"Heaven's Light is Our Guide"

Rajshahi University of Engineering & Technology (RUET)

Department of Electrical & Computer Engineering (ECE)

Lab Report

Lab Title: Basic Documentation using Markdown .

Student Name: Humaira Tasnim Adiba

Roll Number: 2010002

Lab Session: 01
Submission Date: October 20, 2024
Course Code: ECE 3118

Submitted to:
Instructor: Oishi Joyti
Position: Lecturer
Department: Electrical & Computer Engineering (ECE), RUET

❈ *Exploring Markdown Syntax* ❈

Markdown is a *lightweight* tool that simplifies formatting for text-based documents. It's widely used for creating clean, readable documentation across platforms like *GitHub*.

🔍 *Overview of Topics*:

1. 🏷️ Headers
2. 🖌️ Text Formatting Styles
3. 📑 Lists: Ordered and Unordered
4. 🔗 Adding Links and Images
5. 🖥️ Using Code Blocks
6. ✦ Blockquotes for Emphasis
7. 🗂️ Creating Tables
8. ➖ Adding Horizontal Lines
9. ☑️ Task Lists
10. 🛠️ Utilizing Inline HTML for Customization

🏷️ *Headers*

Markdown headers are formatted with # symbols. The number of # affects the header size:

markdown

H1: Main Header

H2: Subheader

H3: Section Header

🖌️ *Text Formatting*

Markdown supports various text styles:

markdown **Bold** text for emphasis.
Italic for a softer highlight.
~~Strikethrough~~ when something is outdated.

📑 *Lists: Ordered & Unordered*

Creating lists in Markdown is super simple:

Ordered List:

markdown

1. First Item
2. Second Item
3. Third Item

Unordered List:

markdown

- Bullet Point 1
- Bullet Point 2
- Bullet Point 3

🔗 *Adding Links & Images*

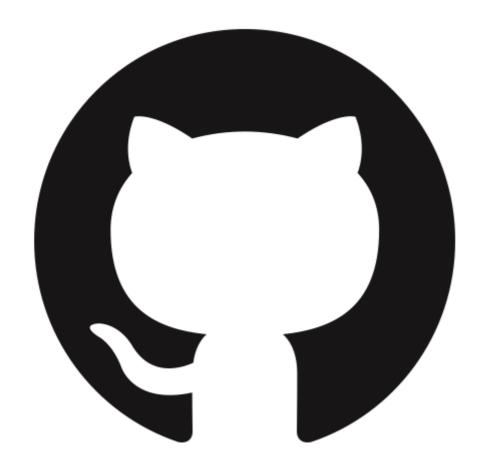Markdown enables you to embed links and images effortlessly:

markdown [Link Text](#)
Alt Text

Example:
[Visit GitHub](#)

💻 *Using Code Blocks*

Display code using either inline or block formats:

Inline Code:

markdown `This is inline code`

Block Code (e.g., JavaScript):

js // This is a block of JavaScript code function sampleFunction() { console.log("Hello, World!"); }

✨ *Blockquotes for Emphasis*

Highlight key notes or quotes with blockquotes:

markdown

> "Markdown makes formatting a breeze!"

🗂 *Tables*

Organize data into neat tables:

markdown

Header 1	Header 2	Header 3
Row 1	Data A	Data B
Row 2	Data C	Data D

➖ *Horizontal Lines*

Separate sections with horizontal lines using three hyphens:

markdown

☑ *Task Lists*

Create checklists for projects or to-do lists:

markdown

- ☑ Completed Task
- ☐ Pending Task
- ☐ Another Pending Task

🛠 *Inline HTML for Customization*

For more advanced styling, you can combine Markdown with HTML:

markdown This text is red!

🚀 Essential Coding Practices & Conventions for Software Engineers

Clean code is the foundation of reliable, scalable software. Mastering best practices ensures your code is readable, maintainable, and easy to collaborate on.

📑 Quick Links

- [What You Should Do](#) ☑️
- [What You Should Avoid](#) 🚫
- [How to Keep Code Clear](#) 👓
- [Top Tips for Cleaner Code](#) 🧼
- [Final Thoughts](#) 💡

What You Should Do ☑️

1. *Use Consistent Naming Conventions*
   Stick to patterns like camelCase for variables and PascalCase for classes. Consistency improves readability and makes your code intuitive.

   js // Example: const maxLength = 50; class UserProfile {}

2. *Keep Functions Short and Focused*
   Break large functions into smaller, single-purpose ones. Each function should do one thing and do it well.

   go // Refactor large functions into smaller, focused tasks func fetchUserData(id string) error { user := getUserByID(id) return sendUserData(user) }

3. *Comment with Purpose*
   Comments should explain why a piece of code exists, not what it does. Well-written code speaks for itself.

   python

This algorithm optimizes searches for large datasets

4. *Always Write Unit Tests*
   Unit tests ensure that each component works as expected and help prevent bugs down the line.

   go func TestCalculateSum(t *testing.T) { result := CalculateSum(2, 3) if result != 5 { t.Errorf("Expected 5 but got %d", result) } }

What You Should Avoid 🚫

1. *Avoid Hardcoding Values*
   Hardcoding magic numbers or strings limits flexibility. Use constants or configuration files to make your code adaptable.

   java // Bad: int maxAttempts = 5;

   // Good: final int MAX_ATTEMPTS = 5;

2. *Don't Overcomplicate Logic*
   If your code is too complex, refactor it into smaller, more readable functions. Avoid deep nesting of loops and conditionals.

   js // Bad: if (user.isLoggedIn && user.hasPermission && user.isAdmin) { // do something }

   // Good: if (isAuthorizedUser(user)) { // do something }

3. *Never Skip Code Reviews*
   Even the best developers make mistakes. Code reviews help catch issues early and improve overall code quality.

How to Keep Code Clear 👓

1. *Use Descriptive Variable Names*
   Choose variable names that clearly describe their purpose. Avoid vague terms or abbreviations.

   python // Good: user_email = "user@example.com"

   // Bad: ue = "user@example.com"

2. *Consistent Indentation*
   Pick an indentation style (2 spaces or 4 spaces) and stick to it across the entire codebase.

3. *Organize Related Code Together*
   Group related functions or blocks of code to make your project easier to navigate.

4. *Use Whitespace to Structure Your Code*
   Separate different sections of your code with empty lines to improve readability.

   go // Organized code with clear sections func main() { initializeApp()

```
handleRequest()

cleanup()
```

   }

Top Tips for Cleaner Code 🎨

1. *Document Key Functions*
   Add brief, meaningful documentation to important functions and public APIs. This helps others quickly understand the purpose of your code.

   js /**

   - Calculate the total cost of items.
   -
   - @param {Array} items - List of items.
   - @returns {Number} - Total cost. */ function calculateTotal(items) { //... }

2. *Follow the DRY Principle*
   Don't Repeat Yourself (DRY). Refactor duplicated logic into reusable functions.

   go // Bad: Duplication func addUser() { ... } func updateUser() { ... }

   // Good: Reuse logic func saveUser() { ... }

3. *Handle Errors Gracefully*
   Always provide meaningful error messages to help with debugging. Avoid using generic error handling methods like panic().

   go // Bad: err := saveUser() if err != nil { panic(err) }

   // Good: err := saveUser() if err != nil { log.Printf("Failed to save user: %v", err) return }

Final Thoughts 💡

- *Use Linters & Formatters*
  Automate code quality checks with tools like linters to maintain consistent style and catch common issues.

- *Write Meaningful Commit Messages*
  Clearly describe what each commit changes. Example: feat: added user login feature

- *Refactor Regularly*
  Continuously refactor your code to improve clarity and performance.

- *Focus on Clarity First, Performance Second*
  Write clean, understandable code before optimizing for performance.