README

HTML clones

> **PREMISE:**
>
> Design an algorithm that will group together HTML documents which are similar from the perspective of a user who opens them in a web browser

1 The Task

Group files that look similar from the user perspective.

1.1 Understanding the Task

An HTML file can be analyzed from two distinct perspectives:

- **Code perspective**: This involves a structural and functional analysis of the underlying code meant to be rendered by the browser.
- **User perspective**: This focuses on the structural and functional analysis of the visual elements as rendered by the browser on the webpage.

The objective of this task is to analyze the contents of HTML files and identify **similarities**. The key point is that these similarities should be evaluated from the **user perspective**.
While the code perspective might seem redundant, at this stage of developing the solution, we will consider both perspectives, as they are not mutually exclusive. The code perspective may still serve as a useful **token** for establishing categories.

1.1.1 The Source Code Perspective

The **source code perspective** refers to how an HTML file is structured and written, focusing on syntax, tags, attributes, and overall organization. This view emphasizes the technical composition of the file, as opposed to how it is visually rendered.

1.1.2 The User Perspective

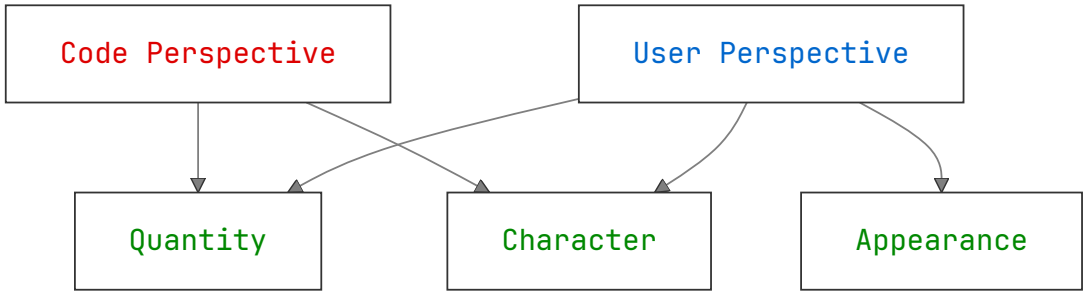
The **user perspective** refers to how the HTML file is rendered by the browser and experienced by the user. Although the source code is accessible, this task focuses only on the **rendered output**—layout, styling, content, and interactions.

1.1.3 Similarity

We define **similarity** as having a **resemblance** in **appearance**, **character**, or **quantity**, without requiring exact identity.
These three aspects are referred to as **tokens** of similarity—each representing a basic unit within our categorization system.

1.1.3.1 Defining the Tokens of Similarity

We define three tokens: **appearance**, **character**, and **quantity**.
We also consider two perspectives: **code** and **user**.



1.1.3.1.1 Tokens in the Code Perspective

We discard **appearance** from the code perspective, as reflected in the diagram above.

- **Quantity** is intuitive—measured by number of lines, elements, or blocks.
- **Character** is more abstract. While code lacks "feel," we treat structure, modularity, and purpose as **essence** to represent its character.

1.1.3.1.2 Tokens in the User Perspective

In the user perspective, the tokens of similarity are more easily observable and often intertwined. To maintain clarity, we adopt stricter definitions:

- **Quantity**: How content-heavy or dense the page appears.
- **Appearance**: The visible aspects—fonts, layout, colors, alignment, etc.
- **Character**: The emotional or psychological *effect* on the user (e.g., joy, boredom, confusion), distinct from visual properties.

1.1.3.1.3 Compound Tokens

A **compound token** is created by combining two or more basic similarity tokens.

The number of possible compound tokens is:

$$CT = \sum_{k=0}^{n} \binom{n}{k} = 2^n - n$$

(See section 1.1.3.2.1 for aggregate usage)

1.1.3.2 Scale of similarity

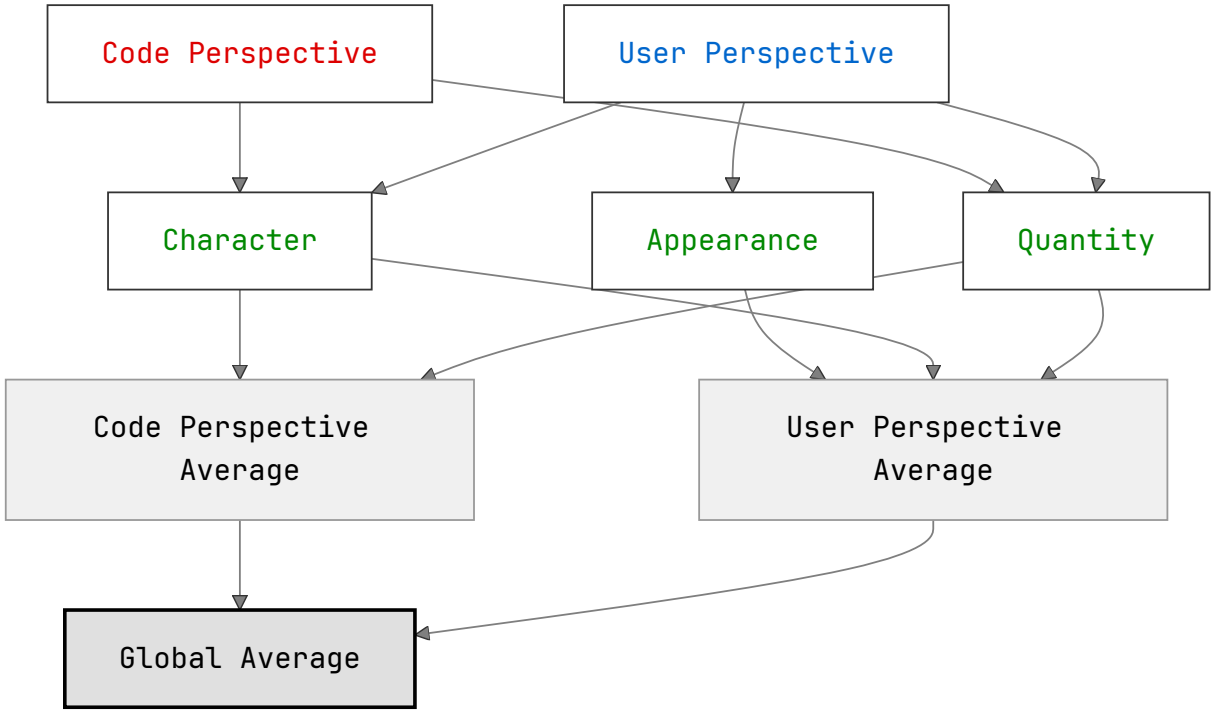
We condider a scale from (0% - 100%) to establish similarity between 2 HTML files. A percent of similarity can only be established by approaching the files from the same perspective, i.e. files can be only similar from the code perspective or similar from the user perspective.
0% similarity and 100% will not be considered as such proportions go outside the scope of the definition.
The 2 possible scales are **independent** of one another, meaning 2 files could be 99% similar from the code perspective and 1% similar from the user perspective and vice versa.

1.1.3.2.1 Methods of determining a final category.

With the similarity tokens established, we now need to determine the final, relevant categories that will be produced by the solution. These categories may be composed of one or more tokens.

We define the ***perspective aggregate*** as the average value of the tokens applied within a specific perspective.
For example, for the user perspective:
(Quantity + Appearance + Character) / 3

We define the ***global aggregate*** as the average of all perspective aggregates:
(code perspective aggregate + user perspective aggregate) / 2



Any of the tokens or aggregates shown above can serve as valid criteria for grouping HTML files.

The most significant point of contention is whether to include the individual tokens derived from the code perspective in the final grouping logic. This concern arises from the fact that aspects of the code perspective may not directly relate to how users experience the rendered HTML.

To address this, I propose that we **exclude individual code perspective tokens** (e.g., code quantity, code character) as *direct* criteria for group formation. In other words, there will be no category that groups files *solely* based on code perspective character or quantity.

However, we **retain the code perspective aggregate (c.p.a.)** as a contributor to the overall global average. This means that although c.p.a. will **not independently define a group**, it may influence groupings through the **global aggregate (g.a.)**.

The rationale for partially including the code perspective is that code is rarely written in isolation. Developers often build upon existing templates or styles, which can transfer some of the **character**—as perceived by users—from one version to another. Additionally, with the increasing use of AI tools, each LLM or code

generation engine tends to imprint its own *style* or *signature* on the generated HTML. These characteristics may manifest not only in code structure but also indirectly in the user-facing experience.

Therefore, while not strictly required, incorporating aspects of the code perspective can lead to a more **robust**, nuanced, and fine-grained grouping system.

> **Info**
>
> u.p. - user perspective
> c.p. - code perspective

That being said, we are left with the following set of candidate tokens:

[

- u.p. charater,
- u.p. appearance,
- u.p. quantity,
- u.p. average,
- global average

]

At this point, another design decision arises: whether to use only **independent tokens** or also allow **compound tokens**. We currently have 5 independent tokens.

If we include compound tokens—combinations of 2 or more independent tokens—we're left with 27 possible combinations. However, many of these compound tokens offer little to no added value and should be discarded. For example, (u.p. character, u.p. appearance, u.p. quantity) is simply a verbose representation of u.p. average.

While introducing compound tokens could add flexibility and allow for more dynamic groupings, it also risks increasing complexity and potentially complicating the codebase unnecessarily. Compound tokens might act as a kind of wildcard grouping mechanism in situations where the available HTML files don't neatly align with existing tokens. In such edge cases, they could help fine-tune categorization.

For now, we will treat the concept of compound tokens as a **future scalability point**, not part of the initial implementation.

We conclude 5 determining grouping tokens:

1. u.p. charater
2. u.p. appearance
3. u.p. quantity
4. u.p. average
5. global average

1.1.3.2.2 Case study

Let's walk through two examples to illustrate how the system evaluates similarity:

Example Case 1

Let there be two HTML files: `a.html` and `b.html`. The intended purpose of both files is to display a simple message to the user, such as "Hello World!"

1. `a.html` contains 10 lines of code and successfully displays the message.
2. `b.html` contains 100 lines of code, also displaying the same message.
3. Both files use the same font, size, layout, and styling.
4. `b.html` includes 90 lines of redundant code that do not contribute any additional functionality.

We analyze their similarity:
Code Perspective:

- `Quantity`: ~10% similarity
- `Character`: ~99% similarity
- `Aggregate`: ~55% similarity

User Perspective:

- `Quantity`: ~99% similarity
- `Character`: ~99% similarity
- `Appearance`: ~99% similarity
- `Aggregate`: ~99% similarity

Global Aggregate:

- ~77% similarity

Example Case 2

Now consider two HTML files, `a.html` and `b.html`, both created to present and advertise a restaurant's food menu.

1. Both contain approximately 1000 lines of code and include numerous high-quality images of food.
2. Both utilize vibrant colors and feature dynamic animations.

We analyze their similarity:
Code Perspective:

- `Quantity`: ~99% similarity
- `Character`: ~30% to 99% similarity (estimated)
- `Aggregate`: ~65% to 99% similarity

User Perspective:

- `Quantity`: ~50% to 99% similarity
- `Character`: ~20% to 90% similarity

- `Appearance`: ~50% to 99% similarity
- `Aggregate`: ~40% to 96.6% similarity

Global Aggregate:

- ~52.5% to 98.6% similarity

These two examples demonstrate both a **static and clearly defined case** (Example 1) and a **more dynamic, fluid case** (Example 2). The values provided are estimates used for illustrative purposes. In practice, the algorithm must rely on well-designed **heuristics** to calculate these similarity percentages with accuracy and consistency.