Week 4: Agent Skills - Teaching Claude New Capabilities

01. Session Goals

- Understand the Agent Skills specification
- Create effective SKILL.md files
- Build skills for data analytics and GTM use cases
- Learn progressive disclosure and skill organization

02. Block 1: Theory - What Are Agent Skills? (30 min)

The Problem Skills Solve
Claude is powerful but generic. Skills let you:

- Teach Claude your specific workflows
- Encode team knowledge and standards
- Create reusable capabilities
- Ensure consistent behavior

Why Skills Matter: SOPs for Your Agent
Think of skills as SOPs (Standard Operating Procedures) for Claude.

In any organization, you don't train every new hire by having them shadow you forever. You write down the process: "Here's how we score leads. Here's our data quality checklist. Here's the format for weekly reports." Then they follow the SOP.

Skills work the same way. You have knowledge Claude doesn't have:

- How your team scores leads
- What makes a good data quality check at your company
- The specific steps for your weekly reporting workflow
- Your industry's benchmarks and red flags

Skills let you write that expertise down once and have Claude use it automatically, forever.

Real examples:

Your Expertise	Without a Skill	With a Skill
Lead scoring criteria	Explain rubric every time	Claude scores leads your way automatically
Data quality standards	Manually check each dataset	Claude applies your standards consistently
Email writing style	Edit every draft Claude writes	Claude writes in your voice from the start
Report format	Copy-paste template, fix formatting	Claude generates reports in your exact format

This is the core skill of this bootcamp: turning what's in your head into something an agent can use.

How to Build Skills: Start With Your Source of Truth
The most important principle for building skills: start with what already works.

Source of Truth	Example
How you already do the work	Your personal process for scoring leads, your email templates
The best person on your team	How your top sales rep qualifies deals, how your best analyst profiles data
A best practices doc	Your company's style guide, your team's SOP for customer research
A trusted external source	A methodology from a respected practitioner, an industry framework you've vetted

The right workflow:

1. Find or create your source of truth (the "golden" version)
2. Write it down as clear, step-by-step instructions
3. Format it as a SKILL.md with the right structure
4. Test it with Claude and iterate

You can use AI to help build the skill. Once you have your source of truth, Claude can help you structure it, identify gaps, and format it correctly. But AI assists the process, it doesn't replace the expertise.

What NOT to do:

Anti-Pattern	Why It Fails
Ask AI to "create a lead scoring skill" from scratch	AI will generate generic content that doesn't reflect your actual criteria
Copy skills from "awesome-claude-code" repos online	Some influencer's workflow isn't your workflow. Their scoring rubric isn't your rubric.
Start with AI and hope it matches your process	You end up with a skill that looks professional but produces wrong outputs

The best skills encode real expertise. They capture what the best person on your team actually does, not what a random AI thinks they should do. Start with your truth, then use AI to help structure and improve it.

The Architecture: Incremental File Exploration
Skills are part of an emerging pattern in agent design: incremental file exploration.

The naive approach is to dump all knowledge into the prompt upfront. "Here's everything you might need to know." This hits context limits fast and wastes tokens on irrelevant information.

The better approach: let the agent explore and load knowledge on-demand.

```
Traditional approach:
+--------------------------------------------+
| System prompt with ALL instructions        |  <- Context bloat
| + ALL examples + ALL rubrics + ALL formats  |
+--------------------------------------------+

Skills approach:
+--------------------------------------------+
| Lean system prompt                         |
| + Skill names and descriptions (lightweight)|
+--------------------------------------------+
              v (when needed)
+--------------------------------------------+
| Load specific SKILL.md                     |  <- On-demand
| -> Load references/ as needed               |
+--------------------------------------------+
```

How it works:

1. At startup, Claude only loads skill `name` and `description` (a few lines each)
2. When your request matches a skill, Claude loads the full SKILL.md
3. If that skill references detailed docs, Claude loads those only when needed
4. The agent manages its own context window by exploring incrementally

This is the same pattern we saw in Week 2 with filesystem + bash. Instead of stuffing everything into the prompt, the agent retrieves context as it needs it. Skills extend this pattern to domain knowledge.

How Skills Work
A skill is just a folder with a SKILL.md file. Claude reads it like a recipe and follows the steps.

```
.claude/skills/lead-scorer/
+-- SKILL.md              # Your instructions
+-- references/
|   +-- scoring-rubric.md # Detailed criteria (loaded on-demand)
+-- scripts/
    +-- validate.py       # Optional automation
```

When you ask Claude something that matches a skill's description, Claude asks permission to use it, loads the instructions, and works from your playbook instead of improvising.

The key difference from tools:

Tools execute and return results. Skills prepare the agent to solve a problem. When Claude invokes a skill, it loads the SKILL.md as new instructions, adjusts its execution context, and continues with this enriched environment. Skills change how Claude thinks, not just what it can do.

The Story Behind Skills
Agent Skills started inside Anthropic as a way to make Claude Code more useful for specialized tasks. As models got more capable, the team needed a scalable way to equip agents with domain expertise without bloating the base prompt.

At an internal hackathon, teams built skills for everything from code review to customer research. The pattern worked so well that Anthropic productized it.

October 2025: Anthropic launched Agent Skills publicly. They also released a "skill-creator" skill that uses Claude to generate new skills interactively.

December 2025: Anthropic published the Agent Skills specification as an open standard at agentskills.io, following the same playbook as MCP.

Skills as an Open Standard
The format is now supported across major AI tools:

Platform	Status
Claude (Claude.ai, Claude Code, Agent SDK)	Native
GitHub Copilot (VS Code, CLI, coding agent)	Native
OpenAI Codex CLI	Native
Cursor, Gemini CLI, and others	Native or compatible

Skills you create work across this ecosystem. Put a SKILL.md in `.claude/skills/` or `.github/skills/`, and most agents pick it up automatically. Your investment in writing skills pays off regardless of which AI tools you use.

Anthropic also launched a directory with skills from commercial partners: Atlassian, Canva, Cloudflare, Figma, Notion, Ramp, and Sentry. These are production-grade examples of how companies package their workflows as skills.

References:

- Anthropic: Introducing Agent Skills (https://www.anthropic.com/news/skills)
- Anthropic Engineering: Equipping Agents for the Real World (https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills)
- Agent Skills Open Standard (https://agentskills.io)
- GitHub: Agent Skills Repository (https://github.com/anthropics/skills)

Skills vs Slash Commands

Aspect	Slash Commands	Agent Skills
Invocation	Explicit: `/command`	Automatic: Claude detects
Structure	Single `.md` file	Directory with `SKILL.md`
Complexity	Simple prompts	Multi-file workflows
Discovery	Manual	Automatic (by description)

SKILL.md Structure

```
---
name: skill-name
description: What it does and when to use it. Include trigger keywords.
---

# Skill Title

## Instructions
Step-by-step guidance for Claude.

## Examples
Concrete input/output examples.
```

YAML Frontmatter Fields

Field	Required	Description
`name`	Yes	Lowercase, hyphens only (max 64 chars)
`description`	Yes	What + when (max 1024 chars)
`allowed-tools`	No	Restrict tools: `Read, Grep, Bash(python:*)`
`context`	No	`fork` for isolated sub-agent context
`hooks`	No	PreToolUse, PostToolUse, Stop handlers

Demo

Show a skill in action:

1. Create a simple skill
2. Trigger it with a matching request
3. See Claude ask permission and execute

---

03. Block 2: Lab 1 - Your First Skill (30 min)

Task: Create a Database Profiler Skill
Build a skill that profiles the startup funding database systematically.

Step 1: Create the skill directory:

```
mkdir -p .claude/skills/database-profiler
```

Step 2: Create `SKILL.md`:

```
---
name: database-profiler
description: Profile SQLite databases to understand structure, quality, and statistics. Use when ana
---

# Database Profiler

When profiling a database, provide:

## 1. Structure Overview
- List all tables
- For each table: column names, data types, row count
- Identify primary keys and foreign key relationships

## 2. Quality Assessment
- Missing values per column (count and percentage)
- Duplicate rows per table
- Referential integrity issues (orphaned foreign keys)

## 3. Statistical Summary
For numeric columns:
- Min, max, mean, median
- Standard deviation
- Outlier candidates (beyond 3 std devs)

For categorical columns:
- Unique value count
- Most common values (top 5)
- Distribution skew

For date columns:
- Date range (earliest to latest)
- Gaps or clustering

## 4. Recommendations
Based on findings, suggest:
- Data quality issues to address
- Interesting patterns to explore
- Next steps for analysis

## Context Management
- Always use LIMIT when exploring individual tables
- Start with aggregations before drilling down
- Report row counts so analyst knows the scale

## Output Format

Use markdown tables for statistics. Be concise but thorough.
```

Step 3: Test the skill:

```
> Profile the startup-funding.db database
```

Watch for:

- Claude asking to use the skill
- Structured output following the template
- Context-aware queries (using LIMIT, aggregations first)

Success Criteria

- [ ] Skill created in correct location
- [ ] Claude discovers and asks to use it
- [ ] Output follows the defined structure

---

04. BREAK (10 min)

---

05. Block 3: Theory - Advanced Skill Patterns (30 min)

Multi-File Skills
For complex skills, add supporting files:

```
.claude/skills/data-analyst/
+-- SKILL.md              # Main instructions
+-- references/
|   +-- sql-patterns.md   # Query templates
|   +-- analysis-loop.md  # Methodology
+-- scripts/
    +-- validate.py       # Validation script
```

Reference files load on demand:

```
For SQL query patterns, see [sql-patterns.md](references/sql-patterns.md).
```

Progressive Disclosure
Keep `SKILL.md` under 500 lines. Structure for efficiency:

```
## Quick Start
[Essential instructions - always loaded]

## Detailed Reference
See [reference.md](references/reference.md) for complete documentation.

## Utility Scripts
Run validation: `python scripts/validate.py input.csv`
```

Description Best Practices

Bad:

```
description: Helps with data.
```

Good:

```
description: Analyze datasets using the Data Analysis Loop (Monitor -> Explore -> Craft -> Impact).
```

Include:

- What it does (capabilities)
- When to use it (trigger scenarios)
- Keywords users might say

Restricting Tools
Use `allowed-tools` for focused skills:

```
---
name: read-only-analyzer
description: Analyze code without making changes
allowed-tools: Read, Grep, Glob
---
```

Patterns:

- `Read` - exact tool name
- `Bash(python:*)` - Bash with python prefix only
- `Bash(git:*)` - Git commands only

Hooks for Skills
Add validation or logging:

```
---
name: secure-operations
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh $TOOL_INPUT"
---
```

---

06. Block 4: Lab 2 - Build a Data Analysis Skill (45 min)

Task: Create a Data Analysis Skill with Guardrails
Build a comprehensive skill that encodes the Data Analysis Loop from Week 2, with context management built in.

Step 1: Create skill structure:

```
mkdir -p .claude/skills/data-analyst/references
```

Step 2: Create `SKILL.md`:

```
---
name: data-analyst
description: Systematic data analysis using the Data Analysis Loop (Monitor -> Explore -> Craft -> I
allowed-tools: Read, Grep, Bash(sqlite3:*), WebSearch, WebFetch
---

# Data Analyst

You are a data analyst. Follow the Data Analysis Loop and context management rules below.

## The Data Analysis Loop

Every analysis follows four phases:

### 1. Monitor
- Run aggregation queries to check key metrics
- Compare current values to historical baselines
- Flag anomalies (significant deviations from average)

### 2. Explore
- When you spot something interesting, dig deeper
- Segment the data: by time, category, cohort
- Look for external context: what else was happening?

### 3. Craft Story
- Synthesize findings into 3-5 key insights
- Support each insight with specific data
- Note confidence level and caveats

### 4. Impact
- Recommend concrete next actions
- Size the opportunity if possible
- Identify what additional data would help

## Context Management Rules (Critical)

1. **Always use LIMIT** - Default to 100 rows for exploration
2. **Track truncation** - Note when results are limited
3. **Aggregate first** - Start with GROUP BY, then drill down
4. **One question at a time** - Don't try to answer everything in one query

## SQL Patterns

For common analysis patterns, see [sql-patterns.md](references/sql-patterns.md).

## Output Format

Structure every analysis as:

**Summary:** 2-3 sentences

**Key Findings:**
- Finding 1 (metric: X, change: Y%)
- Finding 2 (metric: X, change: Y%)
- Finding 3 (metric: X, change: Y%)

**Context:** External factors that might explain trends

**Recommendation:** Next action to take
```

Step 3: Create `references/sql-patterns.md`:

```
# SQL Patterns for Data Analysis

## Time-Series Aggregation
```

SELECT

strftime('%Y-%m', date_column) as month,

COUNT(*) as count,

SUM(amount) as total,

AVG(amount) as average

FROM table

WHERE date_column >= date('now', '-12 months')

GROUP BY month

ORDER BY month DESC;

```
## Window Functions for Rankings
```

-- Rank within categories

SELECT

category,

item,

value,

RANK() OVER (PARTITION BY category ORDER BY value DESC) as rank

FROM table;

-- Running totals

SELECT

date,

amount,

SUM(amount) OVER (ORDER BY date) as running_total

FROM table;

-- Compare to previous period

SELECT

month,

revenue,

LAG(revenue) OVER (ORDER BY month) as prev_month,

revenue - LAG(revenue) OVER (ORDER BY month) as change

FROM monthly_revenue;

```
## Cohort Analysis
```

WITH cohorts AS (

SELECT

startup_id,

MIN(funding_date) as first_funding_date

FROM funding_rounds

GROUP BY startup_id

)

SELECT

strftime('%Y', c.first_funding_date) as cohort_year,

COUNT(DISTINCT c.startup_id) as cohort_size,

COUNT(DISTINCT CASE

WHEN fr.stage = 'Series A' THEN c.startup_id

END) as reached_series_a

FROM cohorts c

LEFT JOIN funding_rounds fr ON c.startup_id = fr.startup_id

GROUP BY cohort_year

ORDER BY cohort_year;

```
## Conversion Funnel
```

WITH stages AS (

SELECT

startup_id,

MAX(CASE WHEN stage = 'Seed' THEN 1 ELSE 0 END) as has_seed,

MAX(CASE WHEN stage = 'Series A' THEN 1 ELSE 0 END) as has_series_a,

MAX(CASE WHEN stage = 'Series B' THEN 1 ELSE 0 END) as has_series_b

FROM funding_rounds

GROUP BY startup_id

)

SELECT

SUM(has_seed) as seed_companies,

SUM(has_series_a) as series_a_companies,

SUM(has_series_b) as series_b_companies,

ROUND(SUM(has_series_a) * 100.0 / SUM(has_seed), 1) as seed_to_a_rate,

ROUND(SUM(has_series_b) * 100.0 / SUM(has_series_a), 1) as a_to_b_rate

FROM stages;

```
## Funding Velocity
```

-- Days between funding rounds

WITH round_sequence AS (

SELECT

startup_id,

stage,

funding_date,

LAG(funding_date) OVER (PARTITION BY startup_id ORDER BY funding_date) as prev_round_date

FROM funding_rounds

)

SELECT

s.name,

rs.stage,

CAST(JULIANDAY(rs.funding_date) - JULIANDAY(rs.prev_round_date) AS INTEGER) as days_since_last_round

FROM round_sequence rs

JOIN startups s ON rs.startup_id = s.id

WHERE rs.prev_round_date IS NOT NULL

ORDER BY days_since_last_round ASC

LIMIT 20;

Step 4: Test with analytical questions:

```
> Analyze the AI/ML funding landscape in the database. What trends do you see?

> Which startups have the fastest funding velocity (shortest time between rounds)?

> Build a cohort analysis: of companies that raised Seed in 2022, how many reached Series A?
```

Deliverable

- Working data-analyst skill with references
- Screenshot showing the Data Analysis Loop in action
- At least one analysis that produces structured output

---

07. Bonus: Data Visualization Skill (Optional Lab Extension)

Building a Visualization Generator Skill
Skills can include scripts that Claude executes. This is powerful for data visualization, where you want consistent chart styles and formats.

Create the skill structure:

```
mkdir -p .claude/skills/data-visualizer/scripts
```

Create `SKILL.md`:

```
---
name: data-visualizer
description: Generate charts and visualizations from data. Use when asked to visualize data, create
allowed-tools: Read, Bash, Write
---

# Data Visualizer

Generate professional visualizations from query results or data files.

## Supported Chart Types

| Type | Use When |
|------|----------|
| Bar chart | Comparing categories |
| Line chart | Showing trends over time |
| Scatter plot | Showing correlations |
| Pie chart | Showing proportions |

## Process

1. Get the data (from query or file)
2. Identify the best chart type for the question
3. Run the visualization script with appropriate parameters
4. Save the output image to `output/charts/`

## Execution

Use the Python script in `scripts/visualize.py`:

\`\`\`bash
python .claude/skills/data-visualizer/scripts/visualize.py \
  --input data.csv \
  --chart-type bar \
  --x-column category \
  --y-column value \
  --title "My Chart" \
  --output output/charts/chart.png
\`\`\`

## Chart Selection Guide

- "Compare X across categories" -> Bar chart
- "Show trend over time" -> Line chart
- "Correlation between X and Y" -> Scatter plot
- "Distribution/proportion" -> Pie chart
```

Create `scripts/visualize.py`:

```
#!/usr/bin/env python3
"""
Data visualization script for the data-visualizer skill.
Generates charts from CSV data using matplotlib.
"""

import argparse
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('Agg')  # Non-interactive backend

def create_bar_chart(df, x_col, y_col, title, output):
    plt.figure(figsize=(10, 6))
    plt.bar(df[x_col], df[y_col], color='steelblue')
    plt.xlabel(x_col)
    plt.ylabel(y_col)
    plt.title(title)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig(output, dpi=150)
    print(f"Chart saved to {output}")

def create_line_chart(df, x_col, y_col, title, output):
    plt.figure(figsize=(10, 6))
    plt.plot(df[x_col], df[y_col], marker='o', color='steelblue')
    plt.xlabel(x_col)
    plt.ylabel(y_col)
    plt.title(title)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig(output, dpi=150)
    print(f"Chart saved to {output}")

def create_scatter_plot(df, x_col, y_col, title, output):
    plt.figure(figsize=(10, 6))
    plt.scatter(df[x_col], df[y_col], alpha=0.6, color='steelblue')
    plt.xlabel(x_col)
    plt.ylabel(y_col)
    plt.title(title)
    plt.tight_layout()
    plt.savefig(output, dpi=150)
    print(f"Chart saved to {output}")

def create_pie_chart(df, label_col, value_col, title, output):
    plt.figure(figsize=(10, 8))
    plt.pie(df[value_col], labels=df[label_col], autopct='%1.1f%%')
    plt.title(title)
    plt.tight_layout()
    plt.savefig(output, dpi=150)
    print(f"Chart saved to {output}")

def main():
    parser = argparse.ArgumentParser(description='Generate charts from CSV data')
    parser.add_argument('--input', required=True, help='Input CSV file')
    parser.add_argument('--chart-type', required=True,
                        choices=['bar', 'line', 'scatter', 'pie'])
    parser.add_argument('--x-column', required=True, help='X-axis column')
    parser.add_argument('--y-column', required=True, help='Y-axis column')
    parser.add_argument('--title', default='Chart', help='Chart title')
```

Test the skill:

```
> Query the funding database for total funding by industry, then visualize it as a bar chart
```

Creating Interactive Dashboards

For dashboards that users can interact with in a browser, Claude can generate HTML, CSS, and JavaScript.

When to use this approach:

- Stakeholders want a visual interface
- Data needs to be presented in multiple views
- You want filtering, sorting, or drill-down capabilities

Example prompt:

```
> Create an HTML dashboard that displays:
> 1. Funding by industry (bar chart)
> 2. Funding over time (line chart)
> 3. A filterable table of all startups
>
> Use Chart.js for the charts. Make it look professional.
> Save to output/dashboard/index.html
```

Claude will generate a complete HTML file with embedded CSS and JavaScript using Chart.js. Open the file in a browser to see your dashboard.

---

08. Wrap-Up (15 min)

Key Takeaways

1. Skills = Reusable Knowledge - Encode workflows Claude can discover
2. Description is Critical - Include what, when, and keywords
3. Progressive Disclosure - Keep SKILL.md lean, reference details
4. Test Thoroughly - Verify discovery and output quality
5. Data Analysis Loop - Encode methodology, not just tools

Homework

Create Two Skills for Your Project:

Build two skills that encode expertise from your domain. Each skill should capture a repeatable workflow.

Domain	Example Skills
GTM/Sales	Lead scorer, email writer, company researcher, pipeline forecaster
Developer Tools	Code reviewer, documentation generator, PR summarizer
Content/Marketing	Content brief writer, SEO analyzer, repurposing guide
Customer Support	Ticket classifier, response drafter, escalation checker
Operations	Invoice processor, compliance checker, report formatter
Data Analytics	Data profiler, anomaly detector, trend analyzer, report generator

For each skill:

- Clear trigger description (when should it activate?)
- Step-by-step instructions Claude can follow
- Defined output format
- Put detailed references in `references/` subdirectory

Submit:

- Skill directories
- Screenshot of each skill in action
- Brief description of when each triggers

Next Week Preview
Week 5: Sub-agents - Orchestrating specialized agents for complex workflows

---

09. Facilitator Notes

Common Issues
1. Skill not triggering: Check description keywords match request
2. Wrong location: Must be `.claude/skills/` not `skills/`
3. YAML errors: Check frontmatter syntax (spaces, not tabs)
4. Reference not loading: Verify relative path is correct

Skill Quality Checklist
- [ ] Name is lowercase with hyphens
- [ ] Description includes what + when + keywords
- [ ] Instructions are step-by-step
- [ ] Output format is defined
- [ ] Examples are included

Timing Adjustments
- Lab 1 is essential - ensure everyone completes
- Lab 2 can be simplified to just SKILL.md if time short

- References can be homework if needed