将包含自定义模块的目录添加到 Python 的导入路径中，让 Python 能够正确找到并导入这些模块，避免因模块搜索路径不正确而导致的导入错误。
* **导入语句自动补全优化**：配合 VS Code 的代码补全功能，根据已配置的导入路径，更精准地提供模块和对象的导入建议。当开发者输入import语句时，它能依据设置的路径，列出该路径下所有可导入的模块和包，提高代码编写效率，减少手动查找和输入导入语句的工作量。
* **导入错误诊断与修复辅助**：当出现导入错误时，该扩展可以帮助开发者分析导入路径问题，提供可能的解决方案。例如，提示缺少某个路径的配置，或者指出路径配置错误，帮助开发者快速定位和解决导入相关的问题，提升调试效率。
* **支持多环境切换**：在开发过程中，开发者可能会使用多个 Python 虚拟环境。Python Path 扩展支持在不同的 Python 环境之间快速切换导入路径配置，确保在每个环境下都能正确导入所需模块，方便管理不同项目或同一项目不同阶段的依赖关系。

##### VS Code 和 PyCharm

都是常用的 Python 开发工具，但它们是**两款不同的软件**，在 VS Code 软件中**不能安装 PyCharm** ，不过在开发时可以依据需求选择使用其中一款，或者在不同场景下结合使用这两款工具，以下是它们的相关对比分析：

VS Code 和 PyCharm 结合使用的情况

* **优势**：在一些复杂的开发场景下，两者结合可以发挥各自的长处。例如，在前期代码原型设计和快速脚本编写阶段，使用 VS Code 的轻量级优势快速实现功能；在进入正式项目开发阶段，尤其是需要进行大规模代码重构、团队协作开发时，切换到 PyCharm，利用其强大的项目管理和专业的 Python 开发功能。
* **适用场景**：对于开发周期较长、需求复杂多变的 Python 项目，开发者可以根据不同阶段的需求，灵活选择使用 VS Code 或 PyCharm，以达到最佳的开发效果。

综上所述，是否在使用 VS Code 时还使用 PyCharm，取决于个人的开发习惯、项目类型和需求，并非一定要安装使用 PyCharm 。

选择 VS Code 单独使用的情况

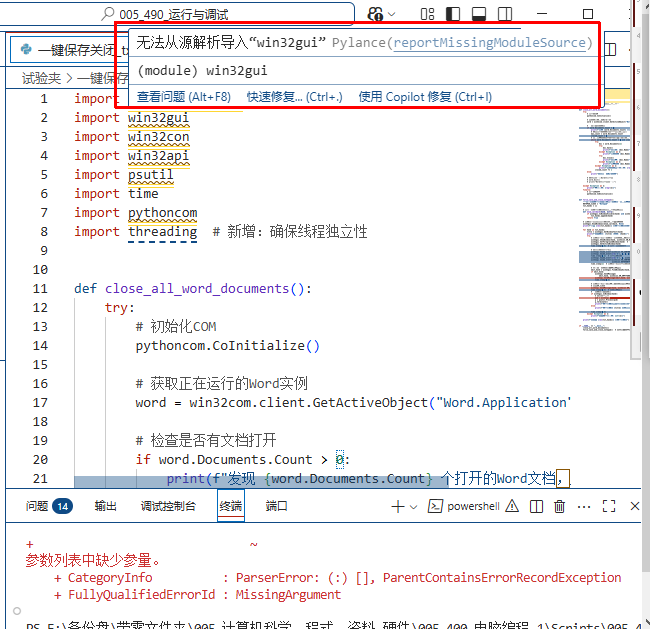
* **优势**
  + **轻量级与高扩展性**：VS Code 本身是轻量级编辑器，启动速度快，占用系统资源少。通过安装丰富的扩展（如 Python、Pylance、Python Debugger 等），可以满足 Python 开发的各种需求，无论是基础的代码编辑、调试，还是高级的代码分析、类型检查等。
  + **跨语言支持**：除了 Python，还支持 JavaScript、C++、Java 等多种编程语言，对于需要在不同语言项目间切换的开发者来说，无需频繁更换工具。
  + **高度定制化**：用户可以根据自己的喜好和工作流程，自定义界面布局、快捷键等，打造个性化的开发环境。
* **适用场景**：适合对工具轻量化有要求，或者需要同时开发多种编程语言项目，以及喜欢灵活定制开发环境的开发者。对于小型 Python 项目，或者日常快速脚本编写，VS Code 能快速启动并高效完成任务。

选择 PyCharm 单独使用的情况

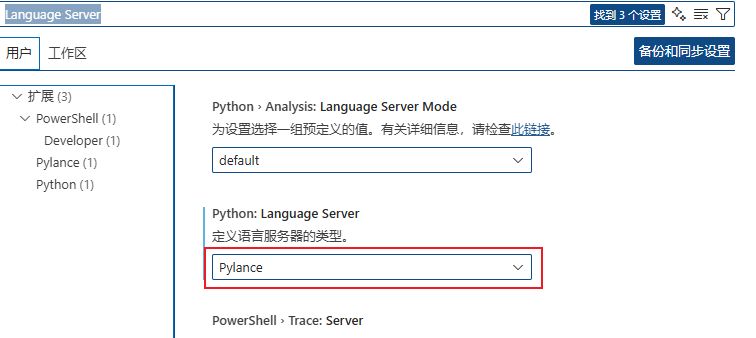
* **优势**
  + **专业的 Python 开发体验**：作为专门针对 Python 开发的集成开发环境（IDE），PyCharm 提供了非常全面且深度集成的功能。比如强大的代码导航（快速定位类、函数、变量的定义和引用）、智能代码重构（一键重命名、提取方法等）功能，能极大提高开发效率。
  + **丰富的框架支持**：对 Django、Flask、Pyramid 等 Python Web 框架，以及 NumPy、Pandas 等数据科学库有很好的支持，提供了专门的项目创建向导和框架特定的功能，方便快速搭建和开发相关项目。
  + **完善的项目管理**：内置了虚拟环境管理、版本控制集成（Git、SVN 等）等功能，能方便地管理项目依赖和代码版本，对于大型 Python 项目的协作开发非常友好。
* **适用场景**：适合专注于 Python 开发，尤其是大型 Python 项目（如企业级 Web 应用、大型数据分析项目）的开发，以及对代码质量和开发效率有较高要求的专业开发者。

### 报错解决方法

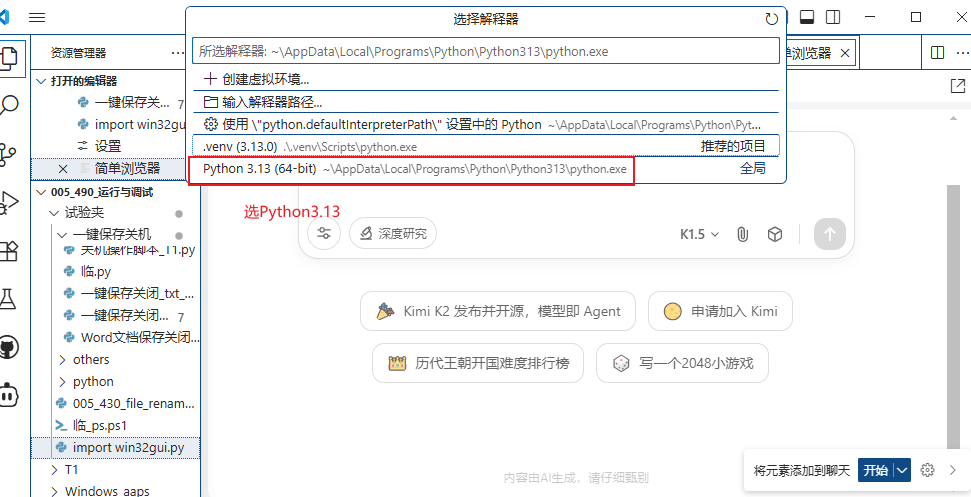
#### 源解析导入错误



Language Server选择



Python: Select Interpreter选择



## Copilot

### 1添加上下文

Copilot 里 “添加上下文” 功能，是为了让 AI 更精准理解代码场景，辅助你写代码、改问题，简单说：

1. 解决 “信息不足” 痛点

代码很复杂，单看某几行（像图里缩进有问题的 force\_save\_and\_close\_notepad() ），AI 有时搞不清全貌。点 “添加上下文”，能把更多相关代码（比如整个脚本里的函数定义、其他关联逻辑、注释说明 ）传给 Copilot，让它 “看” 到更完整的代码环境。

2. 让建议更贴合实际需求

比如你想改这段缩进，但担心改完影响其他逻辑（像和 force\_save\_and\_close\_all\_word() 有没有联动 ）。添加上下文后，Copilot 基于完整代码逻辑分析，给的修复建议（咋缩进、会不会引发新问题 ）会更准，避免 “只改表面，没考虑整体” 的情况。

3. 辅助复杂场景开发

要是写新功能，代码涉及多个模块、调用关系复杂，用 “添加上下文” 把相关类、函数、变量定义都传过去，Copilot 生成的代码（比如补全函数逻辑、写调用示例 ）会更适配你项目的实际结构，减少 “AI 瞎猜、代码接不上” 的尴尬。

简单说，就是给 Copilot “开上帝视角”，让它不光看当前几行，而是结合更多代码背景，给更靠谱的建议，

#### “屏幕截图窗口” 选项的作用和使用方式如下：

功能本质

它是为了让 Copilot 能够获取与屏幕截图操作相关的上下文信息 。当你的代码涉及到对屏幕截图功能的调用、处理截图相关逻辑（比如自动截取屏幕内容辅助办公流程、对截图进行分析等 ）时，选择该选项，能让 Copilot 更好地理解代码所处的场景，进而给出更贴合实际需求的代码建议、逻辑补全或问题解决方案 。

使用方法

点击红框中的 “屏幕截图窗口” 选项，Copilot 会将当前涉及屏幕截图操作相关的环境信息、潜在的系统交互逻辑等纳入到分析范围。之后，在编写或修改与截图功能关联的代码（像调用系统截图 API、处理截图文件保存与读取、对截图内容做识别处理等 ）时，Copilot 就能基于这些补充的上下文，更精准地辅助你，比如给出合理的函数调用示例、代码逻辑优化建议，帮你避免因对截图相关系统交互理解不足而出现的代码错误 。

图片红框内是GitHub Copilot中用于添加上下文的选项，以下是这些选项的功能及使用方法：

#### 1. \*\*打开编辑器\*\*：

- \*\*功能\*\*：选择此选项后，Copilot会将当前打开在编辑器中的所有文件内容作为上下文纳入分析范围。在涉及多个文件协同工作的项目中，能让Copilot了解不同文件间的关联和依赖，从而给出更贴合项目整体逻辑的代码建议 。

- \*\*使用方法\*\*：直接点击该选项，Copilot会自动识别并纳入编辑器内打开的所有文件信息，之后在编写或修改代码时，Copilot会基于这些补充的上下文提供辅助 。

#### 2. \*\*文件和文件夹...\*\*：

- \*\*功能\*\*：允许用户手动指定要添加为上下文的文件或文件夹。可以精准地选择与当前代码编写或问题解决相关的特定文件或整个文件夹（包含多个文件），让Copilot聚焦于相关代码库，给出更有针对性的建议。

- \*\*使用方法\*\*：点击该选项，会弹出文件或文件夹选择窗口，在其中浏览并选中目标文件或文件夹，确认选择后，Copilot就会在后续分析中考虑这些文件的内容 。

#### 3. \*\*MCP资源...\*\*：

- \*\*功能\*\*：如果项目涉及特定的MCP（可能是某种开发平台、框架或资源管理系统 ）资源，选择此选项可将相关的MCP资源信息作为上下文提供给Copilot。帮助Copilot理解项目在该特定资源环境下的运作逻辑，给出适配的代码辅助 。

- \*\*使用方法\*\*：点击该选项，系统会识别并加载与MCP相关的配置、资源文件等信息（如果已配置好相关环境 ），供Copilot分析使用。

#### 4. \*\*屏幕截图窗口\*\*：

- \*\*功能\*\*：当代码涉及屏幕截图相关的操作，如调用截图API、处理截图数据等，选择此选项可让Copilot获取与屏幕截图操作相关的上下文。帮助Copilot理解截图功能的使用场景，给出更合适的代码建议，比如正确的截图函数调用方式、截图后的数据处理逻辑等 。

- \*\*使用方法\*\*：直接点击该选项，Copilot会将截图相关的潜在上下文纳入分析，在后续编写相关代码时提供辅助 。

#### 5. \*\*指令\*\*：

- \*\*功能\*\*：将编辑器中涉及的指令相关操作和上下文提供给Copilot。比如代码中调用的特定命令行指令、编辑器的指令操作逻辑等，有助于Copilot理解代码在指令执行层面的逻辑，给出关于指令调用、参数设置等方面的合理建议 。

- \*\*使用方法\*\*：点击该选项，Copilot会自动识别并分析编辑器中与指令相关的操作和代码，在后续代码编写时提供针对性建议 。

#### 6. \*\*源代码管理...\*\*：

- \*\*功能\*\*：如果项目使用了版本控制系统（如Git ），选择此选项可将源代码管理中的信息，如分支结构、提交历史、文件修改记录等作为上下文提供给Copilot。帮助Copilot了解代码的演变过程和项目的协作情况，从而给出更符合项目开发历史和团队协作模式的代码建议 。

- \*\*使用方法\*\*：点击该选项，Copilot会读取并分析与当前项目关联的源代码管理系统中的相关信息，用于后续的代码辅助 。

#### 7. \*\*问题...\*\*：

- \*\*功能\*\*：将编辑器中检测到的问题（如语法错误、代码警告等 ）作为上下文提供给Copilot。让Copilot针对这些问题进行分析，给出修复建议或优化方案 。

- \*\*使用方法\*\*：点击该选项，Copilot会自动识别编辑器中标记的问题，并在后续提供针对性的代码修复或改进建议 。

#### 8. \*\*符号\*\*：

- \*\*功能\*\*：将当前代码文件中的各类符号（如变量名、函数名、类名等 ）及其定义和使用情况作为上下文提供给Copilot。帮助Copilot更准确地理解代码中的逻辑关系和数据流向，从而给出更贴合代码实际用途的建议 。

- \*\*使用方法\*\*：点击该选项，Copilot会分析当前文件中的符号信息，在后续代码补全、逻辑编写等方面提供更精准的辅助 。

#### 9. \*\*工具\*\*：

- \*\*功能\*\*：如果项目使用了特定的开发工具（如调试工具、测试框架等 ），选择此选项可将这些工具相关的配置和使用逻辑作为上下文提供给Copilot。使Copilot能够理解代码在工具环境下的运行机制，给出适配工具使用的代码建议 。

- \*\*使用方法\*\*：点击该选项，Copilot会识别并加载与项目关联的开发工具相关信息，在后续代码编写时考虑这些工具上下文并提供辅助 。

### 2设置

#### 2.1设置模式

这是 Copilot （或类似 AI 编码助手）的\*\*核心交互功能区\*\*，每个选项对应不同的 AI 协作模式，用法和价值如下：

##### ### 1. Agent（智能代理模式）

- \*\*功能\*\*：让 AI 深度介入开发流程，像“专属小助手”一样主动分析代码、理解需求。

- 比如你改缩进时，AI 能联动检查整个脚本的逻辑（关联函数、主程序入口），甚至帮你优化其他隐藏问题（比如函数调用顺序、资源释放逻辑 ）。

- \*\*使用方法\*\*：按 `Ctrl+Shift+I` 唤起，AI 会扫描代码上下文，自动给出“诊断+优化建议”，直接点建议就能应用修改。

##### ### 2. Ask（提问模式）

- \*\*功能\*\*：手动向 AI 提问，解决“针对性问题”。

- 比如代码报错看不懂，选 `Ask` 后输入“`force\_save\_and\_close\_notepad()` 缩进后还报错咋办？”，AI 会结合代码场景给排查思路（检查函数定义、系统资源占用 ）。

- \*\*使用方法\*\*：点击 `Ask` ，在弹出框输入问题（支持中文/英文），回车等待 AI 回答，直接复制可用代码或方案。

##### ### 3. Edit（智能编辑模式） d

- \*\*功能\*\*：让 AI 直接帮你改代码，基于你的需求“智能重构”。

- 比如你选 `Edit` 后输入“把 `force\_save\_and\_close\_notepad()` 调用优化成更安全的写法”，AI 会自动调整缩进、补全异常处理，直接替换原代码。

- \*\*使用方法\*\*：选中文本/函数 → 点 `Edit` → 输入修改指令（比如“修复缩进并加注释” ）→ AI 生成新代码后，确认替换即可。

### 4. 配置模式（隐藏扩展）

- \*\*功能\*\*：调整 AI 行为，比如“代码风格偏好（严格 PEP8 、简洁实用 ）”“是否自动补全大段逻辑”。

- \*\*使用方法\*\*：点 `配置模式...` ，在设置面板里改参数（比如把“自动补全长度”调短/调长 ），保存后 AI 会按新规则辅助编码。

\*\*一句话总结\*\*：

- 想 AI 主动帮你“体检+优化” → 用 `Agent`

- 遇到问题想“问明白” → 用 `Ask`

- 想 AI 直接替你“改代码” → 用 `Edit`

- 想让 AI 更贴合你的习惯 → 进 `配置模式`

写代码卡壳时，试试这些功能，效率直接起飞～（比如现在改缩进，用 `Agent` 扫一遍，AI 可能连其他潜在问题都顺手解决了 ）

### 2.1设置Python 类型检查模式

#### 2.1.1新手 / 快速写小脚本 → 选 Basic

适合场景：刚学 Python，写简单脚本（比如自动化小工具、临时数据处理），优先 “能跑通、少报错打断思路”。

特点：只检查最基础的类型问题（变量类型完全不匹配才报错），不严格卡类型注解、复杂校验，写代码更自由，不用纠结规范，快速出成果。

#### 2.1.2. 团队协作 / 生产项目 → 选 Standard

适合场景：多人合作的项目、要长期维护的代码（比如公司业务系统、开源库），需要 “代码规范、少埋类型坑”。

特点：比 Basic 严格，会检查函数返回类型、禁用 Any 类型滥用 、要求类型匹配更严谨，能提前发现类型不兼容问题，让代码更健壮、易读，减少后期维护时因类型混乱踩的坑。

#### 2.1.3. 追求极致规范 / 库开发 → 选 Strict

适合场景：开发 Python 库（给别人用的工具包）、对代码质量要求极高（比如金融 / 安全敏感项目），要 “类型绝对严谨、零模糊”。

特点：最严格模式，强制类型注解、禁止宽松类型写法，连微小的类型不匹配都会报错。代码会非常规范，但写起来更 “约束多、费时间”，适合追求极致质量的场景。

# 快捷键

Ctrl+D选中下一个相同的文本片段

Ctrl + Shift + L 选中所有相同的片段。

Ctrl + F2（全选当前单词的所有匹配）

Alt+Shift+F 格式化文档 编辑器会自动修复整个文件的缩进问题。

shutdown /a

保存成功3