**Coding Standard for Medilink (Patient Appointment and Record Management System)**

**1. Introduction**

This coding standard document describes the programming conventions, best practices, and stylistic guidelines that will be followed while developing the Medilink online application.   
The goal is to produce readable, maintainable, reusable, and error-free code that adheres to consistent patterns across all modules, regardless of who produces it.   
The project will be built in Python (Django Framework), with Visual Studio Code (VS Code) as the primary IDE. All contributors must adhere to this standard throughout the project in order to ensure professional and consistent code quality.

**2. General Principles**

1. Code should be simple to read, comprehend, and maintain.  
  
 2. Always follow PEP 8, the standard Python style guide.  
  
 3. Keep functions and classes minimal and focused; each should fulfill a

specific task.  
  
 4. Always offer meaningful comments and documentation.  
  
 5. Avoid code duplication by utilizing reusable components whenever

available.  
  
 All code must be thoroughly tested before being committed to the common

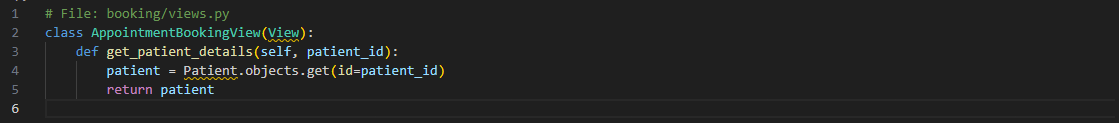
repository (Github ).

**3. Naming Conventions**

Naming is one of the most important aspects of clean code. The following rules ensure clarity and consistency.

| **Type** | **Convention** | **Example** |
| --- | --- | --- |
| **Packages / Modules** | Lowercase, underscores if needed | appointment\_module, report\_manager |
| **Classes** | PascalCase (CapitalizedWords) | PatientRecord, DoctorDashboard |
| **Functions / Methods** | snake\_case (lowercase with underscores) | book\_appointment(), generate\_invoice() |
| **Variables** | snake\_case | patient\_name, appointment\_date |
| **Constants** | ALL\_CAPS | MAX\_LOGIN\_ATTEMPTS = 5 |
| **Database Tables** | lowercase plural nouns | patients, appointments, reports |
| **HTML IDs & Classes** | kebab-case | class="patient-dashboard" |

**Example (in VS Code):**

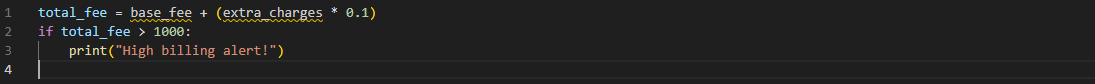


**4. Indentation and Spacing**

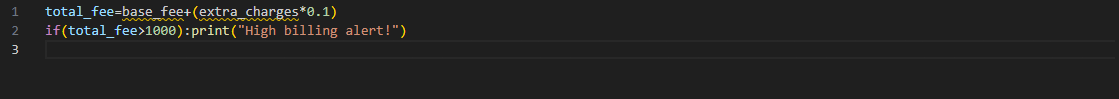
Consistent indentation enhances readability and prevents syntax problems.   
1. Use four spaces per indentation level (no tabs).   
2. Leave a blank line between function or class definitions.   
3. Limit each line to a maximum of 79 characters.   
4. Use spaces around operators and after commas, but not directly within parentheses.

* .

**Good Example:**



**Bad Example:**



**5. Commenting and Documentation**

Comments should describe why something is done, rather than simply what it does.   
Each class, function, and module should have useful docstrings.

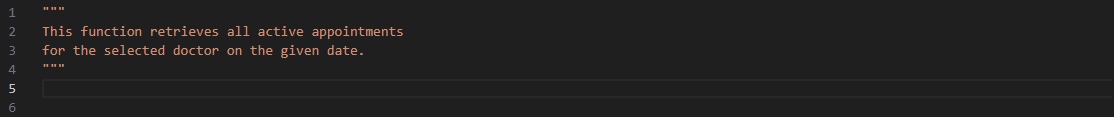
**Single-line comments**

Use # for short explanations.



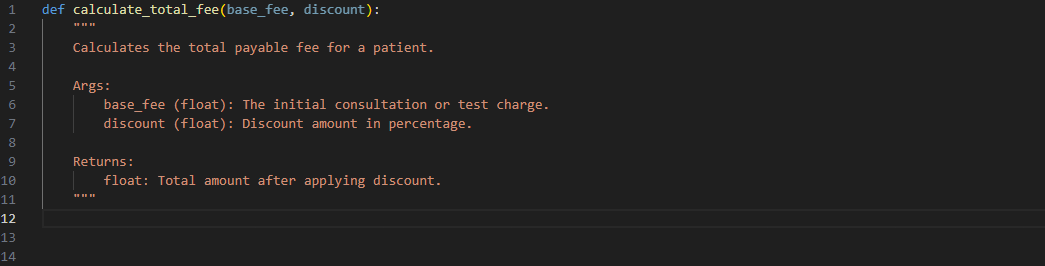
**Multi-line comments**

Use triple quotes """ """ for detailed documentation.



**Function Docstrings**

Use the normal format :



**6. File and Folder Organization**

Organize your Django project with a clear and consistent structure.

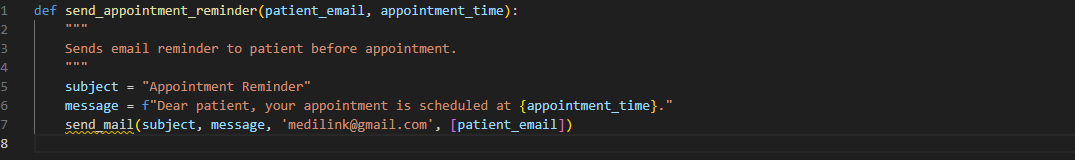
**Example Project Structure (VS Code Sidebar):**



**7. Function and Class Design**

Each function should do a single, well-defined task.   
Use descriptive names to retain modularity.

**Example:**



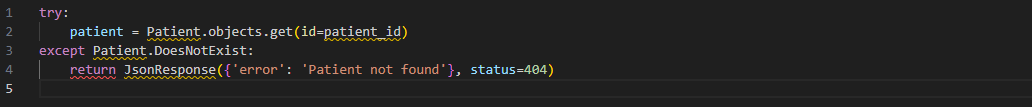
**Do Not:**

1. Limit function length to 30 lines or less.   
2. Use excessive nested loops.   
3. Create helper functions to reuse logic.

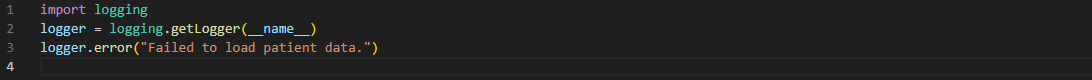
**8. Error Handling**

Exceptions should always be handled appropriately using try-except blocks. Never let the system crash due to runtime faults.

**Example:**



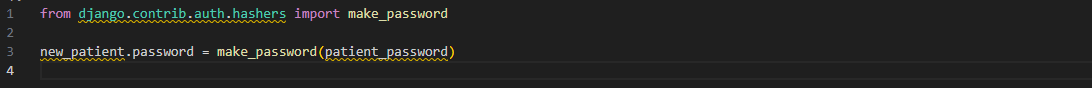
Always log critical errors for the admin to review:



**9. Input Validation and Security**

1. Validate user inputs for both frontend and backend.   
2. Sanitize data before saving to the database.   
3. Avoid relying on user input in SQL queries and instead utilize ORM.   
4. Use Django's authentication system to provide secure logins.   
5. Encrypt important data and passwords with make\_password() and check\_password().

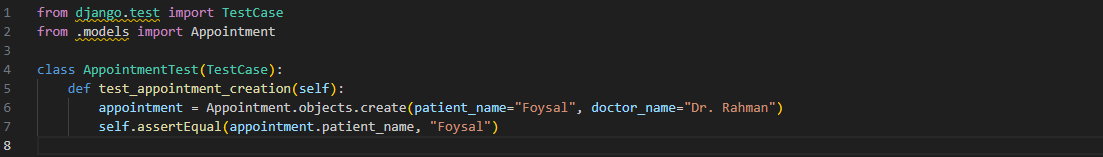
**Example:**



**10. Code Review and Testing**

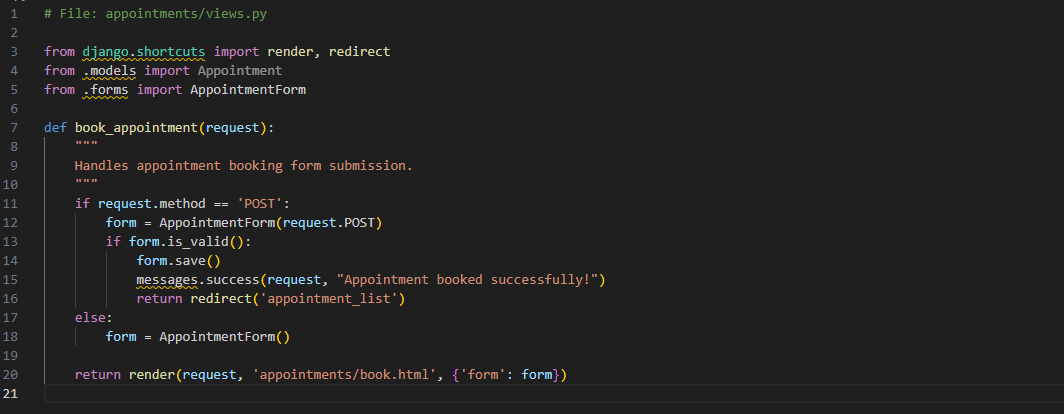
Before merging code, test all important functions locally in the VS Code terminal.   
2. Ensure all pages load successfully in browser.   
3. Check for readability and naming consistency.   
4. Run the python manage.py test for Django unit testing.

**Example Unit Test:**



**11. Coding Example (Visual Studio Code)**

Below is a small real-world example from Medilink showing proper coding style in VS Code:



**13. Conclusion**

Following this coding standard assures that the Medilink system is consistent, professional, and maintainable.   
It encourages collaboration, avoids code disputes, and ensures that the project may be readily expanded or debugged in the future.   
A well-structured codebase shows the developers' discipline — and clean code equals improved healthcare management at Medilink.