# Part 2: Customizing RMarkdown

## Allison Horst, Casey O’Hara - 1/27/2023

###### ```{r setup, include=TRUE}

###### knitr::opts\_chunk$set(echo = TRUE)

###### ```

We've been using default settings to do most of our work in RMarkdown. In this part of the lab, we'll learn a few different ways to spruce up our knitted documents.

## Step 1: Create a new .Rmd & css file

* In your project, create a new folder for Part 2. Add a new .Rmd, leaving everything as the template default for now. Save (e.g. as part\_2.Rmd).
* Create a new CSS file (in RStudio: File > New file > CSS file), also saved in the same folder alongside the .Rmd you just created.
* In the YAML of your .Rmd, add `css: style.css` to the `output` section, which should look like this:

###### title: 'Part 2: Customizing RMarkdown'

###### author: "Allison Horst"

###### date: "1/17/2021"

###### output:

###### html\_document:

###### css: "style.css"

Note the `:` added after `html\_document`.

This lets RMarkdown know that it should pull style specifications from the `style.css` file when it renders your final html.

## Step 2: Get & use google fonts

Go to <https://fonts.google.com/> to check out some fonts. Here we'll just pick two (I don't recommend using more than 2 fonts for any document, though font / weight can vary). For the sake of this lab, I recommend picking two weird and very different fonts so it's clear where they show up in your knitted doc.

* Click on your first font choice.
* Click on 'Select this style' next to the style you want.
* Go back to Google fonts home.
* Click on your second font choice.
* Click on 'Select this style' next to the style you want.
* In the 'Use on the web' box that appears, select the radio button for `@import`
* Copy the text between the two <style> </style> tags (e.g starting with @import and ending with a semi-colon). My copied import information, for example, looks like this (because I had selected the Anton and Playfair Display fonts):

@import url('https://fonts.googleapis.com/css2?family=**Anton**&family=**Playfair+Display**&display=swap');

* **Note:** You can also copy this line for each font separately and include them in your css file.
* Paste the copied @import code into your CSS file. Save.

## Customize your .Rmd

In your css file, add the following:

This will update the style of a level-2 header:

###### h2 {

###### font-family: "Anton";

###### font-size: 30px;

###### color: purple;

###### }

Or to update a level 3 header:

###### h3 {

###### font-family: "Anton";

###### font-size: 25px;

###### color: magenta;

###### }

Try knitting your R Markdown and see if those changes are reflected in the HTML output. (Add a line with a level 3 header to see how that changes!)

And to change the body text font, add something like this to css:

###### body {

###### font-family: "Playfair Display";

###### font-size: 14px;

###### color: teal;

###### }

What about some other aesthetics?

Try adding (to the `body` code) a condition to update the `background-color`.

What else can you customize? EVERYTHING. Check out all kinds of things you can customize here: <https://www.w3schools.com/css/>

## Finding elements you might want to change

Want to know which element you need to change, but unsure what it's called? Open your knitted HTML in your browser, right click and choose 'Inspect Element.'

This will give you a good idea of what the css element is currently called, which you can override in your own CSS! Just try out a bunch of stuff (e.g. changing colors & fonts in code chunks, update static & hover color of hyperlinks, etc.).

The best way to learn CSS is to try weird things in CSS. Have fun!

# Part 3: Customizing your Shiny app with tabs and CSS

**Allison Horst, Casey O’Hara January 2023**

## Step 1: Make a blank app

* Create a new folder where your app will live
* Create a new R script, save as app.R within the folder
* To your empty script, add 4 “base” lines of code to make an empty app:

###### library(shiny)

###### library(tidyverse)

###### 

###### ui <- fluidPage()

###### 

###### server <- function(input, output) {}

###### 

###### shinyApp(ui = ui, server = server)

* Save the file, and you should see ‘Run App’ button appear in RStudio. Press it and see that your totally blank slate app appears

## Step 2: Build some infrastructure (multiple tabs with a navigation bar)

Last week, we made a Shiny app that had a single page. Often (including for your term project) you’ll want to have multiple tabs that a user can interact with. There are a number of ways to do this. I’ll show one here, using [**navbarPage**](https://shiny.rstudio.com/reference/shiny/1.0.5/navbarPage.html).

Within fluidPage in the user interface, add some code to create multiple tabs (note: ignoring server for now!)

###### ui <- fluidPage(

###### navbarPage("My app name",

###### tabPanel("Thing 1"),

###### tabPanel("Thing 2"),

###### tabPanel("Thing 3")

###### )

###### )

## Step 3: Add some widgets & outputs

Create the sidebar layout in Tab 1:

###### ui <- fluidPage(

###### navbarPage("My app name",

###### tabPanel("Thing 1",

###### sidebarLayout(

###### sidebarPanel("WIDGETS"),

###### 

###### mainPanel("OUTPUT!")

###### 

###### ) # end sidebarLayout

###### ), # end tabPanel thing 1

###### tabPanel("Thing 2"),

###### tabPanel("Thing 3")

###### ) # end navbarPage

###### 

###### ) # end ui

Add a widget:

###### ui <- fluidPage**(**

###### navbarPage("My app name",

###### tabPanel("Thing 1",

###### sidebarLayout(

###### sidebarPanel("WIDGETS",

###### checkboxGroupInput(inputId = "pick\_species",

###### label = "Choose species:",

###### choices = unique(starwars$species)

###### ) # end checkboxGroupInput

###### ), # end sidebarPanel

###### 

###### mainPanel("OUTPUT!")

###### 

###### ) # end sidebarLayout

###### 

###### ), # end tabPanel thing 1

###### 

###### tabPanel("Thing 2"),

###### tabPanel("Thing 3")

###### ) # end navbarPage

###### 

###### **)** # end ui

## Step 4: Do some wrangling & make a reactive output in the server

Here, we’ll start to work on our server section. We’ll filter the starwars dataset based on user selection in the checkbox widget we just created, then make a scatterplot height x mass for each of the characters retained.

###### server <- function(input, output) {

###### 

###### sw\_reactive <- reactive({

###### starwars %>%

###### filter(species %in% input$pick\_species)

###### }) # end sw\_reactive

###### 

###### output$sw\_plot <- renderPlot(

###### ggplot(data = sw\_reactive(), aes(x = mass, y = height)) +

###### geom\_point(aes(color = species))

###### ) # end output$sw\_plot

###### 

###### }

## Step 5: Then make the reactive graph show up in the Tab 1 main panel

Now we jump back to the user interface to make our new plot show up!

###### ui <- fluidPage**(**

###### navbarPage("My app name",

###### tabPanel("Thing 1",

###### sidebarLayout(

###### sidebarPanel("WIDGETS",

###### checkboxGroupInput(inputId = "pick\_species",

###### label = "Choose species:",

###### choices = unique(starwars$species)

###### ) # end checkboxGroupInput

###### ), # end sidebarPanel

###### 

###### mainPanel("OUTPUT!",

###### plotOutput("sw\_plot")

###### )

###### 

###### ) # end sidebarLayout

###### 

###### ), # end tabPanel thing 1

###### 

###### tabPanel("Thing 2"),

###### tabPanel("Thing 3")

###### ) # end navbarPage

###### ) # end ui

**And you can similarly add widgets / outputs to additional tabs!** *Keep close track of parentheses and give yourself plenty of space.*

# 

# Customizing your Shiny app with CSS

**Note:** There are a LOT of different ways to customize your Shiny app, including with [shinythemes](https://rstudio.github.io/shinythemes/), [shinyjs](https://deanattali.com/shinyjs/basic) for advanced functionality, and [shinydashboard](https://rstudio.github.io/shinydashboard/get_started.html).

Note for all of these, the general structure, UI + server talking to each other, is pretty much the same. The functions for making different pieces are just a matter of googling & looking at the documentation for each.

What if I want to customize my Shiny app using CSS for full control? We can do that too. See more information on customizing your app with CSS here: <https://shiny.rstudio.com/articles/css.html>

* Add a **www** subfolder within whatever folder your **app.R** file exists in
* Create a new CSS file in RStudio (File > New File > CSS File)
* Save it as **theme\_name.css** (here, I’ll use **ocean.css**) - note that whatever that name is will be the “theme name” that you call to have it work in your Shiny app

To my **www/ocean.css** file, I’ll add:

###### body {

###### background-color: teal;

###### }

* Save the updated CSS
* Add theme = "ocean.css" as the first argument in fluidPage(), e.g.:

###### ui <- fluidPage(theme = "ocean.css",

###### navbarPage("My app name", …

Save & run your app. Notice that the theme is updated. Now you’re on your way to customizing a shiny app with CSS.

# Quick Note on Shiny Apps - one file or two?

You can create a Shiny App in a single file (**app.R**) or two separate files (**ui.R** and **server.R**) - keeping the ui.R and server.R separate might help keep your code a little more organized, but it is entirely up to you and your project team.