

lab_answer

April 3, 2026

1 Lesson 03 — Prompt Engineering Fundamentals

1.1 Lab Answer Notebook

Reference solutions for all exercises. Try lab.ipynb first before consulting these answers.

```
[ ]: # Setup
import os
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) -> str:
    kwargs = {
        "model": MODEL,
        "max_tokens": 1024,
        "messages": [{"role": "user", "content": prompt}]
    }
    if system:
        kwargs["system"] = system
    response = client.messages.create(**kwargs)
    return response.content[0].text

def compare_prompts(original: str, improved: str, label: str = "") -> None:
    print(f"{'='*60}")
    if label:
        print(f" {label}")
    print(f"{'='*60}")
    print(f"\n--- ORIGINAL PROMPT ---\n{original}\n")
    original_out = call_claude(original)
    print(f"--- ORIGINAL OUTPUT ---\n{original_out}\n")
    print(f"--- IMPROVED PROMPT ---\n{improved}\n")
    improved_out = call_claude(improved)
    print(f"--- IMPROVED OUTPUT ---\n{improved_out}\n")
```

```
print("Setup complete.")
```

```
[ ]: # The 5 poor prompts
POOR_PROMPTS = [
    "write something about climate",
    "summarize this",
    "be helpful",
    "fix my code",
    "what do you think about AI?",
]
```

1.2 Exercise 1 — RTCF Rewrites (Reference Solutions)

```
[ ]: # Reference solutions - RTCF rewrites for each poor prompt

improved_prompts = [
    # Prompt 1: "write something about climate"
    # Role: science journalist; Task: write explainer; Context: general
    ↪audience + specific topic;
    # Format: 200 words, one analogy
    (
        "You are a science journalist writing for a general adult audience. "
        "Write a 200-word explainer on why Arctic sea ice loss accelerates
    ↪global warming. "
        "Include one analogy to make the concept concrete. "
        "Tone: clear and engaging, no jargon without definition."
    ),

    # Prompt 2: "summarize this" (no document)
    # Role: research assistant; Task: summarize; Context: document provided +
    ↪target audience;
    # Format: 3 bullets, one sentence each
    (
        "You are a research assistant. "
        "Summarize the following article in exactly 3 bullet points for a busy
    ↪product manager. "
        "Each bullet must be one sentence. Focus on business implications. Omit
    ↪technical methodology.\n\n"
        "ARTICLE:\n"
        "Researchers at MIT published findings showing that transformer models
    ↪fine-tuned on "
        "domain-specific data outperform general models by 23% on specialized
    ↪tasks. "
        "The study evaluated 12 models across healthcare, legal, and financial
    ↪domains. "
    )
]
```

```

        "Fine-tuning costs dropped 40% compared to 2022 baselines due to LoRA_
↪techniques. "
        "The team recommends domain fine-tuning for any production use case_
↪with consistent input formats."
    ),

    # Prompt 3: "be helpful"
    # Role: customer support agent; Task: answer support questions; Context:_
↪SaaS product;
    # Format: concise, escalation path if unknown
    (
        "You are a friendly and professional customer support agent for a B2B_
↪SaaS project management tool. "
        "Answer the user's support question as concisely as possible (1-3_
↪sentences). "
        "If you do not know the answer, say so clearly and offer to escalate to_
↪the technical team. "
        "Do not guess or make up product features. Tone: warm but efficient."
    ),

    # Prompt 4: "fix my code"
    # Role: Python expert; Task: fix specific bug; Context: code + error_
↪description;
    # Format: return corrected function only, no explanation
    (
        "You are an expert Python developer. "
        "The function below raises a KeyError when the 'user_id' key is missing_
↪from the input dict. "
        "Fix it so it returns None when the key is absent instead of raising an_
↪error. "
        "Return only the corrected function - no explanation, no markdown code_
↪fences.\n\n"
        "def get_user(data):\n"
        "    return data['user_id']"
    ),

    # Prompt 5: "what do you think about AI?"
    # Role: technology analyst; Task: list specific risks; Context: economic_
↪focus + 2030 horizon;
    # Format: numbered list, 3 items, one sentence + industry example each
    (
        "You are a technology analyst specializing in labor economics. "
        "List exactly 3 concrete economic risks of widespread AI automation by_
↪2030. "
        "For each risk, name a specific industry most likely to be affected. "

```

```

        "Format: numbered list, one sentence per item. Tone: neutral and
        ↪analytical."
    ),
]

print(f"Loaded {len(improved_prompts)} reference solutions.")

```

```

[ ]: # Run comparisons
for i, (poor, improved) in enumerate(zip(POOR_PROMPTS, improved_prompts),
    ↪start=1):
    compare_prompts(poor, improved, label=f"Prompt {i}")

```

1.3 Exercise 2 — Format Control (Reference Solutions)

```

[ ]: CONTACT_TEXT = """The project kickoff meeting was attended by Sarah Chen (sarah.
    ↪chen@techcorp.io),
lead engineer Marcus Webb at m.webb@devstudio.com, and the client contact
Priya Nair - her email is priya.nair@clientco.org. Also cc'd were Jordan Ellis
(jellis@techcorp.io) and the PM, Dana Okafor, reachable at d.okafor@pm.io."""

```

```

[ ]: # Format A - JSON array
# Key technique: specify exact key names, prohibit markdown fences, require
    ↪parseable JSON
json_prompt = (
    "Extract every person's name and email address from the text below. "
    "Return a JSON array of objects. Each object must have exactly two keys: "
    "'name' (string) and 'email' (string). "
    "Output only the JSON array - no markdown fences, no explanation, no
    ↪preamble. "
    "The output must be valid JSON parseable with json.loads().\n\n"
    f"TEXT:\n{CONTACT_TEXT}"
)

json_output = call_claude(json_prompt)
print("Raw output:")
print(json_output)

parsed = json.loads(json_output)
assert len(parsed) == 5, f"Expected 5 contacts, got {len(parsed)}"
print(f"\nParsed successfully: {len(parsed)} contacts found.")
for c in parsed:
    print(f"  {c['name']} <{c['email']}>")

```

```

[ ]: # Format B - Markdown table
# Key technique: specify header row + separator, prohibit extra text

```

```

table_prompt = (
    "Extract every person's name and email address from the text below. "
    "Return a markdown table with exactly two columns: Name and Email. "
    "Include a header row and a separator row (---|---). "
    "Output only the table - no text before or after it.\n\n"
    f"TEXT:\n{CONTACT_TEXT}"
)

print("Markdown table output:")
print(call_claude(table_prompt))

```

```

[ ]: # Format C - Plain bullet list
# Key technique: specify exact bullet format, no header/footer, no extra
↳ commentary
list_prompt = (
    "Extract every person's name and email address from the text below. "
    "Return only a plain bullet list - no header, no footer, no explanation. "
    "Each bullet must follow this exact format: - [full name]: [email_
↳ address]\n\n"
    f"TEXT:\n{CONTACT_TEXT}"
)

print("Bullet list output:")
print(call_claude(list_prompt))

```