Package 'cascadess'

November 30, 2020

Type Package

Title A Style Pronoun for 'htmltools' Tags

Version 0.1.0

Description Apply styles to tag elements directly or with the
.style pronoun. Using the pronoun, styles are created within
the context of a tag element. Change borders, background colors,
margins, layouts, and more.

License MIT + file LICENSE

URL https://github.com/nteetor/cascadess

BugReports https://github.com/nteetor/cascadess/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Depends R (>= 3.3)

Imports crayon, htmltools (>= 0.4.0), magrittr, rlang, utils

Suggests rmarkdown, shiny, testthat (>= 2.1.0)

Collate 'align.R' 'utils.R' 'background.R' 'border.R' 'cascadess.R'
'display.R' 'eval-style.R' 'flex.R' 'float.R' 'font.R' 'gap.R'
'height.R' 'margin.R' 'overflow.R' 'padding.R' 'position.R'
'reexports.R' 'responsive.R' 'shadow.R' 'text.R' 'utils-docs.R'
'visible.R' 'width.R' 'zzz.R'

NeedsCompilation no

Author Nathan Teetor [aut, cre],
The Bootstrap Authors [cph] (Bootstrap library),
Twitter, Inc [cph] (Bootstrap library)

Maintainer Nathan Teetor <nate@haufin.ch>

Repository CRAN

Date/Publication 2020-11-30 09:00:08 UTC

R topics documented:

`align` *Inline alignment*

Description

The `align()` function adjusts the inline alignment of an element. This applies only to inline elements and may be used to adjust the vertical alignment of an image in a line of text or the contents of a table cell.

For broader alignment purposes use flex box, see `flex()`.

Usage

```
align(x, vertical)
```

Arguments

`x`	A tag element or .style pronoun.
`vertical`	One of `"baseline"`, `"top"`, `"middle"`, `"bottom"`, `"text-top"`, or `"text-bottom"`.

Examples

```
library(htmltools)

div(
  "Text",
  span("Above") %>% align("top"),
  span("Below") %>% align("bottom")
)
```

`background` *Backgrounds*

Description

The `background()` function adjusts the background color of a tag element.

Usage

```
background(x, color)
```

Arguments

`x`	A tag element or .style pronoun.
`color`	One of `"blue"`, `"indigo"`, `"purple"`, `"red"`, `"orange"`, `"yellow"`, `"green"`, `"teal"`, `"cyan"`, `"white"`, or `"transparent"` specifying the background color of the tag element.

Buttons

Use `background()` to modify shiny's action buttons.

```
actionButton("id", "Take action") %>%
  background("green")
```

Take action

With a couple other functions we can take our improvement a step further.

```
actionButton("id", "Take action") %>%
  background("green") %>%
  border("green") %>%
  shadow("small")
```

Take action

Shiny's download buttons include `...`, so we can use the `.style` pronoun!

```
downloadButton(
  .style %>%
    background("white") %>%
    border("blue"),
  outputId = "dwnld",
  label = "Do a download",
  class = NULL
)
```

Panels

```
sidebarLayout(
  sidebarPanel(
    .style %>%
      background("blue"),
    "It's alive"
  ),
  mainPanel(
    .style %>%
      background("red"),
    "It's panel"
  )
)
```

It’s alive

It’s panel

Colors

blue

indigo

purple

red

orange

yellow

green

teal

cyan

white

transparent

Examples

```
library(htmltools)

div(
  .style %>%
    background("white") %>%
    border("blue") %>%
    text("white"),
  "Nunc porta vulputate tellus.",
  "Suspendisse potenti."
)
```

border *Borders*

Description

The `border()` function adjusts a tag element's borders.

Usage

```
border(x, color, sides = TRUE, width = 1, round = "medium")
```

Arguments

`x`	A tag element or .style pronoun.
`color`	One of `"blue"`, `"indigo"`, `"purple"`, `"red"`, `"orange"`, `"yellow"`, `"green"`, `"teal"`, `"cyan"`, or `"white"` specifying the border color.
`sides`	One or more of `"top"`, `"t"`, `"right"`, `"r"`, `"bottom"`, `"b"`, `"left"`, `"l"`, `"all"`, or `"none"` specifying which sides to add borders to, defaults to `TRUE`. `TRUE` and `FALSE` may be used as shorthand for all sides or no sides, respectively.
`width`	One or more of `1`, `2`, `3`, `4`, or `5` specifying the width of the element's border, defaults to `1`.
`round`	One of `"small"`, `"sm"`, `"medium"`, `"md"`, `"large"`, `"lg"`, or `"none"` specifying how to round the corners of the element, defaults to `"medium"`.

Colors

blue
indigo
purple
red
orange
yellow
green
teal
cyan
white

Round

small

sm

medium

md

large

lg

none

Examples

```

library(htmltools)

h3(
  .style %>%
    border("red", "bottom") %>%
    text("red"),
  "A bright, underlined heading!"
)
```

---

cascadess *Cascadess*

Description

Styles for htmltools tags.

For styles to be applied you must include a call to `cascadess()` in your shiny application or html-tools tags.

Usage

```
cascadess()
```

Examples

```

## Not run:
library(shiny)

shinyApp(
  ui = list(
    cascadess(),
    div(
      .style %>%
        padding(3) %>%
        background("indigo") %>%
        font("light"),
      "Etiam laoreet quam sed arcu."
    )
  ),
  server = function(input, output) {

  }
)

## End(Not run)
```

`display` *Display*

Description

The `display()` function adjusts how a tag element is rendered. For example, to use the flex box layout the display must be `"flex"`.

Usage

```
display(x, type)
```

Arguments

`x`	A tag element or .style pronoun.
`type`	A responsive argument. One of `"inline"`, `"inline-block"`, `"block"`, `"grid"`, `"table"`, `"table-cell"`, `"table-row"`, `"flex"`, `"inline-flex"`, or `"none"`.

Block vs inline

```
div(
  div(
    .style %>%
      border("blue"),
    "block"
  ),
  div(
    .style %>%
      border("blue"),
    "block"
  )
)
```

block

block

```
div(
  div(
    .style %>%
      border("blue") %>%
      display("inline"),
    "inline"
  ),
  div(
    .style %>%
      border("blue") %>%
      display("inline"),
    "inline"
  )
)
```

inline

inline

Examples

```
library(htmltools)

# When using flex make sure you specify the flex display.
div(
  .style %>%
    display("flex") %>%
    flex(justify = "center"),
  "Powerful stuff"
)
```

---

flex *Flex*

Description

The `flex()` function adjusts the flex box layout of an element. To use the flex box layout the element must also use the flex display, see `display()`. The flex box layout is incredibly powerful and allows centering of elements vertically or horizontally, automatic adjustment of space between or around child elements, and more.

Direct child elements of a flex box container are automatically considered flex items and may be adjusted with `item()`.

Usage

```
flex(x, direction = NULL, justify = NULL, align = NULL, wrap = NULL)
```

Arguments

`x`	A tag element or .style pronoun.
`direction`	A responsive argument. One of `"row"` or `"column"` specifying the main axis of flex items, defaults to `NULL`, in which case the argument is ignored. If `"row"`, the main axis is horizontal and items are arranged from left to right. The cross axis is the vertical. If `"column"`, the main axis is vertical and items are arranged from top to bottom. The cross axis is the horizontal.
`justify`	A responsive argument. One of `"start"`, `"end"`, `"center"`, `"between"`, `"around"`, or `"evenly"` specifying how items are arranged on the main axis, defaults to `NULL`, in which case the argument is ignored. If `"between"`, `"around"`, or `"evenly"` then items are arranged by distributing the space available on the main axis in-between the element's flex items.
`align`	A responsive argument. One of `"start"`, `"end"`, `"center"`, `"baseline"`, or `"stretch"` specifying how items are arranged on the cross axis, defaults to `NULL`, in which case the argument is ignored.
`wrap`	A responsive argument. One of `TRUE` or `FALSE` specifying if items are forced onto one line or allowed to wrap onto multiple lines, defaults to `NULL`, in which case the argument is ignored.

Centering elements

Center an input above a larger element or next to the element if space allows.

```
div(
  .style %>%
    display("flex") %>%
    flex(direction = c(default = "column", md = "row")),

  radioButtons("id", "A sample input", c("Choice 1", "Choice 2")) %>%
    margin(h = c(xs = "auto", md = 0)),

  div(
    .style %>%
      width(100) %>%
      background("indigo") %>%
      text("white"),
    "Plot placeholder"
  )
)
```

A sample input

Choice 2

Plot placeholder

Details

This section needs pretty specific examples of how to use flex. I don't know that people will want a tutorial on flex.

For the sake of the demo let's create a flex item help function.

```
flexItem <- function(...) {
  div(
    .style %>% padding(3) %>% border("blue"),
    "A flex item",
    ...
  )
}
```

Different `direction`**s:**

Many of `flex()`'s arguments are viewport responsive. On small screens the flex items are placed vertically and can occupy the full width of the mobile device. On medium or larger screens the items are placed horizontally.

```
div(
  .style %>%
    display("flex") %>%
    flex(
      direction = c(xs = "column", md = "row")  # <-
    ),
  flexItem(),
  flexItem(),
  flexItem()
)
```

A flex item

A flex item

A flex item

Resize the browser for this example.

You can keep items as a column by specifying only `"column"`.

```
div(
  .style %>%
    display("flex") %>%
    flex(direction = "column"),  # <-
  flexItem(),
  flexItem(),
  flexItem()
)
```

A flex item

A flex item

A flex item

Spacing items with `justify`:

Below is a series of examples showing how to change the horizontal alignment of your flex items. Let's start by pushing items to the beginning of their parent container.

```
div(
  .style %>%
    display("flex") %>%
    flex(justify = "start"),  # <-
  flexItem(),
  flexItem(),
  flexItem(),
  flexItem()
)
```

A flex item

A flex item

A flex item

A flex item

We can also push items to the **end**.

```
div(
  .style %>%
    display("flex") %>%
    flex(justify = "end"),  # <-
  flexItem(),
  flexItem(),
  flexItem(),
  flexItem()
)
```

A flex item

A flex item

A flex item

A flex item

Without using a table layout we can **center** items.

```
div(
  .style %>%
    display("flex") %>%
    flex(justify = "center"),  # <-
  flexItem(),
  flexItem(),
  flexItem(),
  flexItem()
)
```

A flex item

A flex item

A flex item

A flex item

You can also put space **between** items

```
div(
  .style %>%
    display("flex") %>%
    flex(justify = "between"),  # <-
  flexItem(),
  flexItem(),
  flexItem(),
  flexItem()
)
```

A flex item

A flex item

A flex item

A flex item

... or put space **around** items.

```
div(
  .style %>%
    display("flex") %>%
    flex(justify = "around"),  # <-
  flexItem(),
  flexItem(),
  flexItem(),
  flexItem()
)
```

A flex item

A flex item

A flex item

A flex item

The `"between"` and `"around"` values come from the original CSS values `"space-between"` and `"space-around"`.

`wrap` **onto new lines:**

Using flexbox we can also control how items wrap onto new lines.

```
div(
  .style %>%
    display("flex") %>%
    flex(wrap = TRUE),
  flexItem(),
  flexItem(),
  flexItem(),
  flexItem()
)
```

A flex item

A flex item

A flex item

A flex item

Examples

```
library(htmltools)

div(
  .style %>%
    display("flex") %>%
    flex(justify = "end"),
  div("Aliquam posuere."),
  div("Lorem ipsum dolor sit amet, consectetuer adipiscing elit.")
)
```

`float` *Floats*

Description

The `float()` function places an element to the left or right side of its parent element. Other text and inline elements wrap around floated elements. Note, `float()` has no effect on flex items.

Usage

```
float(x, side)
```

Arguments

`x`	A tag element or .style pronoun.
`side`	A responsive argument. One of `"left"`, `"left"`, `"right"`, `"right"`, or `"none"` specifying the side to float the element.

Examples

```
library(htmltools)

div(
  div(
    .style %>%
      border("red") %>%
      float("left"),
    "Warning"
  ),
  div(
    "Nam a sapien.",
    "Phasellus neque orci, porta a, aliquet quis, semper a, massa.",
    "Phasellus lacus."
  )
)
```

`font` *Font*

Description

The `font()` function adjusts the size, weight, style, case, and family of the font of a tag element.

Usage

```
font(x, size = NULL, weight = NULL, style = NULL, case = NULL, family = NULL)
```

Arguments

`x`	A tag element or .style pronoun.
`size`	One of `1`, `2`, `3`, `4`, `5`, or `6` specifying a font size, defaults to `NULL`, in which case the argument is ignored. The sizes follow the conventions of heading tags, so `1` is the largest font and `6` the smallest.
`weight`	One of `"light"`, `"lighter"`, `"normal"`, `"bolder"`, or `"bold"` specifying the font weight, defaults to `NULL`, in which case the argument is ignored. If `"bolder"` or `"lighter"`, the font weight is changed relative to the current font weight.
`style`	One of `"italic"` or `"normal"` specifying the font style, defaults to `NULL`, in which case the argument is ignored.
`case`	One of `"upper"`, `"lower"`, or `"title"` specifying the font case, default to `NULL`, in which case the argument is ignored.
`family`	One of `"sans-serif"` or `"monospace"` specifying the font family, defaults to `NULL`, in which case the argument is ignored.

Weights

```
p(
  .style %>%
    font(weight = "bold"),
  "Curabitur lacinia pulvinar nibh."
)
```

Curabitur lacinia pulvinar nibh.

```
p(
  .style %>%
    font(weight = "light"),
  "Proin quam nisl, tincidunt et."
)
```

Proin quam nisl, tincidunt et.

Examples

```
library(htmltools)

p(
  .style %>%
    text("indigo") %>%
    font(weight = "bold"),
  "Phasellus at dui in ligula mollis ultricies."
)
```

`gap` *Grid element spacing*

Description

The `gap()` function is used to space child elements of a parent tag element with `display("grid")`. Instead of specifying a margin for each child element a gap may be specified for the parent element. This function will have no effect on element's without display set to `"grid"`.

Usage

```
gap(x, size)
```

Arguments

`x`	A tag element or .style pronoun.
`size`	A responsive argument. One of `0`, `1`, `2`, `3`, `4`, or `5` specifying the amount of gap space.

Details

Internet Explorer does not support the grid display layout.

Examples

```
library(htmltools)

div(
  .style %>%
    display("grid") %>%
    gap(2),
  div("Child 1"),
  div("Child 2"),
  div("Child 3")
)
```

`height` *Height*

Description

The `height()` function adjusts a tag element's height. Heights are specified relative the height of a parent element, an element's content, or the size of the browser window.

Usage

```
height(x, size, min = NULL, max = NULL)
```

Arguments

`x`	A tag element or .style pronoun.
`size`	One of `25`, `50`, `75`, `100`, `"auto"`, or `"viewport"` specifying the height of the tag element. If `25`, `50`, `75`, or `100`, the element's height is a percentage of the height of the parent element must also be specified. These percentages do not account for margins or padding and may cause an element to extend beyond its parent element. If `"auto"`, the element's height is determined by the browser. The browser will take into account the height, padding, margins, and border of the tag element's parent to keep the element from extending beyond its parent. If `"viewport"`, the element's height is determined by the size of the browser window.
`min`	One of `25`, `50`, `75`, `100`, or `"viewport"` specifying the minimum height of the tag element. See `size` for details.
`max`	One of `25`, `50`, `75`, `100`, or `"viewport"` specifying the maximum height of the tag element. See `size` for details.

Examples

```
library(htmltools)

div(
  .style %>%
     height("auto", max = "viewport") %>%
     overflow("auto"),
  "Vivamus id enim.",
  "Nunc rutrum turpis sed pede.",
  "Nunc aliquet, augue nec adipiscing interdum, ",
  "lacus tellus malesuada massa, quis varius mi purus non odio."
)
```

`item` *Flex items*

Description

The `item()` function adjusts a flex item. Unlike `flex()`, which adjusts the flex box layout through the flex container element, `item()` is used to change specific flex items. A flex item may be re-ordered, expanded, or shrunk.

Usage

```
item(
  x,
  align = "stretch",
  order = NULL,
  fill = NULL,
  grow = NULL,
  shrink = NULL
)
```

Arguments

`x`	A tag element or .style pronoun.
`align`	A responsive argument. One of `"auto"`, `"start"`, `"end"`, `"center"`, `"baseline"`, or `"stretch"` specifying how to align the item on the cross axis, defaults to `"stretch"`. Overrides the `flex()` `align` argument.
`order`	A responsive argument. One of `0`, `1`, `2`, `3`, `4`, `5`, `6`, or `7` specifying the order of the item, defaults to `1`. Items of the same order are then sorted by their source code order. Defaults to `NULL`, in which case the argument is ignored.
`fill`	A responsive argument. If `TRUE`, the flex parent element's horizontal space is divided proportionally amongst this tag element and all other flex items with `fill = TRUE`, defaults to `NULL`, in which case the argument is ignored.
`grow`	A responsive argument. One of `TRUE` or `FALSE`, defaults to `NULL`, in which case the argument is ignored.
`shrink`	A responsive argument. One of `TRUE` or `FALSE`, defaults to `NULL`, in which case the argument is ignored.

Examples

```
library(htmltools)
```

```
div(
  .style %>%
    display("flex"),
  div(
    .style %>%
      item(order = 2),
    "Second"
  ),
  div(
    "First"
  )
)
```

---

margin *Margins*

Description

The `margin()` function adjusts the outer spacing of a tag element. The margin of a tag element is the space outside and around the tag element, its border, and its content.

Usage

```
margin(
  x,
  all = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  horizontal = NULL,
  vertical = NULL
)
```

Arguments

`x`	A tag element or .style pronoun.
`all`	A responsive argument. One of `-5:5` or `"auto"` specifying a margin for all sides of the tag element, defaults to `NULL`, in which case the argument is ignored. 0 removes all outer space, 5 adds the most space, and negative values will consume space and pull the element in that direction.
`top, right, bottom, left`	A responsive argument. One of `-5:5` or `"auto"` specifying a margin for the respective side of the tag element.

`horizontal`	A responsive argument. One of `-5:5` or `"auto"` specifying a margin for the left and right sides of the tag element.
`vertical`	A responsive argument. One of `-5:5` or `"auto"` specifying a margin for the top and bottom sides of the tag element.

Auto margins

In most modern browsers you want to horizontally center a tag element using the flex layout. Alternatively, you can horizontally center an element using `margin(..,horizontal = "auto")`.

```
div(
  .style %>%
    margin(v = 2, h = "auto") %>%  # <-
    padding(3) %>%
    background("teal"),
  "Nam a sapien. Integer placerat tristique nisl."
)
```

Nam a sapien. Integer placerat tristique nisl.

Examples

```
library(htmltools)

div(
  .style %>%
    margin(left = 3, right = 3),
  "Mauris mollis tincidunt felis."
)

div(
  .style %>%
    margin(horizontal = 3),
  "Nulla posuere."
)


div(
  .style %>%
    margin(l = 2, b = 1),
  "Sed bibendum."
)


div(
  .style %>%
    margin(h = "auto"),
  "Sed id ligula quis est convallis tempor."
)
```

overflow *Overflow*

Description

The `overflow()` function adjust how an element's content scrolls. Scrolling an element's contents may be helpful to prevent child elements from extending the height or width of the element. The height of the element must be set.

Usage

```
overflow(x, scroll)
```

Arguments

`x`	A tag element or .style pronoun.
`scroll`	One of `"auto"`, `"hidden"`, `"visible"`, or `"scroll"` specifying how the content of the element scrolls. `TRUE` and `FALSE` may be used in place of `"scroll"` or `"hidden"`, respectively.

Examples

```

library(htmltools)

div(
  .style %>%
    width(25) %>%
    overflow(FALSE),
  "Nullam libero mauris, consequat quis, varius et, dictum id, arcu."
)
```

padding *Padding*

Description

The `padding()` function adjusts the inner spacing of a tag element. The padding of a tag element is the space between the tag element's border and its content or child elements.

Usage

```
padding(x, all = NULL, top = NULL, right = NULL, bottom = NULL, left = NULL)
```

Arguments

`x`	A tag element or .style pronoun.
`all`	A responsive argument. One of `1:5` specifying a padding for all sides of the tag element, defaults to `NULL`, in which case the argument is ignored. 0 removes all inner space and 5 adds the most space.
`top, right, bottom, left`	A responsive argument. One of `1:5` specifying a padding for the element's respective side, defaults to `NULL`, in which case the argument is ignored. 0 removes all inner space and 5 adds the most space.

Panels

Well panels.

```
wellPanel(
  radioButtons(
    inputId = "id",
    label = "Radio input",
    choices = c(
      "Choice 1",
      "Choice 2"
    )
  )
)
```

Radio input

Choice 2

Shrink well padding.

```
wellPanel(
  .style %>%
    padding(1),
  radioButtons(
    inputId = "id",
    label = "Radio input",
    choices = c(
      "Choice 1",
      "Choice 2"
    )
  )
)
```

Radio input

Choice 2

Auto width.

```
wellPanel(
  .style %>%
    padding(1),
  radioButtons(
    inputId = "id",
    label = "Radio input",
    choices = c(
      "Choice 1",
      "Choice 2"
    )
  ) %>%
    width("auto")
)
```

Radio input

Choice 2

Examples

```
library(htmltools)

div(
  .style %>%
    margin(2) %>%
    border("green") %>%
    padding(2) %>%
    background("red"),
  "Donec vitae dolor."
)
```

`position` *Position an element*

Description

The `position()` adjusts how an element is positioned. Positioning could be absolute or relative. Furthermore, you can arrange an element within its parent element using `top`, `right`, `bottom`, or `left`.

Usage

```
position(
  x,
  value,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  by = "edge"
)
```

Arguments

`x`	A tag element or .style pronoun.
`value`	One of `"static"`, `"relative"`, `"absolute"`, `"fixed"`, or `"sticky"` specifying how the element is positioned.
`top, right, bottom, left`	One of `0`, `50`, or `100` specifying where the element is positioned. By default these values position an element using the element's edge, see argument `by`. Defaults to `NULL`, in which case the argument is ignored.
`by`	One of `""` or `"by-center"` specifying the element's positioning anchor, defaults to `"edge"`.

Examples

```
library(htmltools)

div(
  div(.style %>% position("absolute", t = 0, r = 0))
)
```

`responsive` *Understanding responsive arguments*

Description

Responsive arguments allow you to apply styles to tag elements based on the size of the viewport (e.g. browser screen). This is important when developing applications for both web and mobile. Specifying a single unnamed value the style will be applied for all viewport sizes. Use the names below to apply a style for viewports of that size and larger. For example, specifying `c(default = "center",md = "left")` will apply `"center"` on extra small and small viewports, but for medium, large, and extra large viewports `"left"` is applied. Styles for larger viewports take precedence.

A responsive argument may be a single value or a named list. Specifying a single unnamed value is equivalent to specifying `default` or `xs`. The possible values will be described in the specific help page. Most responsive arguments default to `NULL` in which case the argument is ignored.

Breakpoints:

extra small

Use the breakpoint with default = or xs = .

The style is always applied, unless supplanted by a style for any other breakpoint.

small
Use the breakpoint with sm =.
The style is applied when the viewport is at least 576px wide, think landscape phones.
medium
Use the breakpoint with md =.
The style is applied when the viewport is at least 768px wide, think tablets.
large
Use the breakpoint with lg =.
The style is applied when the viewport is at least 992px wide, think laptop or smaller desktops.
extra large
Use the breakpoint with xl =.
The style is applied when the viewport is at least 1200px wide, think large desktops.
extra extra large
Use the breakpoint with xxl =.
The style is applied when the viewport is at least 1400px wide, think large desktops.

`shadow` *Shadows*

Description

The `shadow()` function adjusts the box shadow of a tag element. Shadows help distinguish elements or indicate interactivity.

Usage

```
shadow(x, size)
```

Arguments

`x`	A tag element or .style pronoun.
`size`	One of `"small"`, `"sm"`, `"medium"`, `"md"`, `"large"`, `"lg"`, or `"none"` specifying the amount of shadow added.

Sizes

small

sm

medium

md

large

lg

none

Examples

```
library(htmltools)

div(
  .style %>%
    shadow("md"),
  "Donec posuere augue in quam."
)

div(
  .style %>%
    border("red") %>%
    shadow("small"),
  "Praesent augue."
)
```

`style-pronoun` *Style pronoun*

Description

The `.style` pronoun allows you to define styles for a tag element within the context of the element. Without the `.style` pronoun tag styles are applied outside and after constructing a tag element.

```
div(". . .") %>% background("primary") %>% display("flex")
```

However, once the content of a tag element grows to more than a few lines, associating the element's styles with the element becomes less and less intuitive. In these situations, make use of the `.style` pronoun.

```
div(
  .style %>%
    border("primary") %>%
    font("primary"),
  p(". . ."),
  p(". . .")
)
```

Usage

```
.style
```

Prefixing

Complex components such as `shiny::radioButtons()` or `yonder::listGroupInput()` may need a non-standard prefix for the CSS classes applied by cascadess' functions.

text *Text*

Description

The `text()` function adjusts the text color, alignment, line spacing, line wrapping, line height, and decoration of a tag element.

Usage

```
text(
  x,
  color = NULL,
  align = NULL,
  spacing = NULL,
  decoration = NULL,
  wrap = NULL,
  select = NULL
)
```

Arguments

`x`	A tag element or .style pronoun.
`color`	One of `"blue"`, `"indigo"`, `"purple"`, `"red"`, `"orange"`, `"yellow"`, `"green"`, `"teal"`, `"cyan"`, `"black-50"`, `"white-50"`, `"white"`, `"muted"`, `"body"`, or `"reset"` specifying the text color, defaults to `NULL`, in which case the argument is ignored.
`align`	One of `"left"`, `"right"`, or `"center"` specifying the alignment of the text within the element, defaults to `NULL`, in which case the argument is ignored.
`spacing`	One of `"sm"`, `"small"`, `"md"`, `"medium"`, `"lg"`, or `"large"` specifying the text line spacing, defaults to `NULL`, in which case the argument is ignored.
`decoration`	One of `"none"`, `"underline"`, or `"strikethrough"` specifying how the text is decorated, defaults to `NULL`, in which case the argument is ignored.
`wrap`	One of `TRUE` or `FALSE` specifying if an element's text should wrap onto new lines, defaults to `NULL`, in which case the argument is ignored.
`select`	One of `"all"` or `"none"` specifying how the element's text is selected when the user clicks on the element, defaults to `NULL`, in which case the argument is ignored.

Colors

```
div(
  .style %>%
    text("blue") %>%  # <-
    border("blue"),
  p("Nullam tristique diam non turpis.",
    "Pellentesque dapibus suscipit ligula.",
    "Nullam eu ante vel est convallis dignissim."),
  p("Aliquam posuere.")
)
```

Nullam tristique diam non turpis. Pellentesque dapibus suscipit ligula. Nullam eu ante vel est convallis dignissim.

Aliquam posuere.

Examples

```
library(htmltools)

div(
  .style %>%
    text(spacing = "small"),
  "Nam vestibulum accumsan nisl.",
  "Fusce commodo."
)

div(
  .style %>%
    text(spacing = "large"),
  "Suspendisse potenti.",
  "Pellentesque tristique imperdiet tortor."
)

tags$button(
  .style %>%
    text(wrap = FALSE),
  "Aliquam feugiat tellus ut neque."
)
```

`visible` *Element visibility*

Description

The `visible()` function changes the visibility of a tag element. An invisible element is both visually hidden and is also hidden from screen readers.

Usage

```
visible(x, value)
```

Arguments

`x`	A tag element or .style pronoun.
`value`	One of `TRUE` or `FALSE` specifying if the element is visible, defaults to `TRUE`.

Examples

```
library(htmltools)

div("I am hidden") %>%
  visible(FALSE)
```

`width` *Width*

Description

The ‘width() function adjusts a tag element’s width. Widths are specified relative the width of a parent element, an element’s content, or the size of the browser window.

Usage

```
width(x, size, min = NULL, max = NULL)
```

Arguments

`x`	A tag element or .style pronoun.
`size`	One of `25`, `50`, `75`, `100`, `"auto"`, or `"viewport"` specifying the width of the tag element. If `25`, `50`, `75`, `100`, the element’s width is a percentage of the width of the parent element must also be specified. These percentages do not account for margins or padding and may cause an element to extend beyond its parent element. If `"auto"`, the element’s width is determined by the browser. The browser will take into account the height, padding, margins, and border of the tag element’s parent to keep the element from extending beyond its parent. If `"viewport"`, the element’s height is determined by the size of the browser window.
`min`	One of `25`, `50`, `75`, `100`, or `"viewport"` specifying the minimum width of the tag element, defaults to `NULL`, in which case the argument is ignored. See `size` for details.
`max`	One of `25`, `50`, `75`, `100`, or `"viewport"` specifying the maximum width of the tag element, defaults to `NULL`, in which case the argument is ignored. See `size` for details.

Examples

```
library(htmltools)

div(
  .style %>%
    width(c(xs = 100, md = 50)) %>%
    margin(c(xs = 2, md = "auto")),
  "In id erat non orci commodo lobortis.",
  "Suspendisse potenti.",
  "Nam euismod tellus id erat."
)
```

Index