

lab_answer

April 3, 2026

1 Lesson 04 — Advanced Prompting

1.1 Lab Answer Notebook

Reference solutions for all exercises. Try lab.ipynb first.

```
[ ]: import json
from dotenv import load_dotenv
import anthropic

load_dotenv()
client = anthropic.Anthropic()
MODEL = "claude-3-5-haiku-20241022"

def call_claude(prompt: str, system: str = None, max_tokens: int = 1024) -> str:
    kwargs = dict(model=MODEL, max_tokens=max_tokens,
                  messages=[{"role": "user", "content": prompt}])
    if system:
        kwargs["system"] = system
    msg = client.messages.create(**kwargs)
    return msg.content[0].text

print("Setup complete.")
```

1.2 Exercise 1 — Chain-of-Thought (Reference Solution)

```
[ ]: WORD_PROBLEM = """A train leaves City A at 9:00 AM traveling at 75 mph toward_
    ↪City B.
    A second train leaves City B at 10:30 AM traveling toward City A at 55 mph.
    The distance between City A and City B is 280 miles.
    At what time do the two trains meet, and how far from City A is the meeting_
    ↪point?"""
```

```
[ ]: def solve_without_cot(problem: str) -> str:
    """Direct answer - no step-by-step reasoning."""
    # Keep max_tokens low to prevent model from reasoning even when not asked
    prompt = problem + "\n\nAnswer concisely with just the final answer. No_
    ↪step-by-step working."
```

```

return call_claude(prompt, max_tokens=100)

def solve_with_cot(problem: str) -> str:
    """Chain-of-thought - 'Let's think step by step' is the canonical trigger
    ↪(Wei et al. 2022)."""
    prompt = problem + "\n\nLet's think step by step."
    return call_claude(prompt, max_tokens=1024)

print("=" * 60)
print("WITHOUT CoT (3 runs)")
print("=" * 60)
for i in range(3):
    print(f"\nRun {i+1}:")
    print(solve_without_cot(WORD_PROBLEM))

print("\n" + "=" * 60)
print("WITH CoT (3 runs)")
print("=" * 60)
for i in range(3):
    print(f"\nRun {i+1}:")
    print(solve_with_cot(WORD_PROBLEM))

```

1.3 Exercise 2 — Few-Shot Classifier (Reference Solution)

```

[ ]: # 4 labeled examples - one per urgency level
# Each is clearly representative, diverse, and uses consistent format
FEW_SHOT_EXAMPLES = """Ticket: Our entire payment processing system is down. No
    ↪customers can complete purchases. Revenue loss ongoing since 08:00 UTC.
Urgency: CRITICAL

Ticket: The bulk export feature in the admin panel fails for datasets over
    ↪10,000 rows with a timeout. Workaround: export in smaller batches.
Urgency: HIGH

Ticket: The font in the settings panel looks slightly smaller than elsewhere in
    ↪the app. No functional impact.
Urgency: LOW

Ticket: EU timezone users see meeting times off by 1 hour. Affects ~15% of EU
    ↪users. Workaround: manually set timezone.
Urgency: MEDIUM"""

```

```

def classify_ticket(text: str) -> str:
    """Classify ticket urgency using few-shot prompting."""
    # Append new ticket after examples in the same format
    # Ask for only the label - keep max_tokens small
    prompt = (
        f"{FEW_SHOT_EXAMPLES}\n\n"
        f"Ticket: {text}\n"
        "Urgency:"
    )
    result = call_claude(prompt, max_tokens=10)
    result = result.strip().upper()
    for label in ["CRITICAL", "HIGH", "MEDIUM", "LOW"]:
        if label in result:
            return label
    return result

TEST_TICKETS = [
    "The entire API is returning 500 errors for all customers since 14:00 UTC.␣
    ↪Production is down.",
    "Our dashboard export to CSV is producing garbled characters for non-ASCII␣
    ↪names.",
    "Would be great if the dark mode also applied to the mobile app - currently␣
    ↪only on desktop.",
    "I forgot my password and the reset email is not arriving. I need to log in␣
    ↪for a meeting in 10 minutes.",
    "Some users report occasional slow loading of the reports page - not␣
    ↪consistent, maybe 1 in 10 loads.",
]
EXPECTED = ["CRITICAL", "HIGH", "LOW", "HIGH", "MEDIUM"]

correct = 0
for ticket, expected in zip(TEST_TICKETS, EXPECTED):
    result = classify_ticket(ticket)
    match = result == expected
    correct += int(match)
    status = "CORRECT" if match else f"WRONG (expected {expected})"
    print(f"[{result:8}] {status}")
    print(f" {ticket[:80]}")
    print()

print(f"Accuracy: {correct}/{len(TEST_TICKETS)}")

```

1.4 Exercise 3 — XML Extraction (Reference Solution)

```
[ ]: MEETING_NOTES = """Quick sync - Product & Engineering
Tuesday, March 19th, 2024, 10am

Present: Priya (PM), Jake (Eng Lead), Mei (Design), Carlos (Backend)

We kicked off with Priya walking through the Q2 roadmap. The team agreed to
prioritize the mobile redesign over the analytics dashboard - that's decided.
Jake will need to prepare a technical spec document for the mobile work by next
Friday (March 29). Mei is already working on mockups and will share them in_
↳Figma
by end of this week (March 22). Carlos pointed out we have a performance issue
in the search endpoint and he'll fix it by March 21 before it becomes_
↳customer-facing.

We also discussed the API rate limiting feature. No decision yet - needs more
scoping. Priya to write a one-pager on requirements by March 25.

The team agreed to move the weekly standup from Monday to Tuesday going forward.
"""
```

```
[ ]: def extract_with_xml(document_text: str) -> str:
    """Extract structured info from document using XML-tagged prompt.
    Using XML tags prevents Claude from confusing document content with_
    ↳instructions.
    """
    prompt = f"""<instructions>
Extract the key information from the document below.
Return valid JSON matching the schema in <output_format>.
Output only the JSON - no markdown fences, no explanation, no preamble.
</instructions>

<document>
{document_text}
</document>

<output_format>
{{
    "meeting_date": "string (e.g., March 19, 2024)",
    "attendees": ["list of attendee names"],
    "action_items": [
        {{
            "owner": "person's name", "task": "what they need to do", "deadline":_
↳"date string or null"}}
    ],
    "decisions": ["list of decisions made"]
}}
```

```

</output_format>"""
    return call_claude(prompt, max_tokens=1024)

# Run and validate
raw_result = extract_with_xml(MEETING_NOTES)
print("Raw response:")
print(raw_result)
print()

result = json.loads(raw_result)
print("JSON parse: SUCCESS")
print(f" Meeting date: {result.get('meeting_date')}")
print(f" Attendees: {result.get('attendees')}")
print(f" Action items ({len(result.get('action_items', []))}):")
for item in result.get('action_items', []):
    print(f"     - {item.get('owner')}: {item.get('task')} (by {item.
        ↳get('deadline')})")
print(f" Decisions: {result.get('decisions')}")
assert len(result.get('action_items', [])) >= 1
print("\nValidation: PASSED")

```