WORKSHOP OVERVIEW

# Claude Code Hands-On Workshop

Build games. Build skills. Ship something real. Windows & macOS.

SECTION 1

## Claude Code Basics

Install · Start · Your first prompts · 15 min — then continue to Section 2 for prompting patterns

1.1

### Install Claude Code

⏱ 3 min

Follow the step-by-step installation guide for your operating system:

**Claude Code Setup Guide**
Download the desktop app — no terminal, no Node.js needed. Windows & macOS.

1.2

### Start a Session

⏱ 3 min

Open Claude Code and point it at your workshop folder:

**WINDOWS**

```
1. Open the Start menu → type Claude →
press Enter
2. Click "Open Folder" in the app
3. Navigate to Documents and create a
```

**MACOS**

```
1. Open Launchpad or Applications →
click Claude
2. Click "Open Folder" in the app
3. Navigate to Documents and create a
```

```
new folder
   called "my-workshop", then click
Select Folder
```

```
new folder
   called "my-workshop", then click
Open
```

You'll see the chat interface with a text field at the bottom. Claude is now connected to your folder — type in plain English, no commands needed.

**What Claude Code can do**

- Read and write files in your open project folder
- Run commands on your machine (with your permission)
- Remember context across the whole session
- Use skills you've installed to produce better output

1.3

## Your First Prompts

⏱ 9 min

Try these prompts one at a time. Watch what Claude Code does — it shows you which files it reads or creates.

### Prompt 1 — Ask a question

**TYPE THIS:**

*"What files are in my current directory?"*

Claude runs the appropriate list command and describes what it finds. Notice it explains what it's doing before doing it.

### Prompt 2 — Create a file

**TYPE THIS:**

*"Create a file called hello.html that shows 'Hello Workshop!' in big text, centred on the page, with a dark background."*

Claude creates the file. Open it in your browser:

**WINDOWS**

```
Open File Explorer → Documents → my-workshop
Double-click hello.html
```

**MACOS**

```
Open Finder → Documents → my-workshop
Double-click hello.html
```

### Prompt 3 — Iterate on it

**TYPE THIS:**

*"Make the text animate — fade in slowly when the page loads."*

Claude edits the file in place. Refresh your browser — you don't need to open it again.

### Prompt 4 — Ask Claude to explain

**TYPE THIS:**

*"Explain what you just added, in plain English."*

Claude explains the CSS animation it used. You can ask follow-up questions at any time.

**Key things to know about prompting Claude Code**

1. Be specific — include format, style, colours in your request
2. Iterate — each prompt builds on the last in the same session
3. Ask why — Claude explains any decision if you ask
4. Say 'undo that' or 'go back' if you don't like a change
5. Use "+ New Chat" in the app to start fresh without losing your files

SECTION 2

# Prompting Patterns

Generate · Refine · Debug · Explain · 15 min

Before jumping into building the game, this section teaches you the four core prompting patterns you'll use constantly with Claude Code. Each pattern has a specific shape — once you know them, you can apply them to any project.

Keep hello.html open in your browser and the Claude Code app open with your workshop folder loaded. You'll use it as a live sandbox.

**The four patterns**

1. **Generate** — describe what you want, Claude builds it from scratch
2. **Refine** — ask Claude to change or improve something that already exists
3. **Debug** — show Claude a problem, ask it to find and fix the cause
4. **Explain** — ask Claude to walk you through code or a decision in plain English

2.1

## Pattern 1 — Generate

⏱ 3 min

The generate pattern turns a plain-language description into working code. The more specific you are, the closer Claude gets on the first attempt.

### Vague prompt (try this first)

**TYPE THIS:**

*"Add a button to hello.html."*

Claude will add something — but it may not be what you had in mind. Now try a specific version:

### Specific prompt (try this second)

**NOW TYPE THIS:**

*"Add a large centred button below the heading that says 'Click me'. When clicked, the background colour cycles through 5 neon colours one at a time. The button should have rounded corners, a white border, and a subtle pulse animation."*

Notice the difference. Specific prompts include: what the element looks like, where it lives, what it does, and any visual details (colours, shape, animation).

**Generate prompt formula**

Create [what] + that [does/shows] + with [visual style] + in [location]

Example: "Create a countdown timer that shows minutes and seconds, with large red digits, centred below the button."

2.2

## Pattern 2 — Refine

⏱ 3 min

Refining means asking Claude to change something that already exists. You don't need to describe the whole file again — Claude remembers the full context of the session.

### Layer changes one at a time

**TYPE THIS:**

*"Make the button bigger — padding 20px top and bottom, 40px left and right. Change the font to bold 20px."*

**THEN THIS:**

*"The pulse animation is too fast. Slow it down to 2 seconds per cycle and make it more subtle — just a slight brightness change, not a size change."*

**THEN THIS:**

*"Change the 5 cycling colours to a synthwave palette — deep purple, hot pink, electric blue, neon green, bright orange."*

Each prompt refines just one thing. This is intentional — small changes are easy to undo if you don't like the result.

> **Tip:** If a refinement makes things worse, just say: 'Undo that last change' or 'Revert the button back to how it was before.'

2.3

## Pattern 3 — Debug

⏱ 4 min

Debugging means telling Claude what is wrong and asking it to find the cause. You don't need to know what the bug is — describing the symptom is enough.

### Introduce a deliberate bug to practice with

**TYPE THIS TO BREAK SOMETHING ON PURPOSE:**

*"In the button click handler, change the colour cycling logic so it skips every other colour."*

Refresh the page and click the button a few times — you'll see colours being skipped. Now debug it:

### Debug prompt

**TYPE THIS:**

*"The colour cycling is broken — it skips every other colour instead of stepping through them in order. Find the bug and fix it."*

Claude will read the code, identify the off-by-one or logic error, explain what it found, and fix it.

**Debug prompt formula**

Describe the symptom + what you expected + ask Claude to find and fix it

"The X is doing Y. I expected Z. Find the bug and fix it."

You can also paste an error message directly: 'I got this error in the browser console: [paste error]. Fix it.'

### Reading browser console errors

Open DevTools in your browser to see real errors — this is the fastest way to give Claude useful debugging information:

**WINDOWS**

```
# Open DevTools in Chrome/Edge:
Press F12
# OR: Ctrl+Shift+I
# Click the Console tab
# Paste any red errors into Claude
Code
```

**MACOS**

```
# Open DevTools in Chrome/Safari:
Press Cmd+Option+I
# OR: right-click the page > Inspect
# Click the Console tab
# Paste any red errors into Claude
Code
```

2.4

## Pattern 4 — Explain

⏱ 5 min

The explain pattern turns Claude into a teacher. You can ask it to explain any code it wrote, any decision it made, or any concept you don't understand — in as much or as little detail as you want.

### Explain code Claude just wrote

**TYPE THIS:**

*"Explain the colour cycling code you just wrote. What does each line do? Use plain English, no jargon."*

Claude gives a line-by-line walkthrough. If it's still too technical, push further:

**THEN THIS:**

*"Explain what an array is in one sentence using a real-world analogy."*

**THEN THIS:**

*"Explain what the modulo operator (%) does in the cycling logic. Give me a simple example with numbers."*

### Explain a whole file

**TYPE THIS:**

*"Explain what hello.html does from top to bottom, as if I've never seen HTML or JavaScript before."*

Claude will walk through the structure — HTML, CSS, and JS — explaining the purpose of each section.

### Ask Claude to explain a website you didn't build

You can paste any code into Claude Code and ask it to explain:

**TYPE THIS:**

*"Here is some JavaScript code I found online. Explain what it does and how I could adapt it to change the background colour on a timer instead of a click: [paste any JS snippet here]"*

**Explain prompts — useful variations**

- "Explain this to me like I am 10 years old"
- "What does [specific function/line] do and why is it written that way?"
- "What would break if I removed [specific part]?"
- "Why did you use [X approach] instead of [Y approach]?"
- "Give me an analogy for how [concept] works"

2.5

## Putting it all together

⏱ bonus

Real sessions mix all four patterns naturally. Here is an example sequence — notice how each prompt uses a different pattern:

| STEP | PATTERN | PROMPT |
|---|---|---|
| 1 | Generate | Build a landing page for a coffee shop — dark background, cream text, a hero image placeholder, and a menu section with 4 items. |
| 2 | Explain | Explain the CSS flexbox layout you used for the menu section. |
| 3 | Refine | Make the hero section taller and add a subtle parallax scroll effect. |

| STEP | PATTERN | PROMPT |
|---|---|---|
| 4 | Debug | The menu items are stacking vertically on mobile instead of in a 2x2 grid. Fix it. |
| 5 | Refine | Add a sticky header that fades in after the user scrolls down 100px. |
| 6 | Explain | What event listener did you use to detect the scroll? Why that approach? |

SECTION 3

# Build a Retro Shooter Game

5 guided steps — each prompt builds on the last · 25 min

You're going to build a fully playable browser game from scratch using a sequence of prompts. Each step adds one feature. By the end you'll have a Space Invaders-style game you can open in any browser — no setup, no npm, one HTML file.

**Before you start**

Make sure you're in your working folder and Claude Code is running.

All 5 steps happen in the same Claude Code session — don't type /clear between them.

3.1

## Create the base game

⏱ 5 min

**PROMPT 1 — COPY AND PASTE THIS EXACTLY:**

*"Build a retro browser shooter game as a single HTML file called game.html. Space Invaders style. Dark background, green player ship at the bottom, red enemies in rows at the top. Player moves left and right with arrow keys, shoots with spacebar. Enemies move side to side and slowly descend. Score in the top left. Lives as mini ships in the bottom left."*

Wait for Claude to finish, then open the file:

**WINDOWS**

```
Open File Explorer → Documents → my-workshop
Double-click game.html
```

**MACOS**

```
Open Finder → Documents → my-workshop
Double-click game.html
```

**Try it:** Move with arrow keys. Shoot with Space. Make sure the basic game works before moving on.

## 3.2 Add explosions and sound

⏱ 5 min

**PROMPT 2:**

*"Add particle explosions when enemies are destroyed — bursts of orange and yellow squares flying outward. Also add retro sound effects: a short laser sound when I shoot, and a noise burst when an enemy explodes. Use the Web Audio API, no external files."*

Refresh game.html in the browser. Shoot an enemy — you should see particles and hear the laser.

> **Tip:** If sound does not play on first shot, click anywhere on the game page first — browsers require a user interaction before playing audio.

## 3.3 Add waves and difficulty

⏱ 5 min

**PROMPT 3:**

*"When all enemies are destroyed, spawn a new wave. Each new wave should have one more row of enemies, and enemies should move faster and shoot more frequently. Show the current wave number at the top centre. Add a brief 'WAVE 2', 'WAVE 3' message in the middle of the screen between waves."*

Clear the first wave and watch the second wave appear faster. Try to reach wave 3.

## 3.4 Add power-ups

⏱ 5 min

**PROMPT 4:**

*"When an enemy is destroyed, there's a 15% chance it drops a power-up that slowly falls down the screen. Three types: a yellow star for rapid fire (shoots 3x faster for 8 seconds), a cyan shield that makes the player flash and survive one hit, and a magenta spread shot that fires 3 bullets in a fan for 8 seconds. Show an icon and colour for each type. Play a rising chime sound when collected."*

Play for a few minutes and collect power-ups. Rapid fire makes clearing waves much easier.

3.5

## Polish — title screen and hi-score

⏱ 5 min

**PROMPT 5:**

*"Add a title screen that shows 'RETRO SHOOTER' in large yellow pixel-style text, the hi-score, and 'PRESS SPACE TO START'. Save the hi-score to localStorage so it persists between sessions. Add a game over screen showing the final score and hi-score, with 'PRESS SPACE TO PLAY AGAIN'. Add a starfield background of slowly scrolling white dots."*

Close and reopen game.html — your hi-score should still be there.

**Your game is done ✓**

You now have a fully playable retro shooter with:

- 5 waves of escalating difficulty
- Particle explosions and chiptune audio
- 3 power-up types
- Persistent hi-score
- Title and game-over screens

The entire game is in one file — game.html. You can email it or deploy it to Netlify in 30 seconds (see wrap-up).

### Bonus prompts — if you finish early

- **Boss wave:** Add a large boss enemy every 5 waves that takes 10 hits, moves in a sine wave, and fires 3 bullets at once.
- **Combo multiplier:** Add a combo counter that multiplies score when killing enemies in quick succession.
- **Enemy variety:** Add a second enemy type — purple and diamond-shaped — that dive-bombs the player at random.
- **Mobile controls:** Add on-screen touch buttons so the game works on a phone.

SECTION 4

# Personal Demo Project

Design it · Build it · Present it · ~90 min — your own app, your own prompts, your own story

Everything so far has been guided. Now it's your turn. You pick what to build, you write every prompt, and you deal with whatever goes wrong. That's where the real learning happens.

## 4.1 Design Your Goal App

⏱ 15 min

Before you open Claude Code, spend a few minutes getting clear on what you want to build. It doesn't have to be big — it has to be *yours*.

**Answer these three questions first**

1. **What is it?** One sentence. "A tool that..." / "A game where..." / "A page that..."
2. **What does done look like?** What will you see on screen when it works?
3. **What's your first prompt?** Write it out before you type it into Claude.

**Need ideas?** A personal budget tracker · a quiz about something you love · a to-do list with a twist · a random meal picker · a countdown timer for something you're waiting for · a mini portfolio page · a word game · a habit tracker. Anything goes — the simpler the idea, the further you'll get.

## 4.2 Build It with Claude Code

⏱ ~45 min

Open Claude Code, start a new session, and build. Use everything from Sections 1–3: generate, refine, debug, explain. Your instructor is here to help when you're stuck — but the prompts are yours.

**Keep notes as you go**

You'll present at the end, so track these as you work:

- The prompts that worked well — copy them into a notes file
- Anything that broke or surprised you
- What you changed or refined, and why
- Where you are vs. where you wanted to be

**Stuck?** Try: *"This isn't working — [describe what you see]. What's wrong and how do we fix it?"* · Or: *"I wanted [X] but got [Y]. Explain why and suggest two ways to fix it."*

4.3

## Demo Time — Present to the Group

⏱ ~30 min · 3–5 min per person

Everyone shares their screen and walks the group through what they built. No grades, no judgement — just hearing how different people approached the same problem with completely different ideas.

**Your 5-minute demo script**

1. **Show it first.** Open the app and run it. Let people see it before you explain anything.
2. **What was your goal?** What did you set out to build, in one sentence?
3. **Walk through your prompts.** What did you type first? What changed after the first result?
4. **What went wrong?** One thing that broke or didn't work the way you expected — and how you fixed it.
5. **Did you reach your goal?** If yes — what would you add next? If no — what got in the way, and what would you do differently?

**What makes a great demo**

The most useful demos aren't the ones where everything worked perfectly. They're the ones where something broke and the person explains how they figured it out. That's the part everyone learns from.

WRAP-UP

# Deploy · Recap · Next steps

5 min

### Deploy your game to Netlify

⏱ 2 min

Your game is a single HTML file — deploying is just a drag and drop.

1. Go to netlify.com and log in (or create a free account)
2. Click Sites
3. Ask Claude to copy the file: type *"Copy game.html and save it as index.html"*
4. Drag the index.html file onto the Netlify drop zone
5. Netlify gives you a live URL instantly — share it with anyone

**Tip:** To update: drag a new index.html onto the same site in Netlify. The URL stays the same.

## What you built today

**Session recap**

1. Learned Claude Code basics — prompting, iterating, asking for explanations
2. Practised the 4 prompting patterns: generate, refine, debug, and explain
3. Built a fully playable retro shooter from 5 prompts — particles, audio, power-ups, hi-score
4. Deployed a live game to the web

## What to try next

- **More game types:** "Build a Breakout clone" / "Make a top-down dungeon crawler" / "Create a bullet hell with a boss"

QUICK REFERENCE

# Windows & Mac

| TASK | WINDOWS | MACOS |
|---|---|---|
| **Start Claude Code** | Start menu → type `Claude` | Launchpad → `Claude` |
| **New conversation** | `+ New Chat` button in the app | |
| **Open HTML in browser** | File Explorer → double-click file | Finder → double-click file |
| **Copy / rename file** | Ask Claude: *"Copy game.html as index.html"* | |

## Now go build something.

Download the full workshop PDF to keep as a reference.

Download Workshop PDF

Claude Basics · Setup Guide · Event Page

Built with Claude Code — claude.ai/code