

MÓDULO 4

Gestión avanzada del contexto

Prompts · Historial · Límites · Estrategias

Plan de Formación — OpenAI Codex para Desarrolladores

Nivel: Intermedio · Duración estimada: 3–4 horas

Versión 1.0 — Febrero 2026

Índice

1. Introducción

Codex es tan bueno como el contexto que recibe. Un agente de IA con acceso a un repositorio de 500.000 líneas de código no puede procesarlo todo de una vez: necesita saber exactamente dónde mirar, qué restricciones aplicar y cómo verificar su trabajo. La gestión del contexto es la habilidad que separa a un desarrollador que “usa Codex” de uno que “domina Codex”.

En este módulo aprenderemos a diseñar prompts estructurados que producen cambios mínimos, revisables y testeables. Construiremos plantillas reutilizables para los tres flujos más comunes (feature, bugfix, refactor) y dominaremos las técnicas de contexto progresivo para mantener a Codex “en carril” incluso en repos grandes.

La filosofía central es: no le des todo a Codex de golpe; aliméntalo por fases con contexto relevante, constraints claros y una definición de “done” verificable.

2. Objetivos de aprendizaje

#	Objetivo	Evidencia de logro
O1	Mantener a Codex “en carril” en repos grandes usando técnicas de acotación de contexto.	Completa una tarea en un repo multicomponente tocando solo los ficheros permitidos.
O2	Diseñar prompts que producen cambios mínimos, revisables y testeables.	Genera un refactor con diff <50 líneas y tests verdes al primer intento.
O3	Aplicar las plantillas Feature/Bugfix/Refactor en tareas reales.	Usa la plantilla correcta en 3 escenarios del laboratorio.
O4	Evitar degradación por contexto insuficiente o excesivo (ruido).	Identifica y corrige 3 anti-patrones de contexto en ejemplos dados.

3. Contenidos teóricos

3.1 Cómo Codex gestiona el contexto

3.1.1 Fuentes de contexto automático

Codex construye su contexto de trabajo combinando múltiples fuentes automáticamente:

Fuente	Superficie	Descripción
AGENTS.md (cadena de instrucciones)	CLI + IDE	Ficheros AGENTS.md desde ~/.codex hasta el CWD, fusionados en orden de precedencia
Ficheros abiertos en el editor	IDE	La extensión inyecta automáticamente los ficheros que tienes abiertos
Selección de código activa	IDE	Texto seleccionado se incluye como contexto prioritario
@ file references	CLI + IDE	Ficheros referenciados explícitamente con @ruta en el prompt
Output de comandos ejecutados	CLI + IDE	Resultados de comandos shell que Codex ejecuta durante la tarea
Web search (cached/live)	CLI + IDE	Resultados de búsqueda web si está habilitado
MCP server tools	CLI + IDE	Datos de servidores MCP configurados (bases de datos, APIs, etc.)
Historial de la conversación	CLI + IDE	Mensajes anteriores de la sesión actual (sujeto a compaction)

3.1.2 Contexto implícito vs. explícito

IDE: contexto implícito. La extensión inyecta automáticamente los ficheros abiertos y las selecciones. Esto permite prompts más cortos porque Codex ya “ve” tu código.

CLI: contexto explícito. En la terminal, generalmente necesitas mencionar las rutas explícitamente con @ruta/al/fichero o usar el atajo @ + Tab para completar ficheros del workspace.

💡 Consejo práctico

En el CLI, escribe @ en el compositor y empieza a escribir el nombre del fichero. Codex abrirá un buscador fuzzy sobre el workspace y puedes pulsar Tab/Enter para insertar la ruta completa.

3.1.3 Límites del contexto y compaction

Los modelos Codex tienen ventanas de contexto grandes (hasta 400K tokens), pero no infinitas. Cuando una sesión se acerca al límite, Codex ejecuta context compaction automáticamente: resume el historial preservando la información más relevante. Esto permite sesiones de horas, pero tiene implicaciones prácticas:

- Las instrucciones del principio de una sesión larga pueden “difuminarse” tras la compaction.
- Las convenciones críticas deben estar en AGENTS.md (se releen en cada sesión), no solo en el prompt.
- Para tareas de larga duración, usar PLANS.md para persistir el estado (ver Módulo 3).

3.2 Anatomía de un prompt efectivo para Codex

Un prompt efectivo para Codex no es una instrucción abierta; es un encargo profesional con los mismos elementos que darías a un colega humano. La estructura recomendada tiene cuatro bloques:

3.2.1 Los cuatro bloques: Brief + Constraints + Definition of Done + Commands

Bloque	Propósito	Ejemplo
Brief	Qué hacer, en una o dos frases claras.	Implementa un endpoint POST /api/tasks que cree tareas.
Constraints	Límites explícitos que acotan el trabajo.	Solo modifica src/routes/tasks.ts. No cambies la API pública de otros módulos. Máximo 80 líneas nuevas.
Definition of Done	Cómo verificar que está terminado.	Tests verdes con npm test. Sin warnings de ESLint. El endpoint responde 201 con body válido.
Commands	Comandos de validación que Codex debe ejecutar.	Ejecuta: npm test && npm run lint. Muestra el output.

3.2.2 Ejemplo completo: prompt bien estructurado

```
# Prompt para Codex CLI o IDE

## Brief
Añade validación de email al método create_user() en src/services/user.py.
El email debe contener '@' y al menos un '.' después del '@'.
Si es inválido, lanzar ValueError con mensaje descriptivo.

## Constraints
- Solo modifica src/services/user.py y tests/test_user.py.
- No cambies la firma del método create_user (mismos parámetros).
- No añadas dependencias externas (solo stdlib).
- Mantener cobertura de tests > 80%.

## Definition of Done
- create_user('bad-email') lanza ValueError.
- create_user('valid@test.com') funciona como antes.
- Todos los tests existentes siguen pasando.
- Al menos 4 tests nuevos cubriendo: email válido, sin @,
  sin punto, cadena vacía.

## Validation
Ejecuta: PYTHONPATH=. pytest tests/test_user.py -v
Ejecuta: flake8 src/services/user.py
Muestra resultados.
```

3.2.3 Ejemplo: prompt mal estructurado (anti-patrón)

```
# ❌ Anti-patrón: prompt vago sin estructura

"Mejora la validación de usuarios en el proyecto."
```

```
# Problemas:  
# - No especifica QUÉ ficheros tocar  
# - No define QUÉ significa "mejorar"  
# - No indica cómo VERIFICAR que está bien  
# - Codex podría tocar 20 ficheros o 1, no hay guía  
# - Sin constraints, puede cambiar APIs públicas
```

 **Regla del OpenAI Workflows Guide**

Codex funciona mejor cuando le tratas como a un compañero de equipo con contexto explícito y una definición clara de “done”. La documentación oficial enfatiza: proporciona siempre pasos de reproducción (bugfix), constraints (refactor) o deliverables esperados (feature).

3.3 Contexto progresivo: alimentar por fases

La estrategia de contexto progresivo evita sobrecargar a Codex con información irrelevante. En lugar de volcar todo el contexto de una vez, se alimenta al agente en fases secuenciales:

3.3.1 Las cuatro fases

Fase	Acción del desarrollador	Acción de Codex	Propósito
1. Lectura	Pedir a Codex que lea y explique el código relevante.	Analiza ficheros, identifica dependencias, genera un resumen.	Verificar que Codex entiende el contexto antes de actuar.
2. Plan	Pedir un plan de cambios con ficheros y líneas afectadas.	Propone un plan detallado sin ejecutar nada.	Revisar el enfoque antes de permitir cambios.
3. Edición	Aprobar el plan y pedir la implementación con constraints.	Escribe código, crea tests, modifica ficheros.	Ejecución controlada con scope acotado.
4. Validación	Pedir que ejecute tests y reporte resultados.	Ejecuta pytest/npm test/lint y muestra output.	Confirmar que los cambios son correctos.

3.3.2 Ejemplo de flujo progresivo en CLI

```
# FASE 1: Lectura
# Abrir Codex interactivo
codex

# Prompt fase 1:
"Lee @src/auth/middleware.ts y @src/auth/session.ts.
Expícame cómo funciona el flujo de autenticación:
1. Dónde se valida el token.
2. Cómo se carga la sesión.
3. Qué excepciones se manejan."

# FASE 2: Plan (tras revisar la explicación)
"Necesito añadir renovación automática de tokens expirados.
Propone un plan de cambios:
- Qué ficheros tocarás (solo auth/).
- Qué funciones modificarás.
- Qué tests añadirás.
No implementes nada aún."

# FASE 3: Edición (tras aprobar el plan)
"Implementa el plan. Constraints:
- Solo modifica auth/middleware.ts y auth/session.ts.
- No cambies la interfaz AuthResult.
- Añade tests en tests/auth/test_renewal.ts.
- Máximo 60 líneas nuevas."

# FASE 4: Validación
"Ejecuta npm test -- --testPathPattern=auth
y npm run lint.
Muestra los resultados."
```

💡 Mid-turn steering

Si durante la fase de edición ves que Codex toma una dirección incorrecta, puedes pulsar **Enter** mientras está trabajando para inyectar instrucciones correctivas sin perder el contexto de la tarea. Ejemplo: No modifiques session.ts, solo middleware.ts.

3.4 Acotación de contexto en reposos grandes

3.4.1 Técnicas de acotación

Técnica	Cómo aplicarla	Efecto
codex --cd <path>	Lanzar Codex apuntando a un subdirectorio específico	Limita el workspace y el sandbox al subdirectorio
codex --add-dir	Añadir directorios adicionales de solo lectura	Acceso a shared/ sin darle escritura
@ file references	Referenciar solo los ficheros relevantes en el prompt	Codex prioriza esos ficheros en su contexto
AGENTS.md por directorio	AGENTS.md específicos en subdirectorios del monorepo	Instrucciones adaptadas a cada módulo
Constraints explícitos	"Solo modifica src/api/. No toques src/core/."	El agente respeta los límites declarados
sandbox workspace-write	Sandbox que restringe escritura al CWD	Prevención técnica de modificaciones fuera del scope

3.4.2 Ejemplo: monorepo con múltiples servicios

```
# Estructura del monorepo:
# monorepo/
#   services/
#     auth/           ← queremos trabajar aquí
#     api/
#     web/
#     shared/         ← solo lectura
#     infra/

# Lanzar Codex acotado a auth/ con acceso de lectura a shared/
codex --cd services/auth --add-dir ../shared

# El sandbox workspace-write restringe escritura a services/auth/
# Codex puede LEER shared/ pero no ESCRIBIR en él
```

3.5 Plantillas de prompt por tipo de tarea

Estas plantillas son reutilizables y deben adaptarse al proyecto. Se pueden incluir en AGENTS.md o en Skills para que Codex las aplique automáticamente.

3.5.1 Plantilla Feature (nueva funcionalidad)

```
## Brief
[Describe la funcionalidad en 1-2 frases]

## Scope
- Ficheros permitidos: [lista explícita de rutas]
- Nuevos ficheros: [rutas de ficheros nuevos a crear]

## Constraints
- No modificar [APIs/interfaces/módulos protegidos].
- Seguir estilo: [referencia a linter/formatter del proyecto].
- Máximo [N] líneas nuevas.
- No añadir dependencias sin aprobación.

## Acceptance Criteria
- [Criterio 1: comportamiento esperado]
- [Criterio 2: caso edge]
- [Criterio 3: rendimiento si aplica]

## Tests
- Añadir tests en [ruta].
- Cubrir: happy path, caso edge, input inválido.

## Validation Commands
- [comando test]: [descripción]
- [comando lint]: [descripción]
- Mostrar resultados.
```

3.5.2 Plantilla Bugfix (corrección de error)

```
## Bug Description
[Descripción del bug en 1-2 frases]

## Reproduction Steps
1. [Paso 1]
2. [Paso 2]
3. Resultado esperado: [X]
4. Resultado actual: [Y]

## Hypothesis
[Dónde crees que está el problema y por qué]

## Constraints
- No cambiar la API pública.
- Fix mínimo: la menor cantidad de cambios posible.
- No introducir nuevas dependencias.

## Fix + Regression
```

- Implementar el fix.
 - Añadir test de regresión que falle sin el fix y pase con él.
- ```
Validation Commands
- Reproducir el bug ANTES del fix (debe fallar).
- Aplicar fix.
- Reproducir el bug DESPUÉS del fix (debe pasar).
- Ejecutar suite completa de tests.
```

### 3.5.3 Plantilla Refactor (mejora sin cambio funcional)

```
Objective
[Qué mejorar: reducir complejidad ciclomática, eliminar duplicación, extraer módulo, etc.]

Invariant
- NO BEHAVIOUR CHANGE: la funcionalidad externa debe ser idéntica.
- Todos los tests existentes deben seguir pasando sin modificación.
- La API pública (firmas de funciones, tipos exportados) no cambia.

Scope
- Ficheros permitidos: [lista explícita]
- No tocar: [lista de exclusiones]

Metrics (antes/después)
- Líneas de código: mostrar diff stat.
- Complejidad ciclomática (si hay herramienta).
- Número de funciones/métodos.

Validation Commands
- Ejecutar tests existentes (DEBEN pasar sin cambios).
- Ejecutar linter.
- Mostrar git diff --stat para verificar scope acotado.
```

#### 💡 Skills como plantillas reutilizables

Estas plantillas se pueden convertir en **Skills de Codex** (Módulo 5) para invocarse directamente con `$feature`, `$bugfix` o `$refactor` en el composer. Así el equipo las comparte sin copiar/pegar.

## 3.6 El ciclo /review como guardrail integrado

Codex incluye un flujo de revisión integrado que actúa como guardrail de calidad:

1. Pedir a Codex que implemente los cambios.
2. Antes de hacer commit, ejecutar `/review` en el CLI.
3. Codex lanza un reviewer separado que lee el diff y reporta hallazgos priorizados sin tocar el working tree.
4. Corregir los issues detectados.
5. Repetir `/review` hasta limpieza total.

## 6. Hacer commit con confianza.

```
Dentro de una sesión Codex (TUI):

Revisión estándar del working tree
/review

Revisión con instrucciones específicas
/review Focus on security and edge cases

Revisión contra una rama base
/review --base main
```

## 4. Buenas prácticas

### 4.1 Pedir resultados verificables

Cada prompt debe terminar con comandos de validación que Codex ejecute antes de dar el trabajo por terminado:

| Tipo de proyecto | Comando de validación             | Propósito                  |
|------------------|-----------------------------------|----------------------------|
| Python           | PYTHONPATH=. pytest tests/ -v     | Tests unitarios            |
| Python           | flake8 src/ && black --check src/ | Linting + formateo         |
| Node/TypeScript  | npm test && npm run lint          | Tests + lint               |
| Go               | go test ./... && go vet ./...     | Tests + análisis estático  |
| Rust             | cargo test && cargo clippy        | Tests + lint               |
| General          | git diff --stat                   | Verificar scope de cambios |

### 4.2 Acotar el scope con guardrails explícitos

Los guardrails son instrucciones negativas que impiden que Codex se desvíe. Ejemplos de guardrails efectivos:

- "Solo modifica ficheros en src/auth/. No toques ningún otro directorio."
- "No cambies la firma de funciones públicas (parámetros ni tipo de retorno)."
- "No añadas dependencias de producción. Solo devDependencies si es imprescindible."
- "No modifiques tests existentes. Solo añade tests nuevos."
- "Máximo 50 líneas de código nuevas."
- "Mantén compatibilidad backward con la API v2."

### 4.3 Iterar con feedback específico

Cuando Codex produce un resultado incorrecto o subóptimo, el feedback debe ser tan estructurado como el prompt inicial:

```
❌ Feedback vago:
"Esto no está bien, hazlo mejor."

✅ Feedback específico:
"El test test_expired_token no cubre el caso donde el token
expira durante la request (race condition).
Añade un test que simule:
1. Token válido al inicio de la request.
2. Token expirado cuando se valida (mock time.time()).
3. Verificar que se renueva automáticamente."
```

### 4.4 Usar Git como red de seguridad

- Stash o commit ANTES de pedir cambios a Codex.
- Revisar git diff ANTES de hacer commit del resultado.

- Usar ramas feature para cada tarea.
- Si el resultado no es satisfactorio: git checkout . para revertir y reformular el prompt.

## 4.5 Separar la exploración de la ejecución

**Fase de exploración (sandbox read-only):** Usar Codex para entender código, mapear dependencias, explicar flujos. Sin riesgo de modificaciones.

**Fase de ejecución (sandbox workspace-write):** Una vez entendido el problema, cambiar a workspace-write y pedir la implementación con constraints claros.

## 5. Errores comunes y cómo evitarlos

### 5.1 Pedir “refactoriza todo el repo”

**El problema:** El prompt no acota scope. Codex interpreta “todo el repo” literalmente y toca decenas de ficheros, muchos innecesariamente. El diff resultante es imposible de revisar y probablemente rompe cosas.

**La solución:** Descomponer en tareas atómicas. Ejemplo: en lugar de “refactoriza el repo”, pedir “extrae la validación de email de UserService a una función separada en src/utils/validators.py. No toques otros módulos.” Cada tarea toca 1-3 ficheros máximo.

### 5.2 No fijar guardrails

**El problema:** Sin restricciones explícitas, Codex puede: cambiar firmas de funciones públicas (rompiendo clientes), añadir dependencias no deseadas, reformatear ficheros que no debería tocar, o “mejorar” código que no se le pidió.

**La solución:** Incluir siempre la sección Constraints en el prompt. Las restricciones más importantes: ficheros permitidos, API immutable, límite de líneas, prohibición de dependencias nuevas.

### 5.3 No pedir comandos de verificación

**El problema:** Codex genera código que “parece correcto” pero no lo ejecuta. El desarrollador acepta el diff sin validar y descubre errores más tarde.

**La solución:** Siempre terminar el prompt con: “Ejecuta [comando test] y muestra los resultados.” Si los tests fallan, Codex puede iterar automáticamente sobre el error.

### 5.4 Tabla de errores adicionales

| Error                                      | Consecuencia                                         | Prevención                                                     |
|--------------------------------------------|------------------------------------------------------|----------------------------------------------------------------|
| Volcar todo el contexto de golpe           | Ruido: Codex pierde foco entre ficheros irrelevantes | Contexto progresivo: fases de lectura→plan→edición→validación  |
| No usar @ para referenciar ficheros en CLI | Codex adivina qué ficheros leer (puede equivocarse)  | Siempre @ruta explícita en el prompt                           |
| Confiar en la explicación sin ejecutar     | El código puede compilar pero no funcionar           | Pedir ejecución de tests como paso final obligatorio           |
| Dar feedback vago (“hazlo mejor”)          | Codex no sabe qué cambiar específicamente            | Feedback con referencia exacta al problema y solución esperada |
| Olvistar AGENTS.md en sesiones largas      | Convenciones se pierden tras compaction              | Convenciones críticas en AGENTS.md (se releen siempre)         |

## 6. Casos de uso reales

### 6.1 Onboarding: “Explica el flujo de auth y dónde cambiarlo”

**Escenario:** Nuevo desarrollador se incorpora al equipo y necesita entender el subsistema de autenticación para implementar renovación de tokens.

#### Flujo aplicado (contexto progresivo)

1. Fase lectura: “Lee @src/auth/middleware.ts @src/auth/session.ts @src/auth/types.ts. Explícame el flujo de autenticación paso a paso.”
2. Fase plan: “Necesito añadir renovación automática de tokens. Propone un plan de cambios indicando qué funciones modificarás en cada fichero. No implementes nada aún.”
3. Revisión del plan con el tech lead (humano).
4. Fase edición: “Implementa el plan aprobado. Constraints: solo auth/, no cambiar interfaz AuthResult, tests en tests/auth/.”
5. Fase validación: “Ejecuta npm test -- --testPathPattern=auth && npm run lint.”

**Resultado:** El nuevo desarrollador entiende el subsistema ANTES de tocarlo, y el cambio es revisado por el equipo en cada fase.

### 6.2 Aislar cambio a un paquete en monorepo

**Escenario:** Equipo necesita actualizar la lógica de pricing en un monorepo con 15 servicios. Solo debe afectar al servicio de billing.

#### Técnicas aplicadas

- codex --cd services/billing --add-dir ../shared/types (scope físico).
- Constraints en prompt: “Solo modifica services/billing/. Los tipos de shared/types/ son de solo lectura.”
- AGENTS.md en services/billing/ con convenciones específicas: “Usar Decimal para precios. Tests con pytest-mock. No añadir dependencias.”
- Validación: PYTHONPATH=. pytest services/billing/tests/ -v && git diff --stat (verificar que no se tocaron otros servicios).

**Resultado:** El cambio solo afecta a billing/. El diff es revisable en una PR pequeña y los tests del servicio pasan.

## 7. Laboratorio práctico (L4) — Prompting con guardrails y reducción de diffs

### ⌚ Objetivo del laboratorio

Lograr una mejora (refactor) en un componente con deuda técnica produciendo un **diff mínimo** (<50 líneas), **sin cambio funcional** y con **validación reproducible** (tests verdes). El alumno practicará las tres plantillas y el flujo de contexto progresivo.

### 7.1 Prerrequisitos

- Codex CLI instalado y autenticado (Módulos 1-2).
- Python 3.10+ con pytest y flake8.
- Git instalado.

### 7.2 Paso 1 — Crear proyecto con deuda técnica intencionada

*Tiempo estimado: 5 minutos*

```
mkdir /tmp/codex-lab04 && cd /tmp/codex-lab04
git init
mkdir -p src tests

Crear módulo con deuda técnica intencionada:
- Función demasiado larga (>50 líneas)
- Lógica duplicada
- Sin type hints
- Validación mezclada con lógica de negocio

cat > src/order_processor.py << 'PYEOF'
"""Procesador de pedidos (código con deuda técnica intencionada)."""

def process_order(order_data):
 """Procesa un pedido completo. Función demasiado larga y compleja."""
 # Validación de datos (mezclada con lógica de negocio)
 if not order_data:
 return {"status": "error", "message": "Empty order"}
 if "items" not in order_data:
 return {"status": "error", "message": "No items"}
 if not isinstance(order_data["items"], list):
 return {"status": "error", "message": "Items must be a list"}
 if len(order_data["items"]) == 0:
 return {"status": "error", "message": "Empty items list"}

 # Validar cada item (duplicación de lógica)
 validated_items = []
 for item in order_data["items"]:
 if "name" not in item:
 return {"status": "error", "message": "Item missing name"}
 if "price" not in item:
 return {"status": "error", "message": "Item missing price"}
```

```

if "quantity" not in item:
 return {"status": "error", "message": "Item missing quantity"}
if item["price"] < 0:
 return {"status": "error", "message": "Negative price"}
if item["quantity"] < 1:
 return {"status": "error", "message": "Invalid quantity"}
validated_items.append(item)

Calcular totales (lógica de negocio mezclada con formateo)
subtotal = 0
for item in validated_items:
 subtotal = subtotal + item["price"] * item["quantity"]

Descuento (lógica hardcoded)
discount = 0
if subtotal > 100:
 discount = subtotal * 0.1
elif subtotal > 50:
 discount = subtotal * 0.05

total = subtotal - discount

Formatear respuesta (duplicación con la estructura de arriba)
return {
 "status": "ok",
 "items_count": len(validated_items),
 "subtotal": round(subtotal, 2),
 "discount": round(discount, 2),
 "total": round(total, 2),
}
PYEOF

```

```

Crear tests básicos (cobertura parcial intencionada)
cat > tests/test_order_processor.py << 'PYEOF'
from src.order_processor import process_order

def test_valid_order():
 result = process_order({"items": [
 {"name": "Widget", "price": 10.0, "quantity": 3}
]})
 assert result["status"] == "ok"
 assert result["total"] == 30.0

def test_empty_order():
 result = process_order(None)
 assert result["status"] == "error"

def test_discount_over_100():
 result = process_order({"items": [
 {"name": "Expensive", "price": 60.0, "quantity": 2}
]})

```

```

 assert result["discount"] == 12.0 # 120 * 0.1
 assert result["total"] == 108.0
PYEOF

Verificar que pasan los tests
PYTHONPATH=. pytest tests/ -v

Commit inicial
git add -A && git commit -m "chore: scaffold lab04 con deuda técnica"

```

## 7.3 Paso 2 — Ejercicio A: Plantilla Refactor (contexto progresivo)

*Tiempo estimado: 15 minutos*

Objetivo: refactorizar `process_order()` extrayendo validación a funciones separadas, sin cambiar el comportamiento externo.

```

codex --full-auto

=====
FASE 1: Lectura
=====
Prompt:
"Lee @src/order_processor.py y @tests/test_order_processor.py.
Identifica:
1. Problemas de deuda técnica (complejidad, duplicación).
2. Responsabilidades mezcladas.
3. Cobertura de tests actual.
No hagas cambios aún."

```

```

=====
FASE 2: Plan (tras revisar el análisis)
=====
Prompt:
"Propone un plan de refactorización para process_order().
Objetivo: extraer validación y cálculo a funciones separadas.
El plan debe indicar:
- Qué funciones nuevas crearás.
- Qué líneas de código moverás a cada función.
- Que el comportamiento externo NO cambia.
NO implementes nada aún."

```

```

=====
FASE 3: Edición (tras aprobar el plan)
=====
Prompt (plantilla Refactor):
"Implementa el plan de refactorización.

Invariant: NO BEHAVIOUR CHANGE.
Todos los tests existentes deben pasar SIN MODIFICACIÓN.

Constraints:

```

- Solo modifica src/order\_processor.py.
- No cambies la firma de process\_order() ni su tipo de retorno.
- Añade type hints a las funciones nuevas.
- Máximo 60 líneas totales en el fichero resultante.

Añade tests nuevos en tests/test\_order\_processor.py para las funciones extraídas (validate\_order, validate\_item, calculate\_totals, o como las nombres)."

```
=====
FASE 4: Validación
=====
Prompt:
"Ejecuta:
PYTHONPATH=. pytest tests/ -v
flake8 src/order_processor.py
git diff --stat
Muestra todos los resultados."
```

## Criterios de éxito del Ejercicio A

- Todos los tests originales pasan sin modificación.
- Se añaden tests nuevos para las funciones extraídas.
- git diff --stat muestra solo 2 ficheros: src/order\_processor.py y tests/test\_order\_processor.py.
- La función process\_order() es más corta y delega en funciones auxiliares.
- Type hints presentes en funciones nuevas.

## 7.4 Paso 3 — Ejercicio B: Plantilla Bugfix

*Tiempo estimado: 10 minutos*

```
Hacer commit del refactor anterior
git add -A && git commit -m "refactor: extraer validación a funciones"

Nuevo prompt usando la plantilla Bugfix:
Bug Description
process_order() no valida que price sea numérico.
Si alguien pasa price='abc', la multiplicación falla con TypeError.

Reproduction
1. Llamar process_order({'items': [{'name': 'X',
 'price': 'abc', 'quantity': 1}]})
2. Esperado: retorno con status='error'
3. Actual: TypeError en la línea de cálculo

Constraints
- Solo modifica src/order_processor.py.
- Fix mínimo: añadir validación de tipo.
- No cambiar la firma de ninguna función pública.

Validation
- Añade test de regresión que falle SIN el fix.
```

- Ejecuta PYTHONPATH=. pytest tests/ -v.
- Muestra resultados."

## 7.5 Paso 4 — Ejercicio C: Plantilla Feature

*Tiempo estimado: 10 minutos*

```
Hacer commit del bugfix anterior
git add -A && git commit -m "fix: validar tipo numérico de price"

Nuevo prompt usando la plantilla Feature:
Brief
Añade soporte para códigos de descuento a process_order().
Si order_data contiene 'discount_code', aplicar descuento adicional.
Codes: 'SAVE10' (10% extra), 'SAVE20' (20% extra), inválido (→ ignorar).

Scope
- Modificar: src/order_processor.py
- Añadir tests en: tests/test_order_processor.py

Constraints
- No cambiar la firma de process_order() (discount_code va dentro de order_data, no como parámetro nuevo).
- No añadir dependencias.
- Máximo 20 líneas nuevas.

Acceptance Criteria
- SAVE10 aplica 10% extra sobre el total.
- SAVE20 aplica 20% extra.
- Código inválido se ignora (no error).
- Sin código, comportamiento idéntico al actual.

Validation
- Añadir tests para los 4 criterios.
- PYTHONPATH=. pytest tests/ -v
- Mostrar resultados."
```

## 7.6 Resultado esperado

| Verificación           | Resultado esperado                                | Comando                        |
|------------------------|---------------------------------------------------|--------------------------------|
| Git log                | 4 commits: scaffold + refactor + bugfix + feature | git log --oneline              |
| Tests totales          | >12 tests verdes (3 originales + nuevos)          | PYTHONPATH=. pytest tests/ -v  |
| Diff del refactor      | Solo 2 ficheros, <50 líneas cambiadas             | git diff HEAD~3..HEAD~2 --stat |
| API estable            | process_order() mantiene misma firma              | Inspección visual              |
| No dependencias nuevas | Sin requirements nuevos                           | pip list (comparar con inicio) |
| Flake8 limpio          | Sin warnings                                      | flake8 src/                    |

## 7.7 Limpieza (OBLIGATORIA)

### **⚠️ Limpieza de recursos**

**Ejecutar al finalizar el laboratorio:**

```
1. Borrar el repositorio
rm -rf /tmp/codex-lab04

2. Verificar limpieza
ls /tmp/codex-lab04 2>/dev/null && echo 'ERROR' || echo 'OK: limpio'
```

## 8. Resumen del módulo y siguientes pasos

### 8.1 Conceptos clave aprendidos

| Concepto                  | Detalle                                                              |
|---------------------------|----------------------------------------------------------------------|
| Cuatro bloques del prompt | Brief + Constraints + Definition of Done + Validation Commands       |
| Contexto progresivo       | Lectura → Plan → Edición → Validación (nunca todo de golpe)          |
| Guardrails explícitos     | Restricciones negativas: ficheros permitidos, API inmutable, límites |
| @ file references         | Contexto explícito en CLI para dirigir la atención de Codex          |
| --cd y --add-dir          | Acotación física del workspace en monorepos                          |
| /review como guardrail    | Ciclo de revisión integrado antes de commit                          |
| Plantilla Feature         | Scope + constraints + acceptance criteria + validation               |
| Plantilla Bugfix          | Reproducción + hipótesis + fix mínimo + regresión                    |
| Plantilla Refactor        | Invariant (no behaviour change) + scope + métricas                   |
| Mid-turn steering         | Enter durante ejecución para corregir rumbo sin perder contexto      |

### 8.2 Preparación para el Módulo 5

En el Módulo 5 profundizaremos en AGENTS.md, Skills y contexto persistente: cómo dar instrucciones permanentes al agente que sobreviven entre sesiones y se comparten con el equipo. Para prepararse:

- Revisar la guía oficial de AGENTS.md: <https://developers.openai.com/codex/guides/agents-md/>
- Revisar la documentación de Skills: <https://developers.openai.com/codex/skills/>
- Pensar en qué convenciones y plantillas de este módulo querrías convertir en Skills permanentes.

## 9. Referencias y recursos

- Codex Prompting: <https://developers.openai.com/codex/prompting>
- Codex Workflows: <https://developers.openai.com/codex/workflows/>
- Codex Prompting Guide (Cookbook): [https://developers.openai.com/cookbook/examples/gpt-5/codex\\_prompting\\_guide/](https://developers.openai.com/cookbook/examples/gpt-5/codex_prompting_guide/)
- AGENTS.md guide: <https://developers.openai.com/codex/guides/agents-md/>
- PLANS.md for multi-hour tasks: [https://developers.openai.com/cookbook/articles/codex\\_exec\\_plans/](https://developers.openai.com/cookbook/articles/codex_exec_plans/)
- CLI features (@ references, /review, steering): <https://developers.openai.com/codex/cli/features/>
- IDE features (context, modes): <https://developers.openai.com/codex/ide/features/>
- Skills overview: <https://developers.openai.com/codex/skills/>
- Security (sandbox modes): <https://developers.openai.com/codex/security/>
- Config advanced (--cd, --add-dir): <https://developers.openai.com/codex/config-advanced/>