blowdrycss Documentation

Release 1.0.3

Chad Nelson

Mar 08, 2018

Contents

CHAPTER 1

About

blowdrycss is a rapid styling tool that compiles DRY CSS from encoded class selectors in your web project files.

1.1 Read Me

blowdrycss is a rapid styling tool that compiles DRY CSS from encoded class selectors in your web project files.

1.1.1 Getting Started

Quick Start Docs

Full documentation

1.1.2 Version Changelog

See `version.py` for full changelog.

1.0.1 – Enabled support for output file_name and extension customization. Created settings validation utility functions and unit tests. Custom output file examples: _blowdry.scss and custom.css. Added support for Python 3.6.

1.0.2 – Fixed file extension printing error. Now it uses the settings output_extension. Updated unit test.

1.0.3 – Changed requirements.txt so that versions must be >=. The pypandoc requirement in particular caused issues on both windows and linux installs.

1.1.3 Why the name blowdrycss?

Inspiration for the name came from the blow dryer. A blow dryer rapidly drys and styles hair.

Similarly, `blowdrycss` is used to rapidly style HTML and generate DRY CSS files using encoded class names.

1.1.4 Example Usage in HTML Tags:

Use the CSS level 1, 2.1, and 3 syntax that you already know.

```
<div class="text-align-center margin-top-30">
    <p class="font-size-25">
        The font-size is 25px. <span class="green">Green Text</span>
    </p>
</div>
```

`blowdrycss` decodes the class names `text-align-center`, `margin-top-30`, `font-size-25`, and `green`; and generates the following atomic CSS in `blowdry.css`:

```
.text-align-center { text-align: center }
.margin-top-30 { margin-top: 30px }
.font-size-25 { font-size: 25px }
.green { color: green }
```

1.1.5 Advantages of blowdrycss

1. **Rapid Development:** Less time spent writing CSS, and cleaning up unused style rules.
2. **DRY (Don't Repeat Yourself):** Reduces CSS file size by only defining properties once.
3. **Symbiotic:**
    - Can be integrated with the current ecosystem of CSS compilers and frameworks. Compatible with SASS, SCSS, PostCSS, LESS, Foundation, Bootstrap.
    - Supports class selector discovery within HTML, JINJA, XHTML, .NET, Ruby ERB Templates, Javascript, and C#.
4. **Documented:** Hands-on tutorial and sphinx documentation to get you up and running fast.
5. **Robust:** Built for the real world in which deadlines and division of labor is not always taken into account. Can be used across all phases of a products lifecycle from prototype to production.
6. **Customizable: Features can be turned on and off inside of blowdrycss_settings.py. Examples include:**
    - Watchdog file monitoring
    - Logging
    - Unit parsing
    - Color parsing
    - Font parsing
    - CSS Minification
    - Media query parsing.
7. **Atomic:** Generates atomic CSS declarations.

8. **Standardized:** HTML5 compatible. All W3C CSS Level 2.1, and Level 3 properties implemented. PEP8 Compliant.
9. **Tested:** UnitTest Coverage
10. **Permissive:** MIT license

1.1.6 Requirements

- Python 2.7.x or 3.3+ (required)
- cssutils 1.0.1+ (required)
- future 0.15.2+ (required - for Python 2.7)
- pandoc (required - file type conversion)
- pypandoc 1.1.2+ (required - file type conversion)
- watchdog 0.8.2+ (required - monitor directory and auto-generate CSS)

Optional

- tox 2.3.1+ (Multi-environment testing)
- tox-travis 0.4+ (Allows tox to be used on Travis CI.)
- coverage 4.0.2+ (Check test coverage)
- coveralls 1.1+ (Used to report coverage when tox is run via Travis CI.)
- sphinx 1.3.3+ (docs)

1.1.7 Pre-Requisite Knowledge

- Basic HTML and CSS
- Zero programming experience required.

1.1.8 Motivation

This tool was created after seeing how many companies manage their CSS files. The following are some scenarios:

Scenario 1 - WET (Write Everything Twice) CSS

Inside a CSS file you find the following:

```
.header-1 { font-weight: bold; font-size: 12px; font-color: red; }
.header-2 { font-weight: bold; font-size: 16px; font-color: blue; }
.header-3 { font-weight: bold; font-size: 12px; font-color: green; }
```

The property `font-weight: bold;` appears three times, and `font-size: 12px;` appears twice. This is not DRY (Don't Repeat Yourself).

Scenario 2 - Stale or Unused CSS

Inside a CSS file you find the following:

```
.banner-video {
    position: absolute;
    top: 48%;
    left: 50%;
    min-width: 100%;
    min-height: 100%;
    /*width: auto;*/
    /*max-height: 30.5em;*/
    z-index: -100;
    transform: translateX(-50%) translateY(-50%);
    background-color: rgba(0,0,0,1);
    background-size: contain;
    transition: 1s opacity;
}
```

Six months later the person who wrote this CSS is then asked to remove banner-video from the homepage. More often than not the front-end developer will remove the CSS class from the HTML file, but not from the CSS file. This leaves unused CSS lurking in the project.

Reasons include:

- Forgetting to delete the rule from the CSS file.
- Fear that the class is used somewhere else and that it might break the site.
- Being too busy to search all of the files in their project for other potential use cases.

Now 326 bytes worth of stale CSS data lurks in the style files.

Scenario 3 - Modern CSS Pre-compiler:

CSS compilation with SASS/SCSS, PostCSS, or LESS is awesome, and makes writing lots of CSS rules easy. Tools like these allow auto-generation of hundreds of header rules like the ones above. If care is not taken this leverage can rapidly grow the CSS file.

SCSS Mixin example from a recent project:

```
@mixin text($font-color, $font-size, $font-family:"Open Sans", $line-height:inherit) {
    color: $font-color;
    font-size: $font-size;
    font-family: $font-family, $default-font-family;
    line-height: $line-height;
}
```

This mixin is called using `@include` as follows:

```
@include text($color-blue, rem-calc(14px), $default-font-family);
```

It turns out that `@include text(...)` is called 627 times in our SCSS. Most of these `@include` statements include at least one matching input parameter resulting in thousands of duplicate CSS properties.

Auto-generating `font-size: 1rem;` 500 times is now super easy with a pre-compiler and a for-loop. Some might say,

```
Well we minified it to save space.
```

Yes but,

```
Why did you write the same property 500 times in your main CSS file?
```

CSS File size does matter. For consideration:

- Longer download times increase user bounce rates especially on mobile devices.
- Data pollution on the Internet.
- Increased likelihood of style bugs.
- Increased time required to implement new changes and deprecate features.

1.1.9 What it is not

This tool is not designed to replace the need to hand-craft complex CSS property or rule declarations.

- Custom CSS would need to be written for Multi-rule classes, Background images, url() values, multi-word fonts, and some shorthand properties.

The following is an example of something this tool in not intended to generate, and something that still needs to be written by hand.

```
.home-banner {
    background: url("https://somewhere.net/images/banner/home-mainbanner-bg.jpg") no-
↪repeat;
    font-family: "Open Sans","Source Sans Pro",Arial;
    background-repeat: no-repeat;
    background-size: cover;
    min-height: 7rem;
    font-weight: bold;
    font-size: 3.5625rem;
    color: white;
    line-height: 3.6875rem;
    text-align: center;
    text-shadow: -2px 2px 4px rgba(0,0,0,0.5);
}
```

1.1.10 Valuable References

Blowdrycss Documentation

Github Repo

Slides presented at DessertPy

W3C Full CSS property table

Don't Repeat Yourself

Download Python

cssutils 1.0.1+

watchdog 0.8.2+

1.1.11 License

The MIT license

1.1.12 How to Contribute

- Open an Issue first and get community buy-in.
- Write Code
- Write Unit Tests (All tests must pass. 100% coverage preferred.)

1.2 Syntax – Encoded Class Formatting Rules

1.2.1 Dissecting a CSS Statement

```
.margin-10 { margin: 10px !important; }
```

Format	Property Name	Property Value	Priority
CSS	.margin-10	margin: 10px	!important

1.2.2 Dissecting Encoded CSS Classes

Encoded Class	Property Name or Alias	Property Value	CSS Rule Output
font-size-25	font-size-	25	.font-size-25 { font-size: 25px }
green	color-	green	.green { color: green }
p-70-10	p-	70px 10px	.p-70-10 { padding: 70px 10px }

1.2.3 Dashes separate words in multi-word property names and aliases.

A Property Names is a valid CSS property name in accordance with the W3C Full CSS property table

`font-weight, border-bottom-color, border-bottom-style, border-bottom-width, border-collapse`

1.2.4 Dashes are placed at the end of aliases to indicate that it's an alias and not a css property name.

Aliases for property names.

`f-weight-, bg-c-, bg-color-, t-align-`

1.2.5 Property names may be encoded as an alias.

Consider this dictionary key, value pair found in `datalibrary.py` dictionary `DataLibrary.self.custom_property_alias_dict`.

```
'font-weight':  {'fweight-', 'lighter', 'fw-', 'bolder', 'f-weight-',
'font-w-', 'bold'},
```

It maps the alias set `{'fweight-', 'lighter', 'fw-', 'bolder', 'f-weight-', 'font-w-', 'bold'}` to the property name `font-weight`. Meaning that any of the values in the set can be substituted for `font-weight`.

The full property name can also be used directly in the encoded class i.e. `font-weight-`.

1.2.6 Dashes separate CSS property name/alias from property value

Encoded Class Format	CSS Rule Output
property-name-value	.property-name-value { property-name: value }
alias-value	.alias-value { property-name: value }
font-weight-700	.font-weight-700 { font-weight: 700 }
fw-700	.fw-700 { font-weight: 700 }

1.2.7 Dashes separate multiple values for properties that take multiple values.

Encoded Class Format	CSS Rule Output
alias-value-value-valu e-value	.alias-value-va lue-value-value { property-name: value value value value }
padding-10-20-10-10	.padding-10-20- 10-10 { padding: 10px 20px 10px 10px }
p-10-20-10-10	.p-10-20-10-10 { padding: 10px 20px 10px 10px }

1.2.8 Dashes separate `!important` priority indicator `'-i'` (append to the end of the string)

Encoded Class Format	CSS Rule Output
alias-value-i	.alias-value-i { property-name: value !important }
font-weight-bold-i	.font-weight-bold-i { font-weight: bold !important }

1.2.9 Shorthand can be used in cases where the alias is unambiguously the css property value.

Applicable properties include: `color`, `font-weight`, `font-style`, `text-decoration`, and `text-transform`.

Encoded Class Format	CSS Rule Output
alias	.alias { property-name: alias }
purple	.purple { color: purple }
bold	.bold { font-weight: bold }
lighter	.lighter { font-weight: lighter }
underline	.underline { text-decoration: underline }
italic	.italic { font-style: italic }
lowercase	.lowercase { text-transform: lowercase }

1.2.10 Built-in CSS Property Values containing '-' are now valid.

- Implemented: 11/29/2015 version: 0.0.2

See `custom_property_alias_dict` in `datalibrary.py` for a comprehensive list of usable aliases. Note that not every built-in CSS property value defined here is implemented. The reason is that some values like 'right' and 'left' are used for more than one property name. Also, in the case of `font-weight` numbers are defined like '100', '200', and so on. Encoded CSS classes are not permitted begin with a number and are therefore excluded.

Value Encoding Format	CSS Property Value Output
sans-serif	font-family: sans-serif
x-large	font-size: x-large

1.2.11 CSS Web Safe Font Fallbacks are now Auto-Generated for `font-family`

- Implemented: 11/29/2015 version: 0.0.2

Per w3schools.com > The font-family property should hold several font names as a "fallback" system, to ensure maximum compatibility between browsers/operating systems. If the browser does not support the first font, it tries the next font.

Font Name as Alias	CSS Rule Output
arial	.arial { font-family: arial, sans-serif }
papyrus	.papyrus { font-family: papyrus, fantasy }

1.2.12 Color Declarations

Color Format	Encoded Class Format	CSS Rule Output
keyword	color-silver (most efficient)	.color-silver { color: silver }
rgb	color-rgb-0-255-0	.color-rgb-0-255-0 { color: rgb(0, 255, 0) }
rgba	color-rgba-255-0-0-0_5	.color-rgba-255-0-0-0_5 { color: rgba(255, 0, 0, 0.5) }
hex6	color-h0ff23f (prepend `h`)	.color-h0ff23f { color: #0ff23f }
hex6	h0ff23f (no property name)	.h0ff23f { color: #0ff23f }
hex3	color-h03f (prepend `h`)	.color-h03f { color: #03f }
hex3	hfd4	.hfd4 { color: #fd4 }
hsl	color-hsl-120-60p-70p	.color-hsl-120-60p-70p { color: hsl(120, 60%, 70%) }
hsla	color-hsla-120-60p-70p-0_3	.color-hsla-120-60p-70p-0_3 { color: hsl(120, 60%, 70%, 0.3) }

1.2.13 Negative Values

A `-n` prefix for a number becomes a minus sign `-`, and creates a negative value.

Value Encoding Format	CSS Property Value Output
top-n48	top: -48px
margin-n5cm-n6cm	margin: -5cm -6cm
height-n9in	height: -9in

Note: The `'n'` in `margin` and `-9in` are unaffected.

1.2.14 Use underscores to indicate Decimal point.

Decimal points are substituted for all underscores.

Value Encoding Format	CSS Property Value Output
margin-1_32rem	margin: 1.32rem
left-n4_2rem	left: -4.2rem

Note: Underscores can only be used as decimal points.

Other usage of underscores will invalidate the class. e.g. `padding_1`, `*padding-1`, or `padding-1*` are considered invalid and will not be decoded. Classes may still be defined with these names, but CSS would not be generated by this tool.

1.2.15 Using Percentages 'p' becomes '%'

Use a suffix of `p` to indicated a percentage value.

Value Encoding Format	CSS Property Value Output
margin-1p-10p-3p-1p	margin: 1% 10% 3% 1%
right-n3_2p	right: -3.2%

1.2.16 Default Units:

If units are not provided in the class name, then default units were applicable. The default units are defined in `UnitParser.default_property_units_dict` inside `unitparser.py`. This makes it possible to easily change the default units for a particular property name.

Value Encoding Format	CSS Property Value Output
padding-50	padding: 50px
elevation-20	elevation: 20deg

1.2.17 Optional Unit Conversion

- Implemented: 11/28/2015 in version: 0.0.2

To enable 'px' to 'em' unit conversion open `blowdrycss.py` and set `use_em = True`

1.2.18 Explicitly Encoding Units in Class Name

Value Encoding Format	CSS Property Value Output
padding-50cm	padding: 50cm
width-120vmin	width: 120vmin

1.2.19 Pseudo Class and Pseudo Element Selectors

- Implemented as of version: 0.2.0

Pseudo classes and pseudo elements are documented here.

Examples:

```
'color-blue-hover', 'padding-10rem-i-active', 'bgc-h048-visited',
'color-red-after', 'padding-20rem-i-before', 'bgc-h096-selection'
```

Note: Pseudo classes with parenthesis are excluded. Also, chaining pseudo items together is not implemented. Replaced the print statements in FileHandler with logging.debug to increase efficiency.

Value Encoding Format	CSS Rule Output
pink-hover	.pink-hover:hover { color: pink }

1.2.20 Media Queries using Breakpoints

- Implemented: 1/2/2016 in version: 0.0.6
- Valid Formats

```
# General case
'property name/alias' + 'breakpoint_key' + 'limit_key'

# Special case -- Implied property_name is 'display'.
# Allows elements to be visible or hidden.
'breakpoint_key' + 'limit_key'
```

- Breakpoint keys – See `blowdrycss_settings.py` if you want to customize these.

```
'xxsmall-', 'xsmall-', 'small-', 'medium-', 'large-',
'xlarge-', 'xxlarge-', 'giant-', 'xgiant-', 'xxgiant-'
```

- Limit keys

```
'only', 'down', 'up'
```

- Breakpoints and limits combined. CSS property name defaults to `display`.

```
xxsmall-up
medium-only
xxlarge-down
```

- Custom CSS Property with breakpoint and limit suffix.

```
bold-small-only
color-hfff-giant-down
text-align-center-large-up
```

- Set Custom Breakpoints 4/1/2016 as of version 0.1.8. Breakpoints can now be set by specifying a screen width as a custom breakpoint before the limit key `-up` or `-down`. The `-only` limit key is excluded since it would only apply when the width is an exact match. Units default to pixels if not specified. Unit conversion still applies if `use_em = True` in blowdrycss_settings.py.

```
padding-25-820-up
display-480-down
margin-5-2-5-2-1000-up
display-960-up-i
display-3_2rem-down
```

Value Encoding Format

`display-medium-up`

CSS Media Query Output

```
@media only screen and (max-width: 30.0625em) {
    .display-medium-up { display: none }
}
```

Value Encoding Format

`large-up` – Property name `'display-'` is optional.

CSS Media Query Output

```
@media only screen and (max-width: 45.0625em) {
    .large-up { display: none }
}
```

Value Encoding Format

`giant-down` – Property name `'display-'` is optional.

CSS Media Query Output

```
@media only screen and (min-width: 160.0em) {
    .giant-down { display: none }
}
```

Value Encoding Format

`padding-100-large-only`

CSS Media Query Output

```
@media only screen and (min-width: 45.0625em) and (max-width: 64em) {
    .padding-100-large-only { padding: 6.25em }
}
```

1.2.21 Media Queries using Scaling Flag

- Implemented: 1/2/2016 in version: 0.0.6
- Allows scaling of `'font-size'` or other pixel–based CSS property as the screen width is reduced.
- Just add `'-s'` or `'-s-i'` to the end of your encoded class selector.

Value Encoding Format

`font-size-48-s`

CSS Media Query Output

```
.font-size-48-s { font-size: 3em }

@media only screen and (max-width: 45em) {
    .font-size-48-s { font-size: 2.6667em }
}
@media only screen and (max-width: 30em) {
    .font-size-48-s { font-size: 2.4em }
}
```

Value Encoding Format

`font-size-16-s-i` – !important global override case

CSS Media Query Output

```
.font-size-16-s-i { font-size: 1em !important }

@media only screen and (max-width: 45em) {
    .font-size-16-s-i { font-size: 0.8889em !important }
}
@media only screen and (max-width: 30em) {
    .font-size-16-s-i { font-size: 0.8em !important }
}
```

CHAPTER 2

Quickstart

2.1 Quick Start Guide

Documentation home

This guide rapidly shows you how to:

- Setup a virtual environment.
- Install *blowdrycss*.
- Walk through the `/examplesite` demo.
- Auto-generate DRY CSS with blowdrycss.
- Rapidly style HTML with encoded class syntax.
- Access more in-depth information.

Note: If this guide seems too quick, then head over to the *Tutorial* instead.

```
> pip install virtualenv
> mkdir blowdrycss_tutorial
> cd blowdrycss_tutorial
> virtualenv
> source bin/activate
> Install pandoc `click here for os-specific instructions <https://pypi.python.org/
↪pypi/pypandoc/1.1.3#installing-pandoc>`__.
> pip install blowdrycss
```

- Download the zipped version of `blowdrycss` from the github repository.
- Copy and paste the entire `examplesite` folder from the downloaded zip file to the new `blowdrycss_tutorial` folder.
- Look at the files inside of the `examplesite` folder. There should be the following:

```
blowdrycss_tutorial/
    examplesite/
        images/
        index.html
        test.aspx
        test.html
        test.jinja2
```

- Open a new Command Line Interface (CLI).

```
> cd <path to>/blowdrycss_tutorial/examplesite
> python -m http.server 8080            # (Python 3.x)
> python -m SimpleHTTPServer 8080       # (Python 2.x)
```

- Open a web browser and go to localhost:8080 by clicking here.
- The page should contain lots of unstyled text and images. It should basically be a mess.
- Leave the web server running, and go back to the original CLI.
- Ensure that the current folder is `blowdrycss_tutorial`. Run blowdrycss.

```
> blowdrycss
```

- Look at the files inside of the `examplesite` folder again. There should be a new subfolder called `css` containing the files `blowdry.css` and `blowdry.min.css`.

```
blowdrycss_tutorial/
    examplesite/
        css/
            blowdry.css
            blowdry.min.css
        images/
        clashing_aliases.html
        index.html
        property_aliases.html
        test.aspx
        test.html
        test.jinja2
    blowdrycss_settings.py
```

- Navigate to `<path>/blowdrycss_tutorial/examplesite/css`, and verify that `blowdry.css` and `blowdry.min.css` now exist.
- A file `blowdrycss_settings.py` appears. This file can be used to modify or override default settings. Use of this file is documented in the *Advanced Topics* section.
- Two new HTML files `property_aliases.html` and `clashing_aliases.html` also appear. There is more about these files in the *Advanced Topics*.
- Open `<path>/blowdrycss_tutorial/examplesite/index.html`
- Go to line 12 and find:

```
<h1 class="c-blue text-align-center display-medium-up font-size-48-s">
```

- From the class attribute delete `c-blue` and replace it with the word `green`.
- Change `font-size-48-s` to `font-size-148-s`.
- The line should now look like this:

```
<h1 class="green text-align-center display-medium-up font-size-148-s">
```

- Save the changes.
- Ensure that the current folder is `<path>/blowdrycss_tutorial`.
- Run `> blowdrycss`
- Now refresh the browser for the web page running on localhost:8080.
- The title at the top of the page should be large and green.

Syntax

More information about how to write well-form encoded class names is found on the *Syntax – Encoded Class Formatting Rules* page.

Want to learn more?

Go to Part 5 of the *Tutorial*.

Head on over to *Advanced Topics*.

Documentation home

CHAPTER 3

Tutorial

3.1 Tutorial

This guide teaches you how to:

- Setup the tutorial's virtual environment.
- Install `blowdrycss`.
- Walk through the `/examplesite` demo.
- Auto-generate DRY CSS with blowdrycss.
- Rapidly style HTML with encoded class syntax.
- Access more in-depth information.

Note:

No assumptions are made about your level of proficiency with python.

If this tutorial seems too slow, then go to the *Quick Start Guide*.

3.1.1 Part 1 - Setup virtualenv and install blowdrycss

- Python is required. Python 3.x is preferred. It can be downloaded here.
- Check your python installation or version number. Open a command line interface (CLI), and enter the following command.

```
> python
```

  Something *like* the following should appear.

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit␣
↪(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To exit type: `exit()`.

- Create a virtual environment. (For the purposes of this tutorial the project folder should be initially empty.)

```
> pip install virtualenv
> mkdir blowdrycss_tutorial
> cd blowdrycss_tutorial
> virtualenv
```

- Activate the virtual environment. Verify initial state.

```
> source bin/activate (linux, osx) or scripts\activate.bat (windows)
> python
>>> exit()
> pip freeze
```

- Install pandoc click here for os-specific instructions.
- Install `blowdrycss`.

```
> pip install blowdrycss
> pip freeze
```

- Deactivate virtual environment.

```
> deactivate
```

Explanation:

`pip install virtualenv` Install virtual environment package from PyPi.

`mkdir blowdrycss_tutorial` Create a folder for this tutorial.

`cd blowdrycss_tutorial` Sets the current working directory to your web projects directory.

`virtualenv` Setup your project up as a virtual environment.

`source bin/activate` Activates the new virtual environment.

`python` Confirm the version of python that is installed.

`exit()` Exit python console.

`pip freeze` Shows all of the python packages that are currently installed in the virtual environment.

`pip install blowdrycss` Installs blowdrycss and related dependencies 'inside' of your virtual environment.

`pip freeze` Shows blowdrycss and the related dependencies after the install.

`deactivate` deactivates the virtual environment.

Good References:

- Python Guide
- The virtualenv docs
- PyCharm virtualenv How To

3.1.2 Part 2 - Setup examplesite and run a local webserver.

- Download the zip version of `blowdrycss` from the github repository.
- Copy and paste the `examplesite` folder into the `blowdrycss_tutorial` folder created in Step 1.
- `cd examplesite`
- Run `python -m http.server 8080` (Python 3.x) or `python -m SimpleHTTPServer 8080` (Python 2.x) depending on your version of python. On Windows the firewall might complain. Tell it to allow this server to run.
- Open a web browser and go to localhost:8080 by clicking here.
- The page should contain lots of unstyled text and images. It should basically be a mess.
- Go back to the command line interface (CLI). The local webserver can be stopped by pressing `Ctrl + C` or closing the window. If you want to keep the webserver running then you will need to open a separate CLI.

3.1.3 Part 3 - Auto-generate CSS

- Look at the files inside of the `examplesite` folder. There should be the following:

```
blowdrycss_tutorial/
    examplesite/
        images/
        index.html
        test.aspx
        test.html
        test.jinja2
```

- Ensure that the current folder is `blowdrycss_tutorial`.

```
> cd ..
```

- Reactivate the virtualenv and run blowdrycss.

```
> source bin/activate
> blowdrycss
```

- Look at the files inside of the `examplesite` folder again. There should be a new subfolder called `css` containing the files `blowdry.css` and `blowdry.min.css`.

```
blowdrycss_tutorial/
    examplesite/
        css/
```

```
            blowdry.css
            blowdry.min.css
        images/
        clashing_aliases.html
        index.html
        property_aliases.html
        test.aspx
        test.html
        test.jinja2
    blowdrycss_settings.py
```

- Navigate to `<path>/blowdrycss_tutorial/examplesite/css`, and verify that `blowdry.css` and `blowdry.min.css` now exist.
- A file `blowdrycss_settings.py` appears. This file can be used to modify or override default settings. Use of this file is documented in the *Advanced Topics* section.
- Two new HTML files `property_aliases.html` and `clashing_aliases.html` also appear. There is more about these files in the *Advanced Topics*. In general, they document syntax that can (property_aliases) and cannot be used (clashing_aliases).
- Open a web browser and go to localhost:8080.
- The page should now be styled better. Keep in mind that some elements are intentionally left un-styled for tutorial purposes.

Note:

The CSS files `blowdry.css` and `blowdry.min.css` are auto-generated and not intended to be edited by humans.

Any manual changes made to these two files are overwritten when `blowdrycss` is run.

To test this delete the `css` folder, and run `blowdrycss`. The `css` will automatically appear under examplesite.

3.1.4 Part 4 - Apply new styles in `index.html`

Usage of Encoded Classes in HTML Tags

```
<div class="text-align-center margin-top-30">
    <p class="font-size-25">
        The font-size is 25px. <span class="green">Green Text</span>
    </p>
</div>
```

blowdrycss decodes the class names `text-align-center`, `margin-top-30`, `font-size-25`, and `green`; and generates the following CSS in `blowdry.css`:

```
.text-align-center { text-align: center }
.margin-top-30 { margin-top: 30px }
```

```
.font-size-25 { font-size: 25px }
.green { color: green }
```

Lets actually style something.

- Navigate to `<path>/blowdrycss_tutorial/examplesite/`
- Open `index.html`
- Go to line 12 and find:

```
<h1 class="c-blue text-align-center display-medium-up font-size-48-s">
```

- From the class attribute delete `c-blue` and replace it with the word `green`.
- Change `font-size-48-s` to `font-size-148-s`.
- The line should now look like this:

```
<h1 class="green text-align-center display-medium-up font-size-148-s">
```

- Save the changes.
- Now refresh the web page running on localhost:8080.
- What happened? Nothing happened because you need to run `blowdrycss` first. Sorry for the trick, but this is the most common reason why it doesn't seem to be working.
- Ensure that the current folder is `<path>/blowdrycss_tutorial`.
- Run `> blowdrycss`
- Now refresh the browser for the web page running on localhost:8080.
- The title at the top of the page should be large and green.

3.1.5 Part 5 - Exploring the auto-generated CSS

- Navigate to `<path>/blowdrycss_tutorial/examplesite/css`.
- List the items in the directory `ls` or `dir`.
- The following files should appear:

```
blowdry.css
blowdry.min.css
```

- Both of these files contain the exact same style rules. The only difference is that the one with the `*.min.css` extension in minified. This means that it is smaller and takes less time to upload and download over the Internet. However, minified files are not designed to be human-readable. The `*.css` is designed to be human-readable.
- Open each file and see the difference. The `blowdry.css` contains line breaks and whitespace. Whereas, `blowdry.min.css` is written as a single line with whitespace removed.

CSS is Auto-Generated

- Look in `blowdry.css` for `.green`.

```
.green {
    color: green
    }
```

- This is the actual CSS that was generated as a result of adding the `green` CSS class selector to the `<h1>` tag.
- Change `color:  green` to `color:  black`.
- Save `blowdry.css`.
- Navigate back to `<path>/blowdrycss_tutorial`
- Run `blowdrycss`.
- Navigate to `<path>/blowdrycss_tutorial/examplesite/css`.
- Look in `blowdry.css` for the `.green` class selector. The CSS is automatically changed from `color: black` back to `color:  green`. The reason is that `blowdry.css` and `blowdry.min.css` are auto-generated. They are both completely overwritten every time `blowdrycss` is run. The auto-generated CSS files are not human-editable.

```
.green {
    color: green
    }
```

Important: The auto-generated CSS files blowdry.css and blowdry.min.css are not human-editable. They are both overwritten each time blowdrycss is run.

Link Tag

- Navigate back to `<path>/examplesite`
- Open `index.html`
- The following is on line 7:

```
<link rel="stylesheet" type="text/css" href="css/blowdry.min.css" />
```

- This line tells the browser which CSS file to use. In this case, it is `css/blowdry.min.css`. Though this could be replaced with `css/blowdry.css` and the page would still look the same. The minified version causes the web page to load faster since the file is smaller.
- Change line 7 of `index.html` to:

```
<link rel="stylesheet" type="text/css" href="css/blowdry.css" />
```

- Save `index.html`.
- Now refresh the web page running on localhost:8080.
- The page should still look the same.
- Change line 7 of `index.html` back to the way it was.

```
<link rel="stylesheet" type="text/css" href="css/blowdry.min.css" />
```

- Save `index.html`.

3.1.6 Part 6 - Experimentation

- Center the image below the title with the class `text-align-center` in the `<div>` containing the image.
- Now (without running `blowdrycss`) refresh the web page running on localhost:8080.
- It worked. But why? The reason it worked is that `text-align-center` is already used in `index.html`, and is already defined in `blowdry.min.css`.

Padding Percentages and Decimals

- Go back to `index.html` and find the '+ sign' images named `images/plus.png`, and add the class `padding-bottom-3p` directly to the `img` class attribute to both of them. They are located at lines 19 and 21.
- Ensure that the current folder is `blowdrycss_tutorial`.
- Run `> blowdrycss`
- Now refresh the web page running on localhost:8080.
- The '+ sign' images now appear closer to the vertical center, but not quite.
- Open `index.html` and change one of the '+ sign' image class selectors from `padding-bottom-3p` to `padding-bottom-4_5p`.
- Ensure that the current folder is `blowdrycss_tutorial`.
- Run `> blowdrycss`
- Now refresh the web page running on localhost:8080.
- The '+ sign' image with the `padding-bottom-4_5p` is now closer to the vertical center.
- What is going on here, and what do the `p` and the _ do?
- To understand this better open up `blowdry.css` and search for `padding-bottom-3p`. The following CSS is found:

```
.padding-bottom-3p {
    padding-bottom: 3%
    }
```

  The `3p` property value is converted into `3%`. So the letter `p` allows the percentage sign `%` to be encoded.

- Now search for `padding-bottom-4_5p`. The following CSS is found:

```
.padding-bottom-4_5p {
    padding-bottom: 4.5%
    }
```

  The `4_5p` property value is converted into `4.5%`. Meaning that the underscore _ represents the decimal point `.` character.

- Generally, these encodings are necessary because characters like `.` and `%` are not allowed in class selector names (See here).
  - On an advanced note, it is possible to escape the `.` and the `%` characters in the CSS file like so:

```
.padding-bottom-4\.5\%
```

However, this is hard to read and non-standard CSS. Though it is *valid*. Therefore, escape characters are ignored and unsupported by `blowdrycss`. It is possible to learn more about escape characters here.

Shortcut and Multi-value CSS Properties

- Apply these encoded class selectors to an image:

```
border-10px-solid-black p-20-30-20-30 w-50
```

Decomposition

`border-10px-solid-black` Add a solid black border that is 10px thick.

`p-20-30-20-30` Add 20px padding top and bottom. Add 30px padding left and right.

`w-50` Make the image 50px wide.

- Ensure that the current folder is `<path>/blowdrycss_tutorial`.
- Run `> blowdrycss`

More Practice

- Change `border-10px-solid-black` to `border-10px-dashed-cornflowerblue`.
- Apply `display-none` to a div.
- Apply `uppercase` to any paragraph tag.
- Feel free to continue experimenting with different property names and values.

  More information about how to write well-form encoded class names is found on the *Syntax – Encoded Class Formatting Rules* page.

Want to learn more?

Head on over to *Advanced Topics*.

CHAPTER 4

Repository and Slides

- Download from Github
- Slides presented at DessertPy in Chandler, AZ. | *Feel free to use them.*

CHAPTER 5

Contents

5.1 Advanced Topics

- Use watchdog to automate CSS compilation.
- Learn about *Invalid Clashing Aliases* and *Valid Property Aliases*.
- How to change settings in `blowdrycss_settings.py`.
- Customizing the alias dictionary.
- Where are the semicolons?
- How to build a plugin. [todo]
- Pro-tip: Want to share your site with a client, co-worker, or colleague. Use ngrok.
- DRYness
- Syntax Guide

5.1.1 Automate CSS Compilation with Watchdog

- Having to run `blowdrycss` can annoying in a development environment.
- What if it were possible to auto-detect that `index.html` was saved and automatically run `blowdrycss`?
- It is possible with watchdog.

Note: As of version `0.1.3` this is now much easier.

Enable watchdog

- If the virtualenv is not already active, then activate the virtualenv with `source/bin activate`.

- Navigate to `<path>/blowdrycss_tutorial`.
- Open `blowdrycss_settings.py`.
- Set `auto_generate = True`.
- Run `blowdrycss`.
- Notice that what is printed differs from the default mode.
- Test it by saving a change to one of the project files e.g. change `<path>/blowdryexample/examplesite/index.html`.

5.1.2 Setting Customization

The first time the `blowdrycss` command is run a file is auto-generated in the current directory named `blowdrycss_settings.py`. This file provide the ability to override the default settings for a given project.

It is possible to change the directories, file types to discover, unit conversion, output file type, media query breakpoints, and more.

5.1.3 Find Non-matching classes

If the encoded class name contains a typo or invalid value e.g. `ppadding-5`, `margin-A`, `font-color-h000rem`, or `squirrel-gray` it will be placed in `removed_class_set`. The variable `removed_class_set` is found in `ClassPropertyParser()` inside of `classpropertyparser.py`.

One day this may be placed an HTML file for easier discovery of typos or invalid syntax.

5.1.4 Customize Aliases:

New custom aliases can be assigned as shorthand abbreviation for an official CSS property.

- Open the auto-generated settings file `blowdrycss_settings.py`.
- Edit `custom_property_alias_dict`.

Custom Alias Syntax:

custom_property_alias_dict (*dict*) – Contains customized shorthand encodings for a CSS property name. e.g. `'c-'` is an alias for `'color'`. This saves on typing.

These encoded class selectors can be used inside of Web project files matching `blowdrycss_settings.file_type`. They can be customized to your liking.

Custom Alias Rules:

- The dictionary `key` is the official W3C CSS property name. Also the `key` must exist in the set `datalibrary.DataLibrary.property_names`, as follows:

```
{
    'azimuth', 'background', 'background-attachment',
    'background-color', 'background-image', 'background-position',
    'background-repeat', 'border', 'border-bottom',
```

```
    'border-bottom-color', 'border-bottom-style',
    'border-bottom-width', 'border-collapse', 'border-color',
    'border-left', 'border-left-color', 'border-left-style',
    'border-left-width', 'border-right', 'border-right-color',
    'border-right-style', 'border-right-width', 'border-spacing',
    'border-style', 'border-top', 'border-radius',
    'border-top-left-radius', 'border-top-right-radius',
    'border-bottom-right-radius', 'border-bottom-left-radius',
    'border-top-color', 'border-top-style', 'border-top-width',
    'border-width', 'bottom', 'caption-side', 'clear', 'clip',
    'color', 'content', 'counter-increment', 'counter-reset',
    'cue', 'cue-after', 'cue-before', 'cursor', 'direction',
    'display', 'elevation', 'empty-cells', 'float', 'font',
    'font-family', 'font-size', 'font-style', 'font-variant',
    'font-weight', 'height', 'left', 'letter-spacing',
    'line-height', 'list-style', 'list-style-image',
    'list-style-position', 'list-style-type', 'margin',
    'margin-bottom', 'margin-left', 'margin-right',
    'margin-top', 'max-height', 'max-width', 'min-height',
    'min-width', 'opacity', 'orphans', 'outline',
    'outline-color', 'outline-style', 'outline-width',
    'overflow', 'padding', 'padding-bottom', 'padding-left',
    'padding-right', 'padding-top', 'page-break-after',
    'page-break-before', 'page-break-inside', 'pause',
    'pause-after', 'pause-before', 'pitch', 'pitch-range',
    'play-during', 'position', 'quotes', 'richness',
    'right', 'speak', 'speak-header', 'speak-numeral',
    'speak-punctuation', 'speech-rate', 'stress',
    'table-layout', 'text-align', 'text-decoration',
    'text-indent', 'text-shadow', 'text-transform', 'top',
    'unicode-bidi', 'vertical-align', 'visibility',
    'voice-family', 'volume', 'white-space', 'widows',
    'width', 'word-spacing', 'z-index'
}
```

Note: If a new key is added to the official W3C CSS standard, but not listed here feel free to raise an issue in the code repository.

- The dictionary `value` is a `set()` of custom string aliases. For example:

```
{'bgc-', 'bg-c-', 'bg-color-', }
```

- When adding a new alias it must end with a `'-'`. As an example, `'bgc-'` is a valid custom alias format. If the `'-'` is removed, then blowdrycss assumes that `'bgc'` expects it to be a valid and unique CSS property value (*which it is not*). An example of a valid, unique CSS property value would be `'bold'`.
- An alias must be unique across all defined aliases. Any alias that clashes with an alias in this dictionary or the dictionary auto-generated by `DataLibrary.initialize_property_alias_dict()` is removed, and becomes unusable.
- **Clashing aliases are:**
  - Printed when `get_clashing_aliases()` is run.
  - Automatically added to the `project_directory` as `clashing_alias.html`.
  - Automatically added to the sphinx docs and can be found under `/docs/clashing_aliases`.

`rst` (*requires sphinx*).

Custom Alias Examples:

- To add a new alias `'azi'` for CSS property `'azimuth'` add the `{key: value, }` pair `{'azimuth': {'azi-'}, }` to custom_property_alias_dict. Defining `'azi-'` allows the following encoded class selector syntax:

```
'azi-left-side', 'azi-far-left', ..., 'azi-rightwards'

<div class="azi-left-side">Azimuth applied to a DIV</div>
```

Aliases already known to clash are:

```
'background-color': {'bc-'},
'border-color': {'bc-', 'border-c-'},
'border-collapse': {'bc-', 'border-c-'},
'border-style': {'border-s-', 'bs-'},
'border-spacing': {'border-s-', 'bs-'},
'border-right': {'br-'},
'background-repeat': {'br-'},
'font-style': {'fs-', 'font-s-'},
'font-size': {'fs-', 'font-s-'},
'list-style': {'ls-'},
'letter-spacing': {'ls-'},
'max-height': {'mh-'},
'min-height': {'mh-'},
'max-width': {'mw-'},
'min-width': {'mw-'},
'pause-before': {'pb-'},
'padding-bottom': {'pb-'},
'padding-right': {'pr-'},
'pitch-range': {'pr-'},
'white-space': {'ws-'},
'word-spacing': {'ws-'},
```

5.1.5 Where are the semicolons in the CSS file?

After opening `blowdry.css`, it becomes evident that semicolons are not used for most of the css rule declarations.

Sample `blowdry.css` contents

```
.padding-5 {
    padding: 0.3125em
    }
.margin-top-50px {
    margin-top: 3.125em
    }
.t-align-center {
    text-align: center
    }
.padding-10 {
    padding: 0.625em
    }
.display-none {
    display: none
    }
.height-150px {
```

```
    height: 9.375em
    }
.margin-25 {
    margin: 1.5625em
    }
```

Why?

- The only or last css rule { property: value } is not required to end with a semicolon. See section 4.1.8 of the current CSS Standard.
- The auto–generated file `blowdry.css` is not intended to be human-editable. Any manual edits are over–written when `blowdrycss` is run. Generally, when building a CSS file by hand it is considered best practise to always include the final semicolon. The reason being that human–error is reduced the next time a person adds a rule to the CSS block. However, this does not apply for a file that is only machine–edited.
- It is compatible with all browsers.
- It results in slightly faster page loads due to smaller `*.css` file size.

5.1.6 DRY-ness must be balanced with other factors.

Consider the following:

```
<div class="background-size-cover min-h-7rem bold font-size-3_5625rem white line-
↪height-3_6875rem
            talign-center t-shadow-n2px-2px-4px-rgba-0-0-0-0_5">
    <!-- div contents -->
</div>
```

This is a case were the DRY principle is subsumed by the value of readability, brevity, and encapsulation. Creating a custom CSS class selector in this case might be warranted.

Also, just because this tool can decode the class

```
t-shadow-n2px-2px-4px-rgba-0-0-0-0_5
```

that doesn't mean it is intended to be frequently used in this manner.

My CSS is DRY, but my HTML is not.

Copying and pasting something like

```
p-10-20-11-22 h-50 w-50 talign-center orange font-size-16 margin-top-30
```

twenty times in an HTML file is not that DRY from an HTML perspective. If this is happening, then it might be valuable to pause and hand-craft a CSS class for this repeating class selector pattern.

5.1.7 Syntax Guide

Continue to *Syntax – Encoded Class Formatting Rules*.

5.2 Invalid Clashing Aliases

Property Name	Clashing Aliases
background-color	bc-
background-repeat	br- repeat
border-collapse	bc- border-c-
border-color	bc- border-c-
border-radius	br- border-r-
border-right	br- border-r-
border-spacing	bs- border-s-
border-style	bs- border-s-
font-size	font-s- fs-
font-style	font-s- fs-
letter-spacing	ls-
list-style	ls-
max-height	mh-
max-width	mw-
min-height	mh-
min-width	mw-
overflow	visible hidden
padding-bottom	pb-
padding-right	pr-
pause-before	pb-
pitch-range	pr-
play-during	repeat
visibility	visible hidden
white-space	ws-
word-spacing	ws-

5.3 Valid Property Aliases

Property Name	Valid Aliases
azimuth	center-left right-side center-right azi- behind far-right leftwards far-le
background	bac- bg-
background-attachment	ba- background-a-
background-color	background-c- bg-color- bg-c- bgc-
background-image	bi- background-i-
background-position	background-p- bp-
background-repeat	no-repeat background-r- repeat-x repeat-y
border	bor-
border-bottom	bb- border-b-
border-bottom-color	border-b-color- bbc-
border-bottom-left-radius	bbl- border-b-left-radius-
border-bottom-right-radius	bbr- border-b-right-radius-
border-bottom-style	border-b-style- bbs-
border-bottom-width	bbw- border-b-width-
border-collapse	
border-color	

border-left	bl- border-l-
border-left-color	border-l-color- blc-
border-left-style	border-l-style- bls-
border-left-width	border-l-width- blw-
border-radius	
border-right	
border-right-color	border-r-color- brc-
border-right-style	border-r-style- brs-
border-right-width	brw- border-r-width-
border-spacing	
border-style	
border-top	bt- border-t-
border-top-color	border-t-color- btc-
border-top-left-radius	border-t-left-radius- btl-
border-top-right-radius	border-t-right-radius- btr-
border-top-style	border-t-style- bts-
border-top-width	border-t-width- btw-
border-width	bw- border-w-
bottom	bot-
caption-side	cs- caption-s-
clear	
clip	
color	indigo gold firebrick indianred yellow darkolivegreen darkseagreen slategr
content	no-open-quote no-close-quote close-quote con- open-quote
counter-increment	counter-i- ci-
counter-reset	counter-r- cr-
cue	
cue-after	cue-a- ca-
cue-before	cue-b- cb-
cursor	sw-resize n-resize s-resize help e-resize default text move wait nw-resize
direction	rtl dir- ltr
display	table-footer-group xgiant xlarge table-row xsmall table-column table giant
elevation	lower level ele- below above higher
empty-cells	empty-c- ec-
float	
font	
font-family	charcoal copperplate papyrus fantasy tahoma sans-serif impact calibri aria
font-size	fsize- f-size-
font-style	oblique italic
font-variant	font-v- fv- small-caps
font-weight	fw- bold bolder f-weight- lighter fweight- font-w-
height	hei- h-
left	
letter-spacing	letter-s-
line-height	lh- line-h-
list-style	list-s-
list-style-image	list-s-image- lsi-
list-style-position	outside inside lsp- list-s-position-
list-style-type	list-s-type- square lower-roman decimal lower-greek georgian decimal-lead
margin	m- mar-

margin-bottom	mb- m-bot- margin-b-
margin-left	margin-l- ml-
margin-right	mr- margin-r-
margin-top	m-top- margin-t- mt-
max-height	max-h-
max-width	max-w-
min-height	min-h-
min-width	min-w-
opacity	opa-
orphans	orp-
outline	out-
outline-color	outline-c- oc-
outline-style	outline-s- os-
outline-width	ow- outline-w-
overflow	ove- scroll
padding	pad- p-
padding-bottom	padding-b-
padding-left	pl- padding-l-
padding-right	padding-r-
padding-top	pt- padding-t- p-top-
page-break-after	pba- page-b-after-
page-break-before	page-b-before- pbb-
page-break-inside	page-b-inside- pbi-
pause	
pause-after	pause-a- pa-
pause-before	pause-b-
pitch	high x-low low x-high
pitch-range	pitch-r-
play-during	play-d- pd- mix
position	relative pos- static absolute
quotes	quo-
richness	ric-
right	
speak	spell-out
speak-header	always sh- speak-h- once
speak-numeral	speak-n- digits continuous sn-
speak-punctuation	code sp- speak-p-
speech-rate	slow x-slow faster speech-r- fast sr- slower x-fast
stress	str-
table-layout	table-l- tl-
text-align	ta- talign- t-align- text-a-
text-decoration	text-d- td- blink overline line-through underline
text-indent	ti- text-i-
text-shadow	ts- text-s-
text-transform	uppercase lowercase text-t- tt- capitalize
top	
unicode-bidi	ub- embed bidi-override unicode-b-
vertical-align	sub va- text-bottom text-top middle v-align- valign- vertical-a- super base
visibility	collapse vis-
voice-family	vf- voice-f-

volume	`silent` `vol-` `loud` `x-soft` `x-loud` `soft`
white-space	`white-s-`
widows	`wid-`
width	`w-`
word-spacing	`word-s-`
z-index	`z-i-` `zi-`

5.4 Upcoming Features

5.4.1 Make DRYer:

TODO: Implement this essential feature. TODO: Document Currently two classes are being created with the same properties. The preferred solution would be to assign both classes to the same property.

Scenario 1:

Value Encoding Format	CSS Property Value Output
bold	.bold { font-weight: bold }
font-weight-bold	.font-weight-bold { font-weight: bold }

Duplicates the string `{ font-weight: bold }`.

DRY solution 1

```
.bold, font-weight-bold { font-weight: bold }
```

Scenario 2:

Value Encoding Format	CSS Property Value Output
padding-10	.padding-10 { padding: 10px }
padding-10px	.padding-10px { padding: 10px }

Duplicates the string `{ padding: 10px }`

DRY solution 2

```
.padding-10, .padding-10px { padding: 10px }
```

Drop requirement for hexadecimal color values to be prefixed with a property name. Implemented: 11/28/2015

Integrate optional px :point_right: em Unit Conversion. Implemented: 11/28/2015

Integrate option hexidecimal :point_right: rgb() Unit Conversion.

Create Seamless Media Queries for responsive layouts:

TODO: Implement this really cool feature. TODO: Document

Build responsive scaling fonts using -r:

TODO: Implement this really cool feature. TODO: Document

Encoded Class

font-size-25-r

Resulting CSS media query.

TODO: Add CSS here.

```
.font-size-25-r {
    font-size: 25px;
}
```

Sphinx Integration

TODO: Integrate Sphinx (in progress) TODO: Put the docs on readthedocs

5.5 Unsupported Features

5.5.1 Shorthand properties

Use shorthand properties at your own risk. Currently no support is guaranteed for shorthand properties.

5.5.2 No encoding is defined for '/', comma, dash, double quote, '@'.

CSS Property Value	Encodings Not Implemented
font: 12px/14px sans-serif	'/' and '-'
font: 16rem "New Century Schoolbook"	double quote
font-family: Palatino, serif, arial	comma

5.5.3 Properties Values that contain 'url()' are not supported as they are too bulky and verbose. These sorts of

`url()` declarations belong in your custom CSS class definitions.

CSS Property Value	Encodings Not Implemented
background-image: url("/home/images/s ample/image.png")	'/', '(', and double quote

CHAPTER 6

Code

Repository

6.1 blowdrycss

blowdrycss is a rapid styling tool that compiles DRY CSS from encoded class selectors in your web project files.

6.2 blowdry

`blowdry.`**`boilerplate`**`()`

Watchdog wrapper only calls this once to eliminate recurring performance impact.

- Validate the output_file_name and output_extenion settings.
- Generate Markdown documentation files.
- Generate HTML documentation files. (This location is important since it allows encoded css to be included in the documentation files.)
- Generate reStructuredText documentation files.

Returns None

`blowdry.`**`parse`**`(`*recent=True*, *class_set=set()*, *css_text=b"*`)`

It parses every eligible file in the project i.e. file type matches an element of settings.file_types. This ensures that from time to time unused CSS class selectors are removed from blowdry.css.

Order of Operations:

- Initialize settings.

- Start performance timer.
- Define File all file types/extensions to search for in project_directory
- Get all files associated with defined file_types in project_directory
- Get set of all defined classes
- Filter class names only keeping classes that match the defined class encoding.
- Build a set() of valid css properties. Some classes may be removed during cssutils validation.
- Output the DRY CSS file. (user command option)
- Output the Minified DRY CSS file. (user command option)

Depending on the settings this script generates the following:

- **DRY CSS files**
  - blowdry.css **human readable**
  - blowdry.min.css **minified**
- **Clashing Alias files (Encoded class selector aliases that are invalid and cannot be used because they clash.)**
  - Markdown **Github**
  - HTML **Browser**
  - reStructuredText **Sphinx**
- **Property Alias File (Encoded class selector aliases that are valid and can be used to construct class selectors.)**
  - Markdown **Github**
  - HTML **Browser**
  - reStructuredText **Sphinx**
- Temporal Statistics

Note: The default locations of these files can be overridden to suit your needs.

Directory assignments `project_directory` – Allows `blowdrycss` to know where the HTML project is located. It will only search the files in the directory specified here.

6.3 blowdrycss_settings

Usage Notes:

The first time `blowdrycss` is run it auto-builds `blowdrycss_settings.py` via `__init__.py`. This makes it easy to find and customize related settings.

Why such a long name? – blowdrycss_settings.py

Popular web frameworks such as django and flask already auto-generate a settings file called `settings.py`. The longer more specific name is used to prevent naming conflicts, and increase clarity.

Parameters:

markdown_directory (*string*) – Generally used for development purposes and github documentation.

project_directory (*string*) – Path to recursively search for all defined `file_types`.

css_directory (*string*) – Path where the projects CSS files are located.

docs_directory (*string*) – Path where Sphinx docs are located (requires sphinx to be installed and run).

output_file_name (*string*) – Name of the generated output file contain DRY CSS definitions.

output_extension (*string*) – File extension of the generated output file. Must begin with '.'

file_types = (*tuple of strings*) – All file types/extensions to search for in the defined project_directory that contain encoded class selectors.

Example format:

```
('*.html', )
```

timing_enabled (*bool*) – Run performance timer to see the performance of `blowdrycss`.

markdown_docs (*bool*) – Generate a markdown files that provides a quick syntax and clashing alias reference. Normally set to False except when posting to github.

html_docs (*bool*) – Generate a html file that provides a quick syntax and clashing alias reference.

rst_docs (*bool*) – Generate a sphinx rst file that provides a quick syntax and clashing alias reference.

human_readable (*bool*) – Generate a standard human readable css file. This file is named `blowdry.css` by default.

minify (*bool*) – Generate a minified version of the css file. This file is named `blowdry.min.css` by default.

media_queries_enabled (*bool*) – Generate breakpoint and scaling media queries.

use_em (*bool*) – A `pixels` to `em` unit conversion flag. True enables unit conversion. False disables unit conversions meaning any pixel value remains unchanged.

base (*int*) – Base used for unit conversion (typically set to 16). The pixel value will be divided by `base` during unit conversion.

xxsmall (*tuple of floats*) – (0px, upper limit in pixels)

xsmall (*tuple of floats*) – (xxsmall upper limit + 1px, upper limit in pixels)

small (*tuple of floats*) – (xsmall upper limit + 1px, upper limit in pixels)

medium (*tuple of floats*) – (small upper limit + 1px, upper limit in pixels)

large (*tuple of floats*) – (medium upper limit + 1px, upper limit in pixels)

xlarge (*tuple of floats*) – (large upper limit + 1px, upper limit in pixels)

xxlarge (*tuple of floats*) – (xlarge upper limit + 1px, upper limit in pixels)

giant (*tuple of floats*) – (xxlarge upper limit + 1px, upper limit in pixels)

xgiant (*tuple of floats*) – (giant upper limit + 1px, upper limit in pixels)

xxgiant (*tuple of floats*) – (xgiant upper limit + 1px, 1E+6) [Technically the upper limit is infinity, but CSS does not permit it.]

Custom Alias Syntax:

custom_property_alias_dict (*dict*) – Contains customized shorthand encodings for a CSS property name. e.g. `'c-'` is an alias for `'color'`. This saves on typing.

These encoded class selectors can be used inside of Web project files matching `file_type`. They can be customized to your liking.

For more details about how to create custom aliases head on over to *Advanced Topics*.

cssutils Patch:

`cssutils` does not currently support all CSS 3 Units. The patch in this file allows length units of `q`, `ch`, `rem`, `vw`, `vh`, `vmin`, and `vmax`. It also allows angle units of `turn`.

`blowdrycss_settings.px_to_em(pixels)`

Convert a numeric value from px to em using `settings.base` as the unit conversion factor.

Rules:

- `pixels` shall only contain [0-9.-].
- Inputs that contain any other value are simply passed through unchanged.
- Default `base` is 16 meaning `16px = 1rem`

Note: Does not check the `property_name` or `use_em` values. Rather, it blindly converts whatever input is provided. The calling method is expected to know what it is doing.

Rounds float to a maximum of 4 decimal places.

Parameters **pixels** (*str, int, float*) – A numeric value with the units stripped.

Returns

(str)

- If the input is convertible return the converted number as a string with the units `em` appended to the end.
- If the input is not convertible return the unprocessed input.

```
>>> from blowdrycss_settings import px_to_em
>>> # settings.use_em = True
>>> px_to_em(pixels='-16.0')
-1em
>>> # settings.use_em = False
>>> px_to_em(pixels='42px')
42px
>>> # Invalid input passes through.
>>> px_to_em(pixels='invalid')
invalid
```

6.4 datalibrary

class `datalibrary.DataLibrary`

DataLibrary is not intended for use outside of this file as each time its' called it rebuilds the dictionaries.

Attributes:

property_regex_dict (*dict*)

A regex dictionary for detecting more complex value patterns for a given property.

Dictionary Contains:

- The `key` is the official CSS property name.
- The `value` is a `set()` of regex strings.

Regexes Cases:

- Hexidecimal (3 digit) – 'h123', 'h123 bold', 'underline h123 bold'

- Hexidecimal (6 digit) – 'h123456', 'h123456 underline', 'underline h123456 bold'
- Hexidecimal (3 digit + pseudo-class + importance designator) – 'h123-hover-i', 'h123-after-i bold',
- Hexidecimal (6 digit + pseudo-class + importance designator) – 'h12ad56-hover-i', 'h12AD56-hover-i underline'
- Hexidecimal (3 digit + importance designator + pseudo-class) – 'h1f3-i-hover', 'h1F3-i-hover bold'
- Hexidecimal (6 digit + importance designator + pseudo-class) – 'h123456-i-hover', 'h123456-i-hover underline'
- **Hexidecimal Regex explained**
  - `r"(h[0-9a-fA-F]{3} ?)$"` or `r"(h[0-9a-fA-F]{6} ?)$"`
  - `h` – The substring must begin with an `h`.
  - `[0-9a-fA-F]` – The characters that follow must be a hexidecimal characters.
  - `{3}` or `{6}` – Limit the number of hexidecimal characters to either 3 or 6 only.
  - `' ?'` – The substring may optionally be followed by a space.

property_value_as_alias_dict (*dict*)

Maps valid, unique W3C CSS property values to a W3C CSS property name. This enables the use of unique property values as standalone class selectors i.e. `bold` can be used in place of `font-weight-bold`. This makes the syntax succinct while remaining declarative.

Rules:

- The property value alias must be unique across all valid W3C defined property values.
- The property value alias must be unique across all property values defined in the dictionary.
- The `key` is the official CSS property name.
- The `value` is a `set()` of valid, unique W3C CSS property values.

Example:

```
``'font-weight': {'bold', 'bolder', 'lighter', },``
```

Allowed values for font-weight according to W3C. Based on the standard `normal` is a valid property value. However, `normal` is not unique to `font-weight`. To verify that go here and search for `normal`. Also, `100 - 900` are listed as valid values, but CSS class selectors cannot begin with a digit. This implies that the numeric values cannot be included. That leaves `bold, bolder, and lighter`.

property_names (*set*)

The set of all CSS 2.1 property names listed here: http://www.w3.org/TR/CSS21/propidx.html on the W3C website.

clashing_alias_dict (*dict*)

Auto-generated dictionary of clashing aliases. An alias clashes if it exactly equals an alias associated with another property e.g. One alias for `border-right` is `'br-'`. However `background-repeat` has an identical alias of `'br-'`. Therefore `'br-'` is added to `clashing_alias_dict` and is not allowed to be used as an alias.

Dictionary Contains:

- The `key` is the official CSS property name.
- The `value` is a `set()` of custom string aliases.

property_alias_dict (*dict*)

Auto-generated dictionary of property aliases.

Dictionary Contains:

- The `key` is the official CSS property name.
- The `value` is a `set()` of custom string aliases.

alphabetical_clashing_dict (*dict*)

Alphabetized ordered dictionary of clashing aliases.

Ordered Dictionary Contains:

- The `key` is the official CSS property name.
- The `value` is a `set()` of clashing string aliases.

alphabetical_property_dict (*dict*)

Alphabetized ordered dictionary of property aliases.

Ordered Dictionary Contains:

- The `key` is the official CSS property name.
- The `value` is a `set()` of custom string aliases.

clashing_alias_markdown (*str*) – Auto-generated table of clashing aliases in markdown format.

property_alias_markdown (*str*) – Auto-generated table of property names and aliases in markdown format.

clashing_alias_html (*str*) – Auto-generated table of clashing aliases in HTML format.

property_alias_html (*str*) – Auto-generated table of property names and aliases in HTML format.

clashing_alias_rst (*str*) – Auto-generated table of clashing aliases in reStructuredText format.

property_alias_rst (*str*) – Auto-generated table of property names and aliases in reStructuredText form.

ordered_property_dict (*dict*)

Sorted property_alias_dict with the longest items first as the most verbose match is preferred i.e. If `css_class == 'margin-top'`, then match the `property_alias_dict` key that starts with `margin-top` not `margin`.

Ordered Dictionary Contains:

- The `key` is the official CSS property name.
- The `value` is a `set()` of custom string aliases.

static get_property_aliases (*property_name=''*)

Auto-generates and returns a set of aliases based on abbreviation patterns.

Rules:

- Property name does not contain a dash: {First three letters of property_name + `'-'`}
- Property name contains one dashes:

  1st word + 1st letter after dash + `'-'`

  1st letter of 1st word + 1st letter of 2nd word + `'-'`, (single dash case)

  1st letter of 1st word + 1st letter of 2nd word + 1st letter of 3rd word + `'-'`, (double dash case)

- Append dash '-' at the end of each abbreviation.
- Do not abbreviate words less than or equal to 5 characters in length.

Examples:

```
property_name --> {...}

color --> set()

padding --> {'pad-'}

margin-top --> {'margin-t-', 'mt-'}

border-bottom-width --> {'border-b-width', 'bbw-'}
```

Parameters **property_name** (*str*) – A CSS property name.

Returns Return a set() of abbreviation patterns according to the rules defined above.

autogen_property_alias_dict()

Uses `self.property_names` to auto–generate a property aliases. Assigns the result to `self.property_alias_dict`.

Note: The dictionary may contain clashing aliases. More processing is required to remove them.

merge_dictionaries()

Merges the `property_alias_dict` with `property_value_as_alias_dict`.

Note: All keys in both dictionaries much match.

Raises KeyError – Raises KeyError if property name does not exist as a key in `property_alias_dict`.

set_clashing_aliases()

Searches `property_alias_dict` for duplicate / clashing aliases and adds them to `clashing_alias_dict`.

remove_clashing_aliases()

Removes clashing aliases stored in `clashing_alias_dict` from `property_alias_dict` and deep copies the clean dictionary to `property_alias_dict`.

static dict_to_markdown(*h1_text=''*, *key_title=''*, *value_title=''*, *_dict=None*)

Convert a dictionary into a markdown formatted 2-column table.

Markdown Table Format:

```
# h1_text


key_title | value_title
--- | ---
key[0] | value
key[1] | value
```

Parameters

- **h1_text** (*str*) – Title for the table.
- **key_title** (*str*) – Key name.
- **value_title** (*str*) – Value stored at Key.
- **_dict** (*dict*) – A generic dictionary.

Returns (str) – Returns a markdown formatted 2-column table based on the key/value pairs in `_dict`.

static dict_to_html(*h1_text=''*, *key_title=''*, *value_title=''*, *_dict=None*)

Convert a dictionary into an HTML formatted 2-column table.

HTML Table Format:

```
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="icon" type="image/x-icon" href="/images/favicon.ico" />
        <title>value_title - blowdrycss</title>
        <link rel="stylesheet" type="text/css" href="/css/blowdry.min.css"/>
    </head>
    <body>
        <table>
            <thead>
                <tr>
```

```
                    <th>key_title</th>
                    <th>value_title</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>key[0]</td>
                    <td>value</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```

Parameters

- **h1_text** (*str*) – Title for the table.
- **key_title** (*str*) – Key name.
- **value_title** (*str*) – Value stored at Key.
- **_dict** (*dict*) – A generic dictionary.

Returns (str) – Returns a HTML formatted 2-column table based on the key/value pairs in `_dict`.

6.5 filehandler

class filehandler.**FileFinder**(*recent=True*)

Designed to find all `settings.files_types` specified within a particular `project_directory`. All folders within the `project_directory` are searched.

Parameters:

recent (*str*) – Flag that indicates whether to gather the most recently modified files (True Case) or all eligible files (False Case).

Members:

project_directory (*str*) – Set to settings.project_directory.

files (*str list*) – List of all paths to all parsable files.

file_dict (*dict*) – Dictionary of all paths to all parsable files where the file extension e.g. `*.html` is the key and the full file path is the value.

Example:

```
>>> file_finder = FileFinder(recent=False)
>>> files = file_finder.files
```

static print_collection(*collection*)

Takes a list or tuple as input and prints each item.

Parameters **collection** (*iterable*) – A list or tu of unicode strings to be printed.

Returns None

set_files()

Get all files associated with defined `file_types` in `project_directory`. For each `file_type` find the full path to each file in the project directory of the current `file_type`. Add the full path of each file found to the list `self.files`.

Reference: stackoverflow.com/questions/954504/how-to-get-files-in-a-directory-including-all-subdirectories#answer-954948

Returns None

set_file_dict()

Filter and organize files by type in `file_dict`.

Dictionary Format:

```
self.file_dict = {
    '.html': {'filepath_1.html', 'filepath_2.html', ..., 'filepath_n.html'},
    '.aspx': {'filepath_1.aspx', 'filepath_2.aspx', ..., 'filepath_n.aspx'},
    ...
    '.file_type': {'filepath_1.file_type', 'filepath_2.file_type', ...,
↪'filepath_n.file_type'},
}
```

Automatically removes the * wildcard from `file_type`.

Returns None

set_recent_file_dict()

Filter and organize recent files by type in `file_dict`. Meaning only files that are newer than the latest version of blowdry.css are added.

Dictionary Format:

```
self.file_dict = {
    '.html': {'filepath_1.html', 'filepath_2.html', ..., 'filepath_n.html'},
    '.aspx': {'filepath_1.aspx', 'filepath_2.aspx', ..., 'filepath_n.aspx'},
    ...
    '.file_type': {'filepath_1.file_type', 'filepath_2.file_type', ...,
↪'filepath_n.file_type'},
}
```

Automatically removes the * wildcard from `file_type`.

Returns None

class filehandler.**FileConverter**(*file_path=''*)

Converts text files to strings.

On initialization checks the existence of `file_path`.

Example:

```
>>> from os import getcwd, chdir, path
>>> # Valid file_path
>>> current_dir = getcwd()
>>> chdir('..')
>>> file_path = path.join(current_dir, 'examplesite', 'index.html')
>>> chdir(current_dir)     # Change it back.
>>> file_converter = FileConverter(file_path=file_path)
>>> file_string = file_converter.get_file_as_string()
>>> #
>>> # Invalid file_path
>>> file_converter = FileConverter(file_path='/not/valid/file.html')
FileNotFoundError: file_path /not/valid/file.html does not exist.
```

get_file_as_string()

Convert the _file to a string and return it.

Returns (*str*) Return the _file as a string.

Example:

```
>>> from os import getcwd, chdir, path
>>> current_dir = getcwd()
>>> chdir('..')
>>> file_path = path.join(current_dir, 'examplesite', 'index.html')
>>> chdir(current_dir)     # Change it back.
>>> file_converter = FileConverter(file_path=file_path)
>>> file_string = file_converter.get_file_as_string()
```

class filehandler.**CSSFile**

A tool for writing and minifying CSS to files.

Reference: stackoverflow.com/questions/273192/in-python-check-if-a-directory-exists-and-create-it-if-necessary#answer-14364249

Parameters:

css_directory (*str*) – File directory where the .css and .min.css output files are stored.

Members:

file_name (*str*) – Defined in blowdrycss_settings.py as `output_file_name`. Default is 'blowdry'.

extension (*str*) – Defined in blowdrycss_settings.py as `output_extension`. Default is '.css'.

Note: The output file is named `file_name + extension` or `file_name + .min + extension`. ex1: blowdry.css or blowdry.min.css ex2: _custom.scss or _custom.min.scss

Example:

```
>>> from os import getcwd, chdir, path
>>> current_dir = getcwd()
>>> chdir('..')
>>> project_directory = path.join(current_dir, 'examplesite')
>>> css_directory = path.join(project_directory, 'css')
>>> chdir(current_dir)     # Change it back.
>>> css_text = '.margin-top-50px { margin-top: 3.125em }'
>>> css_file = CSSFile(
>>>     file_directory=css_directory, file_name='blowdry'
>>> )
>>> css_file.write(css_text=css_text)
>>> css_file.minify(css_text=css_text)
```

write(*css_text=''*)

Output a human readable version of the css file in utf-8 format.

Notes:

- The file is human readable. It is not intended to be human editable as the file is auto-generated.
- Pre-existing files with the same name are overwritten.

Parameters **css_text** (*str*) – Text containing the CSS to be written to the file.

Returns None

Example:

```
>>> css_text = '.margin-top-50px { margin-top: 3.125em }'
>>> css_file = CSSFile()
>>> css_file.write(css_text=css_text)
```

minify(*css_text=''*)

Output a minified version of the css file in utf-8 format.

Definition:

The term minify "in the context of CSS means removing all unnecessary characters, such as spaces, new lines, comments without affecting the functionality of the source code."

Source: https://www.jetbrains.com/phpstorm/help/minifying-css.html

Purpose:

The purpose of minification is to increase web page load speed.
Reducing the size of the CSS file reduces the time spent downloading the CSS file and waiting for the page to load.

Notes:

- The file is minified and not human readable.
- Pre-existing files with the same name are overwritten.
- Uses the cssutils minification tool.

Important:

- `ser.prefs.useMinified()` is a global setting. It must be reset to `ser.prefs.useDefaults()`. Otherwise, minification will continue to occur. This can result in strange behavior especially during unit testing or in code called after this method is called.

Parameters **`css_text`** (*`str`*) – Text containing the CSS to be written to the file.

Returns None

Example:

```
>>> css_text = '.margin-top-50px { margin-top: 3.125em }'
>>> css_file = CSSFile()
>>> css_file.minify(css_text=css_text)
```

class filehandler.**GenericFile**(*file_directory='/home/docs/checkouts/readthedocs.org/user_builds/blowdrycss/checkouts/l*
file_name='', *extension=''*)

A tool for writing extension-independent files.

Reference: stackoverflow.com/questions/273192/in-python-check-if-a-directory-exists-and-create-it-if-necessary#answer-14364249

Parameters:

file_directory (*str*) – File directory where the output files are saved / overwritten.

file_name (*str*) – The name of the output file. Default is 'blowdry'.

extension (*str*) – A file extension that begins with a `.` and only contains `.`, `0-9` or `a-z`.

Notes:

- `file_name` does not include extension because `write_file()` normalizes and appends the extension.
- `extension` is converted to lowercase.

Example:

```
>>> from os import getcwd, path
>>> file_directory = path.join(getcwd())
>>> css_text = '.margin-top-50px { margin-top: 3.125em }'
>>> markdown_file = GenericFile(
>>>     file_directory=file_directory,
>>>     file_name='blowdry',
>>>     extension='.md'
>>> )
>>> text = '# blowdrycss'
>>> markdown_file.write(text=text)
```

write(*text=''*)

Output a human readable version of the file in utf-8 format.

Converts string to bytearray so that no new lines are added to the file. Note: Overwrites any pre-existing files with the same name.

Raises **TypeError** – Raise a TypeError if `text` input is not of type `str`.

Parameters **text** (*unicode or str*) – The text to be written to the file.

Returns None

class filehandler.**FileModificationComparator**

A Comparator that compares the last modified time of blowdry.css with the last modified time of another file.

Returns None

Example

```
>>> import blowdrycss_settings as settings
>>> from blowdrycss.filehandler import FileModificationComparator
>>> file_age_comparator = FileModificationComparator()
>>> print(file_age_comparator.is_newer(file_path=path.join(settings.project_
↪directory, '/index.html'))
```

is_newer (*file_path*)

Detects if `self.file_path` was modified more recently than blowdry.css. If `self.file_path` is newer than blowdry.css or blowdry.min.css it returns True otherwise it returns false.

If blowdry.css or blowdry.min.css do not exist, then the file under comparison is newer.

Parameters **file_path** (*str*) – The full path to a file.

Returns (*bool*) Returns True if modification time of blowdry.css or blowdry.min.css do not exist, or are older i.e. less than the `self.file_path` under consideration.

6.6 cssbuilder

class cssbuilder.**CSSBuilder** (*property_parser=<blowdrycss.classpropertyparser.ClassPropertyParser object>*)

Builds CSS text with the `cssutils.css` module.

Note: Removes invalid classes. A class is invalid for one of the following reasons:

- It is not valid CSS.
- It does not contain a valid `blowdrycss` encoding.

Object initialization process:

- Build CSS property rules
- Add to css_rules, OR remove invalid css_class from class_set.
- Build a CSS stylesheet based on the CSS `css_rules` set.

Parameters: property_parser (*ClassPropertyParser object*) – Contains a class property parser with a populated class_set.

Returns: None

build_selector(*css_class=''*)

Builds a CSS selector by prepending a `'.'` to `css_class`, and appending an optional pseudo item.

Rules

- Always append a `'.'` to `css_class`.
- If a pseudo class is found append `':'` + `pseuedo_class` to `css_class`.
- If a pseudo element is found append `'::'` + `pseudo_element` to `css_class`.

Parameters **css_class** (*str*) – This value may or may not be identical to the property_value.

Returns *str* – The selector with a '.' prepended and an option pseudo item appended.

build_stylesheet()

Builds the stylesheet by adding CSS rules to the CSS stylesheet.

Returns None

get_css_text()

Returns str – Returns CSS text.

6.7 mediaquerybuilder

class mediaquerybuilder.**MediaQueryBuilder**(*property_parser=<blowdrycss.classpropertyparser.ClassPropertyParser object>*)

Builds a set of CSS media queries from valid classes found in `ClassPropertyParser.class_set`.

Takes a set of classes that may or may not contain media query flags.
Mixing breakpoint and scaling syntax is not allowed. Classes that contain mixed syntax like the following: `small-down-s` or `font-size-28-medium-only-s` are invalidated.

Parameters **property_parser** (`ClassPropertyParser`) – ClassPropertyParser object containing `class_set`.

Returns None

Example Usage:

```
>>> import blowdrycss_settings as settings
>>> from classpropertyparser import ClassPropertyParser
>>> class_set = {'bold', 'large-down', 'font-size-25-s'}
>>> # Filter class names. Only keep classes matching the defined class encoding.
>>> property_parser = ClassPropertyParser(class_set=class_set)
>>> class_set = property_parser.class_set.copy()
>>> # Build Media Queries
>>> if settings.media_queries_enabled:
>>>     unassigned_class_set = class_set.difference(property_parser.class_set)
>>>     # Only use unassigned classes
>>>     property_parser.class_set = unassigned_class_set
>>>     property_parser.removed_class_set = set()
>>>     media_query_builder = MediaQueryBuilder(property_parser=property_parser)
>>>     css_text = bytes(media_query_builder.get_css_text(), 'utf-8')
```

```
>>>     print(media_query_builder.property_parser.class_set)
{'large-down', 'font-size-25-s'}
```

get_css_text()

Joins `css_media_queries` together with an empty separator string `''`.

Returns str – Returns all media queries as CSS text.

Example

```
>>> from classpropertyparser import ClassPropertyParser
>>> class_set = {'bold', 'large-down', 'font-size-24-s'}
>>> # Filter class names. Only keep classes matching the defined class␣
↪encoding.
>>> property_parser = ClassPropertyParser(class_set=class_set)
>>> media_query_builder = MediaQueryBuilder(property_parser=property_parser)
>>> print(media_query_builder.get_css_text())
@media only screen and (min-width: 64.0em) {
    .large-down {
        display: none;
    }
}
.font-size-24-s {
    font-size: 24px;
    @media only screen and (max-width: 45.0em) {
        font-size: 21.3px;
    }
    @media only screen and (max-width: 30.0em) {
        font-size: 19.2px;
    }
}
```

6.8 classpropertyparser

Parser for extracting CSS property names, values, and priorities from set of CSS Class Selectors. These class selectors are gathered externally by the `HTMLClassParser()`.

CSS Unit Reference: http://www.w3schools.com/cssref/css_units.asp
CSS Value Reference: http://www.w3.org/TR/CSS21/propidx.html

Parameters:

class_set (*set*) – A set() of potential css properties.

Returns: Object of Type ClassPropertyParser

Examples:

Give this HTML: `<div class="super-5 h-16-i padding-2 margin-10">Hello</div>` The HTMLClassParser() would extract the following `class_set`.

```
>>> class_set = {'super-5', 'h-16-i', 'PADDING-2', 'margin-10', }
>>> property_parser = ClassPropertyParser(class_set)
>>> # Note that 'super-5' was removed.
>>> print(property_parser.class_set)
{'h-16-i', 'padding-2', 'margin-10'}
>>> css_class = list(property_parser.class_set)[0]
>>> print(css_class)
h-16-i
>>> name = property_parser.get_property_name(css_class=css_class)
>>> # Decodes 'h-' to 'height'
>>> print(name)
height
>>> # Could throw a ValueError.
>>> # See cssbuilder.py for an example of how to handle it.
>>> encoded_property_value = property_parser.get_encoded_property_value(
>>>     property_name=name,
>>>     css_class=css_class
>>> )
>>> print(encoded_property_value)
16
>>> priority = property_parser.get_property_priority(css_class=css_class)
>>> print(priority)
IMPORTANT
>>> value = property_parser.get_property_value(
>>>     property_name=name,
>>>     encoded_property_value=encoded_property_value
>>> )
>>> print(value)    # Note the unit conversion from px_to_em.
1em
```

class classpropertyparser.**ClassPropertyParser**(*class_set=set()*)

class_set_to_lowercase()

Converts member variable self.class_set to lowercase.

Returns None

static underscores_valid(*css_class=''*)

Validate underscore usage in a single css_class. In general, underscores are only allowed to designate a decimal point between two numbers.

Rules:

- Strip all whitespace in front and back.
- **Underscores are only valid between digits**
  - `[0-9]_[0-9]` allowed
  - `_35` not allowed
  - `42_` not allowed
  - `bold_px` not allowed
- **If found in the middle of a string it may begin and/or end with -**
  - `1_2-5_75-1_2-5_75` allowed
- **String may start with n to designate negative numbers.**
  - `n5_25cm` allowed.

- **String may not start with –**
  - `-7_2` not allowed.
- **String may not end with –**
  - `5_4-` not allowed

Parameters **`css_class`** (*`str`*) – Accepts a single CSS class extracted from HTML class attribute.

Returns

boolean

True cases:

```
>>> ClassPropertyParser.underscores_valid('6_3')
True
>>> ClassPropertyParser.underscores_valid('2_456em')
True
>>> ClassPropertyParser.underscores_valid('1_2-5_75-1_2-5_75')
True
>>> ClassPropertyParser.underscores_valid('n5_25cm-n6_1cm')
True
```

False cases:

```
>>> ClassPropertyParser.underscores_valid('-7_2')
False
>>> ClassPropertyParser.underscores_valid('5_4-')
False
>>> ClassPropertyParser.underscores_valid('_b')
False
>>> ClassPropertyParser.underscores_valid('b_')
False
>>> ClassPropertyParser.underscores_valid('padding--_2')
False
>>> ClassPropertyParser.underscores_valid('2_rem')
False
>>> ClassPropertyParser.underscores_valid('m_px')
False
>>> ClassPropertyParser.underscores_valid('__')
False
```

`clean_class_set`()

Detect and Remove invalid css classes from class_set Class names must abide by: http://www.w3.org/TR/CSS2/syndata.html#characters

For purposes of this project only a SUBSET of the CSS standard is permissible as follows:

- Encoded classes shall not be `None` or `''`.
- Encoded shall not contain whitespace (handled implicitly by subsequent rules).
- Encoded classes are only allowed to begin with `[a-z]`
- Encoded classes are only allowed to end with `[a-z0-9]`
- Encoded classes are allowed to contain `[_a-z0-9-]` between the first and last characters.
- Underscores are only valid between digits `[0-9]_[0-9]`

Reference: http://stackoverflow.com/questions/1323364/in-python-how-to-check-if-a-string-only-contains-certain-charact

Returns None

static get_property_name(*css_class=''*)

Extract a property name from a given class.

Rules:

- Classes that use css property names or aliases must set a property value.

Valid:

- `font-weight-700` is valid because `700` is a valid `font-weight` value.
- `fw-700` is valid because it `fw-` is a valid alias for `font-weight`
- `bold` is valid because the `bold` alias implies a property name of `font-weight`

Invalid:

- `font-weight` by itself is a property name, but does not include a property value.
- `fw-` by itself is a property alias, but does not include a property value.
- `700` does imply `font-weight`, but is not a valid CSS selector as it may not begin with a number.

Parameters **css_class** (*str*) – A class name containing a property name and value pair, or just a property value from which the property name may be inferred.

Returns (str) – Class returns the property_name OR if unrecognized returns `''`.

static strip_property_name(*property_name=''*, *css_class=''*)

Strip property name from css_class if applicable and return the css_class.

Raises **ValueError** – If either property_name or css_class are empty or only contain whitespace values.

Parameters

- **property_name** (*str*) – Presumed to match a key or value in the `property_alias_dict`.
- **css_class** (*str*) – This value may or may not be identical to the property_value.

Returns (str) – Returns the encoded property value portion of the css_class.

Examples:

```
>>> ClassPropertyParser.strip_property_name('padding', 'padding-1-2-1-2')
'1-2-1-2'
>>> ClassPropertyParser.strip_property_name('font-weight', 'bold')
'bold'
>>> ClassPropertyParser.strip_property_name('', 'padding-1-2-1-2')
ValueError
>>> ClassPropertyParser.strip_property_name('font-weight', '    ')
ValueError
```

static alias_is_abbreviation(*possible_alias=''*)

Detects if the alias is an abbreviation e.g. `fw-` stands for `font-weight-`. Abbreviated aliases are found in `datalibrary.property_alias_dict`.

Parameters **possible_alias** (*str*) – A value that might be an alias.

Returns (bool) – True if possible_alias ends with a dash –.

Examples:

```
>>> ClassPropertyParser.alias_is_abbreviation('fw-')
True
>>> ClassPropertyParser.alias_is_abbreviation('bold')
False
```

get_property_abbreviations (*property_name=''*)

Returns a list of all property abbreviations appearing in `property_alias_dict`.

Raises **KeyError** – If `property_name` is not found in `property_alias_dict`.

Parameters **property_name** –

Returns (list) – A list of all property abbreviations appearing in `property_alias_dict`.

Example:

Assume the following key, value pair occurs in `property_alias_dict`:

```
'background-color':  {'bgc-', 'bg-c-', 'bg-color-', },
```

```
>>> property_parser = ClassPropertyParser()
>>> property_parser.get_property_abbreviations('background-color')
['bgc-', 'bg-c-', 'bg-color-']
>>> property_parser.get_property_abbreviations('invalid_property_name')
KeyError
```

strip_property_abbreviation (*property_name=''*, *css_class=''*)

Strip property abbreviation from css_class if applicable and return the result.

Raises **ValueError** – If either property_name or css_class are empty or only contain whitespace values.

Parameters

- **property_name** (*str*) – Presumed to match a key or value in the `property_alias_dict`
- **css_class** (*str*) – Initially this value may or may not contain the property_name.

Returns (str) – Returns the encoded property value portion of the css_class.

Examples:

```
>>> property_parser = ClassPropertyParser()
>>> property_parser.strip_property_abbreviation('color', 'c-lime')
'lime'
>>> property_parser.strip_property_abbreviation('', 'c-lime')
ValueError
>>> property_parser.strip_property_abbreviation('color', '  ')
ValueError
```

get_encoded_property_value (*property_name=''*, *css_class=''*)

Strip property name or alias abbreviation prefix from front, pseudo item, and property priority designator. Returns the encoded_property_value.

The term encoded_property_value means a property value that represents a css property value, and may or may not contain dashes and underscores.

Raises **ValueError** – If either property_name or css_class are empty or only contain whitespace values.

Parameters

- **property_name** (*str*) – Name of CSS property that matches a key in `property_alias_dict`.
- **css_class** (*str*) – An encoded css class selector that may contain property name, value, and priority designator.

Returns (str) – Returns only the encoded_property_value after the name and priority designator are stripped.

Examples:

```
>>> property_parser = ClassPropertyParser()
>>> property_parser.get_encoded_property_value('font-weight', 'fw-bold-i')
↪      # abbreviated property_name
'bold'
>>> property_parser.get_encoded_property_value('padding', 'padding-1-10-10-5-i
↪')     # standard property_name
'1-10-10-5'
>>> property_parser.get_encoded_property_value('height', 'height-7_25rem-i')
↪      # contains underscores
'7_25rem'
>>> property_parser.get_encoded_property_value('font-style', 'font-style-
↪oblique')  # no priority designator
'oblique'
>>> property_parser.get_encoded_property_value('', 'c-lime')
ValueError
>>> property_parser.get_encoded_property_value('color', '  ')
ValueError
```

static get_property_value(*property_name=''*, *encoded_property_value=''*)

Accepts an encoded_property_value that's been stripped of it's property named and priority Uses CSSPropertyValueParser, and returns a valid css property value or ''.

Usage Note: Invalid CSS values can be returned by this method. CSS validation occurs at a later step.

Raises **ValueError** – If either property_name or css_class are empty or only contain whitespace values.

Parameters

- **property_name** (*str*) – Name of CSS property that matches a key in `property_alias_dict`.
- **encoded_property_value** (*str*) – A property value that may or may not contain dashes and underscores.

Returns (str) – An unvalidated / potential CSS property value.

Examples:

```
>>> property_parser = ClassPropertyParser(set(), False)                #
↪Turn OFF unit conversion.
>>> property_parser.get_property_value('font-weight', 'bold')          #
↪Special case: alias == property value
'bold'
>>> property_parser.get_property_value('padding', '1-10-10-5')         #
↪Multi-value encoding contains dashes.
```

```
'1px 10px 10px 5px'
>>> property_parser.get_property_value('width', '7_25rem')              #␣
↪Single value contains underscores.
'7.25rem'
>>> property_parser.get_property_value('margin', '1ap-10xp-3qp-1mp3')   #␣
↪Invalid CSS returned.
'1a% 10x% 3q% 1mp3'
>>> property_parser.get_property_value('', 'c-lime')
ValueError
>>> property_parser.get_property_value('color', '  ')
ValueError
```

is_important(*css_class=''*)

Tests whether the css_class ends with the importance_designator.

Raises **ValueError** – If the css_class is empty or only contain whitespace values.

Parameters **css_class** (*str*) – An encoded css class selector that may contain property name, value, and priority designator.

Returns (bool) – Returns True if the css_class ends with the importance_designator. Otherwise, returns False.

Examples:

```
>>> property_parser = ClassPropertyParser()
>>> property_parser.is_important('line-through-i')
True
>>> property_parser.is_important('line-through')
False
>>> property_parser.is_important('')
ValueError
>>> property_parser.is_important('  ')
ValueError
```

strip_priority_designator(*css_class=''*)

Removes the priority designator, if necessary.

Raises **ValueError** – If the css_class is empty or only contain whitespace values.

Parameters **css_class** (*str*) – A css_class that may or may not contain a priority designator.

Returns (str) – If necessary, strip priority designator from the end of css_class and return the string. If the importance_designator is not found, then it returns the unchanged css_class.

```
>>> property_parser = ClassPropertyParser()
>>> property_parser.strip_priority_designator('blink-i')
'blink'
>>> property_parser.strip_priority_designator('blink')
'blink'
>>> property_parser.strip_priority_designator('')
ValueError
>>> property_parser.strip_priority_designator('     ')
ValueError
```

get_property_priority(*css_class=''*)

Returns the keyword `'important'` or `''`. These are in the form that `cssutils` understands.

Raises **ValueError** – If the css_class is empty or only contain whitespace values.

Parameters **`css_class`** (*str*) – An encoded css class selector that may contain property name, value, and priority designator.

Returns (str) – Returns the keyword `important` if the property priority is set to important. Otherwise, it returns ''.

```
>>> property_parser = ClassPropertyParser()
>>> property_parser.get_property_priority('text-align-center-i')
'important'
>>> property_parser.get_property_priority('text-align-center')
''
>>> property_parser.get_property_priority('')
ValueError
>>> property_parser.get_property_priority('     ')
ValueError
```

`is_valid_pseudo_format`(*pseudo_item=''*, *css_class=''*)

Returns True if the CSS class contains a properly formatted pseudo class or element, and False otherwise.

Rules

- The `css_class` may end with '-' + pseudo_item. Example prefix: `color-green-i-hover`
- The `css_class` may end with '-' + pseudo_item + '-i'. Example prefix: `color-green-hover-i`
- The `css_class` must be longer than the suffix or suffix + '-i'.

Parameters

- **`pseudo_item`** (*str*) – Either a pseudo class or pseudo element as defined here
- **`css_class`** (*str*) – This value may or may not be identical to the property_value.

Returns *bool* – Returns True if the CSS class contains the pseudo item, and False otherwise.

`set_pseudo_class`(*css_class=''*)

Check the pseudo class set for a match. Sets `pseudo_class` if found. Otherwise, returns ''.

Raises **`ValueError`** – If either property_name or css_class are empty or only contain whitespace values.

Parameters **`css_class`** (*str*) – This value may or may not be identical to the property_value.

Returns None

`set_pseudo_element`(*css_class=''*)

Check the pseudo element set for a match. Sets the `pseudo_element` if found. Otherwise, returns ''.

Raises **`ValueError`** – If either property_name or css_class are empty or only contain whitespace values.

Parameters **`css_class`** (*str*) – This value may or may not be identical to the property_value.

Returns None

`strip_pseudo_item`(*css_class=''*)

Strip property name from css_class if applicable and return the css_class.

Note: This method must be called after `strip_property_name()`.

Raises **`ValueError`** – If either pseudo_item or css_class are empty or only contain whitespace values.

Parameters **css_class** (*str*) – _value.

Returns (str) – Returns the encoded property value portion of the css_class.

Examples:

```
>>> ClassPropertyParser.strip_pseudo_item('hover-padding-1-2-1-2')
'1-2-1-2'
>>> ClassPropertyParser.strip_pseudo_item('before-bold')
'bold'
>>> ClassPropertyParser.strip_pseudo_item('after-1-2-1-2')
ValueError
>>> ClassPropertyParser.strip_pseudo_item('    ')
ValueError
```

6.9 cssvalueparser

class cssvalueparser.**CSSPropertyValueParser**(*property_name=''*)

Accepts a `property_name` and `use_em` unit conversion flag.

Contains multiple parsers and methods that decodes the CSS property_value.

Parameters **property_name** (*str*) – A CSS property name.

Returns None

Attributes:

property_name (*str*) – A CSS property name. Not allowed to be `''` or None.

color_parser (*ColorParser*) – Parses encoded color values.

unit_parser (*UnitParser*) – Parses encoded unit values, and handles unit conversion.

Important note about methods:

These methods are intended to be called in the order they are defined inside the class.

is_built_in(*value=''*)

Checks if the encoded `value` identically matches a value built-in to the CSS standard. Returns True if `value` matches a CSS built-in valued and False if it does not.

Examples include: 'bold', 'italic', 'w-resize', 'arial', etc.

Parameters **value** (*str*) – Encoded CSS property value.

Returns

(*bool*)

- Returns `True` if `value` matches a CSS built-in valued and `False` if it does not.
- The values 'bold', 'italic', 'w-resize', 'arial' all return `True`.
- The values '-bold', 'fw-', 'color-' all return `False`.
- Invalid `self.property_name` also returns `False` (KeyError Case).

Examples:

```
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='font-weight', use_em=True
>>> )
>>> value_parser.is_built_in('bold')
True
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='padding', use_em=True
>>> )
>>> value_parser.is_built_in('7-4-7-4')
False
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='InvalidCSSPropertyName', use_em=True
>>> )
>>> value_parser.is_built_in('italic')
False
```

static replace_dashes(*value=''*)

Remove leading and trailing dashes. Replace internal dashes with spaces. Return the modified value.

– becomes either `''` or `' '`.

Parameters **value** (*str*) – Encoded CSS property value.

Returns (*str*) – Return the value with dashes removed if necessary.

```
>>> # Delete leading dash '-bold' --> 'bold'
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='font-weight', use_em=True
>>> )
>>> value_parser.replace_dashes('-bold')
'bold'
>>> #
>>> # Delete trailing 'white-' --> 'white'
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='color', use_em=True
>>> )
>>> value_parser.replace_dashes('white-')
'white'
>>> #
>>> # Replace internal '1-5-1-5' --> '1 5 1 5'
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='padding', use_em=True
>>> )
>>> value_parser.replace_dashes('1-5-1-5')
'1 5 1 5'
```

static replace_underscore_with_decimal(*value=''*)

Replace underscore with decimal point. Underscores are used to encode a decimal point

`'_'` becomes `'.'`

Parameters **value** (*str*) – Encoded CSS property value.

Returns (*str*) – Return the value with decimal points added if necessary.

Example

```
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='padding', use_em=True
>>> )
```

```
>>> value_parser.replace_underscore_with_decimal('1_32rem')
'1.32rem'
```

static replace_p_with_percent(*value=''*)

Replace `'p'` suffix with `'%'` if found at the end of any substring containing digits.

`'p '` becomes `'%'`

Mind the space

Parameters **value** (*str*) – Encoded CSS property value.

Returns (*str*) – Return the value with percent signs added if necessary.

Example:

```
>>> # Multi-value: '1p 10p 3p 1p' --> '1% 10% 3% 1%'
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='padding', use_em=True
>>> )
>>> value_parser.replace_p_with_percent(value='1p 10p 3p 1p')
'1% 10% 3% 1%'
>>> #
>>> # Single value ' 1p' --> ' 1%'
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='padding', use_em=True
>>> )
>>> value_parser.replace_p_with_percent(value=' 1p')
' 1%'
```

static replace_n_with_minus(*value=''*)

If a space plus the letter `' n'` is immediately followed by a digit replace it with `' -'`. If `n` is the first letter of the string and followed by digits replace it with `-`. The letter `n` is an encoding for a negative sign. Leaves other `n's` unmodified.

`' n2'` becomes `' -2'` Mind the space.

`'n5'` becomes `'-5'`

Parameters **value** (*str*) – Encoded CSS property value.

Returns (*str*) – Return the value with minus signs added if necessary.

Example:

```
>>> # Multi-value: 'n5cm n6cm' --> '-5cm -6cm'
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='padding', use_em=True
>>> )
>>> value_parser.replace_n_with_minus('n5cm n6cm')
'-5cm -6cm'
>>> #
>>> # 'n9in' --> '-9in' (note that the 'n' at the end is not touched)
>>> value_parser.replace_n_with_minus('n9in')
'-9in'
```

decode_property_value(*value=''*)

Decode the encoded property `value` input e.g. 'bold', '1-5-1-5', '1_32rem', '1p-10p-3p-1p', 'n12px',

'n5_25cm-n6_1cm'. Returns parsed, but non-validated CSS property value.

Parameters **value** (*str*) – An encoded CSS property value.

Returns (*str*) – Returns the decoded, but non-validated CSS property value.

Examples:

```
>>> value_parser = CSSPropertyValueParser(
>>>     property_name='padding', use_em=True
>>> )
>>> value_parser.decode_property_value(value='1-5-1-5')
'0.0625em 0.3125em 0.0625em 0.3125em'
>>> value_parser.unit_parser.use_em = False
>>> value_parser.decode_property_value(value='1-5-1-5')
'1px 5px 1px 5px'
```

static property_is_valid(*name=''*, *value=''*, *priority=''*)

Detects if a given property name, value, and priority combination is valid. Returns True if the combination is valid, and false otherwise.

Validation occurs after the property value is decoded.

Parameters

- **name** – CSS property name
- **value** – Decoded CSS property value
- **priority** – CSS priority designator

Returns (*bool*) – Returns True if the CSS property name, value, and priority combination is valid, and false otherwise.

Examples:

```
>>> value_parser = CSSPropertyValueParser()
>>> value_parser.property_is_valid(
>>>     name='padding', value='1px', priority=''
>>> )
True
>>> value_parser.property_is_valid(
>>>     name='padding', value='invalid', priority=''
>>> )
False
```

6.10 colorparser

Features:

- Validates whether the property_name allows a color property to be set.
- Decodes the following color formats: hexidecimal, rgb, rgba, hsl, hsla.

Exception: English color names are handled separately

Either in the property_alias_dict under the key `color`,
Or they are passed through to cssutils because they are valid CSS and do not require further processing.

Note: The shorthand properties `background`, `border`, and `outline` are supported (as of November 2015).

Assumption: It is assumed that all dashes are removed from the input `value` prior to using this parser.

Example:

```
>>> color_parser = ColorParser('border-color', 'h0df48a')
>>> color_parser.property_name_allows_color()
True
```

class colorparser.**ColorParser**(*property_name=''*, *value=''*)

Extracts plain text, hexidecimal, rgb, rgba, hsl, and hsla color codes from encoded class selectors.

property_name_allows_color()

Detects if the `property_name` allows a color property value.

Reference: http://www.w3.org/TR/CSS21/propidx.html

Returns (bool) – Returns True if the `property_name` is allow to contain colors. Otherwise, it returns False.

Examples:

```
>>> color_parser = ColorParser('border-color', 'h0df48a')
>>> color_parser.property_name_allows_color()
True
>>> color_parser.property_name = 'invalid'
>>> color_parser.property_name_allows_color()
False
```

find_h_index(*value=''*)

Detects if the `value` is a valid hexidecimal encoding.

Note: Supports shorthand properties.

Parameters **value** (*str*) – Expects a `value` of the form: h0ff48f or hfaf i.e. 'h' + a 3 or 6 digit hexidecimal value 0-f.

Returns (int or NoneType) – Returns the index of the `h` to be replaced in the `value` if it matches the hex regex. Otherwise it returns None.

Examples:

```
>>> color_parser = ColorParser()
>>> color_parser.find_h_index(value='h0df48a')
0
>>> color_parser.find_h_index(value='h1invalid')
None
```

replace_h_with_hash(*value=''*)

Replaces the prepended `h` prefix with a hash sign # or octothorpe if you prefer long words.

Includes an internal check to ensure that the value is a valid hexidecimal encoding.

Only replaces the `h` that matches the regex as other `h` characters may be present.

Shorthand properties are supported:

border case: `1px solid hddd` becomes `1px solid #ddd`

Parameters **value** – Encoded hexidecimal value of the form `hf1f` or `hc2c2c2`.

Returns (str) – Returns actually #0ff48f and #faf in the valid case. Returns the input `value` unchanged for the invalid case.

```
>>> color_parser = ColorParser()
>>> # Valid Cases
>>> color_parser.replace_h_with_hash('h0ff24f')
#0ff24f
>>> color_parser.replace_h_with_hash('hf4f')
#f4f
>>> # Valid multiple 'h' case.
>>> color_parser.replace_h_with_hash('13px dashed hd0d')
13px dashed #d0d
>>> # Invalid Cases
>>> color_parser.replace_h_with_hash('bold')
bold
>>> color_parser.replace_h_with_hash('he2z')
he2z
```

add_color_parenthetical(*value=''*)

Convert parenthetical color values: rbg, rbga, hsl, hsla to valid css format

Assumes that color conversion happens after dashes, decimal point, negative signs, and percentage signs are converted.

Note: Currently not compatible with shorthand properties.

Parameters **value** (*str*) – Space delimited rbg, rbga, hsl, hsla values.

Returns (str) – Returns the valid css color parenthetical. Returns the input `value` unchanged for the non-matching case.

Examples:

```
>>> color_parser = ColorParser('color', '')
>>> color_parser.add_color_parenthetical('rgb 0 255 0')
rgb(0, 255, 0)
>>> color_parser.add_color_parenthetical('rgba 255 0 0 0.5')
rgba(255, 0, 0, 0.5)
>>> color_parser.add_color_parenthetical('hsl 120 60% 70%')
hsl(120, 60%, 70%)
>>> color_parser.add_color_parenthetical('hsla 120 60% 70% 0.3')
hsla(120, 60%, 70%, 0.3)
>>> # Pass-through case as no conversion is possible.
>>> color_parser.add_color_parenthetical('hsla')
hsla
```

6.11 fontparser

class fontparser.**FontParser**(*font_value=''*)

Features:

- Parses unquoted font families.

  Unquoted Font-Family References:

http://www.cssfontstack.com/
https://mathiasbynens.be/notes/unquoted-font-family

- **Holds a basic `font_families_dict` (could be extended as desired):**

  Keys: `font-family` category names
  Values: `font-family` member names

- Can generate web safe fallback fonts.

Assumes that the property_name is `font-family`. It does not handle the shorthand property_name `font`

Examples:

```
>>> font_parser = FontParser('papyrus')
>>> font_parser.generate_fallback_fonts()
'papyrus, fantasy'
```

`generate_fallback_fonts`()

Generates web safe fallback fonts

Reference: http://www.w3schools.com/cssref/css_websafe_fonts.asp

Returns (str) – Returns a web safe fallback font string.

Examples:

```
>>> font_parser = FontParser('arial')
>>> font_parser.generate_fallback_fonts()
'arial, sans-serif'
>>> font_parser.font_value = 'monospace'
'monospace'
>>> font_parser.font_value = 'invalid'
''
```

6.12 unitparser

class unitparser.**UnitParser**(*property_name=''*)

Used in these cases:

- No units are provided and default units need to be added to make it valid css.
- The user wants their pixel (px) based units to be converted to em or root em (rem) so that their page scales / zooms properly.

Assumption: The value provided already has negative signs and decimal points. There are no dashes or underscores present in the value e.g. -1.25 can be processed, but n1_25 cannot be processed.

Contains a ‘‘default_property_units_dict‘‘ which maps property names to their default units.

Note: Shorthand properties are not supported.

Why do I want to use em (named after the sound for the letter ‘M’) or root em (rem)?:

Because your webpage will scale with browser and device size.

What does (em) actually stand for?: Source: W3C – http://www.w3.org/WAI/GL/css2em.htm

The foremost tool for writing scalable style sheets is the "em" unit, and it therefore goes on top of the list of guidelines that we will compile throughout this chapter: use ems to make scalable style sheets. Named after the letter "M", the em unit has a long-standing tradition in typography where it has been used to measure horizontal widths. ... In CSS, the em unit is a general unit for measuring lengths, for example page margins and padding around elements. You can use it both horizontally and vertically, and this shocks traditional typographers who always have used em exclusively for horizontal measurements. By extending the em unit to also work vertically, it has become a very powerful unit - so powerful that you seldom have to use other length units.

Source: Wikipedia – https://en.wikipedia.org/wiki/Em_%28typography%29

An em is a unit in the field of typography, equal to the currently specified point size. For example, one em in a 16-point typeface is 16 points. Therefore, this unit is the same for all typefaces at a given point size.

default_units()

Returns the default units "if any" for the assigned `self.property_name`.

Returns (*str*) – Returns default units for the assigned `self.property_name` if they exist. Otherwise, return an empty string `''`.

add_units(*property_value=''*)

If the property_name requires units, then apply the default units defined in default_property_units_dict.

Rules:

- If use_em is False apply the default units for the property name by looking it up in default_property_units_dict.
- Unit that have default units of `px` are converted to `em` if use_em is True.
- If `property_value` has multiple property values, then split it apart.
- If the value already has units, then pass it through unchanged.
- The value provided shall possess negative signs and decimal points.
- Mixed units are allowed, but **not recommended**.
- Values shall only contain [] e.g. -1.25 can be processed, but n1_25 cannot be processed.

Parameters **property_value** (*str*) – A string containing one or more space delimited alphanumeric characters.

Returns (str) – Returns the property value with the default or converted units added.

```
>>> # Convert 'px' to 'em'
>>> unit_parser = UnitParser(property_name='padding', use_em=True)
>>> unit_parser.add_units('1 2 1 2')
0.0625em 0.125em 0.0625em 0.125em
>>> # Use default units
>>> unit_parser.use_em = False
>>> unit_parser.add_units('1 2 1 2')
1px 2px 1px 2px
>>> # Values already have units or are not parsable pass through
>>> # True produces the same output.
>>> unit_parser.use_em = False
>>> unit_parser.add_units('55zp')
55zp
>>> unit_parser.add_units('17rem')
17rem
```

```
>>> # Unitless ``property_name``
>>> # causes ``property_value`` to pass through.
>>> unit_parser.property_name = 'font-weight'
>>> unit_parser.add_units('200')
200
>>> # Mixed units cases - Not a Recommended Practice,
>>> # but represent valid CSS. Be careful.
>>> unit_parser.use_em = False
>>> unit_parser.add_units('5em 6 5em 6')
5em 6px 5em 6px
>>> unit_parser.use_em = True
>>> unit_parser.add_units('1em 100 4cm 9rem')
1em 6.25em 4cm 9rem
```

6.13 breakpointparser

class breakpointparser.**BreakpointParser**(*css_class=", css_property=cssutils.css.Property(name=", value=", priority=")*)

Enables powerful responsive @media query generation via screen size suffixes.

Standard screen breakpoints xxsmall through xgiant:

- `'name--breakpoint_values--limit_key'` – General Format
- `'inline-small-only'` – Only displays the HTML element inline for screen sizes less than or equal to the upper limit_key for `small` screen sizes.
- `'green-medium-up'` – Set `color` to green for screen sizes greater than or equal to the lower limit_key for `medium` size screens.

Custom user defined breakpoint limit_key.

- `'block-480px-down'` – Only displays the HTML element as a block for screen sizes less than or equal to 480px.
- `'bold-624-up'` – Set the `font-weight` to `bold` for screen sizes greater than or equal to 624px.
  - **Note:** If unit conversion is enabled i.e. `use_em` is `True`, then 624px would be converted to 39em.

Important Note about cssutils and media queries

Currently, `cssutils` does not support parsing media queries. Therefore, media queries need to be built, minified, and appended separately.

Parameters

- **css_class** (*str*) – Potentially encoded css class that may or may not be parsable. May not be empty or None.
- **css_property** (*Property*) – Valid CSS Property as defined by `cssutils.css.Property`.

Returns None

Examples:

```
>>> from cssutils.css import Property
>>> from xml.dom import SyntaxErr
>>> # value='inherit' since we do not know if the class is valid yet.
>>> name = 'display'
>>> value = 'inherit'
>>> priority = ''
>>> inherit_property = Property(name=name, value=value, priority=priority)
>>> breakpoint_parser = BreakpointParser(
        css_class='large-up',
        css_property=inherit_property
    )
>>> print(breakpoint_parser.breakpoint_key)
large
>>> print(breakpoint_parser.limit_key)
-up
>>> # Validate encoded syntax.
>>> is_breakpoint = breakpoint_parser.is_breakpoint
>>> if is_breakpoint:
>>>     clean_css_class = breakpoint_parser.strip_breakpoint_limit()
>>>     # Change value to 'none' as display media queries use reverse logic.
>>>     value = 'none'
>>> # Build CSS Property
>>> try:
>>>     css_property = Property(name=name, value=value, priority=priority)
>>>     if css_property.valid:
>>>         if is_breakpoint and breakpoint_parser:
>>>             breakpoint_parser.css_property = css_property
>>>             media_query = breakpoint_parser.build_media_query()
>>>     else:
>>>         print(' (cssutils invalid property value: ' + value + ')')
>>> except SyntaxErr:
>>>     print('(cssutils SyntaxErr invalid property value: ' + value + ')')
>>> print(media_query)
@media only screen and (max-width: 45.0625em) {
    .large-up {
        display: none;
    }
}
```

set_breakpoint_key()

If `self.css_class` contains one of the keys in `self.breakpoint_dict`, then set `self.breakpoint_values` to a breakpoint_values tuple for the matching key. Otherwise, set `is_breakpoint = False`.

Rules:

- Before a comparison is made each key is wrapped in dashes i.e. `-key-` since the key must appear in the middle of a `self.css_class`.
  - This also prevents false positives since searching for `small` could match `xxsmall`, `xsmall`, and `small`.
- The length of `self.css_class` must be greater than the length of the key + 2. This prevents a `css_class` like `'-xsmall-'` or `'-xxlarge-up'` from being accepted as valid by themselves.
- The `key` minus the preceding dash is allowed if it the key is the first word in the string. This allows the shorthand cases, for example: `small-only`, `medium-up`, `giant-down`. These cases imply that the CSS property name is `display`.

- These rules do not catch all cases, and prior validation of the css_class is assumed.
- **Set `is_breakpoint = False` if none of the keys in `self.breakpoint_dict` are found in** `self.css_class`.

  Returns None

Examples:

```
>>> from cssutils.css import Property
>>> # value='inherit' since we do not know if the class is valid yet.
>>> name = 'display'
>>> value = 'inherit'
>>> priority = ''
>>> inherit_property = Property(name=name, value=value, priority=priority)
>>> breakpoint_parser = BreakpointParser(
        css_class='padding-1em-giant-down',
        css_property=inherit_property
    )
>>> # BreakpointParser() sets breakpoint_key.
>>> print(breakpoint_parser.breakpoint_key)
giant
```

`set_limit_key`()

If one of the values in `self.limit_set` is contained in `self.css_class`, then Set `self.limit_key` to the value of the string found. Otherwise, set `is_breakpoint = False`.

Rules:

- The `limit_key` is expected to begin with a dash `-`.
- The `limit_key` may appear in the middle of `self.css_class` e.g. `'padding-10-small-up-s-i'`.
- The `limit_key` may appear at the end of `self.css_class` e.g. `'margin-20-giant-down'`.
- The length of `self.css_class` must be greater than the length of the limit_key + 2. This prevents a `css_class` like `'-up-'` or `'-only-'` from being accepted as valid by themselves.
- These rules do not catch all cases, and prior validation of the css_class is assumed.
- **Set `is_breakpoint = False` if none of the members of `self.limit_set` are found in** `self.css_class`.

  Returns None

Examples:

```
>>> from cssutils.css import Property
>>> # value='inherit' since we do not know if the class is valid yet.
>>> name = 'display'
>>> value = 'inherit'
>>> priority = ''
>>> inherit_property = Property(name=name, value=value, priority=priority)
>>> breakpoint_parser = BreakpointParser(
        css_class='padding-1em-giant-down',
        css_property=inherit_property
    )
>>> # BreakpointParser() sets limit_key.
```

```
>>> print(breakpoint_parser.limit_key)
-down
```

set_custom_breakpoint_key()

Assuming that a limit key is found, but a standard breakpoint key is not found in the `css_class`; determine if a properly formatted custom breakpoint is defined.

Custom Breakpoint Rules:

- Must begin with an integer.
- May contain underscores _ to represent decimal points.
- May end with any allowed unit (em|ex|px|in|cm|mm|pt|pc|q|ch|rem|vw|vh|vmin|vmax).
- Must not be negative.
- Unit conversion is based on the related setting in blowdrycss_settings.py.

Pattern Explained

- `pattern = r'[a-zA-Z].*\-([0-9]*_?[0-9]*?(em|ex|px|in|cm|mm|pt|pc|q|ch|rem|vw|vh)\-(up|down)\-?'`
- `[a-zA-Z]` – css_class must begin with a letter.
- `.*` – First letter may be followed by any number of characters.
- `\-` – A dash will appear before the substring pattern.
- `([0-9]*_?[0-9]*?(em|ex|px|in|cm|mm|pt|pc|q|ch|rem|vw|vh|vmin|vmax)?)` – Substring pattern begins with A number that could contain an _ to encode an optional decimal point followed by more numbers. Followed by an optional unit of measure.
- `\-(up|down)` – Substring pattern must end with either `-up` or `-down`.
- `\-?` – Substring pattern may or may not be followed by a dash since it could be the end of a string or

Returns None

Examples:

padding-25-820-up, display-480-down, margin-5-2-5-2-1000-up, display-960-up-i, display-3_2rem-down

```
>>> from cssutils.css import Property
>>> # value='inherit' since we do not know if the class is valid yet.
>>> name = 'display'
>>> value = 'inherit'
>>> priority = ''
>>> inherit_property = Property(name=name, value=value, priority=priority)
>>> breakpoint_parser = BreakpointParser(
        css_class='padding-25-820-up',
        css_property=inherit_property
    )
>>> # BreakpointParser() sets limit_key.
>>> print(breakpoint_parser.is_breakpoint)
True
>>> print(breakpoint_parser.limit_key)
'-up'
>>> print(breakpoint_parser.breakpoint_dict[breakpoint_parser.breakpoint_
↪key][breakpoint_parser.limit_key])
'51.25em'
```

strip_breakpoint_limit()

Removes breakpoint and limit keywords from `css_class`.

Rules:

- **Return `''` if breakpoint limit key pair == `css_class` i.e. implied `display` property name.**
  - `'xlarge-only'` becomes `''`.
- **Return `property_name + property_value - breakpoint_key - limit_key`.**
  - `'bold-large-up'` becomes `'bold'`.
  - `'padding-12-small-down'` becomes `'padding-12'`.
  - `'margin-5-2-5-2-1000-up'` becomes `'margin-5-2-5-2'` (custom breakpoint case).

Returns (*str*) – Returns a modified version `css_class` with breakpoint and limit key syntax removed.

Examples:

```
>>> from cssutils.css import Property
>>> # value='inherit' since we do not know if the class is valid yet.
>>> name = 'display'
>>> value = 'inherit'
>>> priority = ''
>>> inherit_property = Property(name=name, value=value, priority=priority)
>>> breakpoint_parser = BreakpointParser(
        css_class='xlarge-only',
        css_property=inherit_property
    )
>>> breakpoint_parser.strip_breakpoint_limit()
''
>>> inherit_property = Property(name='font-weight', value=value,␣
↪priority=priority)
>>> breakpoint_parser.css_class='bold-large-up'
>>> breakpoint_parser.strip_breakpoint_limit()
'bold'
```

is_display()

Tests if `css_class` contains character patterns that match the special case when the property name is `display`.

Returns (*bool*) – Returns true if one of the cases is `true`. Otherwise it returns `false`.

Examples:

```
>>> from cssutils.css import Property
>>> # value='inherit' since we do not know if the class is valid yet.
>>> name = 'display'
>>> value = 'inherit'
>>> priority = ''
>>> inherit_property = Property(name=name, value=value, priority=priority)
>>> breakpoint_parser = BreakpointParser(
        css_class='xlarge-only',
        css_property=inherit_property
    )
>>> breakpoint_parser.strip_breakpoint_limit()
''
```

```
>>> inherit_property = Property(name='font-weight', value=value,␣
↪priority=priority)
>>> breakpoint_parser.css_class='bold-large-up'
>>> breakpoint_parser.strip_breakpoint_limit()
'bold'
```

`css_for_only`()

Generates css

Handle Cases:

- **Special Usage with `display`**
  - The css_class `display-large-only` is a special case. The CSS property name `display` without a value is used to show/hide content. For `display` reverse logic is used. The reason for this special handling of `display` is that we do not know what the current `display` setting is if any. This implies that the only safe way to handle it is by setting `display` to `none` for everything outside of the desired breakpoint limit.
  - Shorthand cases, for example: `small-only`, `medium-up`, `giant-down` are allowed. These cases imply that the CSS property name is `display`. This is handled in the `if-statement` via `pair[1:] == self.css_class`.
  - *Note:* `display + value + pair` is handled under the General Usage case. For example, `display-inline-large-only` contains a value for `display` and only used to alter the way an element is displayed.
- General Usage
  - The css_class `padding-100-large-down` applies `padding: 100px` for screen sizes less than the lower limit of `large`.

Note: Unit conversions for pixel-based `self.value` is required **before** the BreakpointParser is instantiated.

Media Query Examples

- *Special Case:* Generated CSS for `display-large-only` or `large-only`:

```
@media only screen and (max-width: 45.0625em) {
    .display-large-only {
        display: none;
    }
}

@media only screen and (min-width: 64.0em) {
    .display-large-only {
        display: none;
    }
}
```

- *General Usage Case:* Generated CSS for `padding-100-large-only`:

```
@media only screen and (min-width: 45.0625em) and (max-width: 64.0em) {
    .padding-100-large-only {
        padding: 100px;
    }
}
```

- *Priority !important Case:* Generated CSS for `large-only-i`:

```
@media only screen and (max-width: 45.0625em) {
    .display-large-only {
        display: none !important;
    }
}

@media only screen and (min-width: 64.0em) {
    .display-large-only {
        display: none !important;
    }
}
```

Returns None

`css_for_down`()

Only display the element, or apply a property rule `below` a given screen breakpoint. Returns the generated css.

Handle Cases:

- **Special Usage with `display`**
  - The css_class `display-medium-down` is a special case. The CSS property name `display` without a value is used to show/hide content. For `display` reverse logic is used. The reason for this special handling of `display` is that we do not know what the current `display` setting is if any. This implies that the only safe way to handle it is by setting `display` to `none` for everything outside of the desired breakpoint limit.
  - Shorthand cases, for example: `small-only`, `medium-up`, `giant-down` are allowed. These cases imply that the CSS property name is `display`. This is handled in the `if-statement` via `pair[1:] == self.css_class`.
  - *Note:* `display + value + breakpoint + limit` is handled under the General Usage case. For example, `display-inline-medium-down` contains a value for `display` and only used to alter the way an element is displayed.
- General Usage
  - The css_class `padding-100-medium-down` applies `padding: 100px` for screen sizes less than the lower limit of `medium`.

Note: Unit conversions for pixel-based `self.value` is required **before** the BreakpointParser is instantiated.

Media Query Examples

- *Special Case:* Generated CSS for `display-medium-down`:

```
@media only screen and (min-width: 45.0em) {
    .display-medium-down {
        display: none;
    }
}
```

- *General Usage Case:* Generated CSS for `padding-100-medium-down`:

```
@media only screen and (max-width: 45.0em) {
    .padding-100-medium-down {
        padding: 100px;
```

```
    }
}
```

Returns None

`css_for_up`()

Only display the element, or apply a property rule `above` a given screen breakpoint. Returns the generated css.

Handle Cases:

- **Special Usage with `display`**
  - The css_class `display-small-up` is a special case. The CSS property name `display` without a value is used to show/hide content. For `display` reverse logic is used. The reason for this special handling of `display` is that we do not know what the current `display` setting is if any. This implies that the only safe way to handle it is by setting `display` to `none` for everything outside of the desired breakpoint limit.
  - Shorthand cases, for example: `small-only`, `medium-up`, `giant-down` are allowed. These cases imply that the CSS property name is `display`. This is handled in the `if-statement` via `pair[1:] == self.css_class`.
  - *Note:* `display + value + breakpoint + limit` is handled under the General Usage case. For example, `display-inline-small-up` contains a value for `display` and only used to alter the way an element is displayed.
- General Usage
  - The css_class `padding-100-small-up` applies `padding: 100px` for screen sizes less than the lower limit of `small`.

Note: Unit conversions for pixel-based `self.value` is required **before** the BreakpointParser is instantiated.

Media Query Examples

- *Special Case:* Generated CSS for `display-small-up`:

```
@media only screen and (max-width: 15.0625em) {
    .display-small-up {
        display: none;
    }
}
```

- *General Usage Case:* Generated CSS for `padding-100-small-up`:

```
@media only screen and (min-width: 15.0625em) {
    .padding-100-small-up {
        padding: 100px;
    }
}
```

Returns None

`build_media_query`()

Pick the css generation method based on the `limit_key` found in `css_class`.

Returns Return CSS media queries as CSS Text.

Examples:

```
>>> from cssutils.css import Property
>>> # value='inherit' since we do not know if the class is valid yet.
>>> name = 'display'
>>> value = 'inherit'
>>> priority = ''
>>> inherit_property = Property(name=name, value=value, priority=priority)
>>> breakpoint_parser = BreakpointParser(
        css_class='padding-1em-giant-down',
        css_property=inherit_property
    )
>>> css_property = Property(name='padding', value='1em', priority='')
>>> if breakpoint_parser.is_breakpoint and css_property.valid:
>>>     breakpoint_parser.css_property = css_property
>>>     media_query = breakpoint_parser.build_media_query()
>>> print(media_query)
@media only screen and (max-width: 160.0em) {
    .padding-1em-giant-down {
        padding: 1em;
    }
}
```

6.14 scalingparser

class scalingparser.**ScalingParser**(*css_class=”, css_property=cssutils.css.Property(name=”, value=”, priority=”)*)

Enables powerful responsive @media query generation via screen size suffixes.

Scaling Flag:

Append `'-s'` to the end of an encoded property values to scale the value up and down based on screen size.

Note: This only works on property values containing distance–based units (pixels, em, etc).

- General format: `<name>-<value>-s`
- Specific case: `font-size-24-s`
- Priority `!important` case: `font-size-24-s-i`
  - (`'-i'` *is always last*)

Responsive Scaling Ratios:

- Assuming `font-size-24-s` is the encoded css class, the font-size will respond to the screen size according to the following table:

Screen Size	Trigger Range	Scaling Factor	px	em
XLarge	> 1024 or 64.0em	1	24	1.5
Large	> 720px & <= 1024	1.043	23.0	1.438
Medium	< 720px or 45.0em	1.125	21.3	1.333
Small	< 480px or 30.0em	1.25	19.2	1.2

Important Note about cssutils

Currently, `cssutils` does not support parsing media queries. Therefore, media queries need to be built, minified, and appended separately.

Parameters

- **`css_class`** (*`str`*) – Potentially encoded css class that may or may not be parsable. May not be empty or None.
- **`css_property`** (*`Property()`*) – Valid CSS Property as defined by `cssutils.css.Property`.

Returns None

Examples:

```
>>> scaling_parser = ScalingParser(css_class='font-weight-24-s')
```

`strip_scaling_flag`()

Remove the `scaling_flag` from `css_class` if possible and return the clean css class. Otherwise, return the `css_class` unchanged.

Rules

- Remove `-s` if found at end of a string
- Remove `-s` if `-s-i` is found at the end of the string.

Returns (*str*) – If the `css_class` is scaling remove the `scaling_flag` and return the clean css class. Otherwise, return the `css_class` unchanged.

Examples:

```
>>> scaling_parser = ScalingParser(css_class='font-size-32-s', name='font-size
↪')
>>> scaling_parser.strip_scaling_flag()
font-size-32
>>> scaling_parser.css_class = 'font-size-56-s-i'
>>> scaling_parser.strip_scaling_flag()
font-size-56-i
>>> scaling_parser.css_class = 'font-size-14'
>>> scaling_parser.strip_scaling_flag()
font-size-14
```

`build_media_query`()

Returns CSS media queries that scales pixel / em values in response to screen size changes.

Generated CSS for ‘‘font-size-24-s‘‘ minus the inline comments & line breaks:

```
// Default size above medium
.font-size-24-s { font-size: 24px; }

// medium screen font size reduction
@media only screen and (max-width: 64.0em) {
    .font-size-24-s { font-size: 23.0px; }
}

// medium screen font size reduction
@media only screen and (max-width: 45.0em) {
    .font-size-24-s { font-size: 21.3px; }
}
```

```
// small screen font size reduction
@media only screen and (max-width: 30.0em) {
    .font-size-24-s { font-size: 19.2px; }
}
```

Priority !important – Generated CSS for ‘‘font-size-24-s-i‘‘ minus the inline comments & line breaks:

```
// Default size above the maximum 'medium' width breakpoint.
.font-size-24-s-i { font-size: 24px !important; }

// medium screen font size reduction
@media only screen and (max-width: 64.0em) {
    .font-size-24-s-i { font-size: 23.0px !important; }
}

// Apply 'medium' screen font size reduction.
@media only screen and (max-width: 45.0em) {
    .font-size-24-s-i { font-size: 21.3px !important; }
}

// Apply 'small' screen font size reduction.
@media only screen and (max-width: 30.0em) {
    .font-size-24-s-i { font-size: 19.2px !important; }
}
```

Returns (*str*) – Returns CSS media queries that scales pixel / em values in response to screen size changes.

6.15 timing

Simple code performance timer that allows for the execution time to be recorded

Credit:

- This is a modified version of Paul’s and Nicojo’s answers on stackoverflow.
- Reference: http://stackoverflow.com/questions/1557571/how-to-get-time-of-a-python-program-execution

Usage Case:

```
>>> # At the beginning of the chunk of code to be timed.
>>> from blowdrycss.timing import Timer
>>> timer = Timer()
>>> timer.report()
Completed @ 2015-12-14 16:56:08.665080
========================================
It took: 0.17296 seconds
========================================
```

class timing.**Timer**

A performance Timer that reports the amount of time it took to run a block of code.

Parameters:

start (*time*) – Time that the program started.

end (*time*) – Time that the program ended.

Returns None

Example

```
>>> from blowdrycss.timing import Timer
>>> timer = Timer()
>>> timer.report()
Completed 2015-12-14 16:56:08.665080
=====================================
It took: 0.17296 seconds
=====================================
>>> timer.reset()       # Resets start time to now.
>>> timer.report()
Completed 2015-12-14 17:05:12.164030
=====================================
It took: 1.45249 seconds
=====================================
```

static seconds_to_string(*seconds_elapsed=0.0*)

Converts the amount of time elapsed to seconds_elapsed, and returns it as a string.

Parameters **seconds_elapsed** (*float*) – A time() value in units of seconds_elapsed.

Returns (*str*) – Returns a string version of the total time elapsed in seconds_elapsed.

elapsed

Calculates the amount of time elapsed (delta T) by subtracting start `time()` from end `time()`.

Math: elapsed = delta T = end - start

Returns (*str*) – Returns delta T in units of seconds as a string.

print_time()

Prints temporal metadata to the console. Including the completion timestamp and delta T in seconds.

Returns None

report()

Sets `end` time and prints the time elapsed (delta T). Calls `print_time()`, and prints temporal metadata.

Returns None

class timing.**LimitTimer**

Timer governs when to perform a full and comprehensive run of blowdry.parse().

Note: This is independent of file modification watchdog triggers which only scan the file(s) that changed since the last run.

** Why is a LimitTimer needed? **

Understanding the Truth Table

1. The project only contains two files: File 1 and File 2.
2. Each file either contains the CSS class selector 'blue' or not i.e. set().
3. File 2 is modified. Either the class `blue` is added or removed i.e. set().
4. X means don't care whether the file contains blue or set().
5. Case #3 is the reason why the LimitTimer is required. The css class selector `blue` was only defined in File 2. Then blue was removed from File 2. Since blue existed in the combined class_set before File 2 was modified, it will remain in the combined class_set after the union with set(). This is undesirable in Case #3 since `blue` is not used anymore in either of the two files. The LimitTimer runs periodically to clear these unused selectors.

Case #	File 1 class_set	File 2 class_set	Combined class_set	File 2 modified	Combined class_set
1	blue	blue	blue	set()	blue
2	blue	set()	blue	X	blue
3	set()	blue	blue	set()	blue
4	set()	set()	set()	blue	blue
5	set()	set()	set()	set()	set()

** Another reason why the LimitTimer is needed. **

On windows and mac watchdog on_modify event gets triggered twice on save. In order to prevent a duplicate run for the same change or set of changes this class is implemented. It can also depend on the IDE being used since some IDEs auto-save.

Members:

time_limit (*str*) – Number of seconds that must pass before the limit is exceeded. Default is settings.time_limit.

start_time (*str*) – Time that the timer started.

Returns None

Example

```
>>> from blowdrycss.timing import LimitTimer
>>> limit_timer = LimitTimer()
>>> if limit_timer.limit_exceeded:
>>>     print("30 minutes elapses.")
>>>     limit_timer.reset()
```

`time_limit`

Getter returns `_time_limit`.

Returns (*int*) – Returns `_time_limit`.

`limit_exceeded`

Compares the current time to the start time, and returns True if `self.time_limit` is exceeded and False otherwise.

Returns (*bool*) – Returns True if `self.time_limit` is exceeded and False otherwise.

reset()

Resets `self.start` to the current time.

Returns None

6.16 utilities

utilities.**contains_a_digit**(*string=''*)

Check if string contains a digit `[0-9]`.

Parameters **string** (*str*) – The string to test.

Returns (bool) – Returns True if string contains at least 1 digit. Otherwise, returns False.

Examples:

```
>>> contains_a_digit('abc1')
True
>>> contains_a_digit('876')
True
>>> contains_a_digit('cat')
False
>>> contains_a_digit('')
False
>>> contains_a_digit('   ')
False
```

utilities.**deny_empty_or_whitespace**(*string=''*, *variable_name=''*)

Prevent `string` or `variable_name` from being empty or only containing whitespace.

Raises **ValueError** – Raises a ValueError if the string or the variable_name is empty or only contains whitespace. The ValueError contains the name of the calling function and the variable name used in the calling function.

Parameters

- **string** (*str*) – The string to test.
- **variable_name** (*str*) – The name of the variable used in the calling function.

Returns None

utilities.**get_file_path**(*file_directory=''*, *file_name='blowdry'*, *extension=''*)

Joins the `file_directory`, `file_name`, and `extension`. Returns the joined file path.

Rules:

- Do not allow `''` empty input for `file_directory`, `file_name`, or `extension`.
- Transform extension to lowercase.
- Extensions must match this regex r"(^[.][.0-9a-z]*[0-9a-z]$)".

Findall regex Decoded:

- `r"(^[.][.0-9a-z]*[0-9a-z]$)"`
- `^[.]` – `extension` must begin with a `.` dot.
- `[.0-9a-z]*` – `extension` may contain any of the character inside the brackets.

- `[0-9a-z]$` – `extension` may only end with the characters inside the brackets.

Parameters

- **file_directory** (*str*) – Directory in which to place the file.
- **file_name** (*str*) – Name of the file (excluding extension)
- **extension** (*str*) – A file extension including the `.`, for example, `.css`, `.min.css`, `.md`, `.html`, and `.rst`

Returns (*str*) – Returns the joined file path.

`utilities.validate_output_file_name_setting()`

Validates output_file_name from blowdrycss_settings.py. First thing that runs.

Raises **SyntaxError** – If settings.output_file_name or settings.output_extension contain '', '/', whitespace or ends with a dot.

Returns None

`utilities.validate_output_extension_setting()`

Validates output_extension from blowdrycss_settings.py. First thing that runs.

Raises **SyntaxError** – If settings.output_extension does not begin with a dot or contains '', '/', whitespace or ends with a dot.

Returns None

`utilities.change_settings_for_testing()`

Change settings directories for testing.

> **Warning:** This method should only be used by the unit_test framework.

Returns None

`utilities.unittest_file_path(folder='', filename='')`

Determines the path of assigned to the folder and file based on the directory in which the unittest command is executed.

Parameters

- **folder** (*str*) – Name of the folder where the file is located.
- **filename** (*str*) – Name of the file including extension e.g. test_aspx.aspx

Returns (*str*) – Return the path of the file to test.

`utilities.print_minification_stats(file_name='', extension='')`

Print before and after minification file size reduction statistics.

Parameters

- **file_name** (*str*) – The file name excluding extension e.g. 'blowdry' or 'site'.
- **extension** (*str*) – Appended to the file_name and begins with a dot e.g. '.css', '.scss', etc.

Returns None

`utilities.print_blow_dryer()`

Prints an image of a blow dryer using ASCII.

A nice png to ascii converter

Returns None

`utilities.make_directory(directory='')`

Try to make a directory or verify its' existence. Raises an error if neither of these are possible.

Raises `OSError` – Raises an OSError if the directory cannot be made or found.

Parameters `directory` (*str*) – A directory path in the file system.

Returns None

`utilities.delete_file_paths(file_paths)`

Delete all file_paths. Use Caution.

Note:

```
Ignores files that do not exist.
```

Parameters `file_paths` (*iterable of strings*) – An iterable containing file path strings.

Returns None

6.17 version

CHAPTER 7

License

7.1 MIT License

7.1.1 The MIT License (MIT)

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

I

L

M

P

Q

R

S

T

U

V

W