Curso GitHub Copilot - Nivel Medio-Alto

Módulo 1: Fundamentos Avanzados de Copilot

1.1 Arquitectura y Funcionamiento

- **Modelo subyacente**: Basado en OpenAI Codex
- **Entrenamiento**: Código público de GitHub y documentación
- **Limitaciones**: No acceso a internet en tiempo real, corte de conocimiento
- **Privacidad**: Tu código no se almacena ni se usa para reentrenar el modelo

1.2 Configuración Avanzada

```
// settings.json en VS Code
{
  "github.copilot.enable": {
    "*": true,
    "yaml": false,
    "plaintext": false
  },
  "github.copilot.editor.enableAutoCompletions": true,
  "github.copilot.advanced": {
    "secret_key": "your_key_here",
    "listCount": 10,
    "inlineSuggestCount": 3
  }
}
```

Módulo 2: Técnicas de Prompting Avanzado

2.1 Prompts Contextuales Efectivos

```
# MALO: Prompt vago
# def process_data():

# BUENO: Prompt específico con contexto
def process_user_analytics_data(raw_json_data):
    """
    Procesa datos de analytics de usuarios desde API
    - Filtra usuarios activos (último login < 30 días)
    - Calcula métricas de engagement
    - Retorna DataFrame con columnas: user_id, engagement_score, last_activity
    """
```

2.2 Uso de Comentarios Estratégicos

```
// Patrón Observer para notificaciones en tiempo real
// Debe manejar múltiples tipos de eventos: user_login, purchase, error
// Implementar rate limiting para evitar spam
class NotificationManager {
```

2.3 Prompts Multi-paso

```
# Paso 1: Definir la estructura de datos
# Paso 2: Implementar validación con Pydantic
# Paso 3: Agregar métodos de serialización/deserialización
# Paso 4: Implementar tests unitarios

from pydantic import BaseModel, validator
from typing import Optional, List
from datetime import datetime
```

Módulo 3: Patrones de Desarrollo con Copilot

3.1 Desarrollo Dirigido por Comentarios (CDD)

```
class PaymentProcessor:
    """
    Procesa pagos con múltiples proveedores (Stripe, PayPal, Square)
    Implementa retry logic y fallback entre proveedores
    Registra todas las transacciones para auditoría
    """

    def __init__(self, primary_provider: str, fallback_providers: List[str]):
        # Copilot generará la implementación basada en estos comentarios
        pass
```

3.2 Refactoring Asistido

```
# Antes: Código monolítico
def handle_user_request(request_data):
    # TODO: Separar en funciones más pequeñas
    # 1. Validar entrada
    # 2. Procesar lógica de negocio
    # 3. Formatear respuesta
    # 4. Logging y métricas
    pass
```

3.3 Generación de Tests

```
def calculate_compound_interest(principal, rate, time, compound_frequency):
    """Calcula interés compuesto con diferentes frecuencias"""
    return principal * (1 + rate/compound_frequency) ** (compound_frequency * time)

# TODO: Generar tests para:
# - Casos normales
# - Casos edge (rate=0, time=0, principal=0)
# - Diferentes frecuencias de capitalización
# - Validación de tipos de entrada
```

Módulo 4: Optimización y Productividad

4.1 Shortcuts y Comandos Clave

- `Ctrl+Enter`: Aceptar sugerencia
- `Alt+]`: Siguiente sugerencia
- `Alt+[`: Sugerencia anterior
- `Ctrl+Shift+P` → "Copilot: Open Completions Panel"

4.2 Copilot Chat Avanzado

```
/explain: Explica el código seleccionado
/fix: Sugiere correcciones para bugs
/optimize: Optimiza el código seleccionado
/tests: Genera tests unitarios
/doc: Genera documentación
```

4.3 Integración con Flujos de Trabajo

```
# .github/workflows/copilot-review.yml
name: Copilot Code Review
on: [pull_request]
jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Copilot Code Review
        uses: github/super-linter@v4
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Módulo 5: Casos de Uso Avanzados

5.1 Desarrollo Full-Stack

```
// Frontend: React + TypeScript
interface UserProfile {
  id: string;
  email: string;
  preferences: {
    theme: 'light' | 'dark';
    notifications: boolean;
  };
}

// TODO: Crear componente UserProfileCard
// - Mostrar avatar, nombre, email
// - Botón para editar preferencias
// - Integrar con contexto de autenticación

# Backend: FastAPI
from fastapi import FastAPI, HTTPException, Depends
from sqlalchemy.orm import Session

# TODO: Crear endpoint CRUD para UserProfile
# - GET /users/{user_id}/profile
# - PUT /users/{user_id}/profile
# - Validación con Pydantic
# - Autenticación JWT
```

5.2 DevOps y Automatización

```
# Multi-stage build para aplicación Python
# Stage 1: Build dependencies
# Stage 2: Runtime optimizado
# Incluir health checks y non-root user
FROM python:3.11-slim as builder

# docker-compose.yml para desarrollo
# Servicios: app, database, redis, nginx
# Volúmenes para hot reload
# Variables de entorno por archivo .env
version: '3.8'
services:
```

5.3 Machine Learning y Data Science

```
import pandas as pd
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# TODO: Pipeline de ML para predicción de churn
# 1. Cargar y limpiar datos de usuarios
# 2. Feature engineering (engagement metrics, usage patterns)
# 3. Entrenamiento con validación cruzada
# 4. Evaluación y métricas de negocio
# 5. Deployment con MLflow
```

Módulo 6: Mejores Prácticas y Limitaciones

6.1 Cuándo NO usar Copilot

- Código que maneja secretos o datos sensibles
- Algoritmos críticos de seguridad
- Lógica de negocio muy específica del dominio
- Debugging de problemas complejos

6.2 Revisión de Código Generado

```
# SIEMPRE revisar:
# 1. Lógica de negocio correcta
# 2. Manejo de errores apropiado
# 3. Seguridad (validación de entrada, escape de datos)
# 4. Performance (complejidad algorítmica)
# 5. Compatibilidad con el stack existente

def user_authentication(username: str, password: str):
    # ⚠️ REVISAR: Hash de contraseña, rate limiting, logging de seguridad
    pass
```

6.3 Mantenimiento de Prompts

```
# Documentar patrones que funcionan bien
# Crear biblioteca de prompts reutilizables
# Mantener consistencia en el equipo

# Patrón: Microservicio con FastAPI
"""
Crear microservicio FastAPI para {funcionalidad}:
- Estructura MVC clara
- Validación con Pydantic
- Documentación automática
- Logging estructurado
```

```
- Health checks
- Containerización
"""
```

Módulo 7: Copilot en Diferentes Lenguajes

7.1 JavaScript/TypeScript

```
// Copilot destaca en:
// - React components y hooks
// - Express.js APIs
// - Type definitions
// - Async/await patterns

interface ApiResponse<T> {
  data: T;
  status: 'success' | 'error';
  message?: string;
}

// TODO: Generic HTTP client con retry y caching
```

7.2 Python

```
# Fortalezas:
# - Data science libraries
# - Web frameworks (Django, FastAPI)
# - Scripts de automatización
# - Machine learning pipelines

from dataclasses import dataclass
from typing import Protocol

# TODO: Repository pattern para acceso a datos
# Debe ser genérico y soportar múltiples backends
```

7.3 Otros Lenguajes

- **Go**: Excelente para microservicios y concurrencia
- **Rust**: Bueno para sistemas de bajo nivel
- **Java**: Spring Boot y enterprise patterns
- **C#**: .NET Core y Azure integrations

Módulo 8: Proyecto Final

Ejercicio Integrador: E-commerce Microservices

Desarrollar sistema de e-commerce con:

1. **User Service** (Python/FastAPI)
   - Autenticación JWT
   - Gestión de perfiles
   - Rate limiting

2. **Product Service** (Node.js/Express)
   - Catálogo de productos
   - Búsqueda y filtros
   - Cache con Redis

3. **Order Service** (Go)
   - Procesamiento de pedidos
   - State machine
   - Event sourcing

4. **Payment Service** (Python)
   - Múltiples proveedores
   - Webhook handling
   - Retry logic

5. **Infrastructure**
   - Docker Compose
   - API Gateway
   - Monitoring
   - CI/CD Pipeline

Recursos Adicionales

Documentación Oficial

- GitHub Copilot Docs
- VS Code Extension

Comunidad y Mejores Prácticas

- GitHub Copilot Community Discussions
- Stack Overflow: `github-copilot` tag
- YouTube: GitHub Copilot tutorials

Herramientas Complementarias

- **Copilot X**: Chat integrado en IDE

- **Copilot for Business**: Funciones empresariales
- **CodeWhisperer**: Alternativa de Amazon
- **TabNine**: Competidor con modelos locales

Evaluación Final

Criterios de Evaluación

1. **Eficiencia**: Reducción del tiempo de desarrollo
2. **Calidad**: Código generado vs código manual
3. **Comprensión**: Explicación de sugerencias de Copilot
4. **Adaptación**: Modificación de prompts según contexto
5. **Juicio crítico**: Cuándo aceptar/rechazar sugerencias

Certificación

Al completar este curso serás capaz de:

- Utilizar Copilot como herramienta de productividad avanzada
- Escribir prompts efectivos para diferentes escenarios
- Integrar Copilot en flujos de trabajo complejos
- Evaluar críticamente el código generado
- Enseñar mejores prácticas a otros desarrolladores