The Family of *Helper Functions*

A small assortment of helper functions is available in the **gt** package. The various `cells_*()` functions are used for targeting cells with the locations argument in the `tab_footnote()` , `tab_style()` , and `text_transform()` functions. The `md()` and `html()` helpers can used be during label creation with the `tab_header()` , `tab_footnote()` , `tab_spanner()` , `tab_stubhead_label()` , and `tab_source_note()` functions.

`md()` : Interpret input text as Markdown-formatted text

Markdown! It's a wonderful thing. We can use it in certain places (e.g., footnotes, source notes, the table title, etc.) and expect it to render to HTML as Markdown does.

EXAMPLE

Use `exibble` to create a **gt** table. When adding a title, use the `md()` helper to use Markdown formatting.

```
exibble %>%
  dplyr::select(currency, char) %>%
  gt() %>%
  tab_header(title = md("Using *Markdown*"))
```

Using *Markdown*

currency	char
49.950	apricot
17.950	banana
1.390	coconut
65100.000	durian
1325.810	NA
13.255	fig
NA	grapefruit
0.440	honeydew

`html()` : Interpret input text as HTML-formatted text

For certain pieces of text (like in column labels or table headings) we may want to express them as raw HTML. In fact, with HTML, anything goes so it can be much more than just text.

EXAMPLE

Use `exibble` to create a **gt** table. When adding a title, use the `html()` helper to use HTML formatting.

```
exibble %>%
  dplyr::select(currency, char) %>%
  gt() %>%
  tab_header(title = html("<em>HTML</em>"))
```

HTML

currency	char
49.950	apricot
17.950	banana
1.390	coconut
65100.000	durian
1325.810	NA
13.255	fig
NA	grapefruit
0.440	honeydew

`px()` : Helper for providing a numeric value as pixels value

For certain parameters, a length value is required. Examples include the setting of font sizes (e.g., in `cell_text()`) and thicknesses of lines (e.g., in `cell_borders()`). Setting a length in pixels with `px()` allows for an absolute definition of size as opposed to the analogous helper function `pct()` .

EXAMPLE

Use `exibble` to create a **gt** table. Use the `px()` helper to define the font size for the column labels.

```
exibble %>%
  gt() %>%
  tab_style(
    style = cell_text(size = px(20)),
    locations = cells_column_labels()
  )
```

num	char	fctr	date	time	datetime	currency	row	group
1.111e-01	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950	row_1	grp_a
2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950	row_2	grp_a

num	char	fctr	date	time	datetime	currency	row	group
3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390	row_3	grp_a
4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000	row_4	grp_a
5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810	row_5	grp_b
NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255	row_6	grp_b
7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA	row_7	grp_b
8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440	row_8	grp_b

`pct()` : Helper for providing a numeric value as percentage

A percentage value acts as a length value that is relative to an initial state. For instance an 80 percent value for something will size the target to 80 percent the size of its ‘previous’ value. This type of sizing is useful for sizing up or down a length value with an intuitive measure. This helper function can be used for the setting of font sizes (e.g., in `cell_text()`) and altering the thicknesses of lines (e.g., in `cell_borders()`).

EXAMPLE

Use `exibble` to create a **gt** table. Use the `pct()` helper to define the font size for the column labels.

```
exibble %>%
  gt() %>%
  tab_style(
    style = cell_text(size = pct(75)),
    locations = cells_column_labels()
  )
```

num	char	fctr	date	time	datetime	currency	row	group
1.111e-01	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950	row_1	grp_a
2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950	row_2	grp_a
3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390	row_3	grp_a

num	char	fctr	date	time	datetime	currency	row	group
4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000	row_4	grp_a
5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810	row_5	grp_b
NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255	row_6	grp_b
7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA	row_7	grp_b
8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440	row_8	grp_b

`cells_title()` : Location helper for targeting the table title and subtitle

The `cells_title()` function is used to target the table title or subtitle when applying a footnote with `tab_footnote()` or adding custom style with `tab_style()` . The function is expressly used in each of those functions' `locations` argument.

EXAMPLE

Use `sp500` to create a **gt** table. Add a header with a title, and then add a footnote to the title with `tab_footnote()` and `cells_title()` (in `locations`).

```
sp500 %>%
  dplyr::filter(
    date >= "2015-01-05" &
      date <="2015-01-10"
  ) %>%
  dplyr::select(
    -c(adj_close, volume, high, low)
  ) %>%
  gt() %>%
  tab_header(title = "S&P 500") %>%
  tab_footnote(
    footnote = "All values in USD.",
    locations = cells_title(groups = "title")
  )
```

S&P 500[1]

date	open	close
2015-01-09	2063.45	2044.81

[1] All values in USD.

S&P 500[1]

2015-01-08	2030.61	2062.14
2015-01-07	2005.55	2025.90
2015-01-06	2022.15	2002.61
2015-01-05	2054.44	2020.58

[1] All values in USD.

`cells_stubhead()` : Location helper for targeting the table stubhead cell

The `cells_stubhead()` function is used to target the table stubhead location when applying a footnote with `tab_footnote()` or adding custom style with `tab_style()` . The function is expressly used in each of those functions' `locations` argument.

EXAMPLE

Use `pizzaplace` to create a **gt** table. Add a stubhead label and then style it with `tab_style()` and `cells_stubhead()` .

```
pizzaplace %>%
  dplyr::mutate(month = as.numeric(substr(date, 6, 7))) %>%
  dplyr::group_by(month, type) %>%
  dplyr::summarize(sold = dplyr::n()) %>%
  dplyr::ungroup() %>%
  dplyr::filter(month %in% 1:2) %>%
  gt(rowname_col = "type") %>%
  tab_stubhead(label = "type") %>%
  tab_style(
    style = cell_fill(color = "#ADD8E6"),
    locations = cells_stubhead()
  )
```

```
## `summarise()` regrouping output by 'month' (override with `.groups` argument)
```

type	month	sold
chicken	1	913
classic	1	1257
supreme	1	1044
veggie	1	1018
chicken	2	875
classic	2	1178

type	month	sold
supreme	2	964
veggie	2	944

`cells_column_spanners()` : Location helper for targeting the column spanners

The `cells_column_spanners()` function is used to target the cells that contain the table column spanners. This is useful when applying a footnote with `tab_footnote()` or adding custom style with `tab_style()` .

EXAMPLE

Use `exibble` to create a **gt** table. Add a spanner column label over three column labels and then use `tab_style()` to make the spanner label text bold.

```
exibble %>%
  dplyr::select(-fctr, -currency, -group) %>%
  gt(rowname_col = "row") %>%
  tab_spanner(
    label = "dates and times",
    id = "dt",
    columns = c(date, time, datetime)
  ) %>%
  tab_style(
    style = cell_text(weight = "bold"),
    locations = cells_column_spanners(spanners = "dt")
  )
```

			dates and times		
	num	char	date	time	datetime
row_1	1.111e-01	apricot	2015-01-15	13:35	2018-01-01 02:22
row_2	2.222e+00	banana	2015-02-15	14:40	2018-02-02 14:33
row_3	3.333e+01	coconut	2015-03-15	15:45	2018-03-03 03:44
row_4	4.444e+02	durian	2015-04-15	16:50	2018-04-04 15:55
row_5	5.550e+03	NA	2015-05-15	17:55	2018-05-05 04:00
row_6	NA	fig	2015-06-15	NA	2018-06-06 16:11
row_7	7.770e+05	grapefruit	NA	19:10	2018-07-07 05:22
row_8	8.880e+06	honeydew	2015-08-15	20:20	NA

`cells_column_labels()`: Location helper for targeting the column labels

The `cells_column_labels()` function is used to target the table’s column labels when applying a footnote with `tab_footnote()` or adding custom style with `tab_style()`.

EXAMPLE

Use `sza` to create a **gt** table. Add a header and then add footnotes to the column labels with `tab_footnote()` and `cells_column_labels()` in `locations`.

```
sza %>%
  dplyr::filter(
    latitude == 20 & month == "jan" &
      !is.na(sza)
  ) %>%
  dplyr::select(-latitude, -month) %>%
  gt() %>%
  tab_footnote(
    footnote = "True solar time.",
    locations = cells_column_labels(columns = tst)
  ) %>%
  tab_footnote(
    footnote = "Solar zenith angle.",
    locations = cells_column_labels(columns = sza)
  )
```

tst[1]	sza[2]
0700	84.9
0730	78.7
0800	72.7
0830	66.1
0900	61.5
0930	56.5
1000	52.1
1030	48.3
1100	45.5
1130	43.6
1200	43.0

[1] True solar time.

[2] Solar zenith angle.

`cells_row_groups()`: Location helper for targeting row groups

The `cells_row_groups()` function is used to target the table's row groups when applying a footnote with `tab_footnote()` or adding custom style with `tab_style()`.

EXAMPLE

Use `pizzaplace` to create a **gt** table with grouped data. Add a summary with the `summary_rows()` function and then add a footnote to the "peppr_salami" row group label with `tab_footnote()` and with `cells_row_groups()` in `locations`.

```
pizzaplace %>%
  dplyr::filter(name %in% c("soppressata", "peppr_salami")) %>%
  dplyr::group_by(name, size) %>%
  dplyr::summarize(`Pizzas Sold` = dplyr::n()) %>%
  gt(rowname_col = "size") %>%
  summary_rows(
    groups = TRUE,
    columns = "Pizzas Sold",
    fns = list(TOTAL = "sum"),
    formatter = fmt_number,
    decimals = 0,
    use_seps = TRUE
  ) %>%
  tab_footnote(
    footnote = "The Pepper-Salami.",
    cells_row_groups(groups = "peppr_salami")
  )
```

```
## `summarise()` regrouping output by 'name' (override with `.groups` argument)
```

	Pizzas Sold
peppr_salami[1]	
L	696
M	428
S	322
TOTAL	1,446
soppressata	
L	405
M	268
S	288
TOTAL	961

[1] The Pepper-Salami.

`cells_stub()` : Location helper for targeting cells in the table stub

The `cells_stub()` function is used to target the table's stub cells and it is useful when applying a footnote with `tab_footnote()` or adding custom style with `tab_style()` .

EXAMPLE

Use `sza` to create a **gt** table. Color all of the `month` values in the table stub with `tab_style()` , using `cells_stub()` in `locations` (`rows = TRUE` targets all stub rows).

```
sza %>%
  dplyr::filter(latitude == 20 & tst <= "1000") %>%
  dplyr::select(-latitude) %>%
  dplyr::filter(!is.na(sza)) %>%
  tidyr::spread(key = "tst", value = sza) %>%
  gt(rowname_col = "month") %>%
  fmt_missing(
    columns = everything(),
    missing_text = ""
  ) %>%
  tab_style(
    style = list(
      cell_fill(color = "darkblue"),
      cell_text(color = "white")
      ),
    locations = cells_stub()
  )
```

	0530	0600	0630	0700	0730	0800	0830	0900	0930	1000
jan				84.9	78.7	72.7	66.1	61.5	56.5	52.1
feb			88.9	82.5	75.8	69.6	63.3	57.7	52.2	47.4
mar			85.7	78.8	72.0	65.2	58.6	52.3	46.2	40.5
apr		88.5	81.5	74.4	67.4	60.3	53.4	46.5	39.7	33.2
may		85.0	78.2	71.2	64.3	57.2	50.2	43.2	36.1	29.1
jun	89.2	82.7	76.0	69.3	62.5	55.7	48.8	41.9	35.0	28.1
jul	88.8	82.3	75.7	69.1	62.3	55.5	48.7	41.8	35.0	28.1
aug		83.8	77.1	70.2	63.3	56.4	49.4	42.4	35.4	28.3
sep		87.2	80.2	73.2	66.1	59.1	52.1	45.1	38.1	31.3
oct			84.1	77.1	70.2	63.3	56.5	49.9	43.5	37.5
nov			87.8	81.3	74.5	68.3	61.8	56.0	50.2	45.3

	0530	0600	0630	0700	0730	0800	0830	0900	0930	1000
dec				84.3	78.0	71.8	66.1	60.5	55.6	50.9

`cells_body()` : Location helper for targeting data cells in the table body

The `cells_body()` function is used to target the data cells in the table body. The function can be used to apply a footnote with `tab_footnote()` , to add custom styling with `tab_style()` , or the transform the targeted cells with `text_transform()` .

EXAMPLE

Use `gtcars` to create a **gt** table. Add a footnote that targets a single data cell with `tab_footnote()` , using `cells_body()` in `locations` (`rows = hp == max(hp)` will target a single row in the `hp` column).

```
gtcars %>%
  dplyr::filter(ctry_origin == "United Kingdom") %>%
  dplyr::select(mfr, model, year, hp) %>%
  gt() %>%
  tab_footnote(
    footnote = "Highest horsepower.",
    locations = cells_body(
      columns = hp,
      rows = hp == max(hp))
  ) %>%
  opt_footnote_marks(marks = c("*", "+"))
```

mfr	model	year	hp
Bentley	Continental GT	2016	500
Aston Martin	DB11	2017	608
Aston Martin	Rapide S	2016	552
Aston Martin	Vanquish	2016	568
Aston Martin	Vantage	2016	430
Lotus	Evora	2017	400
Jaguar	F-Type	2016	340
McLaren	570	2016	570
Rolls-Royce	Dawn	2016	563
Rolls-Royce	Wraith	2016	624*

* Highest horsepower.

`cells_summary()`: Location helper for targeting group summary cells

The `cells_summary()` function is used to target the cells in a group summary and it is useful when applying a footnote with `tab_footnote()` or adding a custom style with `tab_style()`.

EXAMPLE

Use `countrypops` to create a **gt** table. Add some styling to the summary data cells with with `tab_style()`, using `cells_summary()` in `locations`.

```
countrypops %>%
  dplyr::filter(
    country_name == "Japan",
    year < 1970) %>%
  dplyr::select(-contains("country")) %>%
  dplyr::mutate(decade = paste0(substr(year, 1, 3), "0s")) %>%
  dplyr::group_by(decade) %>%
  gt(rowname_col = "year", groupname_col = "decade") %>%
  fmt_number(
    columns = population,
    decimals = 0
  ) %>%
  summary_rows(
    groups = "1960s",
    columns = population,
    fns = list("min", "max"),
    formatter = fmt_number,
    decimals = 0
  ) %>%
  tab_style(
    style = list(
      cell_text(style = "italic"),
      cell_fill(color = "lightblue")
      ),
    locations = cells_summary(
      groups = "1960s",
      columns = population,
      rows = 1
    )
  ) %>%
  tab_style(
    style = list(
      cell_text(style = "italic"),
      cell_fill(color = "lightgreen")
      ),
    locations = cells_summary(
      groups = "1960s",
      columns = population,
      rows = 2
    )
  )
```

	population
1960s	

	population
1960	92,500,572
1961	94,943,000
1962	95,832,000
1963	96,812,000
1964	97,826,000
1965	98,883,000
1966	99,790,000
1967	100,725,000
1968	101,061,000
1969	103,172,000
min	*92,500,572*
max	*103,172,000*

`cells_grand_summary()` : Location helper for targeting cells in a grand summary

The `cells_grand_summary()` function is used to target the cells in a grand summary and it is useful when applying a footnote with `tab_footnote()` or adding custom styles with `tab_style()` .

EXAMPLE

Use `countrypops` to create a **gt** table. Add some styling to a grand summary cell with with `tab_style()` and `cells_grand_summary()` .

```
countrypops %>%
  dplyr::filter(
    country_name == "Spain",
    year < 1970
  ) %>%
  dplyr::select(-contains("country")) %>%
  gt(rowname_col = "year") %>%
  fmt_number(
    columns = population,
    decimals = 0
  ) %>%
  grand_summary_rows(
    columns = population,
    fns = list(change = ~ max(.) - min(.)),
    formatter = fmt_number,
    decimals = 0
  ) %>%
  tab_style(
    style = list(
      cell_text(style = "italic"),
      cell_fill(color = "lightblue")
    ),
    locations = cells_grand_summary(
      columns = population,
      rows = 1
    )
  )
```

	population
1960	30,455,000
1961	30,739,250
1962	31,023,366
1963	31,296,651
1964	31,609,195
1965	31,954,292
1966	32,283,194
1967	32,682,947
1968	33,113,134
1969	33,441,054
change	*2,986,054*

`cells_stub_summary()`: Location helper for targeting the stub cells in a summary

The `cells_stub_summary()` function is used to target the stub cells of summary and it is useful when applying a footnote with `tab_footnote()` or adding custom styles with `tab_style()`. The function is expressly used in each of those functions' `locations` argument. The 'stub_summary' location is generated by the `summary_rows()` function.

EXAMPLE

Use `countrypops` to create a **gt** table. Add some styling to the summary data stub cells with `tab_style()` and `cells_stub_summary()`.

```
countrypops %>%
  dplyr::filter(country_name == "Japan", year < 1970) %>%
  dplyr::select(-contains("country")) %>%
  dplyr::mutate(decade = paste0(substr(year, 1, 3), "0s")) %>%
  dplyr::group_by(decade) %>%
  gt(
    rowname_col = "year",
    groupname_col = "decade"
  ) %>%
  fmt_number(
    columns = population,
    decimals = 0
  ) %>%
  summary_rows(
    groups = "1960s",
    columns = population,
    fns = list("min", "max"),
    formatter = fmt_number,
    decimals = 0
  ) %>%
  tab_style(
    style = list(
      cell_text(
        weight = "bold",
        transform = "capitalize"
      ),
      cell_fill(
        color = "lightblue",
        alpha = 0.5
      )
    ),
    locations = cells_stub_summary(
      groups = "1960s"
    )
  )
```

	population
1960s	
1960	92,500,572
1961	94,943,000

	population
1962	95,832,000
1963	96,812,000
1964	97,826,000
1965	98,883,000
1966	99,790,000
1967	100,725,000
1968	101,061,000
1969	103,172,000
Min	92,500,572
Max	103,172,000

`cells_stub_grand_summary()` : Location helper for targeting the stub cells in a grand summary

The `cells_stub_grand_summary()` function is used to target the stub cells of a grand summary and it is useful when applying a footnote with `tab_footnote()` or adding custom styles with `tab_style()` . The function is expressly used in each of those functions' `locations` argument. The 'stub_grand_summary' location is generated by the `grand_summary_rows()` function.

EXAMPLE

Use `countrypops` to create a **gt** table. Add some styling to a grand summary stub cell with with the `tab_style()` and `cells_stub_grand_summary()` functions.

```
countrypops %>%
  dplyr::filter( country_name == "Spain", year < 1970) %>%
  dplyr::select(-contains("country")) %>%
  gt(rowname_col = "year") %>%
  fmt_number(
    columns = population,
    decimals = 0
  ) %>%
  grand_summary_rows(
    columns = population,
    fns = list(
      change = ~ max(.) - min(.)
    ),
    formatter = fmt_number,
    decimals = 0
  ) %>%
  tab_style(
    style = cell_text(weight = "bold", transform = "uppercase"),
    locations = cells_stub_grand_summary(rows = "change")
  )
```

	population
1960	30,455,000
1961	30,739,250
1962	31,023,366
1963	31,296,651
1964	31,609,195
1965	31,954,292
1966	32,283,194
1967	32,682,947
1968	33,113,134
1969	33,441,054
CHANGE	2,986,054

`cells_footnotes()` : Location helper for targeting the footnotes

The `cells_footnotes()` function is used to target all footnotes in the footer section of the table. This is useful for adding custom styles to the footnotes with `tab_style()` (using the `locations` argument). The 'footnotes' location is generated by one or more uses of the `tab_footnote()` function. This location helper function cannot be used for the `locations` argument of `tab_footnote()` and doing so will result in a warning (with no change made to the table).

EXAMPLE

Use `sza` to create a **gt** table. Color the `sza` column using the `data_color()` function, add a footnote and also style the footnotes section.

```
sza %>%
  dplyr::filter(
    latitude == 20 &
      month == "jan" &
      !is.na(sza)
  ) %>%
  dplyr::select(-latitude, -month) %>%
  gt() %>%
  data_color(
    columns = sza,
    colors = scales::col_numeric(
      palette = c("white", "yellow", "navyblue"),
      domain = c(0, 90)
    )
  ) %>%
  tab_footnote(
    footnote = "Color indicates height of sun.",
    locations = cells_column_labels(
      columns = sza
    )
  ) %>%
  tab_options(table.width = px(320)) %>%
  tab_style(
    style = list(
      cell_text(size = "smaller"),
      cell_fill(color = "gray90")
    ),
    locations = cells_footnotes()
  )
```

tst	sza[1]
0700	84.9
0730	78.7
0800	72.7
0830	66.1
0900	61.5
0930	56.5
1000	52.1
1030	48.3
1100	45.5

[1] Color indicates height of sun.

tst	sza[1]
1130	43.6
1200	43.0

[1] Color indicates height of sun.

`cells_source_notes()`: Location helper for targeting the source notes

The `cells_source_notes()` function is used to target all source notes in the footer section of the table. This is useful for adding custom styles to the source notes with `tab_style()` (using the `locations` argument). The ‘source_notes’ location is generated by the `tab_source_note()` function. This location helper function cannot be used for the `locations` argument of `tab_footnote()` and doing so will result in a warning (with no change made to the table).

EXAMPLE

Use `gtcars` to create a **gt** table. Add a source note and style the source notes section.

```
gtcars %>%
  dplyr::select(mfr, model, msrp) %>%
  dplyr::slice(1:5) %>%
  gt() %>%
  tab_source_note(
    source_note = "From edmunds.com"
  ) %>%
  tab_style(
    style = cell_text(
      color = "#A9A9A9",
      size = "small"
    ),
    locations = cells_source_notes()
  )
```

mfr	model	msrp
Ford	GT	447000
Ferrari	458 Speciale	291744
Ferrari	458 Spider	263553
Ferrari	458 Italia	233509
Ferrari	488 GTB	245400

From edmunds.com

`currency()` : Supply a custom currency symbol to `fmt_currency()`

The `currency()` helper function makes it easy to specify a context-aware currency symbol to `currency` argument of `fmt_currency()` . Since **gt** can render tables to several output formats, `currency()` allows for different variations of the custom symbol based on the output context (which are `html` , `latex` , `rtf` , and `default`). The number of decimal places for the custom currency defaults to `2` , however, a value set for the decimals argument of `fmt_currency()` will take precedence.

EXAMPLE

Use `exibble` to create a **gt** table. Format the `currency` column to have currency values in guilder (a defunct Dutch currency).

```
exibble %>%
  gt() %>%
  fmt_currency(
    columns = currency,
    currency = currency(html = "&fnof;", default = "f"),
    decimals = 2
  )
```

num	char	fctr	date	time	datetime	currency	row	group
1.111e-01	apricot	one	2015-01-15	13:35	2018-01-01 02:22	ƒ49.95	row_1	grp_a
2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	ƒ17.95	row_2	grp_a
3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	ƒ1.39	row_3	grp_a
4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	ƒ65,100.00	row_4	grp_a
5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	ƒ1,325.81	row_5	grp_b
NA	fig	six	2015-06-15	NA	2018-06-06 16:11	ƒ13.26	row_6	grp_b
7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA	row_7	grp_b
8.880e+06	honeydew	eight	2015-08-15	20:20	NA	ƒ0.44	row_8	grp_b

`cell_text()` : Helper for defining custom text styles for table cells

This helper function is to be used with the `tab_style()` function, which itself allows for the setting of custom styles to one or more cells. We can also define several styles within a single call of `cell_text()` and `tab_style()` will reliably apply those styles to the targeted element.

EXAMPLE

Use `exibble` to create a **gt** table. Add styles with `tab_style()` and the `cell_text()` helper function.

```
exibble %>%
  dplyr::select(num, currency) %>%
  gt() %>%
  fmt_number(
    columns = c(num, currency),
    decimals = 1
  ) %>%
  tab_style(
    style = cell_text(weight = "bold"),
    locations = cells_body(
      columns = num,
      rows = num >= 5000
    )
  ) %>%
  tab_style(
    style = cell_text(style = "italic"),
    locations = cells_body(
      columns = currency,
      rows = currency < 100
    )
  )
```

num	currency
0.1	*50.0*
2.2	*17.9*
33.3	*1.4*
444.4	65,100.0
5,550.0	1,325.8
NA	*13.3*
777,000.0	NA
8,880,000.0	*0.4*

`cell_fill()` : Helper for defining custom fills for table cells

The `cell_fill()` helper function is to be used with the `tab_style()` function, which itself allows for the setting of custom styles to one or more cells. Specifically, the call to `cell_fill()` should be bound to the `styles` argument of `tab_style()` .

EXAMPLE

Use `exibble` to create a **gt** table. Add styles with `tab_style()` and the `cell_fill()` helper function.

```
exibble %>%
  dplyr::select(num, currency) %>%
  gt() %>%
  fmt_number(
    columns = c(num, currency),
    decimals = 1
  ) %>%
  tab_style(
    style = cell_fill(color = "lightblue"),
    locations = cells_body(
      columns = num,
      rows = num >= 5000
    )
  ) %>%
  tab_style(
    style = cell_fill(color = "gray85"),
    locations = cells_body(
      columns = currency,
      rows = currency < 100
    )
  )
```

num	currency
0.1	50.0
2.2	17.9
33.3	1.4
444.4	65,100.0
5,550.0	1,325.8
NA	13.3
777,000.0	NA
8,880,000.0	0.4

`cell_borders()`: Helper for defining custom borders for table cells

The `cell_borders()` helper function is to be used with the `tab_style()` function, which itself allows for the setting of custom styles to one or more cells. Specifically, the call to `cell_borders()` should be bound to the `styles` argument of `tab_style()`. The `selection` argument is where we define which borders should be modified (e.g., `"left"`, `"right"`, etc.). With that selection, the `color`, `style`, and `weight` of the selected borders can then be modified.

EXAMPLES

Add horizontal border lines for all table body rows in `exibble`.

```
exibble %>%
  gt() %>%
  tab_options(row.striping.include_table_body = FALSE) %>%
  tab_style(
    style = cell_borders(
      sides = c("top", "bottom"),
      color = "#BBBBBB",
      weight = px(1.5),
      style = "solid"
    ),
    locations = cells_body(
      columns = everything(),
      rows = everything()
    )
  )
```

num	char	fctr	date	time	datetime	currency	row	group
1.111e-01	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950	row_1	grp_a
2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950	row_2	grp_a
3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390	row_3	grp_a
4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000	row_4	grp_a
5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810	row_5	grp_b
NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255	row_6	grp_b
7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA	row_7	grp_b
8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440	row_8	grp_b

Incorporate different horizontal and vertical borders at several locations. This uses multiple `cell_borders()` and `cells_body()` calls within `list()`s.

```
exibble %>%
  gt() %>%
  tab_style(
    style = list(
      cell_borders(
        sides = c("top", "bottom"),
        color = "#FF0000",
        weight = px(2)
      ),
      cell_borders(
        sides = c("left", "right"),
        color = "#0000FF",
        weight = px(2)
      )
    ),
    locations = list(
      cells_body(
        columns = num,
        rows = is.na(num)
      ),
      cells_body(
        columns = currency,
        rows = is.na(currency)
      )
    )
  )
```

num	char	fctr	date	time	datetime	currency	row	group
1.111e-01	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950	row_1	grp_a
2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950	row_2	grp_a
3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390	row_3	grp_a
4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000	row_4	grp_a
5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810	row_5	grp_b
NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255	row_6	grp_b
7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA	row_7	grp_b
8.880e+06	honeydew	eight	2015-08-15	20:20	NA	0.440	row_8	grp_b

`google_font()` : Helper function for specifying a font from the Google Fonts service

The `google_font()` helper function can be used wherever a font name should be specified. There are two instances where this helper can be used: the `name` argument in `opt_table_font()` (for setting a table font) and in that of `cell_text()` (used with `tab_style()`). To get a helpful listing of fonts that work well in tables, use the `info_google_fonts()` function.

EXAMPLES

Use `exibble` to create a **gt** table of eight rows, replace missing values with em dashes. For text in the `time` column, we use the Google font ‘IBM Plex Mono’ and set up the `default_fonts()` as fallbacks (just in case the webfont is not accessible).

```
exibble %>%
  dplyr::select(char, time) %>%
  gt() %>%
  fmt_missing(columns = everything()) %>%
  tab_style(
    style = cell_text(
      font = c(
        google_font(name = "IBM Plex Mono"),
        default_fonts()
      )
    ),
    locations = cells_body(columns = time)
  )
```

char	time
apricot	13:35
banana	14:40
coconut	15:45
durian	16:50
—	17:55
fig	—
grapefruit	19:10
honeydew	20:20

Use `sp500` to create a small **gt** table, using `fmt_currency()` to provide a dollar sign for the first row of monetary values. Then, set a larger font size for the table and use the ‘Merriweather’ font using the `google_font()` function (with two font fallbacks: ‘Cochin’ and the catchall ‘Serif’ group).

```
sp500 %>%
  dplyr::slice(1:10) %>%
  dplyr::select(-volume, -adj_close) %>%
  gt() %>%
  fmt_currency(
    columns = 2:5,
    rows = 1,
    currency = "USD",
    use_seps = FALSE
  ) %>%
  tab_options(table.font.size = px(20)) %>%
  opt_table_font(
    font = list(
      google_font(name = "Merriweather"),
      "Cochin", "Serif"
    )
  )
```

date	open	high	low	close
2015-12-31	$2060.59	$2062.54	$2043.62	$2043.94
2015-12-30	2077.34	2077.34	2061.97	2063.36
2015-12-29	2060.54	2081.56	2060.54	2078.36
2015-12-28	2057.77	2057.77	2044.20	2056.50
2015-12-24	2063.52	2067.36	2058.73	2060.99
2015-12-23	2042.20	2064.73	2042.20	2064.29
2015-12-22	2023.15	2042.74	2020.49	2038.97
2015-12-21	2010.27	2022.90	2005.93	2021.15
2015-12-18	2040.81	2040.81	2005.33	2005.55
2015-12-17	2073.76	2076.37	2041.66	2041.89

`default_fonts()` : A vector of default fonts for use with **gt** tables

The vector of fonts given by `default_fonts()` should be used with a **gt** table that is rendered to HTML. We can specify additional fonts to use but this default set should be placed after that to act as fallbacks. This is useful when specifying `font` values in the `cell_text()` function (itself used in the `tab_style()` function). If using `opt_table_font()` (which also has a `font` argument) we probably don't need to specify this vector of fonts since it is handled by its `add` option (which is `TRUE` by default).

EXAMPLE

Use `exibble` to create a **gt** table. Attempting to modify the fonts used for the `time` column is much safer if `default_fonts()` is appended to the end of the `font` listing in the `cell_text()` call (the “Comic Sansa” and “Menloa” fonts don’t exist, but, we’ll get the first available font from the `default_fonts()` set).

```
exibble %>%
  dplyr::select(char, time) %>%
  gt() %>%
  tab_style(
    style = cell_text(
      font = c(
        "Comic Sansa", "Menloa",
        default_fonts()
      )
    ),
    locations = cells_body(columns = time)
  )
```

char	time
apricot	13:35
banana	14:40
coconut	15:45
durian	16:50
NA	17:55
fig	NA
grapefruit	19:10
honeydew	20:20

`adjust_luminance()`: Adjust the luminance for a palette of colors

This function can brighten or darken a palette of colors by an arbitrary number of steps, which is defined by a real number between -2.0 and 2.0. The transformation of a palette by a fixed step in this function will tend to apply greater darkening or lightening for those colors in the midrange compared to any very dark or very light colors in the input palette.

EXAMPLE

Get a palette of 8 pastel colors from the **RColorBrewer** package.

```
pal <- RColorBrewer::brewer.pal(8, "Pastel2")
```

Create lighter and darker variants of the base palette (one step lower, one step higher).

```
pal_darker  <- pal %>% adjust_luminance(-1.0)
pal_lighter <- pal %>% adjust_luminance(+1.0)
```

Create a tibble and make a **gt** table from it. Color each column in order of increasingly darker palettes (with `data_color()`).

```
dplyr::tibble(a = 1:8, b = 1:8, c = 1:8) %>%
  gt() %>%
  data_color(
    columns = a,
    colors = scales::col_numeric(palette = pal_lighter,domain = c(1, 8))
  ) %>%
  data_color(
    columns = b,
    colors = scales::col_numeric(palette = pal, domain = c(1, 8))
  ) %>%
  data_color(
    columns = c,
    colors = scales::col_numeric(palette = pal_darker, domain = c(1, 8))
  )
```

a	b	c
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8