命名规范：

使用有意义且描述性的变量、函数和类名。

变量和函数名使用小驼峰式命名法（camelCase），例如：myVariable，calculateTotal。

类名使用大驼峰式命名法（PascalCase），例如：MyClass，PersonDetails。

使用下划线命名法（snake\_case）来命名常量或全局变量，例如：MAX\_COUNT，CONFIG\_FILE\_PATH。

缩进和空格：

使用合适的缩进，通常为四个空格或者一个制表符。

在运算符前后和逗号后添加空格，例如：x = y + 1，my\_function(arg1, arg2)。

在函数和类之间留有适当的空行，以提高可读性。

注释：

使用注释来解释代码的目的、逻辑和关键部分。

在函数或方法前添加注释，描述其功能、输入参数、输出结果和可能的异常。

使用注释来标记代码中的重要步骤或需要特别注意的地方。

函数和方法：

函数和方法应当具有清晰的单一职责。

控制函数和方法的长度，避免过长的函数。

使用函数和方法来避免重复代码，提高代码的可维护性。

遵循函数和方法的命名规范，使用动词或动词短语来描述其功能。

条件和循环：

使用适当的条件语句（如if、else）和循环语句（如for、while）来控制程序流程。

在条件语句中使用括号，即使它们是可选的。

在逻辑运算符前后添加空格，例如：if (x > 5 && y < 10)。

异常处理：

使用try-catch块来处理可能的异常情况。

在catch块中捕获并适当处理异常，或者抛出新的异常并提供清晰的错误消息。

文件和目录结构：

组织代码文件的结构，使其易于浏览和维护。

为每个类或模块创建独立的文件。

使用适当的文件和文件夹命名，以反映其内容和功能。

版本控制：

使用版本控制系统（如Git）来管理代码，并定期提交代码变更。

为每个提交添加有意义的提交信息，以方便团队成员理解变更的目的和内容。

这只是一个简要的代码规范示例，可以根据团队的具体需求和项目要求进行定制。重要的是，在整个团队中保持一致性，并确保代码的可读性、可维护性和可扩展性