

- welcome to
claude code

boris cherny

tl;dr

1. programming is changing
2. choose your path with claude code
3. think six months out

```
npm install -g @anthropic-ai/clause-code
```

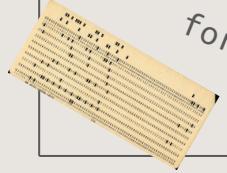
claude.ai/code

1/ programming is at an inflection point

programming language productivity is
increasing exponentially, driven by ai



log(productivity)



1950 1960 1970 1980 1990 2000 2010 2020 2030



ide productivity is following a similar exponential, also driven by ai



claude code
devin cursor
copilot

log(productivity)



1950 1960 1970 1980 1990 2000 2010 2020 2030

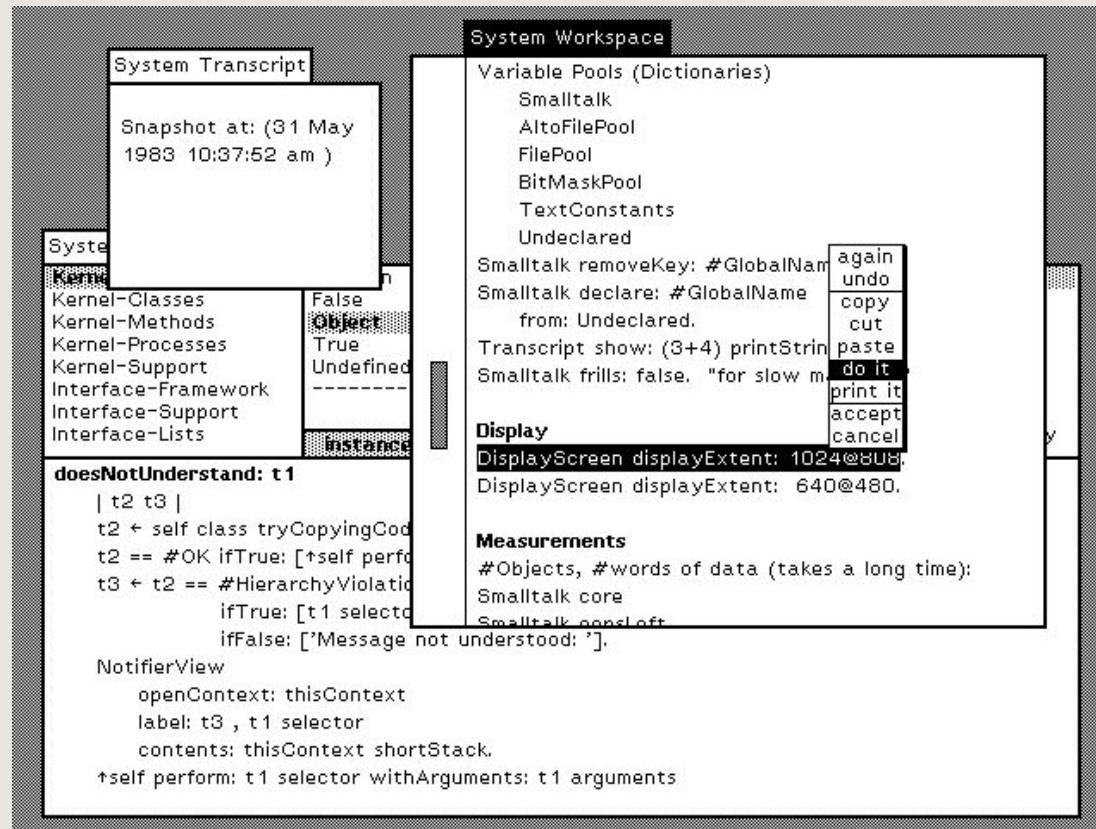
ed vi emacs pascal turbo basic smalltalk-80 vb eclipse idea sublime neovim



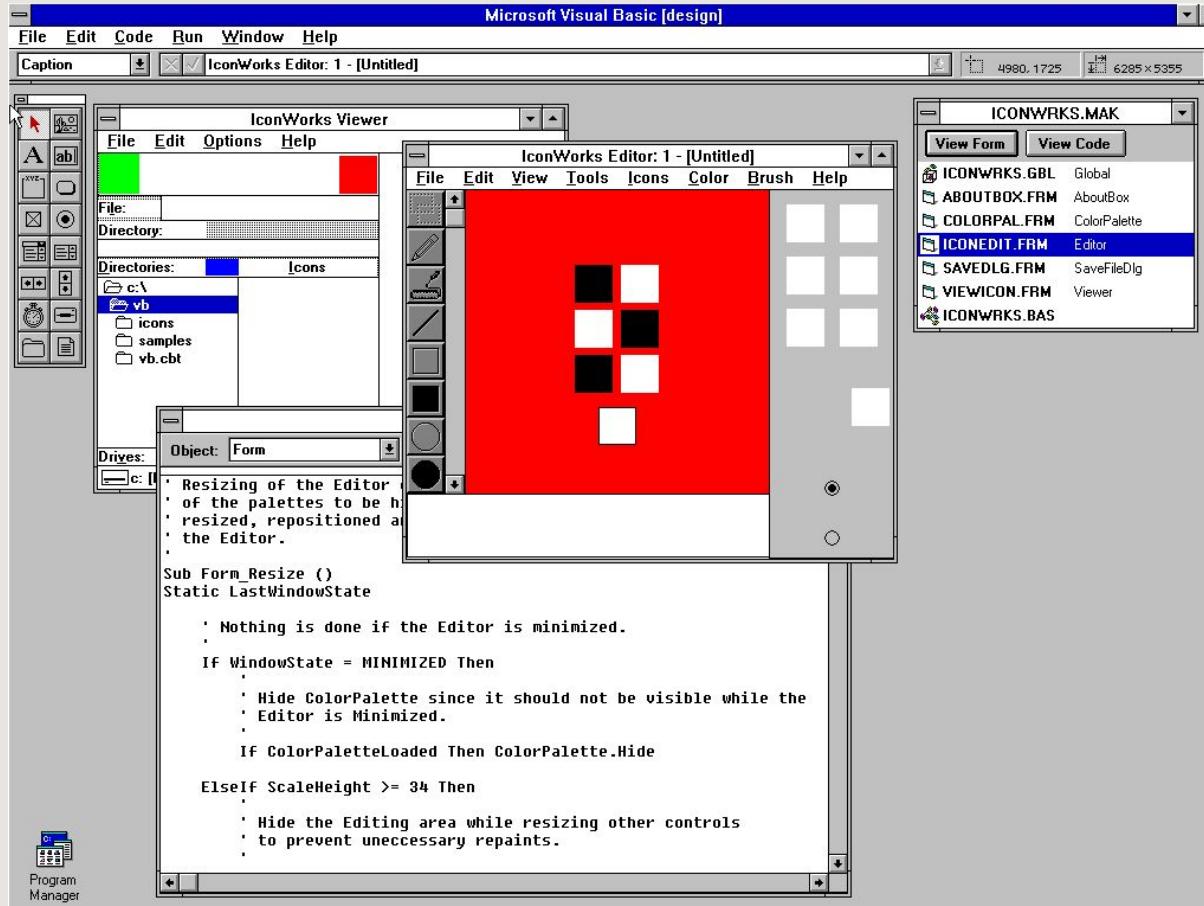
ibm 029 (1964)

```
$ ed fstab
Newline appended
116
z1
/dev/hda2  /      ext2  defaults  1 1\n$#
/dev/hdb1  /home  ext2  defaults  1 2\n$#
/dev/hda1  swap   swap  pri=40    0 0$#
3s/40/42/
w fstab
116
-
```

ed (1969)



smalltalk-80 (1980)



visual basic (1991)

The screenshot shows the Eclipse IDE interface. On the left is the Package Explorer view, displaying a project named 'TestProject' with a source folder 'src' containing a file named 'TestClass.java'. The main workspace shows the code for 'TestClass.java':

```
public class TestClass {  
    public void testMethod() {  
        this.  
    }  
}
```

A code completion dropdown menu is open at the cursor position, showing a list of methods from the Object class:

- clone() : Object - Object
- equals(Object obj) : boolean - Object
- finalize() : void - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- testMethod() : void - TestClass
- toString() : String - Object
- wait() : void - Object

At the bottom of the dropdown, a message reads "Press 'Ctrl+Space' to show Template Proposals".

eclipse (2001)

```
1  from __future__ import print_function
2  import argparse
3  import torch
4  import torch.nn as nn
5  import torch.optim as optim
6  import numpy as np
7  import matplotlib
8  matplotlib.use('Agg')
9  import matplotlib.pyplot as plt
10
11 class Sequence(nn.Module):
12     def __init__(self):
13         super(Sequence, self).__init__()
14         self.lstm1 = nn.LSTMCell(1, 51)
15         self.lstm2 = nn.LSTMCell(51, 51)
16         self.linear = nn.Linear(51, 1)
17
18     def forward(self, input, future = 0):
19         outputs = []
20         h_t = torch.zeros(input.size(0), 51, dtype=torch.double)
```

 **You** May 29, 2024, 2:02 AM

Create a website with a list of the top 10 programming competitions in the world and make sure that each one on the list links to the competition's main website

 **Devin** May 29, 2024, 2:02 AM

Got it! I'll create a website listing the top 10 programming competitions in the world, each linking to their main website. I'll get started on this right away.

 **Devin** May 29, 2024, 2:03 AM

I'm going to set up a React app to create the website listing the top 10 programming competitions, each linking to their main website. I'll use Chakra UI for a modern and user-friendly design.

Quick question: Are there any specific programming competitions you want to be included in the list, or should I choose them independently? Also, do you have any preferences for the design and layout of the website (e.g., color scheme, fonts, etc.)?

If I don't hear back, I'll go ahead and choose the competitions and use a clean, modern design by default.

- Set up a new React app using Create React App.
- Installed Chakra UI and configured it for the project.
- Created a component to list the top 10 programming competitions with links to their main websites.
- Styled the component to ensure a modern and user-friendly design.
- Ran the local development server and verified the website functionality.
- Built the React app for production.

 *Session Ended.*

If you'd like to evaluate this run, [click here](#).

 Start a new conversation with Devin

Devin's Workspace

Following

Shell Browser Editor Planner

App.js

```
home > ubuntu > top10comps > src > App.js

1 import React from 'react';
2 import { ChakraProvider, Box, Link, Text, VStack, Heading } from '@chakra-ui/react'
3
4 function App() {
5   const competitions = [
6     { name: 'ACM ICPC', url: 'https://icpc.global/' },
7     { name: 'Google Code Jam', url: 'https://codingcompetitions.withgoogle.com/cjam' },
8     { name: 'Facebook Hacker Cup', url: 'https://www.facebook.com/codingcompetitions/hacker-cup' },
9     { name: 'Codeforces', url: 'https://codeforces.com/' },
10    { name: 'TopCoder', url: 'https://www.topcoder.com/' },
11    { name: 'AtCoder', url: 'https://atcoder.jp/' },
12    { name: 'LeetCode', url: 'https://leetcode.com/contest/' },
13    { name: 'HackerRank', url: 'https://www.hackerrank.com/domains/tutorials/10-days-of-statistics' },
14    { name: 'Kick Start', url: 'https://codingcompetitions.withgoogle.com/kickstart' },
15    { name: 'CodeChef', url: 'https://www.codechef.com/' },
16  ];
17
18  return (
19    <ChakraProvider>
20      <VStack spacing={8} align="stretch" p={5}>
21        <Heading as="h1" size="2xl" mb={5} textAlign="center">
22          Top 10 Programming Competitions
23        </Heading>
24        {competitions.map((comp, index) => (
25          <Box key={index} px={5} shadow="md" border="1px solid #ccc" border-radius="md" mb={5}>
26            <Link href={comp.url} isExternal>
27              {comp.name}
28            </Text>
29            </Link>
30          </Box>
31        ))}
32      </VStack>
33    </ChakraProvider>
34  )
}

Create machine snapshot 
```

ide devx has evolved quickly, and will
continue to change even more quickly

verification is evolving quickly, too

- manual debugging
- static types (algol)
- formal verification
- abstract interpretation
- automated testing
- continuous integration
- property-based testing (quickcheck)
- dependent typing
- e2e testing
- chaos testing (chaos monkey)
- ai-powered vulnerability testing
- ai-powered unit testing (testgen)
- ai-powered fuzz testing (sapienz)
- self play
- ...

claude code's approach:
works everywhere

1. terminal-native
2. low-level model access
3. infinitely hackable

works across the whole sdlc

1. Discover

Explore codebase
and history

2. Design

Plan project

Search
documentation

Onboard & learn

3. Build

Implement code

Write and execute
tests

Create commits
and PRs

4. Deploy

Automate CI/CD

Configure
environments

Manage
deployments

5. Support & Scale

Debug errors

Large-scale
refactor

Monitor usage &
performance

Using and mastering all of your team's CLI tools (e.g., git, docker, bq) so you can focus on solutions, not syntax

CLAUDE CODE

* Invalid API Keys

Claude Code v2.0.32
Sonnet 4.5 · API Usage Billing
/Users/boris/code/clause-cli

```
> brainstorm then implement a better solution for when api keys are invalid
  ↳ Async hook SessionStart completed
```

- I'll help you brainstorm and implement a better solution for handling invalid API keys. Let me first explore the current implementation to understand how API key validation is handled.

Explore(Explore API key validation)
↳ **Search**(pattern: "401|unauthorized")
↳ **Search**(pattern: "validateApiKey|validate.*key")
+1 more tool use (ctrl+o to expand)

* Moseying... (esc to interrupt)

> []

? for shortcuts

Thinking off (tab to toggle)

terminal

ide

The screenshot shows a code editor window with a purple header bar. The title bar says "claude-code". The left sidebar contains icons for file operations, search, and navigation. The main area displays a file named "README.md". The content of the file is as follows:

```
plugins > learning-output-style > README.md > # Learning Style
1  # Learning Style Plugin
46 ## Example interaction
session timeout behavior is a
security vs. UX trade-off -
should sessions auto-extend on
activity, or have a hard
timeout?
49
50 In `auth/middleware.ts` ,
implement the
`handleSessionTimeout()`
function to define the timeout
behavior.
51
52 Consider: auto-extending
improves UX but may leave
sessions open longer; hard
timeouts are more secure but
might frustrate active users.
53
54 **You:** [Write 5-10 lines
implementing your preferred
approach]
55
56 ## Educational insights
57
58 In addition to interactive
learning, Claude will provide
```

To the right of the code editor, there is a sidebar titled "brainstorm then implement a better solution...". It contains the following text:

handling invalid API keys. Let me first explore the codebase to understand the current implementation.

Task: Find API key validation code

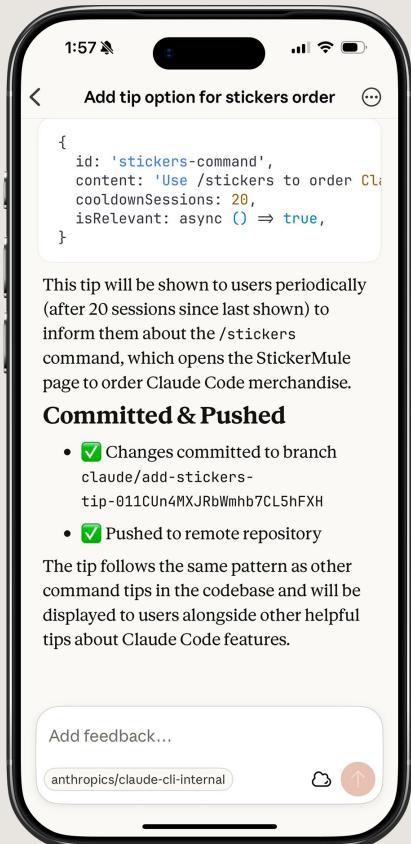
```
IN {
  "subagent_type": "Explore",
  "description": "Find API key validation code",
  "prompt": "Find all code related to API key validation"
}
```

Results of the search:

- Glob pattern: "**/*.ts,tsx,js,jsx,py,go,java" | Found 4 files
- Grep "api\key|apiKey" (type: ts,tsx,js,jsx,py) | 1 line of output
- Grep "invalid.*key|401|403|authentication" (type: ts,tsx,js,jsx,py) | 1 line of output
- Grep "authentication|unauthorized|forbidden" (type: ts,tsx,js,jsx,py)
- Deliberating...

At the bottom of the sidebar, there is a text input field with placeholder text "queue another message..." and a toolbar with buttons for "Ask before edits" and "README.md".

web & ios



A screenshot of the Claude Code web interface. The top navigation bar shows the title "Claude Code | Claude" and the URL "claude.ai/code/session_011CUoA9mVphie4K4ZeFAsF".

The main interface has a search bar "Ask Claude to write code..." and a dropdown menu "anthropics/clade-cli-inter...". Below it is a "Sessions" section with a "All" dropdown. A "Fix a typo" card is visible. The sidebar lists several tips:

- Investigate default model selection for Opus anthropics/clade-desktop
- Investigate Claude model selection logic anthropics/apps
- Configure alternate Claude config directory path anthropics/clade-cli-internal
- Add tip option for stickers order anthropics/clade-cli-internal
- Debugging and identifying error causes anthropics/clade-cli-internal
- Handle API errors in Claude Code UX anthropics/clade-cli-internal
- Add early return for disabled Statsig detection

The right side of the screen displays a configuration tip titled "Configure alternate Claude config directory path". It includes code snippets, usage instructions, and environment variable details:

```
export function getClaudeConfigHomeDir(): string {  
  return process.env.CLAUDE_CONFIG_DIR ?? join(homedir(), '.claude')  
}  
  
In src/constants/macro.ts:55-58:  
  
/**  
 * The directory where Claude Code stores its configuration and data.  
 * If not set, defaults to ~/.claude.  
 */  
CLAUDE_CONFIG_DIR?: string  
  
Usage  
To use an alternate configuration directory, simply set the CLAUDE_CONFIG_DIR environment variable:  
  
# Use a temporary vanilla Claude session  
CLAUDE_CONFIG_DIR=/tmp/claude-vanilla claude  
  
# Or for testing plugin updates  
CLAUDE_CONFIG_DIR=~/claude-test claude  
  
# In your tooling  
export CLAUDE_CONFIG_DIR=/path/to/vanilla/config  
claude
```

Text at the bottom states: "This will make Claude Code use the specified directory for:" followed by a bullet point: "• CLAUDE_CONFIG_DIR/.claude.json (or .claude-oauth.json)"

/install-github-app

A screenshot of a GitHub issue page showing a conversation between users `bcherny` and `claude`.

The URL in the browser bar is `github.com/bcherny/json-schema-to-typescript/issues/660`.

bcherny (Owner) commented 1 minute ago:

@claude implement this

claude (Owner) commented 1 minute ago · edited by claude:

Working on implementing string array item descriptions

I'll implement the fix for issue [#660](#) where string array item descriptions are missing from generated TypeScript.

Todo List:

- Read repository setup instructions (CLAUDE.md) - No CLAUDE.md found, checked README and package.json
- Understand the codebase structure and current description handling
- Find array processing logic and identify where item descriptions should be added
- Implement the fix to include array item descriptions
- Test with the provided example
- Run linting/testing commands (need Bash permissions)

sdk

```
$ claude -p \
"what did i do this week?" \
--allowedTools Bash(git log:*)
--output-format stream-json
```

your app

claude code sdk

anthropic, bedrock, or vertex api

claude models

sdk

```
$ get-gcp-logs 1uhd832d |  
claude -p "correlate errors + commits" \  
--output-format=json |  
jq '.result'
```

your app

claude code sdk

anthropic, bedrock, or vertex api

claude models

use cases

1. codebase q&a + research
2. write code
 - a. 1-shot
 - b. sidekick
 - c. prototype
3. integrate tools & mcps
4. power automation

1. ask claude code about your code

- > how do I make a new `@app/services/ValidationTemplateFactory`?
- > why does `recoverFromException` take so many arguments? look through git history to answer
- > why did we fix issue #18363 by adding the if/else in `@src/login.ts` api?
- > in which version did we release the new `@api/ext/PreHooks.php` api?
- > look at PR #9383, then carefully verify which app versions were impacted
- > what did I ship last week?

2. teach claude to use your tools

> use the barley cli to check for error logs

```
$ claude mcp add barley_server -- node myserver
```

> use the barley mcp server to check for error logs

3. fit the workflow to the task

explore > plan > confirm > code > commit

> figure out the root cause for issue #983, then propose a few fixes. Let me choose an approach before you code. ultrathink

3. fit the workflow to the task

tests > commit > code > iterate > commit

> write tests for `@utils/markdown.ts` to make sure links render properly

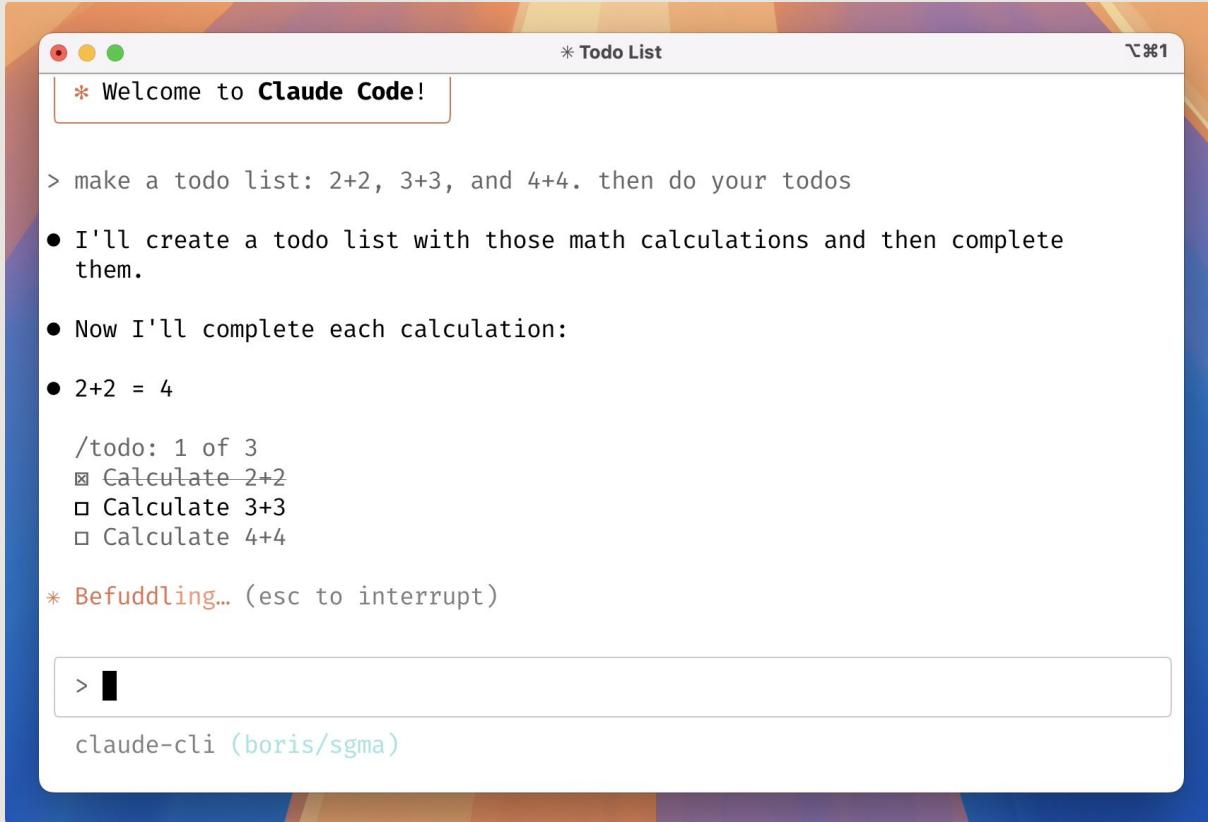
(note the tests won't pass yet, since links aren't yet implemented). then
commit. then update the code to make the tests pass.

3. fit the workflow to the task

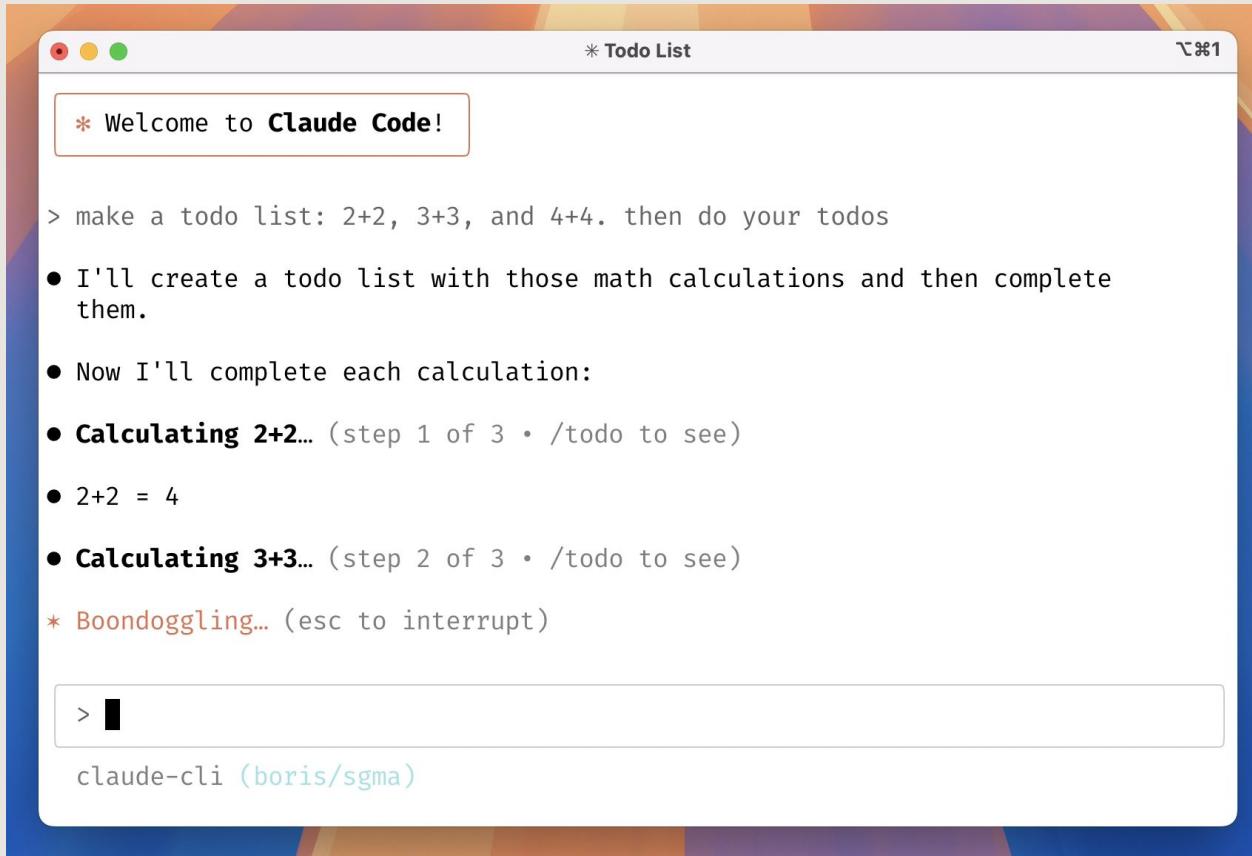
code → screenshot → iterate

> implement [mock.png]. Then screenshot it with puppeteer and iterate till it looks like the mock.

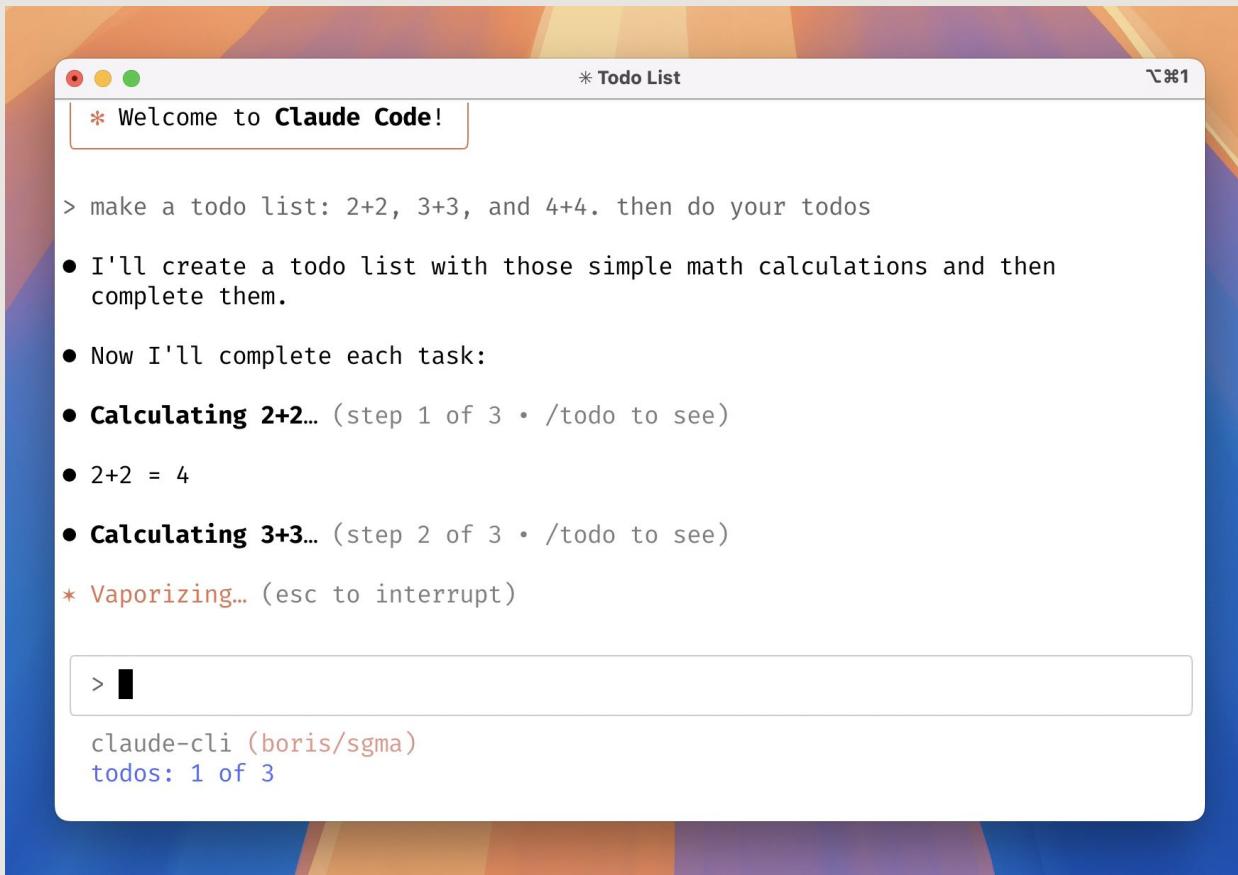
4. use claude code to prototype



> make it so instead of todos showing up as they come in, we hide the tool use and result for todos, and render a fixed todo list above the input. title it "/todo (1 of 3)" in grey



> actually don't show a todo list at all, and instead render the tool uses inline, as bold headings when the model starts working on a todo. keep the "step 2 of 4" or whatever, and add middot /todo to see after in grey



> also add a todo pill under the text input, similar to bg tasks. it should render "todos: 1 of 3" or whatever. make the pill interactive

* Welcome to **Claude Code!**

> make a todo list: 2+2, 3+3, and 4+4. then do your todos

- I'll create a todo list with those arithmetic tasks and then complete them.
- Now I'll complete each task:

* Zesting... (esc to interrupt)

```
> [ ]
```

claude-cli (boris/sgma)

/todo

- Calculate 2+2
- Calculate 3+3
- Calculate 4+4

> actually undo both the pill and headings. instead, make the todo list render to the right of the input, vertically centered with a grey divider. show it when todos are active, hide it when it's done



* Math Todo List

✉⌘1

> make a todo list: 2+2, 3+3, 4+4, 5+5, 6+6, 7+7, 8+8, and 9+9. then do your todos

- I'll create a todo list with those arithmetic operations and then complete them.
- Now I'll complete each calculation:

- Calculate 2+2
- Calculate 3+3
- Calculate 4+4
- Calculate 5+5
- Calculate 6+6
- Calculate 7+7
- Calculate 8+8
- Calculate 9+9

* Bippity-bopping... (esc to interrupt)

> █

claude-cli (boris/sgma)

> actually what if you show the todo list above the input instead. truncate at 5 and show "... and 4 more" or whatever. add a heading "Todo:" in grey text

* Math Todo List

⌘1

* Welcome to Claude Code!

> make a todo list: 2+2, 3+3, 4+4, 5+5, 6+6, 7+7, 8+8, and 9+9. then do your todos

- I'll create a todo list with these math calculations and then solve them.

▫ Calculate 2+2

▫ Calculate 3+3

▫ Calculate 4+4

▫ Calculate 5+5

... and 4 more (ctrl+r to show all)

+ Thundering... (esc to interrupt)

> █

claudie-cli (boris/sgma)

> actually what if you show the todo list above the input instead. truncate at 5 and show "... and 4 more" or whatever. add a heading "Todo:" in grey text

* Sleep Sequence ⌘1

- I'll create a todo list for the sleep commands and then execute them in order.
- Now I'll execute each sleep command in order:

- **Bash(sleep 2)**
 - └ (No content)

Todo:

- Run sleep 2
- Run sleep 3
- Run sleep 4
- Run sleep 3 again

... +3 more

* Forging... (esc to interrupt)

```
> █
```

claude-cli (boris/sgma)

> actually what if you show the todo list above the input instead. truncate at 5 and show "... and 4 more" or whatever. add a heading "Todo:" in grey text

* Sleep Sequence

* Welcome to Claude Code!

> make a todo list to run sleep2, then sleep 3, then sleep 4, then sleep 3 again, then sleep 2, then sleep 1, and finally sleep 2. then do your todos

- I'll create a todo list for running these sleep commands in sequence, then execute them.
- Now I'll execute each sleep command in order:

- Bash(sleep 2)
 - ↳ (No content)

* Running sleep 2... (esc to interrupt ⏎ /todos)

- ↳ Next: Run sleep 3

> █

claude-cli (boris/kync)

> instead of showing todos above the input, merge them into the spinner. show the current todo as the spinner message in active verb form, and the next todo instead of the spinner tip, in passive form. also add a /todos slash command to see all todos, and mention it next to "esc to interrupt"

* Sleep Sequence

- I'll create a todo list for these sleep commands and then execute them.
- Now I'll execute each sleep command in order:

Bash(sleep 2)
└ Running...
ctrl+b to run in background

* Running sleep 2...
└ □ Run sleep 2
 □ Run sleep 3
 □ Run sleep 4
 □ Run sleep 3 again
 □ Run sleep 2 again
 □ Run sleep 1
 □ Run sleep 2 final

```
> █
```

claude-cli (boris/kgyp)

> make it so i can hit `ctrl+t` to expand todos inline. also rm the "esc to interrupt" message when we show the todos, and replace it w "`ctrl+t` to expand"



* Welcome to **Claude Code!**

> Make a todo list 2+2, 3+3, 4+4. dont do it yet

claude-cli (boris/tgez)

1. build for the model six months from now

2. be ready to evolve

3. ask not what the model can do for you

4.

The screenshot shows a web browser window with the title "The Bitter Lesson" in the tab bar. The address bar displays "incompleteideas.net/Incldeas/BitterLesson.html" with a "Not Secure" warning. The page content starts with the title "The Bitter Lesson" in a large, bold, black font, followed by the author's name "Rich Sutton" and the date "March 13, 2019".

The Bitter Lesson

Rich Sutton

March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are psychological commitments to investment in one approach or the other. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation. There were many examples of AI researchers' belated learning of this bitter lesson, and it is instructive to review some of the most prominent.

In computer chess, the methods that defeated the world champion, Kasparov, in 1997, were based on massive, deep search. At the time, this was looked upon with dismay by the majority of computer-chess researchers who had pursued methods that leveraged human understanding of the special structure of chess. When a simpler, search-based approach with special hardware and software proved vastly more effective, these human-knowledge-based chess researchers were not good losers. They said that "brute force" search may have won this time, but it was not a general strategy, and anyway it was not how people played chess. These researchers wanted methods based on human input to win and were disappointed when they did not.

A similar pattern of research progress was seen in computer Go, only delayed by a further 20 years. Enormous initial efforts went into avoiding search by taking advantage of human knowledge, or of the special features of the game, but all those efforts proved irrelevant, or worse, once search was applied effectively at scale. Also important was the use of learning by self play to learn a value function (as it was in many other games and even in chess, although learning did not play a big role in the 1997 program that first beat a world champion). Learning by self play, and learning in general, is like search in that it enables massive computation to be brought to bear. Search and learning are the two most important classes of techniques for utilizing massive amounts of computation in AI research. In computer Go, as in computer chess, researchers' initial effort was directed toward utilizing human understanding (so that less search was needed) and only much later was much greater success had by embracing search and learning.

In speech recognition, there was an early competition, sponsored by DARPA, in the 1970s. Entrants included a host of special methods that took advantage of human knowledge -knowledge of words, of phonemes, of the human vocal tract, etc. On the other side were newer methods that were more statistical in nature and did much more computation, based on hidden Markov models (HMMs). Again, the statistical methods won out over the human-knowledge-based methods. This led to a major change in all of natural language processing, gradually over decades, where statistics and computation came to dominate the field. The recent rise of deep learning in speech recognition is the most recent step in this consistent direction. Deep learning methods rely even less on human knowledge, and use even more computation, together with learning on huge training sets to produce dramatically better speech recognition systems. As in the games, researchers always tried to make systems that worked the way the researchers thought their own

thanks!

```
npm install -g @anthropic-ai/claude-code
```

```
claude.ai/code
```

boris@anthropic.com