How I'd Teach a 10 Year Old to Build Agentic Workflows (Claude Code)

This guide covers everything you need to understand agentic workflows, set up your environment, and build your first agentic workflow using Claude Code, even if you've never written a line of code.

By: Nate Herk

I. What Are Agentic Workflows?

With **traditional workflow automation**, you map out every single step yourself. You drag nodes onto a canvas, connect them together, set up API calls, and make sure the right data flows into the right spots. If you skip a step or mess something up, the whole thing falls apart.

With **agentic workflows**, you just tell the system what you want, and it figures out how to get there on its own. It may ask you clarifying questions to make sure it delivers exactly what you need, but you're not responsible for defining every step.

The Restaurant Analogy: Traditional automation is like cooking dinner yourself from a recipe, you follow every step, measure every ingredient, and if you skip something, the meal is ruined. Agentic workflows are like walking into a restaurant and telling the waiter, "I want a delicious steak dinner." You don't tell them how to season it or what temperature the oven should be. They might ask, "How do you want your steak cooked?" but once you say "medium rare," they handle everything else until you get your end result.

Practical Example: You need contact info for 50 dentists in Chicago so you can send each one a personalized outreach message. With an agentic workflow, you don't map out every step, you just tell the agent what you need. It figures out how to scrape the data, where to scrape it from, how to find the contacts, and how to generate personalized emails. It may ask you, "What services do you want to offer them?" to make the emails better, but that's it.

Side-by-Side Comparison:

- **Traditional automation:** Trigger → Tool Call → Tool Call → AI Step → Conditional Logic → Final Output
- **Agentic workflow:** Input → (Maybe answer a question) → Output

---

II. Setting Up Your Environment

Step 1: Download Visual Studio Code (VS Code)

Go to Google, search for "Visual Studio Code," and download it for free. This is the environment where you'll use Claude Code.

Step 2: Install the Claude Code Extension

Once VS Code is open, click on **Extensions** in the left-hand sidebar, search for **"Claude Code,"** and install the extension. You will need a paid Claude plan (starting at $17/month) which includes Claude Code with Opus 4.5. Once installed, the extension will prompt you to log in with your account.

Step 3: Open a Project Folder

Click on **Explorer** in the left-hand sidebar. Create a new folder on your computer (e.g., "Agentic Workflows Demo") and open it in VS Code. This is where Claude Code will build and organize all your files.

Step 4: Open Claude Code

Click the Claude logo button at the top of VS Code (it says "Claude Code Open"). Close the main welcome window, and you'll see a chat interface on the right-hand side, this is where you interact with your coding agent.

Your screen layout:

- **Left side:** File explorer, where Claude Code creates and organizes files
- **Right side:** Chat interface, where you plan, ask questions, and the agent executes actions

---

III. The WAT Framework (Workflows, Agent, Tools)

This is the framework used to build agentic workflows. It has three components:

Agent, Claude Code itself. This is the AI brain you talk with. It builds workflows and tools, makes decisions, and executes actions.

Workflows, Written in **Markdown** (natural language documents with headers, bullet points, and bold text). These are the processes and instructions, like a recipe that tells the agent what steps to follow.

Tools, Written in **Python** (.py files). These are the actions the agent can take, like individual ingredients. You don't need to read or write this code yourself; Claude Code handles it.

The Cake Analogy: The agent is a chef who needs to make a cake. The chef either reads a pre-existing workflow (recipe) or builds its own. The recipe says things like "crack two eggs into a bowl, add a cup of flour", those ingredients (eggs, flour, sugar) are the tools. The chef (agent) uses a combination of recipes (workflows) and ingredients (tools) to produce the final result.

IV. The CLAUDE.md File

Every new Claude Code project needs a **claude.md** file. This is the agent's job description, it tells Claude Code how to operate within the WAT framework. Think of it like an onboarding document for a new employee.

What the claude.md file contains:

- **Framework definition:** Explains the three layers (workflows, agents, tools)
- **Why it matters:** Without structure, AI accuracy drops fast, if each step is 90% accurate, you're down to 59% success after just five steps
- **How to operate:** Look for existing tools first, learn and adapt when things fail, keep workflows current
- **Self-improvement loop:** The agent updates its own workflows and tools when it encounters and solves problems
- **File structure:** Organized folders for workflows, tools, and temporary files so nothing gets messy

You can reuse this same claude.md file for any project that uses the WAT framework. It's available for download in the free Skool community (linked in the video description).

V. Claude Code Modes

At the bottom of the Claude Code interface, you can select different autonomy levels:

Mode	What It Does
Plan Mode	Agent only thinks and plans, doesn't make any changes. Best for starting a new workflow.
Ask Before Edits	Agent asks your permission before making changes.

Edit Automatically	Agent makes changes without asking.
Bypass Permissions	Full autonomy, agent just goes. Only use this if you're actively watching.

To enable Bypass Permissions: Go to VS Code Settings → search "Claude Code" → enable "Allow Dangerously Skip Permissions."

Best Practice: Always start in **Plan Mode** to define your project requirements before switching to a more autonomous mode for execution.

VI. Setting Up an MCP Server (Firecrawl Example)

What Is MCP?

MCP (Model Context Protocol) gives the agent access to all the tools within a service, so it can figure out which ones to use, when to use them, and what parameters to fill, without you needing to understand the technical details.

The Supermarket Analogy: Instead of sending the agent to the egg store for eggs, the flour store for flour, and the candy store for frosting, you just give the agent access to the supermarket (MCP server) and say, "Whenever you need an ingredient, go grab it. I don't care how, just figure it out."

Installing Firecrawl MCP

Firecrawl is a tool that can scrape, crawl, search, extract, and map website data. Here's how to connect it:

1. Go to Firecrawl's documentation and find the MCP server setup instructions for Claude Code
2. Copy the installation command
3. In Claude Code, say: *"Hey Claude, I want you to help me install the Firecrawl MCP server. You need to install it using this command..."* and paste the command
4. For your API key, create a **.env file** in your project and store the key there rather than giving it directly to Claude Code (security best practice)
5. Sign up at Firecrawl to get your API key (free tier includes 500 credits)
6. Paste your API key into the .env file, save it, and have Claude Code run the installation command

Security Note: Always store API keys in a .env file. If you're working with sensitive keys, have Claude Code walk you through running the command in your own terminal so the key never appears in conversation history.

VII. Building Your First Agentic Workflow (Demo Walkthrough)

Demo 1: Scraping Job Listings

The Goal: Scrape ~200 social media job listings from DailyRemote.com and output them to an Excel spreadsheet.

Step 1, Plan Mode. Paste the URL and describe what you want in natural language:

> *"Hey Claude, I just gave you a URL for a website that has a bunch of job opportunities. There are about 622 job opportunities here, but they're spread across 21 pages. I want you to scrape all of those for me and put them into an Excel sheet. Make sure you're getting all of the relevant fields. Let me know if you can help me make that project requirement more robust, and feel free to ask me any questions."*

Step 2, Answer clarifying questions. The agent asked:

- Should it scrape individual job detail pages or just the listings? → **Just the listings**
- Where to save the Excel file? → **Local, in the temporary folder**
- Any filters, or grab everything? → **Grab 200 for now as a proof of concept**

Step 3, Review and approve the plan. The agent produced a comprehensive plan including a tool (scrape daily remote), a workflow (scrape job listings), and execution steps. Approved and switched to auto-accept.

Result: 209 jobs scraped into an Excel sheet with columns for job title, job type, position, location, experience, category, salary, description, summary, tags, and URL. The agent also created a reusable tool and workflow file.

Demo 2: Filtered Job Search (Sales Jobs in Europe + US)

Given a vaguer prompt with bypass permissions enabled, the agent:

1. Recognized it could reuse the existing scraping tool
2. Found only 52 sales jobs in Europe (fewer than the 500 requested)
3. Proactively asked if it should expand the search to include US-based sales jobs
4. Created temporary Python tools to help filter the data

5. Delivered 372 sales jobs in a formatted Excel sheet with a region column for easy filtering

Demo 3: Dentist Lead Generation (Completely New Use Case)

With no preparation for this type of task, the agent:

1. Checked existing tools and workflows
2. Decided to use Firecrawl to search for dentist directories
3. Tried the ADA site first, discovered it used JavaScript dynamic loading and pivoted to Yellow Pages
4. Created a brand-new scraping tool and workflow for dentist leads
5. Hit a parsing issue (only 2 results), identified the bug, fixed the regex pattern, and updated the tool
6. Delivered 120 unique dentist leads from 4 major cities with phone number, address, city, state, zip code, website, specialties, and listing URL

VIII. Context Management

Claude Code shows you how much context window is remaining (displayed at the bottom of the interface). As conversation history grows, accuracy can degrade, this is called **context rot**.

Best Practice: When context usage exceeds ~60%, compact the conversation. Claude Code will summarize everything important from the conversation history so you can keep going with a fresh context window while retaining key knowledge.

IX. Common Mistakes to Avoid

Mistake 1: Not Being Clear Enough About the Goal

Don't say: *"I need a lead scraper for LinkedIn."*, That's too vague.

Instead, use **Plan Mode** and say: *"Hey, here's a rough idea of what I want. Help me turn this into a solid project requirement doc (PRD)."* The agent will brainstorm, research, and ask you all the right questions so it knows exactly what to build. You're the manager keeping it on the right path, the agent is the expert.

Mistake 2: Not Defining What "Done" Looks Like

Without a clear finish line, agents may overcomplicate things, keep researching, keep looping, or waste time when the answer was simple.

Don't say: *"Search for LinkedIn profiles of CEOs at tech companies."*

Instead say: *"I need exactly 75 LinkedIn profiles of CEOs at tech companies. Put them in a spreadsheet with their name, company, email, and profile link. Once you have 75, you're done."*

Clear input + clear output = consistent results.

X. Why Agentic Workflows Are Better

Self-healing, no more debugging loops. With traditional automation, edge cases break the system and you spend hours reading logs and error messages. Agentic workflows handle errors automatically, the agent identifies what went wrong, thinks about a different approach, fixes it, and updates its workflows and tools so it doesn't happen again.

Natural language control. With traditional tools, you had to learn every node, read API documentation, find endpoints, structure JSON, and set up authentication. With agentic workflows, you just explain what you want and the system figures out the rest using MCP servers or by researching API docs on its own.

Gets smarter over time. Every time the agent runs into an issue, it learns and updates its tools and workflows. No manual reconfiguration needed.

XI. Important Caveat: Triggered vs. Scheduled Automations

There's a key difference between automations you trigger yourself and ones that run on a schedule:

Human-triggered (you're at your desk using Claude Code): The agent is right there with you. You can watch it, talk to it, and it self-heals in real time.

Scheduled or event-triggered (runs at 6 AM, or when a form is submitted): You deploy the code (workflows and tools), but not the agent itself. The Claude Code model that lives in VS Code is not deployed, only the workflows and tools it created are. The agent is what makes things self-healing, so deployed automations won't have that same real-time adaptability.

XII. Pro Tips

- **Run multiple agents simultaneously.** You can open multiple Claude Code instances, have each one try a different approach, test all of them, and keep the best result.
- **Always start in Plan Mode** before building anything, let the agent ask questions and create a solid plan first.
- **Use the WAT framework consistently**, give every new project a claude.md file so the agent knows how to organize and improve its work.
- **You don't need to know how to code.** The agent handles all the Python. You just need to clearly describe what you want and guide the direction.

Want to connect with others building and monetizing AI automation?

Become an AIS Plus Member