C-CSS

C-CSS is a composition-based CSS art library. Styling is a work of art! So should the approach!

## Composability Over Componentization

A component-oriented CSS approach has been common. Here you tie a class or ruleset to a component; and problem is, this becomes non-reusable by another component. So where there are UI components that must be both similar and different, you’re in for either overrides or duplication, or both. This tight coupling is, worse yet, written in two places – the component’s HTML/JS and its CSS – without that benefit of code-separation, as this is still nothing but a one-on-one mapping in cyclic dependency, and a lot of back-and-forth.

C-CSS implements, not monolith styling, but single-purpose, function-specific classes that you easily compose into HTML elements or components right as you define them. Now your work ends in simply factoring-in the right classes for an element.

This is an important difference as one approach opens up everything for reuse while the other inhibits reuse. Furthermore, HTML is about form and function, and truly better with componentization. But CSS is a work of art and design that must be approached with more fine-grained composability and fine-tuning. The specifics are different, and it is in this understanding that we based C-CSS classes in the language of design, not component.

C-CSS makes the difference in:

* Reusability – You reapply classes instead of recreate rulesets. Classes are not components of their own, but are combined to define a component.
* Maintainability – With composition, there’s now one place to code your UI, and one codebase to maintain. CSS gets completely of the way, even with a growing UI.
* Universality – With a UI-agnostic implementation, there can now be one standard CSS powering all your projects.
* Better language – With function-descriptive CSS classes, and an overall design-oriented language, you get more intuitive and your code ready for reading.
* Dynamism – With functional utility classes combined with magical tuning, the possibilities are endless; and every element is ready to transform, even on the fly.
* Extensibility –

## The Caveat

So we broke one rule: we went mainstream with CSS variables! But we bet ahead that you’ll rule in our favour for the wisdom in this.

# Structural classes

## Base Classes

### Position

There are four basic classes for position, and are on the *pos-* namespace.

* **pos-abs**
* **pos-rel**
* **pos-stc**
* **pos-sticky**

### Offsets

These classes offset an element on its coordinates. There is one general *offset-…* family, then four aspect-specific families in the conventionof *offset-<aspect>-…*.

Each family offers a scale of: 2, 4, 6, 8, 10, full, 0, where the numbers are based on pixel units, and the full keyword representing 100%.

These are often used together with the *pos-…* classes.

* **offset-…** -
  + **offset-2**
  + **offset-4**
  + **offset-6**
  + **offset-8**
  + **offset-10**
  + **offset-full**
  + **offset-0**
* **offset-top-…** -
  + **offset- top-2**
  + **offset- top-4**
  + **offset- top-6**
  + **offset- top-8**
  + **offset- top-10**
  + **offset- top-full**
* **offset-0**pos-stc

pos-sticky

## Tuning

## Responsivity

## Extensibility

# Theming classes

## Base Classes

## Magic Tuning

## States

## Extensibility

# Effect classes

# Resets