

lab_answer

April 3, 2026

1 Lab 12 — MCP Concepts & Protocol (ANSWERS)

Explore the MCP protocol: connect to servers, call tools, and survey the ecosystem.

Exercises: 1. Connect to an MCP server and list its tools 2. Call a tool via the MCP protocol 3. Survey 5 popular MCP servers

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

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

```
[ ]: # Mock MCP Server

class MockMCPTool:
    def __init__(self, name, description, input_schema):
        self.name = name
        self.description = description
        self.inputSchema = input_schema

class MockMCPResult:
    def __init__(self, tools):
        self.tools = tools

class MockToolContent:
    def __init__(self, text):
        self.type = "text"
        self.text = text

class MockToolResult:
    def __init__(self, content):
        self.content = content

class MockMCPSession:
    def __init__(self):
```

```

        self._tools = [
            MockMCPTool("read_file", "Read the contents of a file",
                {"type": "object", "properties": {"path": {"type": "string"}}, "required": ["path"]}),
            MockMCPTool("list_directory", "List files in a directory",
                {"type": "object", "properties": {"path": {"type": "string"}}, "required": ["path"]}),
            MockMCPTool("search_files", "Search for files matching a pattern",
                {"type": "object", "properties": {"pattern": {"type": "string"}, "path": {"type": "string"}}, "required": ["pattern"]}),
        ]
        self._files = {
            "/tmp/test.txt": "Hello from MCP!\nThis is a test file.",
            "/tmp/config.json": '{"key": "value", "debug": true}',
        }

    async def initialize(self):
        return {"protocolVersion": "2025-03-26", "capabilities": {"tools": {}}}

    async def list_tools(self):
        return MockMCPResult(self._tools)

    async def call_tool(self, name, arguments):
        if name == "read_file":
            path = arguments.get("path", "")
            content = self._files.get(path, f"Error: file not found: {path}")
            return MockToolResult([MockToolContent(content)])
        elif name == "list_directory":
            return MockToolResult([MockToolContent("test.txt\nconfig.json")])
        elif name == "search_files":
            return MockToolResult([MockToolContent(f"Found: /tmp/test.txt")])
        return MockToolResult([MockToolContent(f"Unknown tool: {name}")])

print("Mock MCP session ready.")

```

1.1 Exercise 1 — MCP Client: Connect & List Tools

```

[ ]: async def exercise_1():
    session = MockMCPSession()

    # Initialize the connection
    init_result = await session.initialize()
    print(f"Connected! Protocol version: {init_result['protocolVersion']}")
    print(f"Capabilities: {json.dumps(init_result['capabilities'])}\n")

```

```

# List available tools
tools_result = await session.list_tools()
print(f"Found {len(tools_result.tools)} tools:\n")
for tool in tools_result.tools:
    print(f"Tool: {tool.name}")
    print(f"  Description: {tool.description}")
    print(f"  Parameters: {json.dumps(tool.inputSchema)}")
    print()

await exercise_1()

```

1.2 Exercise 2 — Call a Tool via MCP

```

[ ]: async def exercise_2():
    session = MockMCPSession()
    await session.initialize()

    # Call read_file with a valid path
    print("=== read_file(/tmp/test.txt) ===")
    result = await session.call_tool("read_file", {"path": "/tmp/test.txt"})
    print(f"Result: {result.content[0].text}\n")

    # Call list_directory
    print("=== list_directory(/tmp) ===")
    result = await session.call_tool("list_directory", {"path": "/tmp"})
    print(f"Result: {result.content[0].text}\n")

    # Handle error: non-existent file
    print("=== read_file (non-existent) ===")
    result = await session.call_tool("read_file", {"path": "/tmp/missing.txt"})
    text = result.content[0].text
    if text.startswith("Error:"):
        print(f"Handled error: {text}")
    else:
        print(f"Result: {text}")

await exercise_2()

```

1.3 Exercise 3 — MCP Server Survey

```

[ ]: # Survey of 5 popular MCP servers
servers = [
    {
        "name": "Filesystem",

```

```

        "source": "Official (Anthropic)",
        "tools": "read_file, write_file, list_directory, search_files,
↳get_file_info, create_directory, move_file",
        "resources": "None",
        "transport": "stdio",
        "use_case": "Let Claude read and write files on your local machine
↳within allowed directories"
    },
    {
        "name": "GitHub",
        "source": "Official (Anthropic)",
        "tools": "create_issue, search_repos, get_file_contents,
↳create_pull_request, list_commits, etc.",
        "resources": "repo://<owner>/<repo> for repository content",
        "transport": "stdio",
        "use_case": "Manage GitHub repos, issues, and PRs directly from Claude"
    },
    {
        "name": "PostgreSQL",
        "source": "Official (Anthropic)",
        "tools": "query (read-only SQL execution)",
        "resources": "db://schema for table schemas",
        "transport": "stdio",
        "use_case": "Query databases and explore schemas for data analysis"
    },
    {
        "name": "Brave Search",
        "source": "Official (Anthropic)",
        "tools": "brave_web_search, brave_local_search",
        "resources": "None",
        "transport": "stdio",
        "use_case": "Give Claude web search capabilities via the Brave Search
↳API"
    },
    {
        "name": "Puppeteer",
        "source": "Official (Anthropic)",
        "tools": "navigate, screenshot, click, fill, evaluate",
        "resources": "None",
        "transport": "stdio",
        "use_case": "Browser automation - navigate pages, fill forms, take
↳screenshots"
    },
]

# Print as a formatted table
print(f"{'Name':<15} {'Source':<20} {'Transport':<10} {'Use Case'}")

```

```

print("-" * 90)
for s in servers:
    print(f"{s['name']:<15} {s['source']:<20} {s['transport']:<10}␣
    ↪{s['use_case'][:50]}")

print("\n\nDetailed capabilities:")
for s in servers:
    print(f"\n--- {s['name']} ---")
    print(f"  Tools: {s['tools']}")
    print(f"  Resources: {s['resources']}")

```