Intro

- Name
- Role (day-to-day)
- Github Copilot, Agents [AI development] (1-5) (1-5) (1-5)

→ what do you want to learn?

Breaks

2 Coffee breaks: (10:30) AM — 15 min, (2:30) PM —

Lunch break — 11:30 am — 1 hr

Models (Foundation Models)

↳ Pre-trained models (GPT, OpenAI, on large volume & Data Architecture, Genuine AI) (Text, file, video, audio) — Multi Modal

[VSC + Github MC (copilot)]

- How — Github copilot — Help you code
- various ways to bigger Github copilot
- Configure & troubleshoot?

Features:

1. Copilot for chat — inline chat → ctrl+I; chat screen.
2. Copilot for pull requests
3. Copilot for CLI

Internet with Copilot

1. Inline suggestions
   - start typing — suggestions — ↵ enter → Tab key; Cancel → Esc key
2. Perform Complex tasks
   - fix the code
   - Document the code
   - generate Unit Tests
   - generate the code review.

Basic JS web app

↳ Task List (Todo) — Add / Complete / Delete

Pre-req:

1. VS Code
2. Node.js
3. Extensions: Github Copilot, Github Copilot chat
4. Github MC

Project folder

```
Copilot-webapp
├ index.html
├ style.css
└ app.js
```

Prompts — 13 elements

1. "Instruction" → Task
2. "Context" → what
3. Input → more details
4. Output → what?

→ Model (Role, Task, Context, Input, Output, Instruction, Example, Length, Tone, Format)

Models → Providers

1. FMs (subset of gpt)
   - Text Summarization
   - Text Creation
   - Image Creation
   - Conversation
   - Video Creation
   - Video Summary
2. "LLMs" (Large Language Model)
   - Text summarization
   - Create a new Content (Code)
   - Conversation (new)

Prompt → App (chatbot) → FCS
- PDF
- Tube
- Text
→ Business Users
→ customer

"Context"? → Model (LLM) → (Reason in Reasoning)

↳ Private session — You & Model.

User ↔ App (chatbot)

Refactoring Code using Github Copilot

1. Prep: Make your Copilot refactor safe.

   Refactor = small, slices

2. use the Github Copilot Chat to create refactor plans

   ✓ Refactor this code without changing behavior → Instruction
   → Goals: readability, smaller functions, clearer naming, remove duplication. ← Context
   → Constraints: keep public API the same, don't add new dependencies, keep it fast
   → Find, propose a step-by-step refactor plan, then apply it in small commits.

3. Refactor using "Selection → Chat
   ↳ Inline chat (Ctrl+I)

   Prompt: Extract functions with clear names (Extract functions) keep logic identical (code) Return the refactored code

   Slash commands: — (Inline chat) / chat → current file

   - ✓ /fix = fix the error in the code
   - ✓ /explain = explain the code
   - ✓ /tests = create the unit tests for the selected code
   - /optimize = optimize the selected code / file

Legacy Code — C, C++, Cobol, etc

→ Repo → Project folder in VSC

1. open the Project folder in VSC
2. Github Copilot Chat →
   - ✓ I don't know <fill> and explain me everything which includes run the code, debug the code, create the test, install / setup tests framework
   - 2. Examine the code, identify the scope of optimization, remove duplicate, fix error

FMs / LLMs → what Model to use?

Model → output/response (Token)

Input (Token) → Model (While a code to form two functions) → token → token

Model: input tokens → output tokens

Hands-on (Flask)

Project folder

```
app.py (server)
templates
└ index.html
static
├ app.js
└ style.css
```

→ frontend

Unit Tests — Github Copilot?

⇒ chat view, inline chat, code completions.

? Examine the unit testing tools & environment

Inv: Copilot Chat —

1. Set up testing framework → /setupTests
2. Generate test cases — unit tests, integration tests, end-to-end tests
3. Handle edge cases
4. Fix failing tests = /fixTestFailure
5. Maintain consistency

Buggy → Expense Tracker

```
↳ Readme.md
↳ expense-tracker
  ↳ app.py
  ↳ storage
  ↳ models
  ↳ utils
Tests
```