Sesión 3 — Código con IA

- Asistentes de código: ChatGPT, Claude, Copilot
- Generación y explicación de funciones
- Validación y pruebas

Flujo recomendado

1. Especificación clara
2. Implementación asistida
3. Pruebas y refactorización

Demostración

- Escribir funciones puras
- Pedir tests y documentación

Buenas prácticas de código (rápido)

- Nombres descriptivos y consistentes
- Funciones pequeñas y puras; manejo explícito de errores
- Type hints en interfaces públicas
- Evitar estados globales; separar I/O de lógica

Docstrings y documentación

- Formatos comunes: Google / NumPy / reStructuredText
- Docstring de alto nivel + ejemplos de uso
- Anotar precondiciones, postcondiciones y errores

Ejemplo:

```
def calcular(expr: str) -> float:
    """Evalúa una expresión aritmética simple.

    Args:
        expr: Expresión con +, -, *, / y paréntesis.

    Returns:
        Resultado como número de punto flotante.

    Raises:
        ValueError: Si la expresión es inválida.
    """
```

Pruebas (pytest) y TDD

- Diseñar casos: nominales, bordes, errores
- Escribir tests antes y después de refactorizar
- Mantener tests rápidos y deterministas

Comandos clave:

```
pytest -q
pytest -q -k calcular
```

Calidad estática: mypy, flake8, ruff

- mypy: chequeo estático de tipos
- flake8: estilo y errores comunes
- ruff: linting/format rápido (puede reemplazar flake8 en muchos casos)

Comandos clave:

```
mypy sesiones/03-codigo-con-ia/ejercicios
flake8 sesiones/03-codigo-con-ia/ejercicios
ruff check sesiones/03-codigo-con-ia/ejercicios
```

Sugerencia: corregir warnings primero en el dominio de la función central.

Prompting para calidad

- "Genera 5 tests que cubran casos borde y errores explícitos"
- "Añade docstrings estilo Google y type hints sin cambiar la lógica"
- "Propón refactorizaciones que reduzcan complejidad cognitiva (<10)"

Checklist de entrega

- Tests pasan (pytest)
- Tipos ok (mypy)
- Linter ok (ruff/flake8)
- Docstrings y README actualizados