Package 'yonder'

May 31, 2019

Type Package

Title A Reactive Web Framework Built on 'shiny'

Version 0.1.1

Description Build 'shiny' applications with the latest Bootstrap components and design utilities. Includes refreshed reactive inputs and outputs. Use responsive layouts to design and construct applications for devices of all sizes.

License GPL-3

URL https://nteetor.github.io/yonder

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

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 3.2), shiny (>= 1.1.0)

Imports htmltools, magrittr, utils

Suggests testthat (>= 2.1.0), knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Nathan Teetor [aut, cre],
The Bootstrap Authors [cph] (Bootstrap library),
Twitter, Inc [cph] (Bootstrap library),
JS Foundation [cph] (jQuery library),
Federico Zivolo [ctb, cph] (popper.js library),
Johann Servoire [ctb, cph] (bs-custom-file-input library)

Maintainer Nathan Teetor <nathanteetor@gmail.com>

Repository CRAN

Date/Publication 2019-05-31 15:40:03 UTC

R topics documented:

---

yonder-package *A new approach to shiny applications*

Description

Yonder is a set of tools for flexible and creative shiny application construction and design.

Inputs

Yonder provides many familiar inputs like `selectInput()` or `radioInput()`. There are also new inputs like `groupTextInput()` or `formInput()`.

Changes to be mindful of

- Input functions have an `id` argument instead of `inputId`.
- Input functions do not include a `label` argument for the purpose of adding a label above the input. Button and menu inputs do include a `label` argument, but these arguments refer to button labels. If you would like to add a label above an input please use `formGroup()`.

Familiar variants

Looking for ... ?

- `actionButton()` or `actionLink()` use `buttonInput()` or `linkInput()`
- `radioButtons()` use `radioInput()`
- `checkboxGroupInput()` use `checkbarInput()` or `checkboxInput()`
- `numericInput()` use `numberInput()`
- `selectizeInput()` use `selectInput()` or `chipInput()`
- `submitButton()` use `formInput()` and `formSubmit()`

Layout and content

Included are a handful of utilities for building applications suited for devices and screens of varying sizes. For real control over spacing elements be sure to check out `flex()`, which gives you the power of flexbox layout.

Familiar variants

Looking for ... ?

- `fluidRow()` or `fixedRow()` use `columns()`
- `fixedPage()`, `fluidPage()`, or `sidebarLayout()` use `container()`, `columns()`, and `column()`
- `navbarPage()` use `navbar()`
- `tabPanel()` use `navContent()` and `navPane()`
- `modalDialog()` use `modal()`

Author(s)

Maintainer: Nathan Teetor <nathanteetor@gmail.com>

Other contributors:

- The Bootstrap Authors (Bootstrap library) [copyright holder]
- Twitter, Inc (Bootstrap library) [copyright holder]
- JS Foundation (jQuery library) [copyright holder]
- Federico Zivolo (popper.js library) [contributor, copyright holder]
- Johann Servoire (bs-custom-file-input library) [contributor, copyright holder]

See Also

Useful links:

- https://nteetor.github.io/yonder
- Report bugs at https://github.com/nteetor/yonder/issues

`active` *Selected choice color*

Description

Use `active()` to change the highlight color of an input's selected choices.

Usage

```
active(tag, color)
```

Arguments

`tag`	A tag element.
`color`	One of `"red"`, `"purple"`, `"indigo"`, `"blue"`, `"cyan"`, `"teal"`, `"green"`, `"yellow"`, `"amber"`, `"orange"`, `"grey"`, `"white"` specifying the active color of selected choices.

See Also

Other design utilities: `affix`, `background`, `border`, `display`, `float`, `font`, `height`, `padding`, `scroll`, `shadow`, `width`

Examples

```
### Radiobar example

radiobarInput(
  id = "radio1",
  choices = c("Hello", "Goodnight", "Howdy")
) %>%
  width(16) %>%
  active("orange")  # <-

### Checkbox example

checkboxInput(
  id = "check1",
  choices = c("Rock", "Paper", "Scissors"),
  selected = "Rock"
) %>%
  active("teal")

### Chip input

chipInput(
  id = "chip1",
  choices = c("Ether", "Bombos", "Quake"),
  selected = "Ether"
) %>%
  width("1/2") %>%
  active("green")
```

`affix` *Position*

Description

The `affix` utility function applies Bootstrap classes to fix elements to the top or bottom of a page. Use `"sticky"` to cause an element to fix to the top of a page after the element is scrolled past. *Important*, the IE11 and Edge browsers do not support the sticky behavior.

Usage

```
affix(tag, position)
```

Arguments

`tag`	A tag element.
`position`	One of `"top"`, `"bottom"`, or `"sticky"` specifying the fixed behavior of an element.

See Also

Other design utilities: active, background, border, display, float, font, height, padding, scroll, shadow, width

Examples

```
### Affix an element

div(
  span("I'm up here!") %>%
    padding(left = 3, right = 3) %>%
    background("teal")
) %>%
  display("flex") %>%
  flex(justify = "center") %>%
  affix("top")
```

---

alert *Alert boxes*

Description

Use an alert element to let the user know of successes or to call attention to problems.

Usage

```
alert(..., dismissible = TRUE, fade = TRUE)
```

Arguments

...	Character strings specifying the text of the alert or additional named arguments passed as HTML attributes to the alert element.
dismissible	One of TRUE or FALSE specifying if the alert may be dismissed by the user, deafults to TRUE.
fade	One of TRUE or FALSE specifying if the alert fades out or immediately disappears when dismissed, defaults to TRUE.

See Also

Other components: badge, blockquote, card, collapsePane, d1, dropdown, img, jumbotron, modal, navContent, popover, pre, toast

Examples

```
### Default alert

alert("Donec at pede.") %>%
  background("blue")

### A more complex alert

alert(
  h4("Etiam vel tortor sodales"),
  hr(),
  p("Fusce commodo.")
) %>%
  background("amber")
```

---

background *Background color*

Description

Use `background()` to modify the background color of a tag element.

Usage

```
background(tag, color)
```

Arguments

`tag`	A tag element.
`color`	One of `"red"`, `"purple"`, `"indigo"`, `"blue"`, `"cyan"`, `"teal"`, `"green"`, `"yellow"`, `"amber"`, `"orange"`, `"grey"`, `"black"` or `"white"` specifying the background color of the tag element, defaults to `NULL`

See Also

Other design utilities: `active`, `affix`, `border`, `display`, `float`, `font`, `height`, `padding`, `scroll`, `shadow`, `width`

Examples

```
### Modifying input elements

checkbarInput(
  id = "bar1",
  choices = c(
    "Nunc rutrum turpis sed pede.",
```

```
    "Etiam vel neque.",
    "Lorem ipsum dolor sit amet."
  )
) %>%
  background("cyan")

### Possible colors

colors <- c(
  "red", "purple", "indigo", "blue", "cyan", "teal", "green",
  "yellow", "amber", "orange", "grey", "white"
)

div(
  lapply(
    colors,
    background,
    tag = div() %>%
      padding(5) %>%
      margin(2)
  )
) %>%
  display("flex") %>%
  flex(wrap = TRUE)
```

---

badge *Badges*

Description

Small highlighted content which scales to its parent's size. A badge may be dynamically updated with `replaceContent()`, in which case be sure to pass an `id` argument as part of `...`.

Usage

```
badge(...)
```

Arguments

`...`	Named arguments passed as HTML attributes to the parent element or tag elements passed as children to the parent element.

Example application

```
ui <- container(
  buttonInput(
    id = "clicker",
    label = list(
```

```
      "Clicks",
      badge(id = "counter") %>%
        margin(left = 2) %>%
        background("teal")
    )
  )
)

server <- function(input, output) {
  observe({
    clicks <- if (is.null(input$clicker)) 0 else input$clicker
    replaceContent("counter", clicks)
  })
}

shinyApp(ui, server)
```

See Also

Other components: alert, blockquote, card, collapsePane, d1, dropdown, img, jumbotron, modal, navContent, popover, pre, toast

Examples

```
### Possible colors

colors <- c(
  "red", "purple", "indigo", "blue", "cyan", "teal", "green",
  "yellow", "amber", "orange", "grey", "white"
)

div(
  lapply(colors, function(color) {
    badge(color) %>%
      background(color) %>%
      margin(2) %>%
      padding(1)
  })
) %>%
  display("flex") %>%
  flex(wrap = TRUE)
```

---

blockquote *Blockquotes*

Description

Stylized blockquotes, an updated builder function for <blockquote>.

Usage

```
blockquote(..., source = NULL, align = "left")
```

Arguments

...	Any number of tags elements or character strings passed as child elements or named arguments passed as HTML attributes to the parent element.
source	The quote source, use tags$cite to format the source title, defaults to NULL.
align	One of "left" or "right", defaults to "left".

See Also

Other components: alert, badge, card, collapsePane, d1, dropdown, img, jumbotron, modal, navContent, popover, pre, toast

Examples

```
### Simple example

blockquote(
  "Anyone can love a thing because.",
  "That's as easy as putting a penny in your pocket.",
  "But to love something despite.",
  "To know the flaws and love them too.",
  "That is rare and pure and perfect.",
  source = tags$span(
    "Patrick Rothfuss,", tags$cite("The Wise Man's Fear")
  )
)
```

---

border *Border color*

Description

Use border() to add or modify tag element borders.

Usage

```
border(tag, color = NULL, sides = "all", round = NULL)
```

Arguments

tag	A tag element.
color	One of "red", "purple", "indigo", "blue", "cyan", "teal", "green", "yellow", "amber", "orange", "grey", "black" or "white" specifying the border color of the tag element, defaults to NULL
sides	One or more of "top", "right", "bottom", "left" or "all" or "none" specifying which sides to add a border to, defaults to "all".
round	One or more of "top", "right", "bottom", "left", "circle", "all", or "none" specifying how to round the border(s) of a tag element, defaults to NULL, in which case the argument is ignored.

See Also

Other design utilities: active, affix, background, display, float, font, height, padding, scroll, shadow, width

Examples

```
### Change border color

div(
  div() %>%
    height(3) %>%
    width(3) %>%
    border("green"),
  div() %>%
    height(3) %>%
    width(3) %>%
    border(
      color = "blue",
      sides = c("left", "right")
    )
)

### Round sides

sides <- c("top", "right", "bottom", "left", "circle", "all")

div(
  lapply(
    sides,
    border,
    tag = div() %>%
      height(3) %>%
      width(3),
    color = "black"
  )
) %>%
  display("flex") %>%
```

```
  flex(wrap = TRUE)
```

`buttonGroupInput` *Button group inputs*

Description

A set of buttons with custom values.

Usage

```
buttonGroupInput(id, labels = NULL, values = labels, ...)

updateButtonGroupInput(id, labels = NULL, values = labels,
  enable = NULL, disable = NULL,
  session = getDefaultReactiveDomain())
```

Arguments

`id`	A character string specifying the id of the reactive input.
`labels`	A character vector specifying the labels for each button in the group.
`values`	A vector of values specifying the values of each button in the group, defaults to `labels`.
`...`	Additional named arguments passed as HTML attributes to the parent element.
`enable`	One of `values` indicating individual buttons to enable or `TRUE` to enable the entire input, defaults to `NULL`.
`disable`	One of `values` indicating individual buttons to disable or `TRUE` to disable the entire input, defaults to `NULL`.
`session`	A reactive context, defaults to `getDefaultReactiveDomain()`.

See Also

Other inputs: `buttonInput`, `checkbarInput`, `checkboxInput`, `chipInput`, `fileInput`, `formInput`, `listGroupInput`, `menuInput`, `navInput`, `radioInput`, `radiobarInput`, `rangeInput`, `selectInput`, `textInput`

Examples

```
### Default input

buttonGroupInput(
  id = "group1",
  labels = c("Once", "Twice", "Thrice"),
  values = c(1, 2, 3)
```

```
)

### Styling the button group

buttonGroupInput(
  id = "group2",
  labels = c("Button 1", "Button 2", "Button 3")
) %>%
  background("blue") %>%
  width("1/3")
```

---

buttonInput *Button and link inputs*

Description

Button inputs are useful as triggers for reactive or observer expressions. The reactive value of a button input begins as NULL, but subsequently is the number of clicks.

Usage

```
buttonInput(id, label, ..., stretch = FALSE, download = FALSE,
  tooltip = NULL)

updateButtonInput(id, label = NULL, value = NULL, disable = NULL,
  enable = NULL, session = getDefaultReactiveDomain())

linkInput(id, label, ..., stretch = FALSE, download = FALSE,
  tooltip = NULL)

updateLinkInput(id, label = NULL, value = NULL, enable = NULL,
  disable = NULL, session = getDefaultReactiveDomain())

tooltip(..., placement = "top", fade = TRUE)
```

Arguments

id	A character string specifying the id of the reactive input.
label	A character string specifying the label text on the button or link input.
...	Additional named arguments passed as HTML attributes to the parent element.
stretch	One of TRUE or FALSE specifying stretched behaviour for the button or link input, defaults to FALSE. If TRUE, the button or link will receive clicks from its containing block element. For example, a stretched button or link inside a card() would update whenever the user clicked on the card.
download	One of TRUE or FALSE specifying if the button or link input is used to trigger a download, defaults to FALSE.

tooltip	A call to tooltip() specifying a tooltip for the button or link input, defaults to NULL.
value	A number specifying a new value for the button, defaults to NULL.
disable	if TRUE the button is disabled and will not react to clicks from the user, default to NULL.
enable	If TRUE the button is enabled and will react to clicks from the user, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().
placement	One of "top", "right", "bottom", or "left" specifying what side of the tag element the tooltip appears on.
fade	One of TRUE or FALSE specifying if the tooltip fades in when shown and fades out when hidden, defaults to TRUE.

See Also

Other inputs: buttonGroupInput, checkbarInput, checkboxInput, chipInput, fileInput, formInput, listGroupInput, menuInput, navInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### A simple button

buttonInput(
  id = "button1",
  label = "Simple"
)

# Alternatively, a button can fill the width of its parent element.

buttonInput(
  id = "button2",
  label = "Full-width",
  fill = TRUE  # <-
) %>%
  background("red")

# Use design utilities to further adjust the width of a button.

buttonInput(
  id = "button3",
  label = "Full and back again",
  fill = TRUE  # <-
) %>%
  background("red") %>%
  width("3/4")  # <-

### Possible colors
```

```
colors <- c(
  "red", "purple", "indigo", "blue", "cyan", "teal", "green",
  "yellow", "amber", "orange", "grey"
)

lapply(
  colors,
  function(color) {
    buttonInput(
      id = color,
      label = color
    ) %>%
      background(color) %>%
      margin(2)
  }
) %>%
  div() %>%
  display("flex") %>%
  flex(wrap = TRUE)

### Reactive links

div("Curabitur ", linkInput("link1", "vulputate"), " vestibulum lorem.")

### Stretched buttons and links

card(
  header = "Card with stretched button",
  p("Notice when you hover over the card, the button also detects ",
    "the hover."),
  buttonInput(
    id = "go",
    label = "Go go go",
    stretch = TRUE
  ) %>%
    background("blue")
) %>%
  width(20)

### Download button

buttonInput(
  download = TRUE,
  id = "download1",
  label = "Download",
  icon("download")
)
```

---

card *Cards, blocks of content*

Description

Create blocks of content with `card`. `deck` is used to group and add padding is placed around any number of cards. Additionally, grouping cards with `deck` has the benefit of aligning the footer of each card.

Usage

```
card(..., header = NULL, footer = NULL)

deck(...)
```

Arguments

`...`	For **card**, `tag$div()`s, tag elements, or list groups to include in the card or additional named arguments passed as HTML attributes to the parent element. For **deck**, any number of `card()`s or additional named arguments passed as HTML attributes to the parent element.
`header`	A character string or tag element specifying the header of the card, defaults to `NULL`, in which case a header is not added.
`footer`	A character string or tag element specifying the footer of the card, defaults to `NULL`, in which case a footer is not added.

See Also

Other components: `alert`, `badge`, `blockquote`, `collapsePane`, `d1`, `dropdown`, `img`, `jumbotron`, `modal`, `navContent`, `popover`, `pre`, `toast`

Examples

```
### A simple card

column(
  width = 4,
  card(
    p("Praesent fermentum tempor tellus.")
  )
)

### Adding a title, subtitle

column(
  width = 4,
  card(
    h5("Mauris mollis tincidunt felis."),
    h6("Phasellus at dui in ligula mollis ultricies."),
    p("Nullam tempus. Mauris mollis tincidunt felis."),
    p("Nullam libero mauris, consequat quis, varius et, dictum id, arcu.")
  )
)
```

```
### Styling cards

deck(
  card(
    header = "Donec pretium posuere tellus",
    p("Donec hendrerit tempor tellus."),
    p("Cras placerat accumsan nulla.")
  ) %>%
    font(color = "teal"),
  card(
    p("Aliquam posuere."),
    p("Phasellus neque orci, porta a, aliquet quis, semper a, massa."),
    p("Pellentesque dapibus suscipit ligula.")
  ) %>%
    border("orange"),
  card(
    header = "Phasellus lacus",
    p("Etiam laoreet quam sed arcu."),
    p("Etiam vel tortor sodales tellus ultricies commodo."),
    footer = "Nam euismod tellus id erat."
  ) %>%
    background("grey") %>%
    font(color = "indigo")
)

### Cards with list groups

column(
  width = 4,
  card(
    listGroupInput(
      id = "lg1",
      flush = TRUE,
      choices = c(
        "Pellentesque tristique imperdiet tortor.",
        "Lorem ipsum dolor sit amet, consectetuer adipiscing elit.",
        "Phasellus purus."
      ),
      values = c(
        "choice1",
        "choice2",
        "choice3"
      )
    )
  )
)

### Tabbed content in cards

card(
  header = navInput(
    id = "tabs",
```

```
    choices = c("Tab 1", "Tab 2", "Tab 3"),
    appearance = "tabs"
  ),
  navContent(
    navPane(
      "Phasellus purus.",
      "Proin neque massa, cursus ut, gravida ut, lobortis eget, lacus.",
      "Phasellus purus."
    ),
    navPane(
      "Donec at pede. Praesent augue.",
      "Pellentesque tristique imperdiet tortor."
    ),
    navPane(
      "Fusce suscipit, wisi nec facilisis facilisis,",
      "est dui fermentum leo, quis tempor ligula erat quis odio.",
      "Donec hendrerit tempor tellus."
    )
  )
)

### Deck of cards

deck(
  card(
    title = "Nullam tristique",
    p("Fusce sagittis, libero non molestie mollis, magna orci ultrices ",
      "dolor, at vulputate neque nulla lacinia eros."),
    p("Nunc rutrum turpis sed pede."),
    footer = "Cras placerat accumsan nulla."
  ),
  card(
    title = "Integer placerat",
    p("Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec ",
      "hendrerit tempor tellus."),
    footer = "Cras placerat accumsan nulla."
  ),
  card(
    title = "Phasellus neque",
    p("Donec at pede. Etiam vel neque nec dui dignissim bibendum."),
    footer = "Cras placerat accumsan nulla."
  )
)
```

---

checkbarInput *Checkbar input*

Description

A stylized checkbox input. The checkbar input appears similar to a group of buttons, but with a checked or highlighted state.

Usage

```
checkbarInput(id, choices = NULL, values = choices, selected = NULL,
  ...)

updateCheckbarInput(id, choices = NULL, values = choices,
  selected = NULL, enable = NULL, disable = NULL,
  session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the reactive input.
choices	A character vector or list of tag element specifying the input's choices, defaults to NULL.
values	A vector of values specifying the values of the input's choices, defaults to choices.
selected	One or more of values specifying the input's default selected values, defaults to NULL.
...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

See Also

Other inputs: buttonGroupInput, buttonInput, checkboxInput, chipInput, fileInput, formInput, listGroupInput, menuInput, navInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### Default checkbar

checkbarInput(
  id = "cb1",
  choices = c("When", "Why", "Where")
)

### Modifying background color

checkbarInput(
```

```
  id = "cb2",
  choices = c("What", "Which")
) %>%
  background("teal")

### Labeling a checkbar

formGroup(
  label = "Toppings",
  checkbarInput(
    id = "fixins",
    choices = c(
      "Sprinkles",
      "Nuts",
      "Fudge"
    )
  )
)
```

---

checkboxInput *Checkbox and switch inputs*

Description

Reactive checkbox and checkbar inputs. Users may select one or more choices. The checkbox input appears as a standard checkbox or set of checkboxes. When a checkbox input has no selected choices the reactive value is NULL. Switch inputs differ from checkboxes only in appearance.

Usage

```
checkboxInput(id, choices = NULL, values = choices, selected = NULL,
  ..., inline = FALSE)

updateCheckboxInput(id, choices = NULL, values = choices,
  selected = NULL, inline = FALSE, enable = NULL, disable = NULL,
  valid = NULL, invalid = NULL, session = getDefaultReactiveDomain())

switchInput(id, choices, values = choices, selected = NULL, ...)

updateSwitchInput(id, choices = NULL, values = choices,
  selected = NULL, enable = NULL, disable = NULL, valid = NULL,
  invalid = NULL, session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the reactive input.
choices	A character string or vector specifying a label or labels for the checkbox or checkbar.

values	A character string or vector specifying values for the checkbox or checkbar input, defaults to choice or values, respectively.
selected	One or more of values specifying which choices are selected by default, defaults to NULL, in which case no choices are initially selected.
...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
inline	One of TRUE or FALSE specifying if the checkbox input choices render inline or stacked, defaults to FALSE, in which case the choices are stacked.
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
valid	A character string specifying a message to the user indicating how the input's value is valid, defaults to NULL.
invalid	A character string specifying a message to the user indicating how the input's value is invalid, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, chipInput, fileInput, formInput, listGroupInput, menuInput, navInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### One option

checkboxInput(
  id = "checkbox1",
  choices = "Choice 1",
  selected = "Choice 1"
)

### Multiple options

checkboxInput(
  id = "checkbox2",
  choices = c("Choice 1", "Choice 2")
)

### Inline checkbox

checkboxInput(
  id = "checkbox3",
  choices = c("Choice 1", "Choice 2", "Choice 3"),
  inline = TRUE
```

```
)

### Switches

switchInput(
  id = "switch1",
  choices = paste("Switch choice", 1:3),
  selected = "Switch choice 3"
) %>%
  active("indigo")
```

---

chipInput *Chip inputs*

Description

The chip input is a selectize alternative. Choices are selected from a dropdown menu and appear as chips below the input's text box. Chips do not appear in the order they are selected. Instead chips are shown in the order specified by the `choices` argument. Use the `max` argument to limit the number of choices a user may select.

Usage

```
chipInput(id, choices = NULL, values = choices, selected = NULL, ...,
  max = Inf, inline = TRUE)

updateChipInput(id, choices = NULL, values = choices,
  selected = NULL, max = NULL, enable = NULL, disable = NULL,
  session = getDefaultReactiveDomain())
```

Arguments

`id`	A character string specifying the id of the reactive input.
`choices`	A character vector or list specifying the possible choices.
`values`	A character vector or list of strings specifying the input's values, defaults to `choices`.
`selected`	One or more of `values` specifying which values are selected by default.
`...`	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
`max`	A number specifying the maximum number of items a user may select, defaults to `Inf`.
`inline`	One of `TRUE` or `FALSE` specifying if chips are rendered inline. If `TRUE` multiple chips may fit onto a single row, otherwise, if `FALSE`, chips expand to fill the width of their parent element, one chip per row.

enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

Example simple application

```
ui <- container(
  chipInput(
    id = "chips",
    choices = paste("Option number", 1:10),
    values = 1:10,
    inline = TRUE
  ) %>%
    width("1/2")
)

server <- function(input, output) {

}

shinyApp(ui, server)
```

Example inline chips

```
ui <- container(
  chipInput(
    id = "chips",
    choices = c(
      "A rather long option, isn't it?",
      "Shorter",
      "A middle-size option",
      "One more"
    ),
    values = 1:4,
    fill = FALSE
  ) %>%
    width("1/2") %>%
    background("blue") %>%
    shadow("small")
)

server <- function(input, output) {

}

shinyApp(ui, server)
```

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, fileInput, formInput, listGroupInput, menuInput, navInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### Default input

chipInput(
  id = "chip1",
  choices = paste("Choice", 1:5),
  selected = c("Choice 3", "Choice 4")
)
```

---

collapsePane *Collapsible panes*

Description

The collapsePane() creates a collapsible container. The state of the container, expanded or collapsed, is toggled using showCollapsePane(), hideCollapsePane(), and toggleCollapsePane().

Usage

```
collapsePane(id, ..., show = FALSE)

hideCollapsePane(id, session = getDefaultReactiveDomain())

showCollapsePane(id, session = getDefaultReactiveDomain())

toggleCollapsePane(id, session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the collapse pane.
...	Tag elements inside the collapsible pane or additional named arguments passed as HTML attributes to parent element.
show	One of TRUE or FALSE specifying if the collapsible pane is shown when the page renders, defaults to FALSE.
session	A reactive context, defaults to getDefaultReactiveDomain().

Details

Padding may not be applied to the collapsible pane div element. To pad a collapsible pane first wrap the pane in another element and add padding to this new element.

App with collapse

```
ui <- container(
  buttonInput(
    id = "demo",
    label = "Toggle collapse"
  ),
  collapsePane(
    id = "collapse",
    p(
      "Pellentesque condimentum, magna ut suscipit hendrerit, ",
      "ipsum augue ornare nulla, non luctus diam neque sit amet urna."
    ),
    p(
      "Praesent fermentum tempor tellus.  Vestibulum convallis, ",
      "lorem a tempus semper, dui dui euismod elit, vitae placerat ",
      "urna tortor vitae lacus."
    )
  )
)

server <- function(input, output) {
  observeEvent(input$demo, {
    toggleCollapsePane("collapse")
  })
}

shinyApp(ui, server)
```

See Also

Other components: alert, badge, blockquote, card, d1, dropdown, img, jumbotron, modal, navContent, popover, pre, toast

Examples

```
### Examples

# As these are server-side utilities, please run the example applications
# above.
```

---

column *Grid layout*

Description

These functions are the foundation of any application. Grid elements are nested as follows: `container() > columns() > co`
A `column()` may be created with an explicit width, 1 through 12. To fit a column automatically to its content use `width = "auto"`. To divide the space in a row evenly amongst all columns leave `width` as `NULL`. For examples and usage tips see the sections below.

Usage

```
column(..., width = NULL)

columns(...)

container(..., centered = FALSE)
```

Arguments

`...`	Any number of tags elements passed as child elements or named arguments passed as HTML attributes to the parent element.
`width`	A responsive argument. One of `1:12` or `"auto"`, defaults to `NULL`.
`centered`	One of `TRUE` or `FALSE` specifying how a container fills the browser or viewport window. If `TRUE` the container is responsively centered, otherwise, if `FALSE`, the container occupies the entire width of the viewport, defaults to `FALSE`.

See Also

Other layout functions: `fieldset`, `flex`, `navbar`, `responsive`

Examples

```
### Equal width columns

container(
  columns(
    column(
      "Aliquam erat volutpat."
    ),
    column(
      "Mauris mollis tincidunt felis."
    ),
    column(
      "Cum sociis natoque penatibus et magnis dis parturient montes,",
      "nascetur ridiculus mus."
    )
  )
)

###  Shiny's panel with sidebar layout

container(
```

```
  columns(
    column(
      width = 4,
      card(
        title = "Sidebar",
        formGroup(
          label = "Control 1",
          selectInput("control1", "...")
        ),
        formGroup(
          label = "Control 2",
          selectInput("control2", "...")
        ),
        formGroup(
          label = "Control 3",
          selectInput("control3", "...")
        )
      )
    ),
    column(
      d4("Main panel")
    )
  )
)

### Mobile friendly grids

# Use `column()`s [responsive] `width` argument to make mobile friendly
# applications.

container(
  columns(
    column(
      width = c(sm = 4),
      "Mauris ac felis vel velit tristique imperdiet."
    ),
    column(
      width = c(sm = 4),
      "Nam vestibulum accumsan nisl."
    ),
    column(
      width = c(sm = 4),
      "Proin neque massa, cursus ut, gravida ut, lobortis eget, lacus."
    )
  )
)

# or

container(
  columns(
    column(
      width = c(sm = 4),
```

```
      "Aenean in sem ac leo mollis blandit."
    ),
    column(
      width = c(sm = 8),
      "Nulla posuere. In id erat non orci commodo lobortis."
    )
  )
)

### Fit columns to their content

container(
  columns(
    column(),
    column(
      width = "auto",
      "Cras placerat accumsan nulla. Aenean in sem ac leo mollis blandit."
    ),
    column()
  )
)
```

---

d1 *Headings*

Description

Display headings are not meant to replace the standard HTML heading tags, they are a stand out alternative for eye-catching titles.

Usage

```
d1(...)

d2(...)

d3(...)

d4(...)
```

Arguments

`...`	Any number of character strings or tag elements or named arguments passed as HTML attributes to the parent element.

See Also

Other components: `alert`, `badge`, `blockquote`, `card`, `collapsePane`, `dropdown`, `img`, `jumbotron`, `modal`, `navContent`, `popover`, `pre`, `toast`

Examples

```
### d1

d1("Eye-catching!")

### d2

d2("Just incredible")

### d3

d3("Wowie, zowie")

### d4

d4("You'll never guess what happens next.")
```

`display` *Display property*

Description

Use the `display()` utility to adjust how a tag element is rendered. All arguments are responsive allowing you to hide elements on small screens or convert elements from inline to block on large screens.

Usage

```
display(tag, type)
```

Arguments

`tag`	A tag element.
`type`	A responsive argument. One of `"inline"`, `"block"`, `"inline-block"`, `"flex"`, `"inline-flex"`, or `"none"`.

See Also

Other design utilities: `active`, `affix`, `background`, `border`, `float`, `font`, `height`, `padding`, `scroll`, `shadow`, `width`

Examples

```
### Using flexbox

# When using `flex()` be sure to set the display, too.

div(
  lapply(
    1:5,
    function(i) {
      div() %>%
        padding(5) %>%
        margin(top = c(xs = 2), bottom = c(xs = 2)) %>%
        background("blue")
    }
  )
) %>%
  display("flex") %>%
  flex(
    direction = c(xs = "column", sm = "row"),
    justify = c(sm = "around")
  )
```

`dropdown` *Dropdown menus*

Description

Dropdown menus are a container for buttons, text, and form inputs. See argument `...` for details on composing dropdown menus.

Usage

```
dropdown(label, ..., direction = "down", align = "left")
```

Arguments

`label`	A character string specifying the label of the dropdown's button.
`...`	Character strings or vectors, header tag elements, button inputs, or form inputs specifying the elements of the dropdown. These elements may be grouped into lists to create a menu with sections. `h6()` is the recommended heading level for menu headers. Character vectors are converted into paragraphs of text. To format menu text use `p()` and utility functions. Additional named arguments are passed as HTML attributes to the parent element.
`direction`	One of `"up"`, `"right"`, `"down"`, or `"left"` specifying the direction in which the menu opens, defaults to `"down"`.

align One of "left" or "right" specifying which side of the button to align the dropdown menu to, defaults to "left".

See Also

Other components: alert, badge, blockquote, card, collapsePane, d1, img, jumbotron, modal, navContent, popover, pre, toast

Examples

```
### Dropdown with buttons

dropdown(
  label = "Choices",
  buttonInput("choice1", "Choice 1"),
  buttonInput("choice2", "Choice 2"),
  buttonInput("choice3", "Choice 3")
)

### Dropdown with links

dropdown(
  label = "Choices",
  linkInput("link1", "Choice 1"),
  linkInput("link2", "Choice 2")
)

### Grouped sections

dropdown(
  label = "Sections",
  h6("Section 1"),
  buttonInput("a", "Option A"),
  buttonInput("b", "Option B"),
  hr(),
  h6("Section 2"),
  buttonInput("c", "Option C"),
  buttonInput("d", "Option D")
)

### Direction variations

dropdown(
  label = "Up!",
  direction = "up",
  buttonInput("up1", "Choice 1"),
  buttonInput("up2", "Choice 2")
)

### Dropdowns with forms
```

```
dropdown(
  label = "Sign in",
  formInput(
    id = "login",
    formGroup(
      label = "Username / Email",
      textInput(
        type = "email",
        id = "user"
      )
    ),
    formGroup(
      label = "Password",
      textInput(
        type = "password",
        id = "pass"
      )
    ),
    formSubmit(
      label = "Sign in",
      value = "signin"
    )
  ) %>%
    padding(3, 4, 3, 4)
)
```

`fieldset` *Group and label multiple inputs*

Description

Use `fieldset` to associate and label inputs. This is good for screen readers and other assistive technologies.

Usage

```
fieldset(..., legend = NULL)
```

Arguments

`...`	Any number of inputs to group or named arguments passed as HTML attributes to the parent element.
`legend`	A character string specifying the fieldset's legend.

See Also

Other layout functions: `column`, `flex`, `navbar`, `responsive`

Examples

```
### Grouping related inputs

fieldset(
  legend = "Pizza order",
  formGroup(
    "What toppings would you like?",
    div(
      checkbarInput(
        id = "toppings",
        choices = c(
          "Cheese",
          "Black olives",
          "Mushrooms"
        )
      )
    )
  ),
  formGroup(
    "Is this for delivery?",
    checkboxInput(
      id = "deliver",
      choice = "Deliver"
    )
  ),
  buttonInput("order", "Place order") %>%
    background("blue")
)
```

`fileInput` *File inputs*

Description

Upload files to the server.

Usage

```
fileInput(id, placeholder = "Choose file", browse = "Browse", ...,
  multiple = TRUE, accept = NULL)
```

Arguments

`id`	A character string specifying the id of the reactive input.
`placeholder`	A character string specifying the text inside the file input, defaults to `"Choose file"`.
`browse`	A character string specifying the label of file input, defaults to `"Browse"`.

...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
multiple	One of TRUE or FALSE specifying whether or not the user can upload multiple files at once, defaults to TRUE.
accept	A character vector of possible MIME types or file extensions, defaults to NULL, in which case any file type may be selected.

Example uploading a file

```
shinyApp(
  ui = container(
    fileInput("upload") %>%
      margin(0, "auto", 0, "auto")
  ),
  server = function(input, output) {
    observe({
      req(input$upload)

      print(input$upload)
    })
  }
)
```

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, formInput, listGroupInput, menuInput, navInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### Standard file input

fileInput(id = "file1")

### Adding a button

fileInput(
  id = "file2",
  left = buttonInput("upload", "Upload") %>%
    background("green")
)

### Customizing text

fileInput(
  id = "file3",
  placeholder = "Pick a file",
  browse = "Go go go!"
```

```
)
```

`flex` *Flex layout*

Description

Use `flex()` to control how a flex container tag element places its flex items or child tag elements. For more on turning a tag element into a flex container see `display()`. By default tag elements within a flex container are treated as flex items.

Usage

```
flex(tag, direction = NULL, justify = NULL, align = NULL,
  wrap = NULL, reverse = NULL)
```

Arguments

`tag`	A tag element.
`direction`	A responsive argument. One of `"row"` or `"column"` specifying the placement of flex items, defaults to `NULL`. If `"row"` items are placed vertically, if `"column"` items are placed horizontally. Browsers place items vertically by default.
`justify`	A responsive argument. One of `"start"`, `"end"`, `"center"`, `"between"`, or `"around"` specifying how items are horizontally aligned, defaults to `NULL`. See the **justify** section below for more on how the different values affect horizontal spacing.
`align`	A responsive argument. One of `"start"`, `"end"`, `"center"`, `"baseline"`, or `"stretch"` specifying how items are vertically aligned, defaults to `NULL`. See the **align** section below for more on how the different values affect vertical spacing.
`wrap`	A responsive argument. One of `TRUE` or `FALSE` specifying whether to wrap flex items inside the flex containter, `tag`, defaults to `NULL`. If `TRUE` items wrap inside the container, if `FALSE` items will not wrap. See the **wrap** section below for more.
`reverse`	A responsive argument. One of `TRUE` or `FALSE` specifying if flex items are placed in reverse order, defaults to `NULL`. If `TRUE` items are placed from right to left when `direction` is `"row"` or bottom to top when `direction` is `"column"`.

See Also

Other layout functions: `column`, `fieldset`, `navbar`, `responsive`

Examples

```
### Different `direction`s

# Many of `flex()``'s arguments are viewport responsive and below we will see
# how useful this can be. 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 once again.

div(
  div("A flex item") %>%
    padding(3) %>%
    border(),
  div("A flex item") %>%
    padding(3) %>%
    border(),
  div("A flex item") %>%
    padding(3) %>%
    border()
) %>%
  display("flex") %>%
  flex(
    direction = list(xs = "column", md = "row")  # <-
  ) %>%
  background("grey") %>%
  border()

# *Resize the browser for this example.*

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

div(
  div("A flex item") %>%
    padding(3) %>%
    border(),
  div("A flex item") %>%
    padding(3) %>%
    border(),
  div("A flex item") %>%
     padding(3) %>%
     border()
) %>%
  display("flex") %>%
  flex(direction = "column")  # <-

### 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(
```

```
  replicate(
    div("A flex item") %>%
      padding(3) %>%
      border(),
    n = 5,
    simplify = FALSE
  )
) %>%
  display("flex") %>%
  flex(justify = "start")  # <-

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

div(
  replicate(
    div("A flex item") %>%
      padding(3) %>%
      border(),
    n = 5,
    simplify = FALSE
  )
) %>%
  display("flex") %>%
  flex(justify = "end")  # <-

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

div(
  replicate(
    div("A flex item") %>%
      padding(3) %>%
      border(),
    n = 5,
    simplify = FALSE
  )
) %>%
  display("flex") %>%
  flex(justify = "center")  # <-

# You can also put space **between** items

div(
  replicate(
    div("A flex item") %>%
      padding(3) %>%
      border(),
    n = 5,
    simplify = FALSE
  )
) %>%
  display("flex") %>%
  flex(justify = "between")  # <-
```

```
# ... or put space **around** items.

div(
  replicate(
    div("A flex item") %>%
      padding(3) %>%
      border(),
    n = 5,
    simplify = FALSE
  )
) %>%
  display("flex") %>%
  flex(justify = "around")  # <-

# *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(
  replicate(
    div("A flex item") %>%
      padding(3) %>%
      border(),
    n = 5,
    simplify = FALSE
  )
) %>%
  display("flex") %>%
  flex(wrap = TRUE)
```

`float` *Float*

Description

Use `float()` to float an element to the left or right side of its parent element. A newspaper layout is a classic usage where an image is floated with text wrapped around.

Usage

```
float(tag, side)
```

Arguments

`tag`	A tag element.
`side`	A responsive argument. One of `"left"` or `"right"` specifying the side to float the element.

See Also

Other design utilities: active, affix, background, border, display, font, height, padding, scroll, shadow, width

Examples

```
### Newspaper layout

div(
  div() %>%
    width(5) %>%
    height(5) %>%
    margin(right = 2) %>%
    background("amber") %>%
    float("left"),
  p(
    "Fusce commodo. Nullam tempus. Nunc rutrum turpis sed pede.",
    "Phasellus lacus.  Cras placerat accumsan nulla.",
    "Fusce sagittis, libero non molestie mollis, ",
    "magna orci ultrices dolor, at vulputate neque nulla lacinia eros."
  ),
  p(
    "Nulla facilisis, risus a rhoncus fermentum, tellus tellus",
    "lacinia purus, et dictum nunc justo sit amet elit."
  ),
  p(
    "Proin neque massa, cursus ut, gravida ut, lobortis eget, lacus.",
    "Aliquam posuere.",
    "Sed id ligula quis est convallis tempor."
  )
)
```

---

font *Font color, size, weight*

Description

The font() utility modifies the color, size, weight, case, or alignment of a tag element's text. All arguments default to NULL, in which case they are ignored. For example, font(.., size = "lg") increases font size without affecting color, weight, case, or alignment.

Usage

```
font(tag, color = NULL, size = NULL, weight = NULL, case = NULL,
  align = NULL)
```

Arguments

tag	A tag element.
color	One of "red", "purple", "indigo", "blue", "cyan", "teal", "green", "yellow", "amber", "orange", "grey", "black" or "white" specifying the text color of the tag element, defaults to NULL
size	One of "xs", "sm", "base", "lg", "xl" specifying a font size relative to the default base page font size, defaults to NULL.
weight	One of "bold", "normal", "light", "italic", or "monospace" specifying the font weight of the element's text, defaults to NULL.
case	One of "upper", "lower", or "title" specifying a transformation of the tag element's text, default to NULL.
align	A responsive argument. One of "left", "center", "right", or "justify", specifying the alignment of the tag element's text, defaults to NULL.

See Also

Other design utilities: active, affix, background, border, display, float, height, padding, scroll, shadow, width

Examples

```
### Changing text color

card(
  header = h3("Important!") %>%
    font(color = "amber"),
  div(
    "This is a reminder."
  )
) %>%
  border(color = "amber")

### Changing font size

div(
  p("Extra small") %>%
    font(size = "xs"),
  p("Small") %>%
    font(size = "sm"),
  p("Medium") %>%
    font(size = "base"),
  p("Large") %>%
    font(size = "lg"),
  p("Extra large") %>%
    font(size = "xl")
)

### Changing font weight
```

```
# Make an element's text bold, italic, light, or monospace.

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

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

---

formGroup *Input labels, help text, and formatting to inputs*

Description

Form groups are a way of labelling an input. Form rows are similar to `columns()`s, but include additional styles intended for forms. The flexibility provided by form rows and groups means you can confidently develop shiny applications for devices and screens of varying sizes.

Usage

```
formGroup(label, input, ..., help = NULL, width = NULL)

formRow(...)
```

Arguments

`label`	A character string specifying a label for the input or `NULL` in which case a label is not added.
`input`	A tag element specifying the input to label.
`...`	For **formGroup**, additional named arguments passed as HTML attributes to the parent element. For **formRow**, any number of `formGroups` or additional named arguments passed as HTML attributes to the parent element.
`help`	A character string specifying help text for the input, defaults to `NULL`, in which case help text is not added.
`width`	A responsive argument. One of `1:12` or "auto" specifying a column width for the form group, defaults to `NULL`.

Examples

```
### Grid layout forms

# Use responsive arguments to adjust form layouts based on viewport size.
# Be sure to adjust the size of your browser window between large and small.
```

```
card(
  formRow(
    formGroup(
      width = c(md = 6),  # <-
      label = "Username",
      textInput(
        id = "user"
      )
    ),
    formGroup(
      width = c(md = 6),  # <-
      label = "Password",
      textInput(
        type = "password",
        id = "pass"
      )
    )
  ),
  formGroup(
    label = "Username",
    groupTextInput(
      id = "username",
      left = "@"
    )
  ),
  buttonInput(
    id = "go",
    label = "Go!"
  ) %>%
    background("blue")
) %>%
  margin(3) %>%
  background("grey")
```

`formInput` *Form inputs*

Description

Form inputs are a new reactive input. Form inputs are an alternative to shiny's submit buttons. A form input is comprised of any number of inputs. The value of these inputs will *not* change until a form submit button within the form input is clicked. A form input's reactive value depends on the clicked form submit button. This allows you to distinguish between different form submission types, think "login" versus "register".

A form submit button, `formSubmit()`, is a special type of button used to control form input submission. A form input and its child reactive inputs will *never* update if a form submit button is not included in `...` passed to `formInput()`.

Usage

```
formInput(id, ..., inline = FALSE)

formSubmit(label, value = label, ...)
```

Arguments

id	A character string specifying the id of the reactive input.
...	Any number of unnamed arguments passed as child elements to the parent form element or named arguments passed as HTML attributes to the parent element. At least one `formSubmit()` must be included.
inline	One of `TRUE` or `FALSE`, if `TRUE` the form and its child elements are rendered in a horizontal row, defaults to `FALSE`. On small viewports, think mobile device, `inline` intentionally has no effect and the form will span multiple lines.
label	A character string specifying the label of the form submit button.
value	A character string specifying the value of the form submit button and the value of the form input when the button is clicked, defaults to `label`.

Details

When `inline` is `TRUE` you may want to adjust the right margin of each child element for viewports larger than mobile, `margin(<TAG>, right = c(sm = 2))`, see `margin()`. You only need to apply extra space for larger viewports because inline forms do not take effect on small viewports.

Frozen inputs with scope

```
ui <- container(
  formInput(
    id = "login",
    formGroup(
      label = "Username",
      textInput(
        id = "user"
      )
    ),
    formGroup(
      label = "Password",
      textInput(
        type = "password",
        id = "pass"
      )
    ),
    formSubmit(
      label = "Login",
      value = "login"
    )
  )
```

```
)

server <- function(input, output) {
  # Will not react until the form submit button is
  # clicked.
  observe({
    print(input$email)
    print(input$password)
  })
}

shinyApp(ui, server)
```

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, fileInput, listGroupInput, menuInput, navInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### A simple form

card(
  header = "Please pick a flavor",
  formInput(
    id = "form1",
    formGroup(
      label = "Ice creams",
      radioInput(
        id = "flavor",
        choices = c("Mint", "Moose tracks", "Marble"),
      )
    ),
    formSubmit("Make choice", "choice") %>%
      background("teal")
  )
) %>%
  border("teal") %>%
  width(50)
```

---

height *Height*

Description

Utility function to change a tag element's height. Height is specified relative to the font size of page (browser default is 16px), relative to their parent element, or relative to the element's content.

Usage

```
height(tag, size)
```

Arguments

tag	A tag element.
size	A character string or number specifying the height of the tag element. Possible values: An integer between 1 and 20, in which case the height of the element is relative to the font size of the page. "full", in which case the element's height is a percentage of its parent's height. The height of the parent element must also be specified. Percentages do not account for margins or padding and may cause an element to extend beyond its parent. "auto", in which case 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. "screen", in which case the element's height is determined by the height of the viewport.

See Also

Other design utilities: active, affix, background, border, display, float, font, padding, scroll, shadow, width

Examples

```
### Numeric values

div(
  lapply(
    seq(2, 20, by = 2),
    function(h) {
      div(h) %>%
        width(2) %>%
        height(h) %>%  # <-
        padding(l = 1) %>%
        border("black")
    }
  )
) %>%
  display("flex") %>%
  flex(justify = "between")
```

img *Responsive images and figures*

Description

A small update to `tags$img` and `tags$figure`. Create responsive images with `img`. `figure` has specific arguments for an image child element and image caption.

Usage

```
img(src, ...)

figure(image, caption = NULL, ...)
```

Arguments

`src`	A character string specifying the source of the image.
`...`	Additional tag elements or named arguments passed as HTML attributes to the parent element.
`image`	An `<img>` tag, typically a call to `img`.
`caption`	A character string specifying the image caption, defaults to `NULL`.

See Also

Other components: `alert`, `badge`, `blockquote`, `card`, `collapsePane`, `d1`, `dropdown`, `jumbotron`, `modal`, `navContent`, `popover`, `pre`, `toast`

jumbotron *Jumbotron*

Description

A showcase banner, good for front or splash pages.

Usage

```
jumbotron(..., title = NULL, subtitle = NULL)
```

Arguments

`...`	Tag elements passed as child elements or named arguments passed as HTML attributes to the parent element.
`title`	A character string specifying a title for the jumbotron, defaults to `NULL`, in which case a title is not added.
`subtitle`	A character string specifying a subtitle for the jumbotron, defaults to `NULL`, in which case a subtitle is not added.

See Also

Other components: alert, badge, blockquote, card, collapsePane, d1, dropdown, img, modal, navContent, popover, pre, toast

Examples

```
### Landing page welcome

jumbotron(
  title = "Welcome, welcome!",
  subtitle = "Here we are showcasing the very showcase itself.",
  tags$p(
    "Now let's talk more about that superb new feature."
  )
)
```

`listGroupInput` *List group inputs*

Description

List group inputs are an actionable list of items. They behave similarly to checkboxes or radios, that is, users may select one or more items from the list. However, list group items may include highly variable content.

Usage

```
listGroupInput(id, choices = NULL, values = choices, selected = NULL,
  ..., layout = "vertical", flush = FALSE)

updateListGroupInput(id, choices = NULL, values = choices,
  selected = NULL, enable = NULL, disable = NULL,
  session = getDefaultReactiveDomain())
```

Arguments

`id`	A character string specifying the id of the reactive input.
`choices`	A vector of character strings or list of tag elements specifying the content of the list group's items.
`values`	A character vector specifying the values of the list items, defaults to `choices`.
`selected`	One or more of `values` specifying which choices are selected by default, defaults to `NULL`, in which case no choice is selected.
`...`	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.

layout	A responsive argument. One of "vertical" or "horizontal" specifying how list items are laid out, defaults to "vertical". Note, if layout is "horizontal" and the flush argument is ignored.
flush	One of TRUE or FALSE specifying if the list group is rendered without an outside border, defaults to FALSE. Removing the list group border is useful when rendering a list group inside a custom parent container, e.g. inside a card().
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

Navigation with a list group

A list group can also control a set of panes. Be sure to set multiple = FALSE. This layout is reminiscent of a table of contents.

```
ui <- container(
  columns(
    column(
      width = 3,
      listGroupInput(
        id = "nav",
        selected = "pane1",
        choices = c(
          "Item 1",
          "Item 2",
          "Item 3"
        ),
        values = c(
          "pane1",
          "pane2",
          "pane3"
        )
      )
    ),
    column(
      navContent(
        navPane(
          id = "pane1",
          p("Pellentesque tristique imperdiet tortor.")
        ),
        navPane(
          id = "pane2",
          p("Sed bibendum. Donec pretium posuere tellus.")
        ),
        navPane(
          id = "pane3",
```

```
            p("Pellentesque tristique imperdiet tortor.")
          )
        )
      )
    )
  )

  server <- function(input, output) {
    observeEvent(input$nav, {
      showPane(input$nav)
    })
  }

  shinyApp(ui, server)
```

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, fileInput, formInput, menuInput, navInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### An actionable list group

listGroupInput(
  id = "list1",
  choices = paste("Item", 1:5)
)

### List group within a card

card(
  header = h6("Pick an item"),
  listGroupInput(
    id = "list2",
    flush = TRUE,
    choices = paste("Item", 1:5),
  )
)

### Horizontal list group

listGroupInput(
  id = "list3",
  choices = paste("Item", 1:4),
  layout = "horizontal"
)
```

menuInput *Menu inputs*

Description

A togglable dropdown menu input. Menu inputs may be used as standalone reactive inputs or within a navInput(). For building custom, more complex dropdown elements please see dropdown().

Usage

```
menuInput(id, label, choices = NULL, values = choices,
  selected = NULL, ..., direction = "down", align = "left")

updateMenuInput(id, choices = NULL, values = choices,
  selected = NULL, enable = NULL, disable = NULL,
  session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the reactive input.
label	A character string or tag element specifying the label of the menu's toggle button.
choices	A character vector specifying the choice text of the menu's items.
values	A character vector specifying the values of the menu's items, defaults to choices.
selected	One or more of values specifying which choices are selected by default, defaults to NULL, in which case no choices are initially selected.
...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
direction	One of "up", "right", "down", or "left" specifying which direction the menu opens, defaults to "down".
align	One or "right" or "left" specifying which side of the toggle button the menu aligns to, defaults to "left".‘
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, fileInput, formInput, listGroupInput, navInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### A simple menu

menuInput(
  id = "menu1",
  label = "Menu",
  choices = c(
    "Choice 1",
    "Choice 2",
    "Choice 3"
  )
)

### Use in navigation

navInput(
  id = "nav1",
  choices = list(
    "Tab 1",
    menuInput(
      id = "navOptions",
      label = "Tab 2",
      choices = c(
        "Option 1",
        "Option 2",
        "Option 3"
      )
    ),
    "Tab 3",
    "Tab 4"
  ),
  values = paste0("tab", 1:4)
)
```

`modal` *Modal dialogs*

Description

Modals are a flexible alert window, which disable interaction with the page behind them. Modals may include inputs, buttons, or simply text. Each modal may be assigned an `id`. By default `hideModal()` will hide all modals, but you may instead specify a modal's `id` in which case only that modal is closed. Additionally, when `id` is not `NULL` observers and reactives may watch for the modal's close event.

Usage

```
modal(id, title, ..., footer = NULL, center = FALSE, size = "md",
  fade = TRUE)

showModal(modal, session = getDefaultReactiveDomain())

closeModal(id = NULL, session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the modal, when closed `input[[id]]` is set to `TRUE`.
title	A character string or tag element specifying the title of the modal.
...	Unnamed values passed as tag elements to the body of the modal. or named values passed as HTML attributes to the body element of the modal.
footer	A character string or tag element specifying the footer of the modal.
center	One of `TRUE` or `FALSE` specifying whether the modal is vertically centered on the page, defaults to `FALSE`.
size	One of `"sm"` (small), `"md"` (medium), `"lg"` (large), or `"xl"` (extra large) specifying the relative width of the modal, defaults to `"md"`.
fade	One of `TRUE` or `FALSE` specifying if the modal fades in when shown and fades out when closed, defaults to `TRUE`.
modal	A modal tag element created using `modal()`.
session	A reactive context, defaults to `getDefaultReactiveDomain()`.

Example application

```
ui <- container(
  buttonInput(
    id = "open",
    "Open modal",
    icon("plus")
  )
)

server <- function(input, output) {
  modal1 <- modal(
    title = "A simple modal",
    p(
      "Cras mattis consectetur purus sit amet fermentum.",
      "Cras justo odio, dapibus ac facilisis in, egestas",
      "eget quam. Morbi leo risus, porta ac consectetur",
      "ac, vestibulum at eros."
    )
  )
```

```
  observeEvent(input$open, ignoreInit = TRUE, {
    showModal(modal1)
  })
}

shinyApp(ui, server)
```

See Also

Other components: alert, badge, blockquote, card, collapsePane, d1, dropdown, img, jumbotron, navContent, popover, pre, toast

Examples

```
### Simple modal

modal(
  id = NULL,
  title = "Title",
  p("Cras placerat accumsan nulla.")
)

### Modal with container body

modal(
  id = NULL,
  size = "lg",
  title = "More complex",
  container(
    columns(
      column("Cras placerat accumsan nulla."),
      column("Curabitur lacinia pulvinar nibh."),
      column(
        "Aliquam posuere.",
        "Praesent fermentum tempor tellus."
      )
    )
  )
)
```

---

navbar *Navigation bar*

Description

Add a navigation bar to your application with `navbar()`. Navigation bars may include a tab toggle (useful for multi-page applications), inline forms (perhaps a search feature or login item), or

character strings to add simple text. Navbars are responsive and will collapse on small screens, think mobile device. A button is automatically added to toggle between the collapsed and expanded states.

Usage

```
navbar(..., brand = NULL, collapse = NULL)
```

Arguments

...	A tab toggle, inline forms, or text to add to include as part of the navigation bar.
brand	A tag element or text placed on the left end of the navbar, defaults to NULL, in which case nothing is added.
collapse	One of "sm", "md", "lg", "xl", or NULL specifying the breakpoint at which the navbar collapes, defaults to NULL, in which case the navbar is always expanded.

See Also

Other layout functions: column, fieldset, flex, responsive

Examples

```
### Navbar with tabs

div(
  navbar(
    brand = "Navbar",
    navInput(
      id = "tabs",
      choices = c("Home", "About", "Our process")
    ) %>%
      margin(right = "auto"),
    formInput(
      inline = TRUE,
      id = "navForm",
      textInput(
        type = "search",
        id = "site_search",
        placeholder = "Search"
      ) %>%
        margin(right = c(sm = 2)),
      formSubmit(
        label = "Search",
        value = "search"
      ) %>%
        background("amber")
    )
  ) %>%
    background("teal"),
  container(
```

```
    navContent(
      navPane(
        h3("Home")
      ),
      navPane(
        h3("About")
      ),
      navPane(
        h3("The process")
      )
    )
  )
)
```

---

navContent *Navigation panes*

Description

These functions pair with navInput(). Use navContent() and navPane() to create the pane layout. To show a new pane use showNavPane() from within an observer. showNavPane() will also hide a previously active pane. If needed you can hide an active pane with hideNavPane(). hideNavPane() is useful when you do not have a new pane to show, but want to hide the current active pane.

Usage

```
navContent(...)

navPane(id, ..., fade = TRUE)

showNavPane(id, session = getDefaultReactiveDomain())

hideNavPane(id, session = getDefaultReactiveDomain())
```

Arguments

...	For **navContent**, any number of nav panes passed as child elements to the nav parent element or named arguments passed as HTML attributes to the parent element. For **navPane**, any number of unnamed arguments passed as tag elements to the parent element or named arguments passed as HTML elements to the parent element.
id	A character string specifying the id of the nav pane.
fade	One of TRUE or FALSE specifying if the pane fades in when shown and fades out when hidden, defaults to TRUE.
session	A reactive context, defaults to getDefaultReactiveDomain().

App with pills

```
ui <- container(
  navInput(
    id = "tabs",
    choices = paste("Tab", 1:3),
    values = paste0("pane", 1:3),
    appearance = "pills"
  ),
  navContent(
    navPane(
      id = "pane1",
      "Nullam tristique diam non turpis.",
      "Cum sociis natoque penatibus et magnis dis parturient montes, ",
      "nascetur ridiculus mus.",
      "Etiam laoreet quam sed arcu.",
      "Curabitur vulputate vestibulum lorem."
    ),
    navPane(
      id = "pane2",
      "Praesent fermentum tempor tellus.",
      "Proin neque massa, cursus ut, gravida ut, lobortis eget, lacus.",
      "Phasellus lacus.",
      "Nam euismod tellus id erat."
    ),
    navPane(
      id = "pane3",
      "Nullam eu ante vel est convallis dignissim.",
      "Phasellus at dui in ligula mollis ultricies.",
      "Fusce suscipit, wisi nec facilisis facilisis, est dui ",
      "fermentum leo, quis tempor ligula erat quis odio.",
      "Donec hendrerit tempor tellus."
    )
  )
)

server <- function(input, output) {
  observeEvent(input$tabs, {
    showNavPane(input$tabs)
  })
}

shinyApp(ui, server)
```

App with dropdown

```
ui <- container(
  navInput(
    id = "tabs",
```

```
      choices = list(
        "Tab 1",
        dropdown(
          label = "Tab 2",
          buttonInput("action", "Action"),
          buttonInput("another", "Another action")
        ),
        "Tab 3"
      ),
      values = paste0("pane", 1:3),
      appearance = "tabs"
    ),
    navContent(
      navPane(
        id = "pane1",
        "Donec at pede.",
        "Pellentesque tristique imperdiet tortor.",
        "Lorem ipsum dolor sit amet, consectetuer adipiscing elit."
      ),
      navPane(
        id = "pane2",
        "Nullam tristique diam non turpis.",
        "Cras placerat accumsan nulla.",
        "Donec at pede."
      ),
      navPane(
        id = "pane3",
        "Phasellus purus.",
        "Etiam laoreet quam sed arcu.",
        "Donec pretium posuere tellus."
      )
    )
  )

  server <- function(input, output) {
    observeEvent(input$tabs, {
      showNavPane(input$tabs)
    })

    observeEvent(c(input$action, input$another), {
      if (input$action > 0 || input$another > 0) {
        showNavPane("pane2")
      }
    })
  }

  shinyApp(ui, server)
```

App with multiple sets of panes

```
ui <- container(
  navInput(
    id = "tabs",
    choices = paste("Tab", 1:3),
    values = paste0("pane", 1:3)
  ),
  columns(
    column(
      navContent(
        navPane(
          id = "pane1_1",
          "Aenean eu leo quam. Pellentesque ornare sem lacinia quam ",
          "venenatis vestibulum. Praesent commodo cursus magna, vel ",
          "scelerisque nisl consectetur et. Vivamus sagittis lacus vel ",
          "augue laoreet rutrum faucibus dolor auctor."
        ),
        navPane(
          id = "pane2_1",
          "Nullam quis risus eget urna mollis ornare vel eu leo. ",
          "Maecenas faucibus mollis interdum. Praesent commodo cursus ",
          "magna, vel scelerisque nisl consectetur et."
        ),
        navPane(
          id = "pane3_1",
          "Lorem ipsum dolor sit amet, consectetur adipiscing elit. ",
          "Vivamus sagittis lacus vel augue laoreet rutrum faucibus ",
          "dolor auctor. Etiam porta sem malesuada magna mollis euismod."
        )
      )
    ),
    column(
      navContent(
        navPane(
          id = "pane1_2",
          "Aenean eu leo quam. Pellentesque ornare sem lacinia quam ",
          "venenatis vestibulum. Praesent commodo cursus magna, vel ",
          "scelerisque nisl consectetur et. Vivamus sagittis lacus vel ",
          "augue laoreet rutrum faucibus dolor auctor."
        ),
        navPane(
          id = "pane2_2",
          "Nullam quis risus eget urna mollis ornare vel eu leo. ",
          "Maecenas faucibus mollis interdum. Praesent commodo cursus ",
          "magna, vel scelerisque nisl consectetur et."
        ),
        navPane(
          id = "pane3_2",
```

```
            "Lorem ipsum dolor sit amet, consectetur adipiscing elit. ",
            "Vivamus sagittis lacus vel augue laoreet rutrum faucibus ",
            "dolor auctor. Etiam porta sem malesuada magna mollis euismod."
          )
        )
      )
    )
  )

  server <- function(input, output) {
    observeEvent(input$tabs, {
      showNavPane(paste0(input$tabs, "_1"))
      showNavPane(paste0(input$tabs, "_2"))
    })
  }

  shinyApp(ui, server)
```

See Also

Other components: alert, badge, blockquote, card, collapsePane, d1, dropdown, img, jumbotron, modal, popover, pre, toast

Examples

```
### Examples

# Because these are server-side utilities please see the example applications
# above.
```

---

navInput *Page navigation inputs*

Description

A reactive input styled as a navigation control. The navigation input can be styled as links, tabs, or pills. A nav input is paired with navContent() and showNavPane() to create tabbed user interfaces. Observers and reactives are triggered when a nav choice or menu item is clicked. The reactive value of a nav input is NULL or a singleton character string. The value of any menus in the nav input must be retrieved with its own reactive id.

Usage

```
navInput(id, choices = NULL, values = choices,
  selected = values[[1]], ..., appearance = "links", fill = FALSE)
```

```
updateNavInput(id, choices = NULL, values = choices, selected = NULL,
  enable = NULL, disable = NULL,
  session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the reactive input.
choices	A character vector or list of tag elements specifying the navigation items of the input.
values	A character vector specifying the values of the input's chocies, defaults to choices.
selected	One of values specifying which choice is selected by default, defaults to values[[1]].
...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
appearance	One of "links", "pills", or "tabs" specifying the appearance of the nav input, defaults to "links".
fill	One of TRUE or FALSE specifying if the nav input fills the width of its parent element. If TRUE, the space is divided evenly among the nav items.
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

Including a menu

Use the reactive id of any nav menus to know when a menu item is clicked.

```
ui <- navInput(
  id = "navigation",
  choices = list(
    "Item 1",
    "Item 2",
    menuInput(
      id = "navMenu",  # <-
      label = "Item 3",
      choices = c("Choice 1", "Choice 2")
    )
  ),
  values = c("item1", "item2", "item3")
)

server <- function(input, output) {
  observeEvent(input$navMenu, {
    cat(paste("Click menu item:", input$navMenu, "\n"))
  })
}

shinyApp(ui, server)
```

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, fileInput, formInput, listGroupInput, menuInput, radioInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### Nav styled as tabs

navInput(
  id = "tabs1",
  choices = c(
    "Tab 1",
    "Tab 2",
    "Tab 3"
  ),
  selected = "Tab 1",
  appearance = "tabs"
)

### Nav styled as pills

navInput(
  id = "tabs2",
  choices = paste("Tab", 1:3),
  selected = "Tab 1",
  appearance = "pills"
)

### Nav with dropdown

navInput(
  id = "tabs3",
  choices = list(
    "Tab 1",
    menuInput(
      id = "menu1",
      label = "Tab 2",
      choices = c(
        "Action",
        "Another action"
      )
    ),
    "Tab 2"
  ),
  values = c("tab1", "tab2", "tab3")
)

### Full width nav input

navInput(
```

```
  id = "tabs4",
  choices = paste("Tab", 1:5),
  values = paste0("tab", 1:5),
  appearance = "pills",
  fill = TRUE
)

### Centering a nav input

navInput(
  id = "tabs5",
  choices = paste("Tab", 1:3)
) %>%
  flex(justify = "center")
```

---

padding *Margin and padding*

Description

Use the `margin()` and `padding()` utilities to change the margin or padding of a tag element. The margin of a tag element is the space outside and around the tag element, its border, and its content. The padding of a tag element is the space between the tag element's border and its content or child elements. All arguments default to `NULL`, in which case they are ignored.

Usage

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

margin(tag, all = NULL, top = NULL, right = NULL, bottom = NULL,
  left = NULL)
```

Arguments

`tag`	A tag element.
`all, top, right, bottom, left`	A responsive argument. For **padding()**, one of `0:5` or `"auto"` specifying padding for one or more sides of the tag element. 0 removes all inner space and 5 adds the most space. For **margin()**, one of `-5:5` or `"auto"` specifying a margin for one or more sides of the tag element. 0 removes all outer space, 5 adds the most space, and negative values will consume space pulling the element in that direction.

See Also

Other design utilities: `active`, `affix`, `background`, `border`, `display`, `float`, `font`, `height`, `scroll`, `shadow`, `width`

Examples

```
### Centering an element

# 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(.., right = "auto", left = "auto")`.

div(
  "Nam a sapien. Integer placerat tristique nisl.",
  style = "height: 100px; width: 200px;"
) %>%
  margin(top = 2, r = "auto", b = 2, l = "auto") %>%  # <-
  padding(3) %>%
  background("indigo")

### Building an inline form

# Inline form elements automatically use the flex layout providing you a
# means of creating condensed sets of inputs. However, you may need to adjust
# the spacing of the form's child elements.

# Here is an inline form without any additional spacing applied.

formInput(
  id = "form1",
  inline = TRUE,
  textInput(
    id = "name",
    placeholder = "Full name"
  ),
  groupTextInput(
    id = "username",
    left = "@",
    placeholder = "Username"
  ),
  checkboxInput(
    id = "remember",
    choice = "Remember me"
  ),
  formSubmit("Login", "login")
)

# Without any adjustments the layout is not great. But, with some styling we
# can make this form sparkle. Notice we are also adjusting the default submit
# button added to the form input.

formInput(
  id = "form2",
  inline = TRUE,
  textInput(
    id = "name",
```

```
    placeholder = "Full name"
  ) %>%
    margin(r = c(sm = 2), b = 2),  # <-
  groupTextInput(
    id = "username",
    left = "@",
    placeholder = "Username"
  ) %>%
    margin(r = c(sm = 2), b = 2),  # <-
  checkboxInput(
    id = "remember",
    choice = "Remember me"
  ) %>%
    margin(r = c(sm = 2), b = 2),  # <-
  formSubmit(
    label = "Login",
    value = "login"
  ) %>%
    margin(b = 2)  # <-
)
```

`popover` *Popovers*

Description

Popovers are small windows of content associated with a tag element. Use `popover()` to construct the element and `showPopover()` to add it to any tag element with an HTML id. Popovers are great for explaining inputs and giving hints to the users. Popovers are hidden with `closePopover()`.

Usage

```
popover(..., title = NULL)

showPopover(id, popover, placement = "top", duration = NULL,
  session = getDefaultReactiveDomain())

closePopover(id, session = getDefaultReactiveDomain())
```

Arguments

`...`	Character strings or tag elements specifying the content of the popover or additional named arguments passed as HTML attributes to the parent element.
`title`	A character string specifying a title for the popover, defaults to `NULL`, in which case a title is not added.
`id`	A character string specifying the id of a popover's target tag element.
`popover`	The popover element to show, typically a call to `popover()`.

placement	One of "top", "left", "bottom", or "right" specifying where the popover is positioned relative to the target tag element indicated by id, defaults to "top".
duration	A positive integer specifying the duration of the popover in seconds or NULL, in which case the popover is not automatically removed. When NULL the popover must be removed with closePopover().
session	A reactive context, defaults to getDefaultReactiveDomain().

Example application

```
ui <- container(
  buttonInput("showHelp", "Help!"),
  div(
    id = "textBlock1",
    "Sociis natoque penatibus et magnis"
  ) %>%
    padding(3)
) %>%
  display("flex") %>%
  flex(justify = "around")

server <- function(input, output) {
  observeEvent(input$showHelp, ignoreInit = TRUE, {
    showPopover(
      target = "textBlock1",
      popover(title = "Hint", "I am a <div> element!"),
      placement = "bottom",
      duration = 4
    )
  })
}

shinyApp(ui, server)
```

See Also

Other components: alert, badge, blockquote, card, collapsePane, d1, dropdown, img, jumbotron, modal, navContent, pre, toast

Examples

```
### Examples

# Please see example application above.
```

pre *Scrollable code snippets*

Description

The pre function adds a maximum height and scroll bar to the standard <pre> element.

Usage

```
pre(...)
```

Arguments

...	Text, tag elements, or named arguments passed as HTML attributes to the tag.

See Also

Other components: alert, badge, blockquote, card, collapsePane, d1, dropdown, img, jumbotron, modal, navContent, popover, toast

Examples

```
### Simple example

pre(
  "shinyApp(",
  "  ui = container(",
  "    columns(",
  "      column(",
  "      ",
  "      )",
  "    )",
  "  )",
  "  server = function(input, output) {",
  "  ",
  "  }",
  ")"
)
```

radiobarInput *Radiobar inputs*

Description

A stylized group of radio inputs. A radiobar input is similar to a button group, but with a checked or highlighted stated. Additionally, only one value at most may be selected at any given time.

Usage

```
radiobarInput(id, choices, values = choices, selected = values[[1]],
  ...)

updateRadiobarInput(id, choices = NULL, values = choices,
  selected = NULL, enable = NULL, disable = NULL,
  session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the reactive input.
choices	A character vector or list of tag elements specifying the labels of the input's choices.
values	A vector specifying the values of the input's choices, defaults to choices.
selected	One of values specifying the input's default selected choice, defualts to values[[1]].
...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, fileInput, formInput, listGroupInput, menuInput, navInput, radioInput, rangeInput, selectInput, textInput

Examples

```
### Radiobars

radiobarInput(
  id = "radiobar1",
  choices = c(
```

```
    "fusce sagittis",
    "libero non molestie",
    "magna orci",
    "ultrices dolor"
  ),
  selected = "ultrices dolor"
) %>%
  background("grey")
```

---

radioInput *Radio inputs*

Description

A stylized radio input. A reactive input with multiple choices where only one choice and value at most may be selected.

Usage

```
radioInput(id, choices = NULL, values = choices,
  selected = values[[1]], ..., inline = FALSE)

updateRadioInput(id, choices = NULL, values = choices,
  selected = NULL, inline = FALSE, enable = NULL, disable = NULL,
  valid = NULL, invalid = NULL, session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the reactive input.
choices	A character vector or list of tag elements specifying the input's choices.
values	A character vector, list of character strings, vector of values to coerce to character strings, or list of values to coerce to character strings specifying the values of the radio input's choices, defaults to choices.
selected	One of values indicating the default selected value of the radio input, defaults to NULL, in which case the first choice is selected by default.
...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
inline	If TRUE, the radio input renders inline, defaults to FALSE, in which case the radio controls render on separate lines.
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
valid	A character string specifying a message to the user indicating how the input's value is valid, defaults to NULL.

invalid	A character string specifying a message to the user indicating how the input's value is invalid, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, fileInput, formInput, listGroupInput, menuInput, navInput, radiobarInput, rangeInput, selectInput, textInput

Examples

```
### Out-of-the-box radios

radioInput(
  id = "radio1",
  choices = c(
    "Vehicula adipiscing mattis",
    "Magna nullam",
    "Aenean venenatis",
    "Tristique quam porta"
  )
)

### Inline radio input

radioInput(
  id = "radio2",
  choices = c(
    "Choice 1",
    "Choice 2",
    "Choice 3"
  ),
  inline = TRUE  # <-
)
```

---

rangeInput *Range input*

Description

rangeInput() creates a simple numeric range input.

Usage

```
rangeInput(id, min = 0, max = 100, default = min, step = 1, ...)

updateRangeInput(id, value = NULL, enable = NULL, disable = NULL,
  session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the reactive input.
min	A number specifying the minimum value of the input, defaults to 0.
max	A number specifying the maximum value of the input, defaults to 100.
default	A number between min and max specifying the default value of the input, defaults to min.
step	A number specifying the interval step of the input, defaults to 1.
...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
value	A number specifying a new value for the input, defaults to NULL.
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().

Details

The sophistication of this input will improve as browsers adopt the latest HTML standards.

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, fileInput, formInput, listGroupInput, menuInput, navInput, radioInput, radiobarInput, selectInput, textInput

Examples

```
### Range inputs

# Select from a range of numeric values.

rangeInput(id = "range1")

### Custom step

rangeInput(id = "range2", step = 10)
```

replaceContent *Dynamic content*

Description

An application may require dynamic content. This content may be quite simple. The content could also be quite variable. These tools are an alternative to the standard `output` related `render*()` functions.

Usage

```
replaceContent(id, ..., session = getDefaultReactiveDomain())

removeContent(id, session = getDefaultReactiveDomain())
```

Arguments

`id`	A character string specifying a reactive id.
`...`	Additional named arguments passed as HTML attributes to the parent element or unnamed arguments passed as the new contents of the output element.
`session`	A reactive context, defaults to `getDefaultReactiveDomain()`.

Details

These functions are experimental and are subject to change. Additionally, they may be moved from this package entirely.

responsive *Understanding responsive arguments*

Description

A responsive argument may be a single value or a named list. Possible names includes `default` or `xs`, `sm`, `md`, `lg`, and `xl`. 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 will default to `NULL` in which case no corresponding style is applied.

Responsive arguments allow you to apply styles to tag elements based on the size of the viewport. 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 above to apply a style for viewports of that size and larger. For example, specifying `list(default = x, md = y)` will apply `x` on extra small and small viewports, but for medium, large, and extra large viewports `y` is applied.

Styles for larger viewports take precedance. See below for details about each breakpoint.

extra small

How: pass a single value, use name `xs`, or use name `default`.

When: the style is always applied, unless supplanted by a style for any other viewport size.

small

How: use name `sm`.

When: the style is applied when the viewport is at least 576px wide, think landscape phones.

medium

How: use name `md`.

When: the style is applied when the viewport is at least 768px wide, think tablets.

large

How: use name `lg`.

When: the style is applied when the viewport is at least 992px wide, think laptop or smaller desktops.

extra large

How: use name `xl`.

When: the style is applied when the viewport is at least 1200px wide, think large desktops.

See Also

Other layout functions: `column`, `fieldset`, `flex`, `navbar`

`scroll` *Vertical and horizontal scroll*

Description

Many of the applications you build depsite a complex layout will still fit onto a single page. To help scroll long content alongside shorter content use the `scroll()` utility function.

Usage

```
scroll(tag, direction = "vertical")
```

Arguments

`tag`	A tag element.
`direction`	One of `"horizontal"` or `"vertical"` specifying which direction to scroll overflowing content, defaults to `"vertical"`, in which case the content may croll up and down.

See Also

Other design utilities: `active`, `affix`, `background`, `border`, `display`, `float`, `font`, `height`, `padding`, `shadow`, `width`

Examples

```
### A simple scroll

div(
  lapply(
    rep("Integer placerat tristique nisl.", 20),
    . %>% p() %>% margin(bottom = 2)
  )
) %>%
  height(20) %>%
  border("black") %>%
  scroll()
```

`selectInput` *Select inputs*

Description

Create a select input. Select elements typically appear as a simple menu of choices and may have one or more selected values, see the `multiple` argument. A group select input is a select input with one or two additional components. These addon components are used to change the reactivity or value of the input, see Details for more information.

Usage

```
selectInput(id, choices = NULL, values = choices,
  selected = values[[1]], ...)

updateSelectInput(id, choices = NULL, values = choices,
  selected = NULL, enable = NULL, disable = NULL, valid = NULL,
  invalid = NULL, session = getDefaultReactiveDomain())

groupSelectInput(id, choices, values = choices, selected = values[[1]],
  ..., left = NULL, right = NULL)

updateGroupSelectInput(id, choices = NULL, values = choices,
  selected = NULL, enable = NULL, disable = NULL, valid = NULL,
  invalid = NULL, session = getDefaultReactiveDomain())
```

Arguments

`id`	A character string specifying the id of the reactive input.
`choices`	A character vector specifying the input's choices.
`values`	A character vector specifying the values of the input's choices, defaults to `choices`.
`selected`	One of `values` indicating the default value of the input, defaults to `values[[1]]`.

`...` Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.

`enable` One of `values` specifying particular choices to enable or `TRUE` specifying the entire input is enabled, defaults to `NULL`.

`disable` One of `values` specifying particular choices to disable or `TRUE` specifying the entire input is disabled, defaults to `NULL`.

`valid` A character string specifying a message to the user indicating how the input's value is valid, defaults to `NULL`.

`invalid` A character string specifying a message to the user indicating how the input's value is invalid, defaults to `NULL`.

`session` A reactive context, defaults to `getDefaultReactiveDomain()`.

`left, right` A character vector specifying static addons or `buttonInput()` or `dropdown()` elements specifying dynamic addons. Addons affect the reactive value of the group input, see the Details section below for more information.

`left` **is character or** `right` **is character**

If `left` or `right` are character vectors, then the group input functions like a text input. The value will update and trigger a reactive event when the text box is modified. The group input's reactive value is the concatention of the static addons specified by `left` or `right` and the value of the text input.

`left` **is button or** `right` **is button**

The button does not change the value of the group input. However, the input no longer triggers event when the text box is updated. Instead the value is updated when a button is clicked. Static addons are still applied to the group input value.

`left` **is a dropdown or** `right` **is a dropdown**

The value of the group input does chance depending on the clicked dropdown menu item. The value of the input group is the concatentation of the dropdown input value, the value of the text input, and any static addons.

See Also

Other inputs: `buttonGroupInput`, `buttonInput`, `checkbarInput`, `checkboxInput`, `chipInput`, `fileInput`, `formInput`, `listGroupInput`, `menuInput`, `navInput`, `radioInput`, `radiobarInput`, `rangeInput`, `textInput`

Examples

```
### Simple select input

selectInput(
  id = "select1",
  choices = c(
    "Choice 1",
    "Choice 2",
    "Choice 3"
  ),
  values = list(1, 2, 3)
```

```
)

### Group select input

groupSelectInput(
  id = "select2",
  choices = 1:5,
  left = "$",
  right = ".00"
) %>%
  width(10)
```

---

shadow *Shadows*

Description

The shadow utility applies a shadow to a tag element. Elements with a shadow may appear to pop off the page. The material design set of components, used on Android and for Google applications, commonly uses shadowing. Although "none" is an allowed size, most elements do not have a shadow by default.

Usage

```
shadow(tag, size = "regular")
```

Arguments

tag	A tag element.
size	One of "none", "small", "regular", or "large" specifying the amount of shadow added, defaults to "regular".

See Also

Other design utilities: active, affix, background, border, display, float, font, height, padding, scroll, width

Examples

```
### Styling a navbar

div(
  navbar(brand = "Navbar") %>%
    background("cyan") %>%
    shadow("small") %>%
    margin(bottom = 3),
  p(
```

```
    "Cras mattis consectetur purus sit amet fermentum. Donec sed ",
    "odio dui. Lorem ipsum dolor sit amet, consectetur adipiscing ",
    "elit. Aenean eu leo quam. Pellentesque ornare sem lacinia quam ",
    "venenatis vestibulum."
  )
)

### Different shadows

div(
  lapply(
    c("small", "regular", "large"),
    shadow,
    tag = div() %>%
      padding(5) %>%
      margin(2)
  )
) %>%
  display("flex")
```

`textInput` *Text inputs*

Description

A text input. A group text input is an alternative text input. The group text input allows you to include static prefixes or buttons with a standard text input.

`numberInput()` is a simple wrapper around `textInput()` with `type` set to `"number"` and explicit arguments for specifying a min value, max value, and the step amount. Use `updateTextInput()` to update a number input.

Usage

```
textInput(id, value = NULL, placeholder = NULL, ..., type = "text")

numberInput(id, value = NULL, placeholder = NULL, ..., min = NULL,
  max = NULL, step = 1)

updateTextInput(id, value = NULL, enable = NULL, disable = NULL,
  valid = NULL, invalid = NULL, session = getDefaultReactiveDomain())

groupTextInput(id, value = NULL, placeholder = NULL, ...,
  type = "text", left = NULL, right = NULL)

updateGroupTextInput(id, value = NULL, enable = NULL, disable = NULL,
  valid = NULL, invalid = NULL, session = getDefaultReactiveDomain())
```

Arguments

id	A character string specifying the id of the reactive input.
value	A character string or a value coerced to a character string specifying the default value of the textual input.
placeholder	A character string specifying placeholder text for the input, defaults to NULL, in which case there is no placeholder text.
...	Additional named arguments passed as HTML attributes to the parent element or tag elements passed as child elements to the parent element.
type	One of "color", "date", "datetime-local", "email", "month", "number", "password", "search", "tel", "text", "time", "url" or "week" specifying the type of text input, defaults to "text". For details on a particular type please see https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input.
min	A number specifying the minimum allowed value of the number input, defaults to NULL.
max	A number specifying the maximum allowed value of the number input, defaults to NULL.
step	A number specifying the increment step of the number input, defaults to 1.
enable	One of values specifying particular choices to enable or TRUE specifying the entire input is enabled, defaults to NULL.
disable	One of values specifying particular choices to disable or TRUE specifying the entire input is disabled, defaults to NULL.
valid	A character string specifying a message to the user indicating how the input's value is valid, defaults to NULL.
invalid	A character string specifying a message to the user indicating how the input's value is invalid, defaults to NULL.
session	A reactive context, defaults to getDefaultReactiveDomain().
left	A character vector specifying static addons or buttonInput() or dropdown() elements specifying dynamic addons. Addons affect the reactive value of the group input, see the Details section below for more information. **left is character or right is character** If left or right are character vectors, then the group input functions like a text input. The value will update and trigger a reactive event when the text box is modified. The group input's reactive value is the concatention of the static addons specified by left or right and the value of the text input. **left is button or right is button** The button does not change the value of the group input. However, the input no longer triggers event when the text box is updated. Instead the value is updated when a button is clicked. Static addons are still applied to the group input value. **left is a dropdown or right is a dropdown** The value of the group input does chance depending on the clicked dropdown menu item. The value of the input group is the concatentation of the dropdown input value, the value of the text input, and any static addons.

right — A character vector specifying static addons or buttonInput() or dropdown() elements specifying dynamic addons. Addons affect the reactive value of the group input, see the Details section below for more information.

left is character or right is character

If left or right are character vectors, then the group input functions like a text input. The value will update and trigger a reactive event when the text box is modified. The group input's reactive value is the concatention of the static addons specified by left or right and the value of the text input.

left is button or right is button

The button does not change the value of the group input. However, the input no longer triggers event when the text box is updated. Instead the value is updated when a button is clicked. Static addons are still applied to the group input value.

left is a dropdown or right is a dropdown

The value of the group input does chance depending on the clicked dropdown menu item. The value of the input group is the concatentation of the dropdown input value, the value of the text input, and any static addons.

See Also

Other inputs: buttonGroupInput, buttonInput, checkbarInput, checkboxInput, chipInput, fileInput, formInput, listGroupInput, menuInput, navInput, radioInput, radiobarInput, rangeInput, selectInput

Examples

```
### Default text input

textInput(id = "text")

### Default number input

numberInput(id = "num1")

### Specify `min`, `max`, and `step`

numberInput(
  id = "num2",
  min = 1,
  max = 10,
  step = 2
)
```

---

toast *Toasts*

Description

Send notifications to the user. Create notification elements, toasts, with the `toast()` function. Display toasts with `showToast()` and remove all active toasts with `closeToast()`.

Usage

```
toast(header, ...)

showToast(toast, duration = 4, action = NULL,
  session = getDefaultReactiveDomain())

closeToast(session = getDefaultReactiveDomain())
```

Arguments

`header`	A character string or tag element specifying a header for the toast, defaults to `NULL`. A close button is always included in the header.
`...`	Any number of character strings or tag elements to include in the body of the toast. Any number of named arguments passed as HTML attributes to the parent element.
`toast`	A toast element, typically built with `toast()`.
`duration`	A positive integer or `NULL` specifying the duration of the toast in seconds by default a toast is removed after 4 seconds. If `NULL` the toast is not automatically removed.
`action`	A character string specifying a reactive id. If specified, the hiding or closing of the toast will set the reactive id `action` to `TRUE`.
`session`	A reactive context, defaults to `getDefaultReactiveDomain()`.

Showing notifications

```
ui <- container(
  buttonInput(
    id = "show",
    label = "Show notification"
  ) %>%
    margin(3)
)

server <- function(input, output) {
  observeEvent(input$show, {
    showToast(
      toast(
        list(
          span("Notification") %>%
            margin(right = "4"),
          span(strftime(Sys.time(), "%H:%M")) %>%
```

```
            margin(right = 1)
        ),
        "This is notification ", input$show
      ) %>%
        margin(right = 2, top = 2)
    )
  })
}

shinyApp(ui, server)
```

Reacting to notifications

When a notification is not automatically closed you may want to know when the notification is manually closed.

```
ui <- container(
  buttonInput(
    id = "show",
    label = "Show notification"
  ) %>%
    margin(3)
)

server <- function(input, output) {
  observeEvent(input$show, {
    showToast(
      action = "undo",
      duration = NULL,
      toast(
        tags$strong("Close") %>%
          margin(right = "auto"),
        "When closing this notification, see the console"
      ) %>%
        margin(right = 2, top = 2)
    )
  })

  observeEvent(input$undo, {
    print("The notification was closed")
  })
}

shinyApp(ui, server)
```

See Also

Other components: alert, badge, blockquote, card, collapsePane, d1, dropdown, img, jumbotron, modal, navContent, popover, pre

Examples

```
### A simple toast

# The `"fade"` and `"show"` classes have been added for the sake of
# these examples.

toast(
  class = "fade show",
  header = div("Header") %>%
    margin(right = "auto"),
  "Hello, world!"
)

### Styling pieces of a toast

toast(
  class = "fade show",
  list(
    div("Notification") %>%
      font(weight = "bold") %>%
      margin(right = "auto"),
    tags$small("1 min ago")
  ),
  "Hello, world!"
)
```

`width` *Width*

Description

Utility function to change a tag element's width. Widths are specified relative to the font size of page (browser default is 16px), relative to their parent element (i.e. 1/2 the width of their parent), or relative to the element's content.

Usage

```
width(tag, size)
```

Arguments

`tag`	A tag element.
`size`	A character string or number specifying the width of the tag element. Possible values: An integer between 1 and 20, in which case the width of the element is relative to the font size of the page.

"1/2", "1/3", "2/3", "1/4", "3/4", "1/5", "2/5", "3/5", "4/5", or "full", in which case the element's width is a percentage of its parent's width. The height of the parent element must be specified for percentage widths to work. Percentages do not account for margins or padding and may cause an element to extend beyond its parent.

"auto", in which case the element's width is determined by the browser. The browser will take into account the width, padding, margins, and border of the tag element's parent to keep the element from extending beyond its parent.

See Also

Other design utilities: active, affix, background, border, display, float, font, height, padding, scroll, shadow

Examples

```
### Numeric `size` values

# When specifying a numeric value the width of the element is relative to the
# default font size of the page.

div(
  lapply(
    1:20,
    width,
    tag = div() %>%
      border("black") %>%
      height(4)
  )
) %>%
  flex(
    direction = "column",
    justify = "between"
  )

### Fractional `size` values

# When specifying width as a fraction the element's width is a percentage of
# its parent's width.

div() %>%
  margin(b = 3) %>%
  background("red") %>%
  height(5) %>%
  width("1/3")  # <-
```

Index