Inicio

1. Presentación
2. Introducción al tema repaso de la clase anterior
3. Objetivos
    a. **Primero**: comprender las herramientas de las que dispongo para trabajar con copilot.
    b. **Segundo**: utilizar diferentes estrategias de prompting para debugging y generación de código.
    c. **Tercero**: crear una app sencilla que me permita interactuar con un modelo de Openai a través de un endpoint utilizando FastAPI.

Entorno de Trabajo

4. Github:
    a. Introducción a la arquitectura de trabajo(perfil del que toma las formas en que sugiere código según como lo he hecho antes)
    b. Diferentes formas de dar contexto
    c. Estrategias de Prompting para generación de código
    d. Extensiones útiles
        i. Github Copilot
        ii. Github Copilot Chat
    e. Copilot Contextual (Ctrl + I):
        i. /fix (predispone al modelo a reparar en el system), cuanto más contexto le demos para arreglar, mejor va a actuar. Óptimo para pequeños trozos de código
        ii. Chat para refactorizar grandes porciones de código
    f. Cómo lee Copilot el repositorio (archivos abiertos, comentarios, etc.). Siempre es mejor comenzar con una planificación donde el modelo vea la totalidad del repo
    g. Comenzar con un flujograma o explicando lo que la función debería hacer es ideal para que comprenda el objetivo.
    h. Manejo de historial para ordenar
    i. Comandos útiles
5. FastAPI:
    a. Breve Explicación de FastAPI: una de las herramientas más extendidas para servir modelos a través de endpoints.
    b. Ejercicio práctico:
        i. Crear un endpoint @*app.post("/chat")* para interactuar con la API de GPT utilizando FastAPI (se puede hacer con chat, comentarios, Agent o Edit utilizando diferentes modelos)

Recursos:

c. Prompt Engeneering: Tecnicas y mejores practicas para generar codigo:
    i. CoT prompting

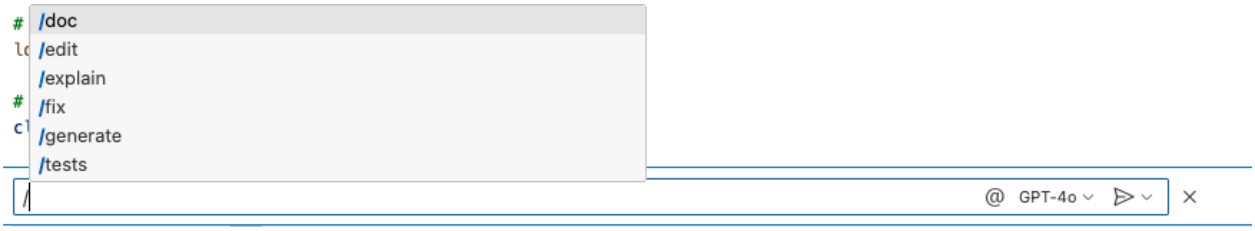
https://drive.google.com/file/d/1AbaBYbEa_EbPelsT40-vj64L-2IwUJHy/view?pli=1

0:00 - 0:10 | Bienvenida e introducción

- Presentación
- Objetivos de la clase:
    d. **Primero**: comprender las herramientas de las que dispongo para trabajar con copilot.
    e. **Segundo**: utilizar diferentes estrategias de prompting para debugging y generación de código.
    f. **Tercero**: crear una app sencilla que me permita interactuar con un modelo de Openai a través de un endpoint utilizando FastAPI.
- Repaso de lo que haremos

0:10 - 0:30 | Introducción a GitHub Copilot para interactuar con los modelos - Tips y comandos

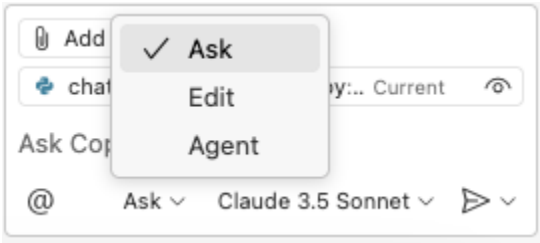
- Cómo utilizar Copilot para desarrollo, debugging y refactorización
- Demostración: Copilot Chat, /fix, sugerencias en línea



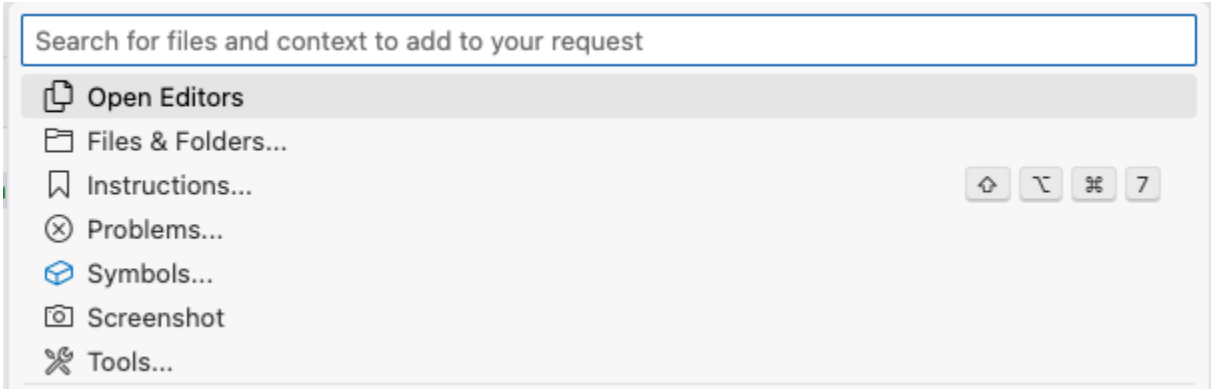
Cuando seleccionamos una porción de código podemos utilizar los comandos Cmd + I (en Mac) o Ctrl + I (en Windows) para abrir la modalidad inline de la herramienta en Code Space. Allí podemos utilizar estos atajos para predisponer a nuestro modelo a una tarea en particular:

- **/doc**: Genera o muestra documentación para el código seleccionado o el archivo activo.
- **/edit**: Permite editar o modificar el código seleccionado según una instrucción dada.
- **/explain**: Explica el funcionamiento del código seleccionado o del archivo activo.
- **/fix**: Sugiere o aplica correcciones a errores detectados en el código seleccionado.
- **/generate**: Genera nuevo código, funciones, clases o archivos según una instrucción.

- **/tests**: Crea pruebas unitarias o de integración para el código seleccionado.



Junto al botón de enviar, tienes un menú desplegable con tres opciones:
• **Ask**: para hacer preguntas (modo por defecto). Ideal para debugging, explicación de errores, sugerencias de código.
• **Edit**: para pedirle que modifique código automáticamente.
• **Agent**: para tareas más complejas o interactivas (puede implicar múltiples pasos).



En la parte superior de la ventana de Copilot Chat verás el botón “Add Context…”. Al hacer clic:
• Puedes seleccionar archivos abiertos, carpetas, instrucciones, problemas de código, símbolos, etc.
• Esto le da más información a Copilot sobre el proyecto actual para que sus respuestas sean más relevantes.

Una vez que agregaste contexto (como un archivo .py), verás ese archivo asociado a la consulta. Esto es útil para pedirle a Copilot que:

• Revise el código completo.
• Sugiera mejoras o refactorización.
• Explique por qué un error ocurre.

Prompting Tips:

✅ Usa comentarios detallados sobre lo que el código debería hacer.
✅ Especifica qué tipo de error estás tratando de resolver.
✅ Sé claro, específico y directo: Copilot responde mejor cuando entiende el propósito de la función
Evita esto: # Arregla esto
Prefiere esto: # Corrige esta función para que devuelva el área de un triángulo
Reescribe prompts ambiguos o genéricos:
Evita esto: “Hazlo mejor”
Prefiere esto: “Refactoriza esta función para mejorar legibilidad y evitar duplicación de lógica”
✅ Usa comentarios con contexto (Contextual Prompting):
Ejemplo:
# La siguiente función debería devolver la suma de dos enteros positivos:
def sumar(a, b):
return a – b
Luego puedes usar Copilot Chat con /fix o /explain para ayudar al modelo a identificar la contradicción entre comentario y código.
✅ Usa prompting de rol o sistema cuando sea posible:
Ejemplo:
«Actúa como un ingeniero de software senior. Revisa este código y sugiere mejoras.»
✅ Usa Chain of Thought para depuración compleja cuando el error no es evidente, invita al modelo a razonar paso a paso, esto ayuda con errores semánticos ocultos o fallos lógicos sutiles.
Ejemplo:
Explica paso a paso por qué esta función lanza un error en tiempo de ejecución y cómo solucionarlo.
✅ Usa ejemplos (few-shot prompting): Si ves que Copilot no interpreta correctamente tu necesidad, podés proporcionarle un ejemplo directamente en el comentario, esto aclara el patrón que debe seguir:
Ejemplo:
# Entrada: a = 3, b = 5
# Salida esperada: 8
def sumar(a, b):
return a + b

Guia: https://drive.google.com/file/d/1AbaBYbEa_EbPelsT40-vj64L-2IwUJHy/view?pli=1

0:30 - 0:45 | Ejercicio práctico guiado: depuración

- Código con errores intencionales
- Identificación y corrección con Copilot
- Evaluación de soluciones

0:45 - 1:00 | Aplicación práctica: chatbot con FastAPI

- Qué es FastAPI y su potencialidad
- Breve demostración de /docs

1:05 - 1:25 | Creación de Chatbot básico con Github Copilot
- Creación de endpoints básicos con Copilot utilizando FastAPI:
- Utilizar API de Openai para acceder al modelo GPT

Prompt:

Crea una API en Python utilizando FastAPI que funcione como un chatbot usando la API de OpenAI. El objetivo es obtener una demostración simple, no complejizar demasiado. El endpoint debe llamarse `/chat` y aceptar un mensaje del usuario en el cuerpo de la solicitud. El endpoint debe enviar ese mensaje al modelo de OpenAI (por ejemplo, GPT-4) y devolver la respuesta generada. Lee la clave de API de OpenAI desde un archivo .env usando `python-dotenv`. Agrega instrucciones breves sobre cómo probar el endpoint ejecutando uvicorn como servidor y curl para la demostración.

- Propuesta de validaciones y mejoras con Copilot
- Uso de comentarios y contexto para refactorización

1:25 - 1:30 | Cierre y feedback
- Tips finales para usar Copilot
- Recursos recomendados
- Preguntas