Clase 3 - Bootcamp AI Engineer con Python

Prompt Engineering para Sistemas

Instructor: Ramsés Camas | codigofacilito

MÓDULO I

¿Dónde estamos?

Módulo I — Fundamentos de AI Engineering

- **Clase 1:** Setup profesional + Hello AI System
- **Clase 2:** Tokens, contexto, costo y latencia
- **Clase 3:** Prompt engineering para sistemas
- **Clase 4:** Outputs estructurados y contratos

Hoy: los prompts dejan de ser "vibes" y se convierten en código evaluable.

OBJETIVO

Objetivo de hoy

1. Un módulo prompting/promptkit.py en tu repo
2. Un registry de prompts versionable
3. Plantillas reutilizables con variables
4. Chains (secuencias de prompts) componibles
5. Capacidad de medir si un prompt mejoró o empeoró

Lo que NO haremos: "tips para chatear mejor con ChatGPT"

CONCEPTO CLAVE

La diferencia fundamental

Prompt para chat

"Resúmeme este texto de forma clara y concisa"

Prompt para sistema

- Template con variables
- Contrato de salida esperado
- Versionado en el repo
- Testeable contra golden set
- Cambios rastreables

Un AI Engineer no "chatea" — programa instrucciones reproducibles.

/01

Anatomía de un System Prompt de producción

Las 6 secciones que todo prompt robusto necesita

ANATOMÍA

Las 6 secciones de un System Prompt

1 ROL

Quién es el modelo en este contexto

2 CONTEXTO

Información que el modelo necesita

3 INSTRUCCIONES

Qué debe hacer exactamente

4 FORMATO

Cómo estructurar la respuesta

5 RESTRICCIONES

Qué NO debe hacer

6 EJEMPLOS

Few-shot que anclan el comportamiento

Cada sección resuelve un tipo de error diferente.

ROL: Más que "eres un experto en X"

Malo

"Eres un experto en marketing"

Mejor

```
"Eres un analista de marketing B2B
para SaaS en LATAM.
Presupuestos < $10k USD/mes.
Audiencia: CTOs de 50-200 empleados."
```

¿Por qué funciona?

El rol reduce el espacio de respuestas posibles.
Cuanto más específico, menos variación.

Piensa: ¿qué persona real contratarías para esta tarea?

CONTEXTO: el diferenciador de sistemas serios

En un sistema de producción, el contexto es dinámico:

```
CONTEXTO:
- Fecha actual: {fecha}
- Usuario: {nombre_usuario} (rol: {rol})
- Documentos relevantes: {chunks_recuperados}
- Historial reciente: {ultimos_3_turnos}
```

Clave

El contexto se inyecta programáticamente.

No es algo que escribas una vez — es algo que tu código construye por request.

Esto es lo que haremos en RAG (clases 5-7): inyectar el contexto correcto.

INSTRUCCIONES: la parte que todo mundo subestima

Ambiguo

"Analiza el documento"

Claro

```
"Extrae las 3 ideas principales.
Para cada idea:
(a) resumen en 1 oración
(b) cita textual de respaldo
(c) confianza: alto/medio/bajo"
```

Principio: escribe instrucciones como si fueras un tech lead dando specs a un dev junior.

Regla: si dos personas razonables interpretarían tu instrucción diferente, el LLM también lo hará.

FORMATO: controla la salida

Sin formato

El LLM decide cómo responder
(markdown, lista, prosa...)

Con formato explícito:

```
{
  "resumen": "string (máx 100 palabras)",
  "entidades": ["string"],
  "sentimiento": "positivo | negativo | neutro",
  "confianza": 0.0 a 1.0
}
```

Esto habilita parsing programático. Si la respuesta no parsea, puedes reintentar.

En la Clase 4 formalizaremos esto con Pydantic + function calling.

RESTRICCIONES: los guardrails del prompt

- "NO inventes información que no esté en el contexto proporcionado."
- "Si no tienes suficiente información, responde: INSUFFICIENT_CONTEXT."
- "NO incluyas disclaimers ni explicaciones adicionales."
- "Máximo 200 tokens en la respuesta."
- "Responde SOLO en español."

Las restricciones son tan importantes como las instrucciones.
Un prompt sin restricciones es un programa sin validación de inputs.

EJEMPLOS (Few-shot): el arma más poderosa

Few-shot prompting: incluir 2-5 ejemplos de input/output correcto dentro del prompt.

¿Por qué funciona?

- Entiende el formato exacto esperado
- Calibra el tono y la profundidad
- Desambigua instrucciones

Regla práctica

0-shot: Solo tareas muy simples

2-3 shot: Sweet spot para la mayoría

5+ shot: Tareas muy específicas

Los ejemplos hacen el trabajo pesado. Las instrucciones solo clarifican edge cases.

FEW-SHOT EN ACCIÓN

Few-shot: ejemplo real

```
TAREA: Clasifica el ticket de soporte.

EJEMPLO 1:
Ticket: "No puedo iniciar sesión desde ayer"
Categoría: acceso | Prioridad: alta

EJEMPLO 2:
Ticket: "El reporte muestra datos de hace 2 meses"
Categoría: datos | Prioridad: media

EJEMPLO 3:
Ticket: "Se cayó toda la plataforma, 200 afectados"
Categoría: infraestructura | Prioridad: crítica

AHORA CLASIFICA:
Ticket: "{ticket_nuevo}"
```

TEMPLATES

Templates: prompts como código

```python
def build_prompt(ticket: str, ejemplos: list) -> str:
    examples_str = format_examples(ejemplos)
    return f"""
      ROL: Eres un clasificador de tickets.
      EJEMPLOS: {examples_str}
      TICKET: {ticket}
      RESPONDE en JSON: categoria, prioridad
    """
```

Ventajas

- Testeable
- Versionable en Git
- Reutilizable
- Cambios rastreables

Un prompt template NO es un string hardcodeado. Es una función.

REGISTRY

Prompt Registry: organiza tus prompts

¿Por qué un registry?

Un sistema real tiene 10-50 prompts diferentes

Sin registry: prompts regados en 20 archivos

Con registry: un solo lugar donde viven, se versionan y se obtienen

```
prompts/
  ├── registry.py
  ├── templates/
  │   ├── clasificador.py
  │   ├── resumidor.py
  │   └── extractor.py
  └── chains/
      └── analisis_completo.py
```

El registry es un diccionario centralizado de todos los prompts del sistema.

/02

Prompt Chaining y Evaluación

Secuencias de prompts + cómo medir si mejoran

CHAINING

Prompt Chaining: secuencias de prompts

Un solo prompt rara vez resuelve una tarea compleja.

Prompt 1

Extraer

Extrae entidades clave del texto

Prompt 2

Clasificar

Clasifica cada entidad por tipo

Prompt 3

Sintetizar

Genera resumen ejecutivo

¿Por qué no hacer todo en un solo prompt?

- Cada paso es más simple → menos errores
- Puedes testear cada paso independientemente
- Puedes reusar pasos en diferentes chains
- Puedes poner lógica de Python entre pasos

COMPARACIÓN

Chaining vs. un solo prompt largo

Un solo prompt largo

- ✓ Más barato (1 llamada)
- ✓ Más rápido (menos latencia)
- ✗ Si falla, falla todo
- ✗ Difícil de debuggear

Chain de 3 prompts

- ✗ Más caro (3 llamadas)
- ✗ Más lento
- ✓ Fallo aislado y recuperable
- ✓ Cada paso es testeable

En sistemas de producción, la confiabilidad gana casi siempre.

EVALUACIÓN

"Prompts evaluables" — el concepto clave

¿Puedes responder OBJETIVAMENTE: "¿Este prompt es mejor o peor que la versión anterior?"

Dataset de test

inputs + outputs esperados

Métrica de calidad

exactitud, F1, similitud...

Runner

ejecuta prompt contra dataset

Sin esto: "vibe-based prompt engineering" — cambias algo, pruebas con 2 inputs, dices "se ve bien".

Con esto: cambias algo, corres 50 tests, y sabes si mejoró un 12% o empeoró un 3%.

GOLDEN SET

Golden set: tu dataset de verdad

```
[
  {"input": "No puedo entrar a mi cuenta",
   "expected": {"categoria": "acceso",
                "prioridad": "alta"}},
  {"input": "El botón de pago no funciona",
   "expected": {"categoria": "bug",
                "prioridad": "crítica"}},
  {"input": "¿Tienen plan enterprise?",
   "expected": {"categoria": "ventas",
                "prioridad": "baja"}}
]
```

Composición

Inputs representativos
de tu caso de uso real

Outputs esperados
(la respuesta correcta)

Mínimo: 20-50 ejemplos
Ideal: 100+ con edge cases

Profundizaremos en Clases 14-15, pero el concepto empieza aquí.

MÉTRICAS

Métricas simples para evaluar prompts

Para empezar (hoy)

Exact match: ¿salida idéntica al expected?

JSON parseable: ¿salida es JSON válido?

Campos presentes: ¿tiene todos los campos?

Valores válidos: ¿valores en rango correcto?

Para después (Clase 15)

Faithfulness: ¿respuesta basada en el contexto?

Semantic similarity: ¿qué tan parecida al expected?

RAGAS: suite completa para RAG

Hoy nos enfocamos en las primeras. Son simples pero poderosas.

ITERACIÓN

El loop de mejora de prompts

1. Escribe prompt v1
2. Corre contra golden set
3. Mide las métricas
4. Identifica los fallos
5. Modifica el prompt
6. Corre de nuevo
7. Compara: ¿mejoró?
8. Si mejoró: commit

Esto es CI/CD para prompts. No "vibes". Cada versión queda en Git con su score.

/03

Patrones Avanzados

Chain of Thought · Role + Persona · Self-Consistency

CHAIN OF THOUGHT

Chain of Thought (CoT)

Sin CoT

"¿Cuál es el sentimiento de esta reseña?"

Con CoT

```
"Analiza paso a paso:
1. Identifica frases positivas
2. Identifica frases negativas
3. Pesa cuáles dominan
4. Clasificación final"
```

¿Cuándo usarlo?

- Tareas que requieren razonamiento
- Clasificaciones con matices
- Cuando necesitas explicabilidad

Trade-off

Costo: más tokens de salida

Beneficio: más precisión

ROLE + PERSONA

Role + Persona Technique

```
Eres María, ingeniera de soporte senior
con 8 años de experiencia en SaaS B2B.
Tu estilo de comunicación:
- Directo y técnico, sin rodeos
- Verificas información antes de responder
- Cuando no estás segura, lo dices
- Priorizas la solución sobre la explicación
```

Especialmente útil cuando:

- Necesitas consistencia de tono
- El estilo importa tanto como el contenido
- Construyes un chatbot con personalidad definida

SELF-CONSISTENCY

Self-Consistency

Ejecutar el mismo prompt N veces y tomar la respuesta más frecuente (votación por mayoría).

```
respuestas = [
    llm.call(prompt, temperature=0.7)
    for _ in range(5)
]
respuesta_final = majority_vote(respuestas)
```

¿Por qué funciona?

Los LLMs son estocásticos. Los errores aleatorios se cancelan entre sí. La respuesta consistente suele ser la correcta.

Trade-off: 5x más caro, 5x más lento, pero significativamente más confiable. Úsalo para decisiones de alto impacto.

ANTI-PATRONES

Anti-patrones: lo que NO hacer

- **"Sé creativo"** Ambiguo. ¿Qué significa para un clasificador?
- **Prompts de 5000 tokens sin estructura** El modelo pierde el hilo. Secciona.
- **Instrucciones contradictorias** "Sé conciso" + "Explica en detalle" → conflicto
- **Copiar prompts de internet** Tu caso de uso es único.
- **No versionar prompts** Cambiaste algo y empeoró. ¿Qué? Nadie sabe.
- **Evaluar con 1-2 ejemplos** Sesgo de confirmación garantizado.
- **Ignorar el system prompt** Es tu primera y mejor línea de control.

MODELOS

Gemini y Gemma 3: particularidades

Gemini (cloud, free tier)

- Muy bueno siguiendo formato
- System instruction como parámetro separado
- Respeta JSON con response_mime_type
- Free tier: ~15 RPM, ~1M tokens/día

Gemma 3 27B (local)

- Excelente para su tamaño
- Más sensible al system prompt
- Puede necesitar ejemplos explícitos
- Sin response_mime_type nativo

En el promptkit: abstraemos estas diferencias.

ARQUITECTURA

Arquitectura del promptkit.py

PromptTemplate

Clase que maneja variables + renderizado

PromptRegistry

Diccionario centralizado de templates

PromptChain

Secuencia de templates en orden

evaluate_prompt()

Corre prompt contra mini golden set

Todo vive en: prompting/promptkit.py — Usa llm_client.py (Clase 1) y tokenlab.py (Clase 2)

LIVE CODING

Código en vivo

Lo que vamos a construir en los próximos 30 minutos:

1. Definir la clase PromptTemplate con variables y renderizado
2. Crear un PromptRegistry con 2-3 templates de ejemplo
3. Implementar un PromptChain simple (2 pasos)
4. Escribir evaluate_prompt() con un mini golden set
5. Correr el prompt contra Gemini Y Gemma 3 y comparar

Repo actual → módulo nuevo: prompting/promptkit.py

RECAP

Los 5 conceptos clave

1. **TEMPLATES:** Los prompts son código, no strings sueltos.
2. **REGISTRY:** Centraliza, versiona y organiza todos los prompts.
3. **CHAINS:** Divide tareas complejas en pasos simples y componibles.
4. **EVALUACIÓN:** Sin métricas, no hay mejora — solo vibes.
5. **GOLDEN SET:** Tu dataset de verdad para medir progreso.

CONEXIÓN

Conexión con lo que viene

Clase 4 (siguiente)

Outputs validados con Pydantic · Function calling · Contratos para el resto del bootcamp

Clases 5–6 (RAG)

El promptkit construye prompts de generación con contexto recuperado

Clase 10 (ReAct)

Los prompts del agente ReAct son chains sofisticados con loops

El promptkit que construimos hoy es la base de TODO lo que viene.

¿Preguntas antes de la práctica?

- ¿Por qué es mejor un prompt template que un string hardcodeado?
- ¿Cuándo usarías chaining vs un solo prompt?
- ¿Qué es un golden set y por qué es necesario?

PRÁCTICA · 30 MIN

Práctica guiada

"Mejora el clasificador de tickets iterativamente"

Baseline Correr prompt v1, medir accuracy (~30-40%)

Few-shot Agregar 3 ejemplos + formato JSON + categorías válidas

Restricciones Solo JSON, categorías exactas, reglas de prioridad

Chain 2 pasos Extraer info → Clasificar con extracción

Gemini vs Gemma Comparar accuracy, latencia y tokens

Al final: ¿cuál versión ganó? ¿Por cuánto?

Para la próxima clase

Tarea (opcional pero recomendada)

- Agrega 1 template nuevo al registry para un caso de uso tuyo
- Escribe un golden set de 10 ejemplos para ese template
- Corre evaluate_prompt() y anota el score

Clase 4

"Outputs estructurados y contratos — Function calling + Pydantic"

Los prompts se vuelven CONTRATOS verificables programáticamente.