Build your first AI agent

An agent that finds internships for you — @hinumpy

This is the exact code from the reel. It pulls real internship listings that are live right now, filters them, and tells you which one to apply to first.

It needs zero installs. No pip, no frameworks. Just python and one file.

Part 1 — get it running

Step 1: save the file

1. Copy everything in part 4 of this doc
2. Open any text editor and paste it in
3. Save it as agent.py on your desktop

Step 2: try it with no setup

Open terminal, go to the folder you saved it in, and run:

```
cd Desktop
python agent.py --demo "machine learning internships"
```

This runs the search without any ai. It proves the tools work. You should see real companies with real dates.

Step 3: turn on the actual agent

Get a key at console.anthropic.com — sign in, click api keys, create key. Add about five dollars of credit. Each run costs way less than a cent.

```
export ANTHROPIC_API_KEY="sk-ant-your-key-here"
python agent.py "find me ai internships posted this week"
```

How you know it worked The second line says: mode live agent — claude-sonnet-4-5. If it still says no ANTHROPIC_API_KEY found, the export did not stick.

Export only lasts for that terminal window. To make it permanent:

```
echo 'export ANTHROPIC_API_KEY="sk-ant-your-key-here"' >> ~/.zshrc
source ~/.zshrc
```

Other things you can run

```
python agent.py "software engineering internships in nyc"
python agent.py "remote data science internships"
python agent.py --source all "ai internships"    # merges both job boards
```

Part 2 — what an agent actually is

Strip away the hype and an agent is four things. That is it.

1. one small job

Not a general assistant. One specific task. Mine finds internships. The smaller the job, the easier it is to build and debug.

2. a model that already exists

Do not train your own. Use claude, gpt, gemini, whatever. You need one that can reason and return structured output.

3. tools

This is the part everyone skips. A tool is literally just a python function. Mine has two: one that searches jobs, one that saves them to a file. You describe each function to the model in plain english and it decides when to call it.

4. the loop

```
goal  ->  model picks a tool
      ->  tool runs
      ->  result goes back to the model
      ->  repeat until it has an answer
```

That loop is the heartbeat of every agent you have ever heard of. Everything else is decoration.

Part 3 — the three things that broke

I am including these because nobody shows you the broken part, and the broken part is where you actually learn.

Bug 1: the model ignored my tool

I gave it a function to save my shortlist. It just did not use it. No error, no crash. It decided it did not need to.

You do not call your tools. The model picks them. I had to write "you MUST call save_shortlist" into the system prompt before it would.

Bug 2: two job boards, two spellings

One board writes NYC. The other writes New York, NY. Same city. Searching one format against the other returned zero results, silently.

Fixed with a small alias map. This is why the boring data cleanup part matters more than the ai part.

Bug 3: a field that lied by omission

The sponsorship field has four values, and 231 of 371 roles say Other — which means the company said nothing at all. The model was about to read that as good news.

Now the tool returns "not stated — check the posting" and the prompt says stay quiet unless it is explicit. Never let your agent make a claim your data cannot back up.

> **The lesson** In all three cases the model was fine. The data and the instructions were the problem. That is where you will spend most of your time.

Part 4 — the full code

Copy everything below into agent.py.

```
"""
internship agent — an AI agent that hunts internships for you.

the loop every agent runs:
    goal -> model decides -> tool runs -> result goes back to model -> repeat ->
answer

run it:
    export ANTHROPIC_API_KEY="sk-ant-..."
    python agent.py "find me machine learning internships in nyc"

no api key? run the tool layer on its own:
    python agent.py --demo "ml internships nyc"

pull from a different board:
    python agent.py --source simplify "ml internships"
    python agent.py --source all "ml internships"      # merges both, dedupes
"""

import json
import os
import sys
import time
import urllib.request

# ---------------------------------------------------------------- config

SOURCES = {
    "vansh": "https://raw.githubusercontent.com/vanshb03/"
             "Summer2027-Internships/dev/.github/scripts/listings.json",
    "simplify": "https://raw.githubusercontent.com/SimplifyJobs/"
                "Summer2027-Internships/dev/.github/scripts/listings.json",
```

```
}
DEFAULT_SOURCE = "vansh"
CACHE_HOURS = 6
MODEL = "claude-sonnet-4-5-20250929"

P = "\033[38;5;205m"   # pink
M = "\033[38;5;138m"   # mauve
D = "\033[38;5;245m"   # grey
B = "\033[1m"
X = "\033[0m"


def log(tag, msg, color=P):
    print(f"{color}{B}{tag:>9}{X} {msg}")


# ---------------------------------------------------------------- data
# the two boards store the same job differently. normalize before searching,
# so the rest of the code never has to care where a listing came from.

def normalize(job, source):
    season = job.get("season") or ""
    terms = job.get("terms") or ([season] if season else [])
    return {
        "company": job.get("company_name", "unknown"),
        "title": job.get("title", "unknown"),
        "locations": job.get("locations") or [],
        "terms": [t for t in terms if t],
        "category": job.get("category", ""),
        "sponsorship": job.get("sponsorship", "unknown"),
        "date_posted": job.get("date_posted", 0),
        "url": job.get("url", ""),
        "board": source,
    }


def fetch(source):
    """download one board, cache it, return normalized listings."""
    cache = f"cache_{source}.json"
    fresh = os.path.exists(cache) and time.time() - os.path.getmtime(cache) <
CACHE_HOURS * 3600
    if fresh:
        raw = json.load(open(cache))
    else:
        log("fetching", f"pulling {source} listings from github...", M)
        with urllib.request.urlopen(SOURCES[source], timeout=60) as r:
            raw = json.loads(r.read().decode())
        json.dump(raw, open(cache, "w"))
```

```
    return [normalize(j, source) for j in raw if j.get("active") and
j.get("is_visible")]


def load_listings(source=DEFAULT_SOURCE):
    names = list(SOURCES) if source == "all" else [source]
    jobs, seen = [], set()
    for name in names:
        for job in fetch(name):
            if job["url"] and job["url"] not in seen:
                seen.add(job["url"])
                jobs.append(job)
    log("loaded", f"{len(jobs)} open roles from {' + '.join(names)}", M)
    return jobs


# the boards spell places differently. "nyc" and "new york, ny" are one city.
LOCATION_ALIASES = {
    "new york": ["nyc", "new york"],
    "nyc": ["nyc", "new york"],
    "san francisco": ["sf", "san francisco"],
    "sf": ["sf", "san francisco"],
    "bay area": ["sf", "san francisco", "san jose", "palo alto",
                 "santa clara", "mountain view", "sunnyvale"],
    "seattle": ["seattle", "bellevue", "redmond"],
    "la": ["los angeles", "santa monica"],
    "los angeles": ["los angeles", "santa monica"],
    "dc": ["washington, dc", "arlington", "mclean"],
    "remote": ["remote"],
}


# ---------------------------------------------------------------- tools
# a "tool" is just a python function. that is the whole secret.

LISTINGS = []


def search_internships(keywords="", location="", season="", max_days_old=90,
limit=8):
    """filter the live listings. returns a list of dicts."""
    kw = [k.strip().lower() for k in keywords.split(",") if k.strip()]
    loc = location.strip().lower()
    loc_terms = LOCATION_ALIASES.get(loc, [loc]) if loc else []
    sea = season.strip().lower()
    cutoff = time.time() - max_days_old * 86400
    hits = []

    for job in LISTINGS:
```

```
        if job["date_posted"] < cutoff:
            continue
        blob = f"{job['title']} {job['category']} {job['company']}".lower()
        if kw and not any(k in blob for k in kw):
            continue
        if loc_terms:
            places = " ".join(job["locations"]).lower()
            if not any(t in places for t in loc_terms):
                continue
        if sea and sea not in " ".join(job["terms"]).lower():
            continue
        hits.append(job)

    hits.sort(key=lambda j: j["date_posted"], reverse=True)
    return [
        {
            "company": j["company"],
            "title": j["title"],
            "location": ", ".join(j["locations"] or ["remote/unlisted"])[:60],
            "term": ", ".join(j["terms"] or ["n/a"]),
            "sponsorship": SPONSORSHIP_PLAIN.get(j["sponsorship"], "not stated"),
            "days_ago": int((time.time() - j["date_posted"]) / 86400),
            "url": j["url"],
        }
        for j in hits[:limit]
    ]


SPONSORSHIP_PLAIN = {
    "Offers Sponsorship": "offers sponsorship",
    "Does Not Offer Sponsorship": "does NOT offer sponsorship",
    "U.S. Citizenship is Required": "us citizenship required",
    "Other": "not stated — check the posting",
}


def save_shortlist(jobs_json, filename="my_shortlist.md"):
    """write the agent's picks to a markdown file you can actually open.

    the model retypes the jobs to pass them here, and it drops fields when it
    does. so we look each one back up by url and use the real record. the file
    is ground truth, not whatever the model remembered.
    """
    jobs = json.loads(jobs_json) if isinstance(jobs_json, str) else jobs_json
    index = {j["url"]: j for j in LISTINGS}
    lines = ["# my internship shortlist", ""]

    for pick in jobs:
        real = index.get(pick.get("url", ""))
```

```
        if real:
            company = real["company"]
            title = real["title"]
            location = ", ".join(real["locations"] or ["remote/unlisted"])
            days = f"{int((time.time() - real['date_posted']) / 86400)} days ago"
            spons = SPONSORSHIP_PLAIN.get(real["sponsorship"], "not stated")
            url = real["url"]
        else:
            company = pick.get("company", "?")
            title = pick.get("title", "?")
            location = pick.get("location", "?")
            days = "not verified"
            spons = "not verified"
            url = pick.get("url", "")

        lines += [
            f"### {company} — {title}",
            f"- location: {location}",
            f"- posted: {days}",
            f"- sponsorship: {spons}",
            f"- apply: {url}",
            "",
        ]

    open(filename, "w").write("\n".join(lines))
    return {"saved": filename, "count": len(jobs)}


TOOLS = {"search_internships": search_internships, "save_shortlist":
save_shortlist}

TOOL_SCHEMAS = [
    {
        "name": "search_internships",
        "description": "search live internship postings. call this before
answering.",
        "input_schema": {
            "type": "object",
            "properties": {
                "keywords": {"type": "string", "description": "comma separated,
e.g. 'machine learning,data'"},
                "location": {"type": "string", "description": "city, e.g. 'nyc' or
'seattle'"},
                "season": {"type": "string", "description": "e.g. 'summer' or
'winter'"},
                "max_days_old": {"type": "integer"},
                "limit": {"type": "integer"},
            },
        },
```

```
    },
    {
        "name": "save_shortlist",
        "description": "save the final picks to a markdown file for the user.",
        "input_schema": {
            "type": "object",
            "properties": {"jobs_json": {"type": "string", "description": "json
array of job objects"}},
            "required": ["jobs_json"],
        },
    },
]

SYSTEM = (
    "you are an internship hunting agent for a cs student. "
    "always search before answering. never invent a role or a link — "
    "only report what the tool returns. if the search comes back empty, "
    "loosen one filter and try again. "
    "you MUST call save_shortlist with your picks before you give a final answer.
"
    "only mention visa sponsorship if the field explicitly says it is offered — "
    "if it says not stated, say nothing about sponsorship at all. "
    "keep the final answer under 80 words, plain text, no markdown headers or "
    "bold. end by naming which one to apply to first and why."
)


# ---------------------------------------------------------------- the loop

API_URL = "https://api.anthropic.com/v1/messages"


def call_model(messages):
    """one plain https post. no sdk, no dependencies, nothing to break."""
    payload = json.dumps({
        "model": MODEL,
        "max_tokens": 2000,
        "system": SYSTEM,
        "tools": TOOL_SCHEMAS,
        "messages": messages,
    }).encode()

    req = urllib.request.Request(API_URL, data=payload, method="POST", headers={
        "content-type": "application/json",
        "anthropic-version": "2023-06-01",
        "x-api-key": os.environ["ANTHROPIC_API_KEY"],
    })
    try:
        with urllib.request.urlopen(req, timeout=90) as r:
```

```
            return json.loads(r.read().decode())
    except urllib.error.HTTPError as e:
        detail = e.read().decode()[:300]
        raise SystemExit(f"\n{P}api error {e.code}{X}\n{detail}\n")


def run_agent(goal, max_steps=6):
    messages = [{"role": "user", "content": goal}]
    log("goal", goal, P)

    for step in range(1, max_steps + 1):
        reply = call_model(messages)
        messages.append({"role": "assistant", "content": reply["content"]})

        if reply.get("stop_reason") != "tool_use":
            text = "".join(b["text"] for b in reply["content"] if b["type"] ==
"text")
            print(f"\n{P}{B}answer{X}\n{text}\n")
            return text

        results = []
        for block in reply["content"]:
            if block["type"] != "tool_use":
                continue
            log(f"step {step}", f"calling
{block['name']}({json.dumps(block['input'])[:70]})", P)
            out = TOOLS[block["name"]](**block["input"])
            if isinstance(out, list):
                log("result", f"{len(out)} matches", D)
                for j in out[:8]:
                    print(f"  {M}{j['company'][:22]:<24}{X}"
                          f"{j['title'][:40]:<42}{D}{j['days_ago']}d ago{X}")
            else:
                log("result", out, D)
            results.append({"type": "tool_result", "tool_use_id": block["id"],
                            "content": json.dumps(out)})
        messages.append({"role": "user", "content": results})

    return "hit step limit"


# ---------------------------------------------------------------- demo mode

STOPWORDS = {
    "find", "me", "my", "a", "an", "the", "for", "in", "at", "on", "to", "get",
    "show", "some", "any", "please", "i", "want", "need", "looking", "look",
    "internship", "internships", "intern", "role", "roles", "job", "jobs",
    "position", "positions", "posted", "this", "last", "past", "week", "weeks",
    "month", "months", "day", "days", "new", "recent", "open", "near",
```

```
}


def parse_goal(goal):
    """demo-mode stand-in for the model: pull keywords + location out of a
sentence."""
    text = goal.lower()
    location = ""
    for alias in sorted(LOCATION_ALIASES, key=len, reverse=True):
        if alias in text:
            location = alias
            text = text.replace(alias, " ")
            break
    words = [w.strip(",.?!") for w in text.split()]
    keywords = [w for w in words if w and w not in STOPWORDS and len(w) > 1]
    return ",".join(keywords), location


def run_demo(goal):
    """same tool layer, no api key. proves the tools work on their own."""
    log("goal", goal, P)
    keywords, location = parse_goal(goal)
    log("parsed", f"keywords={keywords or 'any'} location={location or 'any'}", D)
    log("step 1", f"calling search_internships(keywords='{keywords}',
location='{location}')", P)
    jobs = search_internships(keywords=keywords, location=location, limit=5,
max_days_old=30)
    if not jobs:
        log("retry", "no hits in 30 days — widening to 90", D)
        jobs = search_internships(keywords=keywords, location=location, limit=5,
max_days_old=90)
    log("result", f"{len(jobs)} matches", D)
    for j in jobs:
        print(f"
{M}{j['company'][:22]:<24}{X}{j['title'][:40]:<42}{D}{j['days_ago']}d ago{X}")
    log("step 2", "calling save_shortlist(...)", P)
    log("result", save_shortlist(jobs), D)
    return jobs


# ---------------------------------------------------------------- main

if __name__ == "__main__":
    argv = sys.argv[1:]
    source = DEFAULT_SOURCE
    if "--source" in argv:
        i = argv.index("--source")
        source = argv[i + 1]
        del argv[i:i + 2]
```

```
    demo = "--demo" in argv
    goal = " ".join(a for a in argv if a != "--demo") or "machine learning
internships"

    LISTINGS = load_listings(source)
    has_key = bool(os.environ.get("ANTHROPIC_API_KEY"))

    if demo:
        log("mode", "demo — tool layer only, no model", D)
        run_demo(goal)
    elif not has_key:
        log("mode", "no ANTHROPIC_API_KEY found, running demo instead", D)
        log("", 'set it with:  export ANTHROPIC_API_KEY="sk-ant-..."', D)
        run_demo(goal)
    else:
        log("mode", f"live agent — {MODEL}", D)
        run_agent(goal)
```

Part 5 — make it yours

Do not just run mine. Change one thing and it becomes your project, and a real resume bullet.

- Add a filter for roles that do not require a resume
- Add a tool that scores each job against your actual skills
- Have it write a tailored cover letter line for the top pick
- Swap the job board for a different dataset entirely — grants, scholarships, research positions
- Make it email you the shortlist every monday morning

Resume bullet you can use once you have shipped it

> **Bullet** Built an llm agent with tool-calling that queries a live 371-role dataset, filters by role and location, and returns a ranked shortlist — reduced manual job board searching to a single command.

Swap in your own numbers. Only use numbers you can actually point to.

The data source

The listings come from open source github repos that people maintain by hand. Star them, they do a lot for us:

github.com/vanshb03/Summer2027-Internships

github.com/SimplifyJobs/Summer2027-Internships

Built it? Tag me @hinumpy. I want to see it.