

S0 Practice — AI Quick Start: Your First generate → verify → integrate → commit Loop

Produce a static review card (HTML/CSS) and ship it in one commit

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-14

Abstract Your first hands-on in the course. You create a small static mini-project (HTML + CSS, no framework, no programming language), then you run the uniform generate → verify → integrate → commit loop exactly once: you ask a chat assistant (a CRISP prompt) to produce a small styled review card, you verify the rendering in the browser, you integrate it into your files, and you commit it. The deliverable is the card rendering correctly and a clean commit.

Table of contents

1 Posture: You Drive, the AI Assists	1
2 Goal	2
3 Setup: Create the Mini-Project	2
4 The Small Change	3
5 The CRISP Prompt (Copy It As Is)	3
6 The AI Writes It	4
7 Checkpoint (Verify in the Browser)	5
8 Critique (What Would You Spot First?) ...	5
9 Commit	5
10 Self-Assessment: It's Done When... ..	6

This exercise comes with the S0 lesson AI Quick Start, where we met **CRISP** and the uniform **generate → verify → integrate → commit** loop. Here, you run this loop **exactly once**, for real, on a **static** deliverable (HTML/

CSS). One goal, one file, one commit — keep it winnable.

1 Posture: You Drive, the AI Assists

This exercise is **chat-based assistance**: you write the prompt, you reread the answer, you decide what goes into your files, you verify the result yourself. **There is no autonomous agent** acting on your behalf: the assistant only sees what you paste to it, and nothing is integrated or committed until you have done it and validated it. That discipline — *generate, verify, integrate, commit* — is the point of the course.

2 Goal

Run the first **generate** → **verify** → **integrate** → **commit** cycle on a **static** deliverable: a **styled review card** in HTML + CSS, integrated into an `index.html` page. The deliverable is the card rendering correctly in the browser **and** a clean commit.

3 Setup: Create the Mini-Project

Everything happens in a local folder, under git. No server, no framework, no build.

1. Create the project and initialize git:

```
mkdir -p webdev-s0 && cd webdev-s0
&& git init
```

2. Create the starting file `index.html` with the exact content below (a reviews-page skeleton, with an insertion point marked by a comment):

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1">
  <title>Product Reviews – Sample
Review</title>
  <style>
    body {
      margin: 0;
      font-family: system-ui,
      -apple-system, "Segoe UI", sans-
      serif;
      background: #f8fafc;
      color: #0f172a;
    }
    header {
      padding: 1rem 1.5rem;
```

```
      background: #ffffff;
      border-bottom: 1px solid
      #e2e8f0;
    }
    header h1 { margin: 0; font-
size: 1.25rem; }
    main {
      max-width: 40rem;
      margin: 0 auto;
      padding: 1.5rem;
    }
  </style>
</head>
<body>
  <header>
    <h1>Product Reviews</h1>
  </header>
  <main>
    <section class="reviews" aria-
label="Reviews">
      <!-- REVIEW: paste the
generated review card here -->
    </section>
  </main>
</body>
</html>
```

3. Commit this starting point, so that your working commit is clean:

```
git add index.html
git commit -m "chore: add static
reviews skeleton"
```

Open `index.html` in the browser (double-click, or open `index.html`). You should see the title “Product Reviews” and an empty area under the title: that is where the card will appear.

4 The Small Change

You will have the assistant produce a **review card**: an HTML block (an `<article class="review-card">` containing the review title, the author, the rating, the review body, the date, and a verified-purchase badge) **and** the CSS rules that style it. Two pastes, one file: the HTML replaces the `<!-- REVIEW: ... -->` comment, the CSS is added inside the `<style>` block.

That is the entire change. No framework, no programming language, no external resource.

5 The CRISP Prompt (Copy It As Is)

Copy the whole block below into your chat assistant and run it. (CRISP = **C**ontext, **R**ole, **I**ntput, **S**teps, **P**arameters — the full framework is in the lesson AI Quick Start.)

CRISP PROMPT — copy everything between the rules and paste it into your assistant

Context:

This is a 100% static reviews page, in HTML + CSS, with no framework, no preprocessor, and no programming language at all. Everything fits in a single

index.html file, with a `<style>` block in the `<head>`. The file already contains

a `<header>` with a title, then a `<main>` with a `<section class="reviews">` inside

which is the marker comment `<!-- REVIEW: paste the generated review card here -->`.

I want to add a small styled review card there.

Role:

You are a senior front-end developer. You write minimal semantic HTML and pure CSS, conforming to conventions. You never add a framework, a preprocessor, an external resource, or a programming language — only the smallest addition that satisfies the request.

Input:

The index.html file (I paste it here). The review to display, following this schema (review_id: string; source: one of

Amazon, BestBuy, Walmart; author: string; rating: integer 1-5; title: string;

body: string; created_at: ISO date; verified_purchase: boolean):

review_id "AMZ_0001", source "Amazon", author "Alice M.", rating 5,

title "Solid and quiet", body "Compact mechanical keyboard with linear

switches. Typing is pleasant and the board stays quiet in an open space.", created_at "2026-03-14",

verified_purchase true.

Steps:

1. Do not change anything that already exists.

2. Produce the review card as an `<article class="review-card" data-review-id="AMZ_0001">`

containing, in this order: an `<h2>` with the review title, the rating rendered

as text (for example "5 out of 5"),

the author, the review body, the date (a `<time>` element with the datetime attribute set to `created_at`), and a verified-purchase badge (a plain `<span>`, shown only because

`verified_purchase is true`). Optionally show the source (Amazon). 3. Produce the CSS rules for `.review-card` and its children: white background,

thin border, rounded corners, a light shadow, padding, and a small pill-style

badge for the verified-purchase label. The card must stay readable on mobile

(no overflow).

4. Indicate precisely where to paste each part: the HTML replaces the comment

"`<!-- REVIEW: ... -->`", the CSS is added inside the `<style>` block.

Parameters:

- Pure CSS only (no framework, no preprocessor, no CDN).
- Semantic HTML5, no `<script>` tags, no event handlers.
- No external resources (no remote image, no remote font).
- Keep the diff minimal: one HTML block + a few CSS rules, nothing else.
- Add no superfluous comments, no refactoring, no unrequested "improvement".

6 The AI Writes It

You can expect the model to return (a) the HTML block of the card and (b) the CSS rules. A good output looks like this:

```
<article class="review-card" data-  
review-id="AMZ_0001">  
  <h2>Solid and quiet</h2>  
  <p class="review-rating">Rating: 5  
out of 5</p>  
  <p class="review-author">by  
Alice M. <span class="review-  
source">(Amazon)</span></p>  
  <p class="review-body">Compact  
mechanical keyboard with linear  
switches. Typing is  
pleasant and the board stays quiet  
in an open space.</p>  
  <p class="review-meta">  
    <time datetime="2026-03-14">14  
Mar 2026</time>  
    <span class="review-  
badge">Verified purchase</span>  
  </p>  
</article>
```

```
.review-card {  
  background: #ffffff;  
  border: 1px solid #e2e8f0;  
  border-radius: 0.75rem;  
  box-shadow: 0 1px 3px rgb(15 23  
42 / 0.08);  
  padding: 1rem 1.25rem;  
  max-width: 26rem;  
}  
.review-card h2 { margin: 0 0 0.25rem;  
font-size: 1.1rem; }  
.review-rating { margin: 0 0 0.25rem;  
font-weight: 600; color: #b45309; }  
.review-author { margin: 0 0 0.5rem;  
font-size: 0.9rem; color: #475569; }  
.review-body { margin: 0 0 0.75rem;  
font-size: 0.95rem; }  
.review-meta { margin: 0; font-size:  
0.85rem; color: #475569; }  
.review-badge {  
  display: inline-block;
```

```
margin-left: 0.5rem;
padding: 0.1rem 0.5rem;
border-radius: 999px;
background: #dcfce7;
color: #166534;
font-size: 0.75rem;
}
```

Paste the HTML in place of the `<!-- REVIEW: ... -->` comment inside `<section class="reviews">`, and add the CSS at the end of the `<style>` block.

::: callout-tip title="The Golden Rule" Take the model's output **as is** and take it straight to the checkpoint. Do not "improve" it, do not rewrite it in your head first. The point of this first exercise is to run the loop — generate → verify → integrate → commit — exactly as the AI produced it. If something is off, you will see it at the checkpoint, and *that* is the lesson. :::

7 Checkpoint (Verify in the Browser)

Save `index.html`, then **open it (or reload it) in the browser** and check:

- ☐ **The card renders:** you see the review title, the rating, the author, the body, the date, and the verified-purchase badge.
- ☐ **It is styled:** white background, border, rounded corners, a light shadow, a green pill badge — not unstyled plain text.
- ☐ **It is responsive:** shrink the window, the card stays readable and does not overflow the screen.
- ☐ **Nothing else is broken:** the "Product Reviews" title is still at the top, the area stays centered.

It is you who decide, by looking at the screen, that the result is correct. The

assistant cannot know that for you: verification is your work.

8 Critique (What Would You Spot First?)

Apply the lens of the classic AI errors from the lesson to *this* change:

- **Invented framework.** Does the model add a utility-classes framework, a preprocessor, a component library, or a `<link>` tag to a CDN? The deliverable is pure HTML/CSS: any reference to a framework is a red flag.
- **Wrong stack / conventions.** Did it use inline `style="..."` attributes instead of the `<style>` block? Did it produce preprocessor CSS (syntax that is invalid in pure CSS)? Did it use a non-semantic `<div>` instead of an `<article>`?
- **Especially the superfluous.** Did it "helpfully" create a whole page, add a menu, multiple review cards, a dark mode, or copyright text? The diff must stay **one HTML block + a few CSS rules**.
- **Disguised interactivity.** Did it slip in a `<script>` tag, an event handler (`onclick`), or a network call? A static review card needs none of it — the verified badge is decorative here.

9 Commit

```
git add index.html
git commit -m "feat(reviews): add styled review card"
```

One file, one commit.

::: callout-warning title="The Trap (Read Before You Get Stuck)" Two things will trip you up on the first pass:

1. **The insertion point.** The HTML must replace the `<!-- REVIEW: ... -->` comment

inside `<section class="reviews">`, not after `</main>` and not in the `<head>`. If the card does not appear, first check where you pasted it.

2. **The CSS must be inside the `<style>` block.** Pasting the rules in the `<body>` (or after the card) styles nothing: in HTML, style rules must live inside a `<style>` element (here, in the `<head>`). If the card is present but “bare”, it is almost always that the CSS is in the wrong place. :::

10 Self-Assessment: It’s Done When...

You have finished S0 when all of the following is true:

- ☐ `index.html` opens in the browser and the review card renders and is styled (title, rating, author, body, date, verified badge).
- ☐ The card is responsive (stays readable when you shrink the window).
- ☐ `git log --oneline` shows your card commit, and `git status` is clean (only `index.html` has changed since the skeleton).
- ☐ You verified the rendering yourself — you did not simply trust the assistant’s output.