Chapter 3: Text Elements

Hello and welcome back! In this chapter, we are going to continue learning more about Components. In this chapter, we are going to learn about Text Elements.

So, you might have been familiar with HTML elements such as `h1`, `h2`, `p`, right? Well, what Streamlit did, is that they created components by implementing these existing tags to make a cooler text system. They not only used these HTML elements, but they even extended this to many other components, including `LaTeX`! We will now start exploring such text elements.

No description has been provided for this image

Part I: Headings and body text

In this part we are going to learn about some components that are used in headings and body text. These are fundamentals which we are going to require.

`st.markdown`

- It is a component to render Github-like MarkDown (.md) in the app's body. Syntax about this can be found in here. It can be quite helpful for real-life uses like any online guide.

- The function signature is: st.markdown(`body`, `unsafe_allow_html` = `False`, `*`, `help` = `None`, `width` = `"stretch"`)

- To render the markdown, just enter the expression you want in the `body` parameter. It will work.

- An example is provided below:

In [ ]:
```
import streamlit as st

st.markdown("*Streamlit* is **really** ***cool***.") # These st.markdown()-s r
st.markdown('''
```

```
    :red[Streamlit] :orange[can] :green[write] :blue[text] :violet[in]
    :gray[pretty] :rainbow[colors] and :blue-background[highlight] text.''')
st.markdown("Here's a bouquet &mdash;\
            :tulip::cherry_blossom::rose::hibiscus::sunflower::blossom:")

multi = '''If you end a line with two spaces,
a soft return is used for the next line.

Two (or more) newline characters in a row will result in a hard return.
'''
st.markdown(multi)
```

No description has been provided for this image

`st.title`

- It is used to display a big text generally used to title the page.
- It is based on `st.markdown`.
- The function signature is: st.title(`body` , `anchor` = `None` , `*` , `help` = `None` , `width` = `"stretch"`)
- Like in most components, we have to use the `body` parameter to set the value of the data to be displayed.
- An example is given below:

In [ ]:
```
import streamlit as st

st.title("_Streamlit_ is :blue[cool] :sunglasses:") # Will render the title
# And also, when you will learn about `Paging` in the upcoming chapters, you w
# find out that this can be pretty important.
```

No description has been provided for this image

`st.header`

- It is used to display a quite smaller text relative to st.title but in header formatting.

- It is based on `st.markdown`.

- The function signature is: st.header(`body` , `anchor` = `None` , `*` , `help` = `None` , `divider` = `False` , `width` = `"stretch"`)

- Like in most components, we have to use the `body` parameter to set the value of the data to be displayed.

- An example is given below:

In [ ]:
```
import streamlit as st

st.header("_Streamlit_ is :blue[cool] :sunglasses:") # A basic header
st.header("This is a header with a divider", divider="gray") # This one has a
st.header("These headers have rotating dividers", divider=True) # Here, in thi
                                                                # This happens whe
st.header("One", divider=True)
st.header("Two", divider=True)
st.header("Three", divider=True)
st.header("Four", divider=True)
```

No description has been provided for this image

`st.subheader`

- It is used to display a sub-header, which is basically a smaller header. You can visually compare this to a header in the pictures and try to figure out the differences in the sizes.

- It is based on `st.markdown`.

- The function signature is: st.subheader(`body` , `anchor` = `None` , `*` , `help` = `None` , `divider` = `False` , `width` = `"stretch"`)

- Like in most components, we have to use the `body` parameter to set the value of the data to be displayed.

- An example is given below:

In [ ]:
```
import streamlit as st

st.subheader("_Streamlit_ is :blue[cool] :sunglasses:") # A subheader
st.subheader("This is a subheader with a divider", divider="gray") # This one
st.subheader("These subheaders have rotating dividers", divider=True) # Simila
                                                                      # This hap
st.subheader("One", divider=True)
st.subheader("Two", divider=True)
st.subheader("Three", divider=True)
st.subheader("Four", divider=True)
```

No description has been provided for this image

Part II: Formatted text

`st.badge`

- It displays a colored badge with a label and an icon (optionaally).
- It is just a wrapper around the Markdown color badge directive. These are the same:
- `st.markdown(":blue-badge[Home]")` `st.badge("Home", color="blue")`
- But remember, this is just a `Markdown Directive`, means it's a special command which is customised by Streamlit, in this case. So, if you try to render `:blue-badge[Home]` in normal MD, it will not work.
- But generally, it is actually better and simpler to use `st.badge` instead.
- The function signature is: st.badge(`label` , `*` , `icon` = `None` , `color` = `"blue"` , `width` = `"content"`)
- An example is as follows:

In [ ]:
```
import streamlit as st

st.badge("New") # A `New` icon with a default blue-themed palette color system
                # and a lighter shade for the background.

st.badge("Success", icon=":material/check:", color="green") # A Success-themed
```

```
# A whole list of them is available at https://fonts.google.com/icons.

st.markdown(
    ":violet-badge[:material/star: Favorite] :orange-badge[⚠ Needs review] :gr
) # A set of Markdown badges as well.
```

No description has been provided for this image

`st.caption`

- Displays very small text which can be used for captions, footnotes etc.
- The function signature is: st.caption(`body` , `unsafe_allow_html` = `False` , `*` , `help` = `None` , `width` = `"stretch"`)
- An example is given below:

In [ ]:
```
import streamlit as st

# Caption with normal text.
st.caption("This is a string that explains something above.")

# Caption with much formatting using Streamlit's system.
st.caption("A caption with _italics_ :blue[colors] and emojis :sunglasses:")
```

No description has been provided for this image

`st.code`

- Display code block with option syntax highlighting.
- This is quite useful as Streamlit exposes this function to display code in a beautifully colored manner and highlights keywords very well.
- Also, if you hover towards the top right corner, you will see an icon-button with a copy sign appearing. If you click that, the code will get copied.

- The functions signature is: st.code(`body` , `language` = `"python"` , `*` , `line_numbers` = `False` , `wrap_lines` = `False` , `height` = `"content"` , `width` = `"stretch"`)

Available languages that you can use in the `language` parameter:

- abap
- abnf
- actionscript
- ada
- agda
- al
- antlr4
- apacheconf
- apex
- apl
- applescript
- aql
- arduino
- arff
- asciidoc
- asm6502
- asmatmel
- aspnet
- autohotkey
- autoit
- avisynth
- avroIdl (avro-idl)
- bash
- basic
- batch
- bbcode
- bicep
- birb
- bison
- bnf
- brainfuck
- brightscript
- bro
- bsl

- c
- cfscript
- chaiscript
- cil
- clike
- clojure
- cmake
- cobol
- coffeescript
- concurnas
- coq
- cpp
- crystal
- csharp
- cshtml
- csp
- cssExtras (css-extras)
- css
- csv
- cypher
- d
- dart
- dataweave
- dax
- dhall
- diff
- django
- dnsZoneFile (dns-zone-file)
- docker
- dot
- ebnf
- editorconfig
- eiffel
- ejs
- elixir
- elm
- erb
- erlang
- etlua
- excelFormula (excel-formula)

- factor
- falselang (false)
- firestoreSecurityRules (firestore-security-rules)
- flow
- fortran
- fsharp
- ftl
- gap
- gcode
- gdscript
- gedcom
- gherkin
- git
- glsl
- gml
- gn
- goModule (go-module)
- go
- graphql
- groovy
- haml
- handlebars
- haskell
- haxe
- hcl
- hlsl
- hoon
- hpkp
- hsts
- http
- ichigojam
- icon
- icuMessageFormat (icu-message-format)
- idris
- iecst
- ignore
- inform7
- ini
- io
- j

- java
- javadoc
- javadoclike
- javascript
- javastacktrace
- jexl
- jolie
- jq
- jsExtras (js-extras)
- jsTemplates (js-templates)
- jsdoc
- json
- json5
- jsonp
- jsstacktrace
- jsx
- julia
- keepalived
- keyman
- kotlin
- kumir
- kusto
- latex
- latte
- less
- lilypond
- liquid
- lisp
- livescript
- llvm
- log
- lolcode
- lua
- magma
- makefile
- markdown
- markupTemplating (markup-templating)
- markup
- matlab
- maxscript

- mel
- mermaid
- mizar
- mongodb
- monkey
- moonscript
- n1ql
- n4js
- nand2tetrisHdl (nand2tetris-hdl)
- naniscript
- nasm
- neon
- nevod
- nginx
- nim
- nix
- nsis
- objectivec
- ocaml
- opencl
- openqasm
- oz
- parigp
- parser
- pascal
- pascaligo
- pcaxis
- peoplecode
- perl
- phpExtras (php-extras)
- php
- phpdoc
- plsql
- powerquery
- powershell
- processing
- prolog
- promql
- properties
- protobuf

- psl
- pug
- puppet
- pure
- purebasic
- purescript
- python
- q
- qml
- qore
- qsharp
- r
- racket
- reason
- regex
- rego
- renpy
- rest
- rip
- roboconf
- robotframework
- ruby
- rust
- sas
- sass
- scala
- scheme
- scss
- shellSession (shell-session)
- smali
- smalltalk
- smarty
- sml
- solidity
- solutionFile (solution-file)
- soy
- sparql
- splunkSpl (splunk-spl)
- sqf
- sql

- squirrel
- stan
- stylus
- swift
- systemd
- t4Cs (t4-cs)
- t4Templating (t4-templating)
- t4Vb (t4-vb)
- tap
- tcl
- textile
- toml
- tremor
- tsx
- tt2
- turtle
- twig
- typescript
- typoscript
- unrealscript
- uorazor
- uri
- v
- vala
- vbnet
- velocity
- verilog
- vhdl
- vim
- visualBasic (visual-basic)
- warpscript
- wasm
- webIdl (web-idl)
- wiki
- wolfram
- wren
- xeora
- xmlDoc (xml-doc)
- xojo
- xquery

- yaml
- yang
- zig

- An example is given below:

In [ ]:
```python
import streamlit as st

# The code we want to render inside the code block.
code = '''def hello():
    print("Hello, Streamlit!")'''

# The code block.
st.code(code, language="python")
```

No description has been provided for this image

`st.divider`

- This displays a horizontal rule, similar to the `hr` element in HTML. You can also achieve this using `st.write("---")` or even just `"---"` via the `Magic` feature as discussed earlier.

- The function signature is: st.divider(`*` , `width` = `"stretch"`)

- You can also control the width of the divider by setting the `width` parameter to a certain value in `pixels` or `px` .

- For example:

In [ ]:
```python
import streamlit as st

st.title("I love Streamlit! 💖") # The Title
st.divider() # The Divider
st.header("Streamlit has many awesome features! 🌟") # A Header
```

No description has been provided for this image

`st.echo`

- Use it in a `with` block to render some code on the app and also execute it later on.

- This is pretty useful for code demos within a Streamlit app.

- The function signature is: st.echo(`code_location` = `"above"`)

- You can set the value of `code_location` to either `"above"` or `"below"` depending on whether to show the echoed code before or after the results of the executed code block respectively.

- For example:

In [ ]:
```
import streamlit as st

with st.echo(): # Start echoing with the `with` block
    st.write('This code will be printed') # The body of the app will render a
                                          # and then will also render the output
```

No description has been provided for this image

`st.latex`

- This component renders LaTeX. Useful for rendering Mathematical equations. The input can only be based on a string or a SymPy expression. SymPy is a popular library for symbolic mathematics.

- Function signature: st.latex(`body` , `*` , `help` = `None` , `width` = `"stretch"`)

- For example:

In [ ]:
```
import streamlit as st

# This will render the equation for the sum of a geometrical progression.
st.latex(r'''
    a + ar + a r^2 + a r^3 + \cdots + a r^{n-1} =
    \sum_{k=0}^{n-1} ar^k =
    a \left(\frac{1-r^{n}}{1-r}\right)
```

```
    ''')
```

No description has been provided for this image

`st.text`

- This writing component is used to display text without `Markdown` or `HTML` parsing, which means that there is no way to render any `Markdown` or `HTML` code by using it, in simpler words.
- This could be used for many safety purposes and prevent wrong and harmful code from being executed.
- The function signature is: st.text(`body` , `*` , `help` = `None` , `width` = `"content"`)
- For example:

In [ ]:
```
import streamlit as st

# Safe Text
st.text("This is text\n[and more text](that's not a Markdown link).")
```

No description has been provided for this image

`st.help`

- Used to display help and other information for a given object. Helpful for a clear understanding of a Pythonic object.
- The function signature is: st.help(`obj` =, `*` , `width` = `"stretch"`)
- To get help for a certain Pythonic object, use it in the `obj` parameter.
- An example is provided below:

In [ ]:
```
import streamlit as st
import pandas
```

```
st.help(pandas.DataFrame) # get help for the pandas DataFrame class
```

No description has been provided for this image

`st.html`

- Used to insert `HTML` in your app.
- The `HTML` is sanitized using `DOMPurify`.
- `DOMPurify`'s primary service is sanitizing user-supplied or third-party `HTML` content. It removes dangerous tags, attributes, and inline scripts, ensuring that the `HTML` is safe for rendering.
- There is no `JavaScript` support yet at the time of writing.
- The function signature is: st.html(`body` , `*` , `width` = `"stretch"`)
- An example is given below:

In [ ]:
```
import streamlit as st

st.html(
    "<p><span style='text-decoration: line-through double red;'>Oops</span>!</
) # An `Oops!` text with 2 red strikethroughs
```

No description has been provided for this image

Summary:

- We learnt to leverage Streamlit's native Text Element components. Futher on, we will learn about Data Elements.