

lab_answer

April 3, 2026

1 Lesson 06 — Structured Outputs & JSON Mode (ANSWERS)

Complete, working implementations of all three exercises with explanatory comments.

```
[ ]: import json
import anthropic
from pydantic import BaseModel, ValidationError
from dotenv import load_dotenv

load_dotenv()
client = anthropic.Anthropic()

JD1 = """Senior Python Developer at TechCorp, London. Remote OK. £80-100K.
Must have: 5+ yrs Python, FastAPI, PostgreSQL.
Nice to have: Kubernetes, Rust."""

JD2 = """Marketing Manager - GrowthCo, NYC. On-site. $75,000-$90,000/yr.
Requirements: 3+ years B2B SaaS marketing, HubSpot experience, strong writing
↳skills.
Lead a team of 2."""

JD3 = """Junior ML Engineer, Berlin, Germany. Hybrid (3 days office). €45K-€60K.
Requirements: Python, PyTorch or TensorFlow, math fundamentals.
Recent graduates welcome."""

JOB_DESCRIPTIONS = [JD1, JD2, JD3]
print('Setup complete.')
```

1.1 Exercise 1 — Prompt-Based JSON Extraction (Solution)

```
[ ]: class JobPosting(BaseModel):
    """Structured representation of a job posting."""
    title: str
    company: str
    location: str
    salary_range: str | None = None # Optional - many JDs don't list salary
    requirements: list[str]         # List of individual requirement strings
    remote: bool                   # True only if explicitly remote/remote-OK
```

```

def _strip_fences(text: str) -> str:
    """Remove markdown code fences that some models add around JSON."""
    text = text.strip()
    if text.startswith("```"):
        # Remove opening fence line (e.g., ```json)
        text = text.split("\n", 1)[1] if "\n" in text else text[3:]
        # Remove closing fence
        if text.rstrip().endswith("```"):
            text = text.rstrip()[:-3].strip()
    return text

def extract_prompt_based(text: str) -> JobPosting:
    """Extract a JobPosting using a prompt-only approach."""
    schema = {
        "title": "str",
        "company": "str",
        "location": "str",
        "salary_range": "str or null",
        "requirements": "[str]",
        "remote": "bool"
    }
    response = client.messages.create(
        model="claude-haiku-4-5",
        max_tokens=500,
        messages=[{
            "role": "user",
            "content": (
                "Extract structured job posting data from the text below. "
                "Respond with ONLY valid JSON - no markdown fences, no prose, no preamble.\n"
                f"Schema: {json.dumps(schema)}\n\n"
                f"Job description:\n{text}"
            )
        }]
    )
    raw = _strip_fences(response.content[0].text)
    data = json.loads(raw) # may raise JSONDecodeError
    return JobPosting.model_validate(data) # may raise ValidationError

print("=== Exercise 1: Prompt-Based Extraction ===")
for i, jd in enumerate(JOB_DESCRIPTIONS, 1):
    posting = extract_prompt_based(jd)
    print(f"JD{i}:")

```

```

for k, v in posting.model_dump().items():
    print(f" {k}: {v}")
print()

```

1.2 Exercise 2 — Tool Use Extraction (Solution)

```

[ ]: # Define the tool using a hand-crafted JSON Schema.
# (Alternatively: start from JobPosting.model_json_schema() and clean it up.)
EXTRACT_TOOL = {
    "name": "extract_job_posting",
    "description": "Extract structured job posting data from raw job_
↳description text.",
    "input_schema": {
        "type": "object",
        "properties": {
            "title": {
                "type": "string",
                "description": "Job title (e.g., 'Senior Python Developer')"
            },
            "company": {
                "type": "string",
                "description": "Company name"
            },
            "location": {
                "type": "string",
                "description": "City and country or region (e.g., 'London, UK')"
            },
            "salary_range": {
                "type": ["string", "null"],
                "description": "Salary range as a string (e.g., '£80-100K'), or_
↳null if not stated"
            },
            "requirements": {
                "type": "array",
                "items": {"type": "string"},
                "description": "List of individual requirements or must-have_
↳skills"
            },
            "remote": {
                "type": "boolean",
                "description": "True if the role is fully remote or_
↳remote-friendly"
            }
        },
        "required": ["title", "company", "location", "requirements", "remote"]
    }
}

```

```

def extract_tool_based(text: str) -> JobPosting:
    """Extract a JobPosting using Claude tool use - schema is enforced by the
    API."""
    response = client.messages.create(
        model="claude-haiku-4-5",
        max_tokens=500,
        tools=[EXTRACT_TOOL],
        # Force the model to call this specific tool:
        tool_choice={"type": "tool", "name": "extract_job_posting"},
        messages=[{
            "role": "user",
            "content": f"Extract the job posting data:\n\n{text}"
        }]
    )
    # Find the ToolUseBlock in the response
    for block in response.content:
        if block.type == "tool_use" and block.name == "extract_job_posting":
            # block.input is already a dict that matches the schema
            return JobPosting.model_validate(block.input)
    raise RuntimeError("No tool_use block found in response")

print("=== Exercise 2: Tool Use Extraction ===")
for i, jd in enumerate(JOB_DESCRIPTIONS, 1):
    posting = extract_tool_based(jd)
    print(f"JD{i}:")
    for k, v in posting.model_dump().items():
        print(f"  {k}: {v}")
    print()

```

1.3 Exercise 3 — Retry on Validation Error (Solution)

```

[ ]: def extract_with_retry(text: str, max_retries: int = 3) -> JobPosting:
    """Extract a JobPosting with retry, feeding errors back to Claude for
    self-correction."""
    initial_prompt = (
        "Extract job posting data as JSON matching this schema exactly:\n"
        '{"title": str, "company": str, "location": str, '
        '"salary_range": str|null, "requirements": [str], "remote": bool}\n\n'
        "Return ONLY valid JSON. No markdown, no prose.\n\n"
        f"Job description:\n\n{text}"
    )
    messages = [{"role": "user", "content": initial_prompt}]
    last_exception = None

```

```

for attempt in range(1, max_retries + 1):
    print(f" Attempt {attempt}/{max_retries}...")
    response = client.messages.create(
        model="claude-haiku-4-5",
        max_tokens=500,
        messages=messages
    )
    raw = response.content[0].text.strip()

    try:
        raw_clean = raw
        if raw_clean.startswith("`"):
            raw_clean = raw_clean.split("\n", 1)[1] if "\n" in raw_clean
        ↪else raw_clean[3:]
            if raw_clean.rstrip().endswith("`"):
                raw_clean = raw_clean.rstrip()[:-3].strip()
        data = json.loads(raw_clean)
        posting = JobPosting.model_validate(data)
        print(f" Success on attempt {attempt}.")
        return posting

    except json.JSONDecodeError as e:
        last_exception = e
        error_msg = f"JSON parse error: {e}\nReturn valid JSON only - no ↪
        ↪markdown fences, no preamble."
    except ValidationError as e:
        last_exception = e
        error_msg = (
            f"The JSON was valid but failed schema validation:\n{e}\n\n"
            "Please fix the specific fields mentioned and return corrected ↪
        ↪JSON only."
        )

    print(f" Error: {str(last_exception)[:120]}")
    # Append assistant's bad response and the correction request
    messages.append({"role": "assistant", "content": raw})
    messages.append({"role": "user", "content": error_msg})

    raise last_exception

# Test on normal JDs
print("=== Exercise 3: Retry Extraction (normal JDs) ===")
for i, jd in enumerate(JOB_DESCRIPTIONS, 1):
    print(f"\nJD{i}:")
    posting = extract_with_retry(jd)
    print(f" Result: {posting.model_dump()}")

```

```
# Test on ambiguous input
print("\n=== Ambiguous Input Test ===")
AMBIGUOUS_JD = "We're looking for someone awesome to join our growing team.␣
↳Good pay. Cool office. Tech stuff. Apply now!"
try:
    result = extract_with_retry(AMBIGUOUS_JD)
    print("Extracted (best effort):", result.model_dump())
except Exception as e:
    print(f"Failed after all retries: {e}")
```