

lab_answer

April 3, 2026

1 Lab 07 — Tool Use & Function Calling (ANSWERS)

Build a multi-tool, multi-turn agent loop with Claude.

Exercises: 1. Implement `execute_tool()` for three tools 2. Build the full tool loop (`run_with_tools()`) 3. Test parallel tool calls

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

load_dotenv()
client = anthropic.Anthropic()
MODEL = "claude-sonnet-4-20250514"
print("Setup complete.")
```

```
[ ]: # Tool Definitions (provided)

TOOLS = [
    {
        "name": "calculator",
        "description": "Perform basic arithmetic: add, subtract, multiply, ↵
↵divide.",
        "input_schema": {
            "type": "object",
            "properties": {
                "operation": {
                    "type": "string",
                    "enum": ["add", "subtract", "multiply", "divide"],
                    "description": "The arithmetic operation to perform."
                },
                "a": {"type": "number", "description": "First operand."},
                "b": {"type": "number", "description": "Second operand."}
            },
            "required": ["operation", "a", "b"]
        }
    },
]
```

```

{
    "name": "unit_converter",
    "description": "Convert a value between common units (km/miles, kg/lb,↵
↵celsius/fahrenheit).",
    "input_schema": {
        "type": "object",
        "properties": {
            "value": {"type": "number", "description": "The numeric value↵
↵to convert."},
            "from_unit": {
                "type": "string",
                "enum": ["km", "miles", "kg", "lb", "celsius",↵
↵"fahrenheit"],
                "description": "Source unit."
            },
            "to_unit": {
                "type": "string",
                "enum": ["km", "miles", "kg", "lb", "celsius",↵
↵"fahrenheit"],
                "description": "Target unit."
            }
        },
        "required": ["value", "from_unit", "to_unit"]
    },
}

{
    "name": "string_tools",
    "description": "Perform string operations: upper, lower, reverse,↵
↵count_words.",
    "input_schema": {
        "type": "object",
        "properties": {
            "operation": {
                "type": "string",
                "enum": ["upper", "lower", "reverse", "count_words"],
                "description": "The string operation to perform."
            },
            "text": {"type": "string", "description": "The input text."}
        },
        "required": ["operation", "text"]
    }
}

]
print(f"{len(TOOLS)} tools defined: {[t['name'] for t in TOOLS]}")

```

1.1 Exercise 1 — Implement execute_tool()

```
[ ]: def execute_tool(tool_name: str, tool_input: dict) -> str:
    """Execute a tool by name with the given input dict. Return result as a
    ↪ string."""

    # Calculator
    if tool_name == "calculator":
        op = tool_input["operation"]
        a, b = tool_input["a"], tool_input["b"]
        if op == "add":
            return str(a + b)
        elif op == "subtract":
            return str(a - b)
        elif op == "multiply":
            return str(a * b)
        elif op == "divide":
            return str(a / b) if b != 0 else "Error: division by zero"
        else:
            return f"Unknown operation: {op}"

    # Unit Converter
    elif tool_name == "unit_converter":
        value = tool_input["value"]
        from_u = tool_input["from_unit"]
        to_u = tool_input["to_unit"]
        conversions = {
            ("km", "miles"): lambda v: v * 0.621371,
            ("miles", "km"): lambda v: v * 1.60934,
            ("kg", "lb"): lambda v: v * 2.20462,
            ("lb", "kg"): lambda v: v * 0.453592,
            ("celsius", "fahrenheit"): lambda v: (v * 9 / 5) + 32,
            ("fahrenheit", "celsius"): lambda v: (v - 32) * 5 / 9,
        }
        key = (from_u, to_u)
        if key in conversions:
            converted = conversions[key](value)
            return f"{converted:.4f} {to_u}"
        else:
            return f"Unsupported conversion: {from_u} -> {to_u}"

    # String Tools
    elif tool_name == "string_tools":
        op = tool_input["operation"]
        text = tool_input["text"]
        if op == "upper":
            return text.upper()
```

```

elif op == "lower":
    return text.lower()
elif op == "reverse":
    return text[::-1]
elif op == "count_words":
    return str(len(text.split()))
else:
    return f"Unknown operation: {op}"

return f"Unknown tool: {tool_name}"

```

```

[ ]: # Test Exercise 1
print(execute_tool("calculator", {"operation": "multiply", "a": 17, "b": 843}))
# Expected: 14331

print(execute_tool("unit_converter", {"value": 100, "from_unit": "km",
    ↪ "to_unit": "miles"}))
# Expected: 62.1371 miles

print(execute_tool("string_tools", {"operation": "reverse", "text": "Hello,
    ↪ World!"}))
# Expected: !dlroW ,olleH

print(execute_tool("string_tools", {"operation": "count_words", "text": "the
    ↪ quick brown fox"}))
# Expected: 4

```

1.2 Exercise 2 — The Full Tool Loop

```

[ ]: def run_with_tools(user_message: str, tools: list, max_iterations: int = 10) ->
    ↪ str:
    """Send user_message to Claude with tools. Handle the full tool loop.
    ↪ Return final text."""
    messages = [{"role": "user", "content": user_message}]

    for iteration in range(max_iterations):
        # Call Claude with tools
        response = client.messages.create(
            model=MODEL,
            max_tokens=1024,
            tools=tools,
            messages=messages,
        )

        print(f" Round {iteration + 1}: stop_reason={response.stop_reason}")

```

```

    # If Claude is done, extract and return the text
    if response.stop_reason == "end_turn":
        for block in response.content:
            if hasattr(block, "text"):
                return block.text
        return "(no text in response)"

    # Handle tool_use: execute each tool call and collect results
    if response.stop_reason == "tool_use":
        tool_results = []
        for block in response.content:
            if block.type == "tool_use":
                print(f"    Tool call: {block.name}({json.dumps(block.
↪input))}")

                result = execute_tool(block.name, block.input)
                print(f"    Result: {result}")
                tool_results.append({
                    "type": "tool_result",
                    "tool_use_id": block.id,
                    "content": result,
                })

        # Append assistant message (with tool_use blocks) and tool results
        messages.append({"role": "assistant", "content": response.content})
        messages.append({"role": "user", "content": tool_results})

    return "(max iterations reached - tool loop did not converge)"

```

```

[ ]: # Test Exercise 2
answer = run_with_tools(
    "What is the square root of 144 multiplied by the number of words in "
    "'the quick brown fox'? Use your tools to compute the exact answer.",
    TOOLS
)
print("FINAL ANSWER:", answer)

```

1.3 Exercise 3 — Parallel Tool Calls

```

[ ]: # Exercise 3: Parallel tool calls - observe whether Claude parallelises
# two independent tool calls in a single round.

query = (
    "Convert 100 USD to EUR (use 1 USD = 0.92 EUR, so multiply 100 by 0.92) "
    "AND calculate 15% tip on $47.50 (multiply 47.50 by 0.15). "
    "Give me both results. Use your tools."
)

```

```

)

messages = [{"role": "user", "content": query}]
round_num = 0

while round_num < 10: # guard against infinite loop
    response = client.messages.create(
        model=MODEL, tools=TOOLS, messages=messages, max_tokens=512
    )

    tool_calls_this_round = [b for b in response.content if b.type == "tool_use"]
    text_blocks = [b for b in response.content if hasattr(b, "text")]

    print(f"Round {round_num}: stop_reason={response.stop_reason}, "
          f"tool_calls={len(tool_calls_this_round)}")
    for tc in tool_calls_this_round:
        print(f"  -> {tc.name}({json.dumps(tc.input)})")

    # If done, print final text and break
    if response.stop_reason == "end_turn":
        for tb in text_blocks:
            print(f"\nFINAL: {tb.text}")
        break

    # Execute all tool calls and feed results back
    tool_results = []
    for tc in tool_calls_this_round:
        result = execute_tool(tc.name, tc.input)
        print(f"    result: {result}")
        tool_results.append({
            "type": "tool_result",
            "tool_use_id": tc.id,
            "content": result,
        })

    messages.append({"role": "assistant", "content": response.content})
    messages.append({"role": "user", "content": tool_results})
    round_num += 1

# Comment: If Round 0 shows 2 tool calls, Claude parallelised them.
# If it shows 1, the calls were sequential across rounds.

```