

MÓDULO 2

Instalación y uso eficiente de herramientas

CLI · IDE Extension · App · Cloud

Plan de Formación — OpenAI Codex para Desarrolladores

Nivel: Introductorio-Intermedio · Duración estimada: 3–4 horas

Versión 1.0 — Febrero 2026

Índice

1. Introducción

En el Módulo 1 establecimos que Codex es un agente autónomo de desarrollo, no un simple chat. En este segundo módulo pasamos a la acción: instalación, configuración y dominio operativo de las cuatro superficies de Codex. El objetivo no es solo “que funcione”, sino que funcione de forma estándar, reproducible y segura para todo el equipo.

El hilo conductor es la configuración por capas de Codex: un sistema diseñado para que las preferencias del usuario, las políticas del proyecto y los requisitos de la empresa coexistan sin conflictos. Dominar este sistema es la clave para que Codex se comporte de forma predecible independientemente de quién lo ejecute o desde dónde.

2. Objetivos de aprendizaje

#	Objetivo	Evidencia de logro
O1	Instalar y operar Codex CLI y la extensión IDE de forma funcional.	El alumno ejecuta tareas desde ambas superficies con el mismo resultado.
O2	Entender y configurar el sistema de capas de config.toml (usuario, proyecto, sistema, perfiles, CLI flags).	El alumno ejecuta codex config show --effective y explica cada capa.
O3	Saber cuándo usar la App de escritorio y cuándo Codex Cloud según el escenario.	El alumno selecciona la superficie adecuada para 3 escenarios distintos.
O4	Crear una configuración de proyecto reproducible y compatible con el equipo.	El alumno entrega un .codex/config.toml versionable con políticas estándar.

3. Contenidos teóricos

3.1 Codex CLI: agente local en tu terminal

3.1.1 Instalación

Codex CLI es una herramienta open source (github.com/openai/codex) construida en Rust que se distribuye como paquete npm. Requiere Node.js 22+ como runtime.

```
# Instalación global
npm i -g @openai/codex

# Verificar instalación
codex --version

# Primera ejecución (inicia flujo de autenticación)
codex
```

Plataformas soportadas:

Plataforma	Estado	Sandbox nativo	Notas
macOS (ARM/Intel)	Estable	Seatbelt (sandbox-exec)	Experiencia de referencia
Linux (x86_64/ARM)	Estable	Landlock + Bubblewrap (bwrap)	Ideal para CI/CD y servidores
Windows (nativo)	Experimental	Restricted token (AppContainer)	Sandbox limitado; preferir WSL2
Windows (WSL2)	Estable	Landlock (kernel Linux)	Mantener repos en ~/code/, no en /mnt/c/

3.1.2 Autenticación

Codex ofrece dos métodos de autenticación:

- **ChatGPT OAuth (recomendado):** Al ejecutar `codex` por primera vez, se abre el navegador para login con tu cuenta ChatGPT. Los planes Plus, Pro, Business, Edu y Enterprise incluyen acceso a los modelos Codex sin coste API adicional.
- **API key:** Usar `OPENAI_API_KEY` como variable de entorno o autenticarse con `codex login --with-api-key`. Se cobra por tokens consumidos según pricing del modelo.

Las credenciales se almacenan localmente según la configuración de `cli_auth_credentials_store`: file (auth.json, por defecto), keyring (OS keychain) o auto (keyring si está disponible, si no file).

💡 Consejo

En entornos headless (servidores CI), usa `codex login --device-code` como fallback: genera un código de dispositivo que introduces en el navegador de otra máquina.

3.1.3 Modos de operación del CLI

Modo	Comando	Descripción	Caso de uso
------	---------	-------------	-------------

Interactivo (TUI)	codex	Interfaz terminal a pantalla completa, conversacional	Desarrollo diario, pair programming
Full-auto	codex --full-auto	workspace-write + on-request approvals	Tareas bien definidas con confianza
Ejecución no interactiva	codex exec "tarea"	Sin TUI, output a stdout, scriptable	CI/CD, automatizaciones, scripts
Cloud desde CLI	codex cloud exec	Lanza tarea en entorno cloud remoto	Tareas largas sin consumo local
Reanudar sesión	codex resume --last	Continua última conversación guardada	Retomar trabajo tras interrupción
Bifurcar sesión	codex fork --last	Crea nuevo thread preservando historial	Explorar alternativas sin perder original

3.1.4 Comandos y atajos esenciales del TUI

Acción	Cómo hacerlo
Buscar ficheros en workspace	@ + escribir nombre → Tab/Enter para insertar
Invocar skill	\$ + nombre del skill
Ejecutar comando shell local	! + comando (ej. !ls, !git status)
Inyectar instrucción durante ejecución	Enter mientras Codex trabaja (mid-turn steering)
Encolar follow-up sin interrumpir	Tab mientras Codex trabaja
Editar mensaje anterior	Esc Esc (componer vacío), seguir con Esc para navegar
Abrir editor externo para prompt largo	Ctrl+G (usa \$VISUAL o \$EDITOR)
Cambiar modelo	Comando /model
Revisar diff	Comando /review
Ver estado del workspace	Comando /status
Cambiar aprobaciones	Comando /permissions o /approvals

3.2 Extensión IDE: Codex integrado en tu editor

3.2.1 Instalación y configuración

La extensión oficial está disponible en el VS Code Marketplace (publisher: OpenAI, ID: openai.chatgpt) y es compatible con VS Code, Cursor, Windsurf y otros editores basados en VS Code. Para JetBrains (IntelliJ, PyCharm, WebStorm, Rider), existe una integración dedicada.

```
# VS Code: instalar desde línea de comandos
code --install-extension openai.chatgpt

# O buscar "Codex" en el panel de Extensions (Ctrl+Shift+X)
# Reiniciar VS Code si no aparece inmediatamente
```

Tras instalar, el ícono de Codex aparece en la barra lateral. Al hacer clic se abre el panel de chat. La autenticación se realiza mediante el botón “Sign in with ChatGPT” (o API key).

i Dato clave

CLI e IDE Extension comparten la misma configuración. Ambos leen `~/.codex/config.toml` y los `.codex/config.toml` del proyecto. Cambiar un valor en `config.toml` afecta a ambas superficies.

3.2.2 Los tres modos de la extensión

Modo	Capacidades	Aprobación	Cuándo usarlo
Chat	Solo conversación. No lee, escribe ni ejecuta.	No aplica	Consultas rápidas, explicaciones de código
Agent (default)	Lee ficheros, escribe cambios, ejecuta comandos	Pide aprobación según política	Desarrollo diario (recomendado)
Agent (Full Access)	Todo lo anterior sin pedir aprobación + red	Ninguna	Tareas de confianza en repos conocidos

3.2.3 Contexto automático en el IDE

La extensión inyecta automáticamente en el contexto de Codex:

- Los ficheros actualmente abiertos en el editor.
- Las selecciones de código que tengas activas.
- Los ficheros referenciados explícitamente con `@fichero.ts` en el prompt.

Esto significa que puedes escribir prompts más cortos que en CLI, porque Codex ya “ve” tu código abierto:

```
# En CLI necesitarías ser explícito:
"Refactoriza la función processOrder en src/services/order.ts"

# En IDE, con el fichero abierto, basta con:
"Refactoriza processOrder para usar async/await"
# Codex ya sabe qué fichero es porque está abierto
```

3.2.4 Delegación IDE → Cloud

Desde la extensión puedes delegar tareas largas a Codex Cloud sin salir del IDE. Codex recuerda el contexto de la conversación local y lo traspasa al cloud. Cuando la tarea cloud termina, puedes revisar los diffs, pedir follow-ups y aplicar los cambios localmente para testear.

3.2.5 Configuración específica de la extensión

Se accede desde el icono de engranaje en la esquina superior del panel Codex:

- Codex Settings > Open config.toml: abre directamente el fichero de configuración compartido.
- MCP Settings: gestionar servidores MCP.
- Keyboard Shortcuts: ver y personalizar atajos de teclado.
- Reasoning effort: selector bajo el input de chat (low / medium / high).
- Model selector: dropdown bajo el input para cambiar de modelo.

Limitación actual

Los **perfils** ([profiles.nombre]) definidos en config.toml **no están soportados aún en la extensión IDE**. Solo funcionan con el CLI via codex --profile nombre. Para la extensión, los valores del perfil deben estar en el nivel raíz del config.toml.

3.3 Sistema de configuración por capas

El sistema de configuración de Codex está diseñado para que múltiples fuentes coexistan con una precedencia clara. Esto es fundamental para equipos donde cada proyecto puede tener reglas distintas y la empresa impone políticas de seguridad.

3.3.1 Orden de precedencia (mayor a menor)

Prioridad	Capa	Ubicación	Quién la gestiona
1 (máxima)	Managed preferences	macOS: MDM perfil; Linux: /etc/codex/managed_config.toml	Administrador de sistemas
2	CLI flags / --config	Línea de comandos (ej. codex -m gpt-5.3-codex)	Desarrollador (por ejecución)
3	Perfil activo	[profiles.nombre] en config.toml (solo CLI)	Desarrollador
4	Config proyecto (CWD)	Closest .codex/config.toml al CWD (solo si trusted)	Equipo (versionado en repo)
5	Config proyecto (root)	.codex/config.toml en la raíz Git	Equipo (versionado en repo)
6	Config usuario	~/.codex/config.toml	Desarrollador (personal)
7	Config sistema	Unix: /etc/codex/config.toml	Administrador
8 (mínima)	Defaults de Codex	Compilados en el binario	OpenAI

💡 Regla práctica

Las capas superiores **sobreesciben** las inferiores para la misma clave. Si el repo tiene `model = "gpt-5.2-codex"` y el usuario tiene `model = "gpt-5.3-codex"`, gana el repo (prioridad 4–5 > prioridad 6). Si se pasa `codex -m gpt-5.3-codex`, gana el flag (prioridad 2).

3.3.2 Config de usuario: `~/.codex/config.toml`

Define los valores por defecto personales del desarrollador. Se aplican a todos los proyectos salvo que el proyecto o un admin los sobreescriban.

```
# ~/.codex/config.toml - Config personal del desarrollador

# Modelo por defecto
model = "gpt-5.3-codex"

# Política de aprobación y sandbox
approval_policy = "on-request"
sandbox_mode = "workspace-write"

# Personalidad del agente
personality = "friendly"

# Variables de entorno expuestas (seguridad)
[shell_environment_policy]
include_only = ["PATH", "HOME", "LANG", "TERM", "NODE_ENV"]

# Feature flags opcionales
```

```
[features]
shell_tool = true
shell_snapshot = true      # Cachear entorno para comandos repetidos
```

3.3.3 Config de proyecto: .codex/config.toml

Se ubica en la raíz del repositorio Git (o en subdirectorios para overrides específicos). Se versiona con el repo para que todo el equipo comparta las mismas políticas. Solo se carga si el proyecto está marcado como trusted.

```
# .codex/config.toml – Config de proyecto (versionado en Git)

# Modelo estandarizado para el equipo
model = "gpt-5.3-codex"

# Políticas de seguridad del proyecto
approval_policy = "on-request"
sandbox_mode = "workspace-write"

[sandbox_workspace_write]
network_access = false    # Sin red por defecto

# Web search en modo cache (menor riesgo de prompt injection)
web_search = "cached"
```

⚠ Seguridad: proyectos trusted vs. untrusted

Codex solo carga las capas .codex/ del proyecto si este está marcado como **trusted**. Si un proyecto no es trusted, Codex ignora silenciosamente su config.toml (y también los .codex/config.toml de subdirectorios). La confianza se establece en el primer arranque o vía la configuración del usuario. **Nunca confiar ciegamente en repos desconocidos**: el config.toml del proyecto podría contener políticas permisivas.

3.3.4 Perfiles: [profiles.nombre]

Los perfiles permiten agrupar conjuntos de configuración bajo un nombre y activarlos con --profile:

```
# ~/.codex/config.toml

# Valores por defecto
model = "gpt-5.3-codex"
approval_policy = "on-request"

# Perfil para revisión profunda
[profiles.deep-review]
model = "gpt-5.3-codex"
model_reasoning_effort = "high"
approval_policy = "never"    # No interrumpir durante review

# Perfil ligero para consultas rápidas
[profiles.quick]
model = "gpt-5.3-codex"
```

```
model_reasoning_effort = "low"

# Perfil CI (para pipelines automatizados)
[profiles.ci]
model = "gpt-5.2-codex"
approval_policy = "never"
sandbox_mode = "danger-full-access"
```

```
# Uso:
codex --profile deep-review      # Revisión profunda
codex --profile quick           # Consulta rápida
codex --profile ci exec "Run tests" # CI automatizado
```

3.3.5 Verificar configuración efectiva

Para ver exactamente qué valores están activos tras resolver todas las capas:

```
# Desde cualquier directorio de proyecto
codex config show --effective

# Muestra: modelo, sandbox, approval_policy, features, MCP servers...
# Cada valor indica su fuente (user, project, default, etc.)
```

3.3.6 Gobernanza enterprise: requirements.toml

Para empresas, los administradores pueden definir restricciones no sobreescrivibles:

```
# /etc/codex/requirements.toml (Linux/macOS, gestionado por MDM)

[approval_policy]
allowed = ["untrusted", "on-request", "on-failure"] # "never" prohibido

[sandbox_mode]
allowed = ["read-only", "workspace-write"] # full-access prohibido

# Allowlist de MCP servers (solo estos se permiten)
# [[mcp_server_allowlist]]
# id = "context7"
# url = "..."
```

Si un usuario intenta seleccionar un valor prohibido (vía config.toml, CLI flags, perfiles o UI), Codex rechaza el cambio. Para Business/Enterprise con autenticación ChatGPT, los requirements también pueden descargarse desde el backend de Codex.

3.4 Codex App: centro de mando multi-agente

La Codex App es una aplicación de escritorio exclusiva para macOS (Apple Silicon) diseñada para gestionar múltiples agentes trabajando en paralelo.

Características principales

- Barra lateral de proyectos con worktrees integrados: cada proyecto mantiene su propio contexto.
- Lista de threads: múltiples conversaciones con agentes ejecutándose simultáneamente.
- Panel de revisión: visualización de diffs con preview de PDFs y ficheros generados.
- Lanzamiento cloud: delegar tareas directamente al entorno cloud sin salir de la app.
- Instalación desde CLI: codex app <path> en macOS (descarga automática del DMG si falta).

Cuándo usar la App vs. otras superficies

Escenario	Superficie recomendada	Razón
Gestionar 3+ tareas en paralelo sobre el mismo repo	App	Threads paralelos con visión unificada
Pair programming interactivo	IDE Extension	Contexto visual del código junto al chat
Script o automatización en CI	CLI (codex exec)	No requiere GUI, output parseable
Tarea larga (horas) sin consumo local	Cloud (web o CLI)	Container aislado, sin bloquear tu máquina
Review de PR desde GitHub	Cloud (@codex en PR)	Integrado con GitHub, sin setup local
Onboarding rápido a repo desconocido	CLI o IDE (modo read-only)	Exploración segura sin riesgo de modificación

3.5 Codex Cloud: delegación en entorno remoto

Codex Cloud ejecuta tareas en contenedores aislados gestionados por OpenAI. Es la superficie para tareas que requieren larga duración, paralelismo o aislamiento total.

Setup inicial

1. Acceder a chatgpt.com/codex.
2. Conectar cuenta GitHub (permite acceso a repositorios).
3. Configurar un entorno: seleccionar repo, rama, setup script (para instalar dependencias).
4. Lanzar primera tarea.

Control de acceso a internet

Por defecto, internet está deshabilitado durante la ejecución de tareas. Solo está habilitado en la fase de setup (para instalar dependencias). Se puede expandir a: acceso total a internet o lista de dominios permitidos.

Integración con GitHub

- @codex en comentarios de PR: lanza una tarea cloud de revisión o implementación.
- @codex en issues: puede analizar la issue y proponer un fix como PR.

- Codex propone PRs directamente desde el cloud.
- Desde CLI: `codex cloud exec --env ENV_ID "tarea"` lanza tareas cloud sin navegador.

4. Buenas prácticas

4.1 Definir proyecto con mínimo alcance

- **Separar monorepos por subproyectos:** Si tu monorepo tiene services/auth, services/api, services/web, configura Codex con `codex --cd services/api` para limitar el scope. Usa `-add-dir` si necesitas acceso a directorios compartidos: `codex --cd services/api --add-dir ../shared`.
- **Un directorio = un contexto:** El sandbox `workspace-write` limita la escritura al CWD. Ejecutar Codex desde la raíz de un monorepo le da acceso de escritura a todo. Ejecutar desde un subdirectorio lo acota.

4.2 Usar configuración repo-scoped para estandarizar

- **Versionar .codex/config.toml:** Incluir en el repo la configuración del proyecto (modelo, sandbox, approval_policy). Esto garantiza que todo el equipo usa las mismas políticas.
- **No incluir secretos:** config.toml es un fichero de políticas, no de credenciales. No incluir API keys, tokens ni URLs de servicios internos.
- **Versionar AGENTS.md junto con config.toml:** Las instrucciones del proyecto (comandos de test, lint, convenciones) deben estar en el repo.

4.3 Elegir la superficie adecuada

- **CLI para automatizar:** codex exec es ideal para rutinas repetibles, scripts y CI. Output parseable con `--json`.
- **IDE para pair-programming:** La extensión inyecta contexto automáticamente y permite refactorización interactiva con feedback visual inmediato.
- **Cloud para delegar:** Tareas largas, revisiones de PR y trabajo en paralelo sin bloquear tu máquina.

4.4 Preparar el entorno antes de lanzar Codex

Codex rinde mejor cuando el entorno ya está configurado. Antes de lanzar una sesión:

- Instalar dependencias (`npm install`, `pip install`, etc.) antes de ejecutar Codex.
- Asegurarse de que el build pasa y los tests corren.
- Tener el linter configurado y funcionando.

Esto evita que Codex gaste tokens diagnosticando problemas de entorno en lugar de resolver tu tarea.

5. Errores comunes y cómo evitarlos

5.1 Mezclar config global con config del repo sin control

El problema: Un desarrollador tiene `approval_policy = "never"` en su config personal. El repo tiene `approval_policy = "on-request"`. Como el repo tiene mayor precedencia, la config personal se ignora. Pero si el repo no está trusted, se usa la config personal permisiva sin saberlo.

La solución: Verificar siempre con `codex config show --effective` antes de empezar. Marcar repos del equipo como trusted de forma explícita. Usar `requirements.toml` en entornos enterprise para establecer mínimos no negociables.

5.2 Ejecutar en directorio equivocado (alcance excesivo)

El problema: Lanzar `codex` desde `~/code/` (directorio padre de todos los repos) en lugar de `~/code/mi-proyecto/`. Codex tiene acceso de escritura a todos los repos.

La solución: Siempre `cd` al directorio del proyecto antes de ejecutar Codex. Usar `codex --cd path` si necesitas lanzar desde otro lugar. Verificar con `/status` qué directorios están en el workspace.

5.3 No versionar convenciones (lint, formato)

El problema: El equipo no tiene ni `AGENTS.md` ni `.codex/config.toml` en el repo. Cada desarrollador tiene su propia configuración. Codex genera código con estilos diferentes según quién lo ejecute: tabs vs. espacios, comillas simples vs. dobles, etc. Luego culpan a la IA de inconsistencias.

La solución: Crear `AGENTS.md` con las convenciones explícitas del proyecto (`linter`, `formatter`, comandos de test). Versionar `.codex/config.toml` con las políticas estándar. Codex respetará estas instrucciones consistentemente para todo el equipo.

5.4 Tabla resumen de errores adicionales

Error	Consecuencia	Prevención
No reiniciar Codex tras cambiar config	Config anterior sigue activa	Reiniciar TUI / recargar extensión
Poner secrets en <code>.codex/config.toml</code>	Se versionan y exponen en Git	Usar variables de entorno para secrets
Instalar extensión no oficial	Riesgo de seguridad, comportamiento imprevisible	Solo instalar <code>openai.chatgpt</code> del Marketplace
Usar perfiles en la extensión IDE	No están soportados; se ignoran silenciosamente	Usar perfiles solo con CLI
No verificar trust del proyecto	Config del repo se ignora sin aviso	Marcar como trusted en primer arranque

6. Casos de uso reales

6.1 Setup corporativo: baseline config + políticas

Escenario: Empresa de 50 desarrolladores adopta Codex. El equipo de plataforma necesita establecer una configuración base que garantice seguridad y consistencia sin restringir la productividad individual.

Solución implementada

1. Config de sistema (/etc/codex/config.toml): modelo por defecto, personality y feature flags estándar.
2. Requirements.toml (/etc/codex/requirements.toml): prohibir sandbox danger-full-access y approval_policy never. Allowlist de MCP servers aprobados.
3. Template .codex/ para repos: plantilla con config.toml + AGENTS.md que cada equipo clona al crear un repo nuevo.
4. Documentación interna: guía de onboarding con la instalación, primer login y validación de codex config show --effective.

Resultado: Cualquier desarrollador, en cualquier repo, obtiene el mismo comportamiento base de Codex. Los equipos pueden refinar las políticas de su proyecto via .codex/config.toml sin violar las restricciones corporativas.

6.2 Homogeneizar comandos y flujos entre equipos

Escenario: Tres equipos (frontend React, backend Python, infra Terraform) con diferentes stacks pero necesidad de un flujo Codex uniforme: mismo modelo, misma política de aprobación, mismos slash commands de revisión.

Solución implementada

- Config global (~/.codex/config.toml): modelo gpt-5.3-codex, approval_policy on-request para todos.
- AGENTS.md específico por repo: el equipo frontend indica “usar npm test && npm run lint”; el backend indica “usar pytest && flake8”; infra indica “usar terraform validate && terraform plan”.
- Skills compartidos: skill de code review estandarizado instalado en ~/.codex/skills/ que todos los equipos usan con \$code-review.

Resultado: Codex se adapta al stack de cada equipo mientras mantiene el flujo de trabajo (revisión, aprobación, sandbox) uniforme para toda la organización.

7. Laboratorio práctico (L2) — Setup corporativo mínimo

⌚ Objetivo del laboratorio

Dejar listo un repositorio con configuración estándar (.codex/config.toml + AGENTS.md) para que Codex trabaje de forma **consistente y reproducible** en CLI e IDE. Al finalizar, ambas superficies respetan las mismas políticas y convenciones.

7.1 Prerrequisitos

- Codex CLI instalado y autenticado (completado en Lab 1).
- VS Code con extensión Codex instalada y autenticada.
- Python 3.10+ y Node.js 22+.
- Git instalado y configurado.

7.2 Paso 1 — Crear proyecto con estructura estándar

Tiempo estimado: 5 minutos

```
# Crear directorio y repo
mkdir /tmp/codex-lab02 && cd /tmp/codex-lab02
git init

# Crear estructura Python de ejemplo
mkdir -p src tests

cat > src/calculator.py << 'EOF'
"""Módulo de calculadora básica para el laboratorio."""

def add(a: float, b: float) -> float:
    return a + b

def subtract(a: float, b: float) -> float:
    return a - b

def multiply(a: float, b: float) -> float:
    return a * b

def divide(a: float, b: float) -> float:
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
EOF

cat > tests/test_calculator.py << 'EOF'
import pytest
from src.calculator import add, subtract, multiply, divide
```

```

def test_add():
    assert add(2, 3) == 5

def test_divide_by_zero():
    with pytest.raises(ValueError):
        divide(1, 0)
EOF

# Setup Python
python3 -m venv .venv && source .venv/bin/activate
pip install pytest flake8 black

# Verificar que funciona
PYTHONPATH=. pytest tests/ -v

# Commit inicial
git add -A && git commit -m "chore: scaffold proyecto lab02"

```

7.3 Paso 2 — Crear configuración de proyecto

Tiempo estimado: 10 minutos

```

# Crear directorio de configuración Codex
mkdir -p .codex

# Crear config.toml del proyecto
cat > .codex/config.toml << 'EOF'
# .codex/config.toml – Configuración estándar del proyecto
# Versionado en Git. Aplicable a CLI e IDE Extension.

# Modelo estandarizado
model = "gpt-5.3-codex"

# Políticas de seguridad
approval_policy = "on-request"
sandbox_mode = "workspace-write"

[sandbox_workspace_write]
network_access = false

# Web search en modo cache (seguridad)
web_search = "cached"
EOF

```

```

# Crear AGENTS.md con convenciones del proyecto
cat > AGENTS.md << 'EOF'
# AGENTS.md – Instrucciones para Codex

```

```

## Convenciones del proyecto
- Lenguaje: Python 3.12 con type hints obligatorios.
- Formatter: black (línea máxima 88 chars).
- Linter: flake8 con max-line-length 88.
- Tests: pytest. Ejecutar `PYTHONPATH=. pytest tests/ -v` antes de cada cambio.
- Docstrings: formato Google style.

## Estructura del proyecto
- src/: código fuente
- tests/: tests unitarios (ficheros test_*.py)

## Reglas
- No añadir dependencias de producción sin aprobación explícita.
- Mantener cobertura de tests > 80%.
- Usar snake_case para funciones y variables.
- No usar import * en ningún caso.
EOF

# Commit de la configuración
git add -A && git commit -m "chore: añadir config Codex y AGENTS.md"

```

7.4 Paso 3 — Verificar configuración desde CLI

Tiempo estimado: 5 minutos

```

cd /tmp/codex-lab02
source .venv/bin/activate

# Verificar configuración efectiva
codex config show --effective

# Debería mostrar:
#   model = gpt-5.3-codex  (fuente: project)
#   approval_policy = on-request  (fuente: project)
#   sandbox_mode = workspace-write  (fuente: project)

```

```

# Lanzar Codex en modo interactivo
codex --full-auto

# Pedir que muestre las instrucciones cargadas:
"Resume las instrucciones que tienes cargadas de AGENTS.md"

# Verificar que respeta convenciones pidiendo una tarea:
"Añade una función power(base, exp) a src/calculator.py con su test.
Sigue las convenciones del proyecto.
Ejecuta pytest y black para validar."

# Verificar que:
# 1. Usa type hints (convención AGENTS.md)
# 2. Ejecuta pytest (convención AGENTS.md)
# 3. Ejecuta black (convención AGENTS.md)
# 4. Usa snake_case (convención AGENTS.md)

```

7.5 Paso 4 — Verificar configuración desde IDE

Tiempo estimado: 5 minutos

```
# Abrir el proyecto en VS Code
code /tmp/codex-lab02
```

1. Abrir el panel de Codex desde la barra lateral.
2. Verificar que está en modo Agent (no Chat ni Full Access).
3. Abrir src/calculator.py en el editor.
4. En el panel Codex, escribir el mismo prompt que en CLI:

```
"Añade una función modulo(a, b) a calculator.py con su test.
Sigue las convenciones del proyecto.
Ejecuta pytest y black para validar."
```

5. Observar que Codex respeta las mismas convenciones que en CLI: type hints, snake_case, pytest, black.
6. Verificar el modelo y reasoning effort en el selector bajo el input de chat.

Criterio de éxito

El agente se comporta igual en CLI e IDE: mismo modelo, mismas convenciones, mismos comandos de validación. La configuración del proyecto (.codex/config.toml + AGENTS.md) controla el comportamiento, no la preferencia individual de cada desarrollador.

7.6 Paso 5 — Probar override con CLI flags

Tiempo estimado: 3 minutos

Verificar que las capas de precedencia funcionan como se espera:

```
# El proyecto establece gpt-5.3-codex, pero podemos overridear con flag:
codex -m gpt-5.2-codex

# Dentro de la sesión, ejecutar /model para verificar:
/model
# Debe mostrar: gpt-5.2-codex (override via CLI flag)

# Salir y lanzar sin override:
codex
/model
# Debe mostrar: gpt-5.3-codex (valor del proyecto)
```

7.7 Resultado esperado

Verificación	Resultado esperado	Comando de comprobación
--------------	--------------------	-------------------------

Config efectiva muestra valores del proyecto	model, sandbox, approval del .codex/config.toml	codex config show --effective
AGENTS.md cargado correctamente	Codex menciona convenciones (pytest, black, type hints)	Pedir resumen de instrucciones
Mismas convenciones en CLI e IDE	Ambos generan código con type hints, snake_case, ejecutan pytest	Comparar output visual
CLI flag overridea modelo del proyecto	codex -m cambia modelo; sin flag vuelve al del proyecto	/model en cada sesión
Git tiene 3 commits limpios	scaffold + config + funciones generadas	git log --oneline

7.8 Limpieza (OBLIGATORIA)

⚠️ Limpieza de recursos

Ejecutar al finalizar el laboratorio:

```
# 1. Desactivar entorno virtual
deactivate

# 2. Borrar el repositorio completo
rm -rf /tmp/codex-lab02

# 3. Revertir cambios en config global (si hiciste pruebas)
# Si modificaste ~/.codex/config.toml, restaurar el backup:
# cp ~/.codex/config.toml.bak ~/.codex/config.toml

# 4. Verificar limpieza
ls /tmp/codex-lab02 2>/dev/null && echo 'ERROR: aún existe' || echo 'OK: limpio'
```

8. Resumen del módulo y siguientes pasos

8.1 Conceptos clave aprendidos

Concepto	Detalle
CLI: agente local	Open source, TUI interactivo, codex exec para CI, sandbox OS-level
IDE Extension	Mismo agente y config que CLI, contexto automático de ficheros abiertos
Config por capas	8 capas de precedencia: managed > CLI flags > profile > project > user > system > defaults
.codex/config.toml	Config de proyecto versionable, solo se carga si trusted
Perfiles	[profiles.nombre] para alternar configuraciones (solo CLI)
requirements.toml	Restricciones admin no sobreescribibles (enterprise)
App: multi-agente	Escritorio macOS con threads paralelos y panel de revisión
Cloud: delegación	Container aislado, tareas largas, integración GitHub
codex config show	Comando para verificar la configuración efectiva resuelta

8.2 Preparación para el Módulo 3

En el Módulo 3 profundizaremos en prompt engineering aplicado a programación: patrones efectivos, gestión del contexto, mid-turn steering y web search integrado. Para prepararse:

- Tener el entorno del Lab 2 funcionando (o recrearlo).
- Revisar brevemente la guía oficial de prompting: <https://developers.openai.com/codex/prompting>
- Revisar los workflows documentados: <https://developers.openai.com/codex/workflows/>

9. Referencias y recursos

- Codex CLI (overview): <https://developers.openai.com/codex/cli>
- Codex CLI (features): <https://developers.openai.com/codex/cli/features/>
- Codex CLI (command reference): <https://developers.openai.com/codex/cli/reference/>
- Codex IDE Extension: <https://developers.openai.com/codex/ide/>
- Codex IDE (features): <https://developers.openai.com/codex/ide/features/>
- Config basics: <https://developers.openai.com/codex/config-basic/>
- Advanced config: <https://developers.openai.com/codex/config-advanced/>
- Config reference: <https://developers.openai.com/codex/config-reference/>
- Sample config.toml: <https://developers.openai.com/codex/config-sample/>
- Security (sandbox, approvals, enterprise): <https://developers.openai.com/codex/security/>
- Codex Cloud: <https://developers.openai.com/codex/cloud>
- MCP configuration: <https://developers.openai.com/codex/mcp/>
- AGENTS.md guide: <https://developers.openai.com/codex/guides/agents-md/>
- Windows setup: <https://developers.openai.com/codex/windows>
- Codex on GitHub (open source): <https://github.com/openai/codex>