

A Minimal Book Example

John Doe

2023-10-05

Contents

1	About	5
1.1	Usage	5
1.2	Render book	5
1.3	Preview book	6
2	Learning R	7
2.1	Environment configuration	7
2.2	Basic manipulations	8
2.3	More advanced manipulations	10
2.4	How to explore a data frame?	10
2.5	About stringr package	11
2.6	About Regex in R	11
2.7	Creating samples	12
2.8	About dates	12
2.9	Useful packages or datasets	12
2.10	Useful libraries	12
2.11	Advance programming	13
3	Learning Python	15
4	Learning Git/Github	17
5	Footnotes and citations	21
5.1	Footnotes	21
5.2	Citations	21

6	Learning Markdown	23
7	Learning linux commands	25
8	Statistics with R	27
9	Learning Shiny	31
9.1	outputs	31
9.2	Basic reactivity	32
10	Learning html	35
10.1	different sorts of inputs	36
11	Learning CSS	37
11.1	simple selectors	37
11.2	CSS combined selectors also called combinators	38
11.3	pseudo class selectors	38
11.4	pseudo elements	39
11.5	attribute selectors	40
11.6	Css font properties	40
11.7	Css positions	40
11.8	Css float	41
11.9	Css display	41
11.10	Css units	41
11.11	Css specify colors	42
11.12	Css box models	42
12	Keyboard shortcuts	43

Chapter 1

About

This is a *sample* book written in **Markdown**. You can use anything that Pandoc’s Markdown supports; for example, a math equation $a^2 + b^2 = c^2$.

1.1 Usage

Each **bookdown** chapter is an .Rmd file, and each .Rmd file can contain one (and only one) chapter. A chapter *must* start with a first-level heading: **# A good chapter**, and can contain one (and only one) first-level heading.

Use second-level and higher headings within chapters like: **## A short section** or **### An even shorter section**.

The `index.Rmd` file is required, and is also your first book chapter. It will be the homepage when you render the book.

1.2 Render book

You can render the HTML version of this example book without changing anything:

1. Find the **Build** pane in the RStudio IDE, and
2. Click on **Build Book**, then select your output format, or select “All formats” if you’d like to use multiple formats from the same book source files.

Or build the book from the R console:

```
bookdown::render_book()
```

To render this example to PDF as a `bookdown::pdf_book`, you'll need to install XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): <https://yihui.org/tinytex/>.

1.3 Preview book

As you work, you may start a local server to live preview this HTML book. This preview will update as you edit the book when you save individual .Rmd files. You can start the server in a work session by using the RStudio add-in “Preview book”, or from the R console:

```
bookdown::serve_book()
```

Chapter 2

Learning R

This is notes about learning R.

Please build this simple boook!!!! ahhhhhhhhhhhhha

2.1 Environment configuration

- install a package: `install.packages("tidyverse")` Once a package installed you don't need to install it again
- load a library Once the package is installed, you need to "load" it in your current environment `library(tidyverse)` It was a think that was disturbing me when starting R/ `install.packages` takes a double quote surrounding the package name whereas `library` doesn't take double quote around the package name
- `options(digits =3) # 3 significant digits`
- installing two or more packages at once with `c()`:
`install.packages(c("tidytext","gutenbergr"))`
- question mark + name of the function to get the help of the function
It opens in the bottom right panel in most configurations of R Studio.
example : `?gutenberg_metadata()`
- get the list of files in the current working directory `list.files()`
- get the path of the working directory `getwd()`
- set the path to the directory you want to be the working dir:
`setwd("C:/users/etc/")` in Windows, files are displayed with backslashes \ whereas to indicate a file to R, you need to replace backslashes with slashes

- taper simplement **library()** :
nous donne la liste de nos packages disponibles
- to get all the packages installed in our pc: `installed.packages()` may be important when changing version of R or Rstudio (wip)
- `data()` to list all the datasets available in base R in base R datasets available are stored in package datasets! Use '`data(package = .packages(all.available = TRUE))`' to list the data sets in all *available* packages.
- to discover which datasets are available in a specific package `data(package="HistData")` or `data(package = "Lahman")`
- to understand what is currently in your environment `search()`
- check your R version `R.Version()`

2.2 Basic manipulations

- remove all objects from current environment:
`rm(list=ls())`
- create a sequence `seq(1,100)` sequence from one to one hundred with a default step of one `seq(1,100,2)` idem with step 2
- create a vector with `rep` (repeat)

`my_vec = rep(c(1,2), times = 7)` or

`my_vec2 = rep(c(1,2), times=c(3,8))`

- print and cat

`cat('hello')` hello `print('hello')` [1] "hello"

- Display a field with two or three... digits:
`format(x, nsmall = 2)` example `df %>% group_by(mois) %>% summarise(format(sum(resultats), nsmall=2))`
- save a data frame in Rda format :
`save(my_df, file = "df_xxx_xxx.Rda")`
- load a data frame you previously saved in Rda format :
`load(file = "df_xxx_xxx.Rda")`
- see built-ins datasets t :
`ls("package:datasets")`

- give name to each field of a vector or dataframe: `setNames(vec,c('titi','toto','tata'))`
`setNames(df,c('titi','toto','tata'))`
- `pull()` `pull()` is similar to `$` . It's mostly useful because it looks a little nicer in pipes, it also works with remote data frames, and it can optionally name the output. `data <- data.frame(x1 = 1:5, x1=LETTERS[1:5])` `data <- setNames(data,c("c1","c2"))`

`pull(data,c1)` # apply function with column name (extraire le vecteur c1 du dataframe) `pull(data,c2)`

`pull(data,1)` # apply function with index

`pull` returns a vector

- `sort`, `order` and `rank` use `sort` to sort the vector ascending use `order` to return the index (position) that sorts input vector that is in which position we would need to reposition the elements to make them ordered use `rank` to obtain the rank of each element in the vector if sorted

`my_letters <- c("A","G","C","D")` `sort(my_letters)` `order(my_letters)`
`rank(my_letters)`

`sort_index <- order(my_letters)` `my_letters[sort_index]`

- `min`, `max` and `which.min` and `which.max`

`set.seed(1)` you need to `set.seed` each time you launch the creation of the vector again if you want to obtain the same sequence `my_numbers <- round(runif(n=6,min=0,max=100),digits = 0)` `which.max(my_numbers)` `which.max` return the indices of the max so `my_numbers[which.max(my_numbers)]` returns the max just like

`max(my_numbers)`

- `which ind <- which(murders$state == "California")` `murder_rate[ind]`
`#> [1] 3.37`
- `match ind <- match(c("New York", "Florida", "Texas"), murders$state)`
`ind #> [1] 33 10 44` `murder_rate[ind] #> [1] 2.67 3.40 3.20`

`match` returns indices instead we could also use a logical vector

- `%in% c("Boston", "Dakota", "Washington") %in% murders$state #> [1] FALSE FALSE TRUE`
- Are two objects identical? `identical(vect1,vect2)` returns `TRUE` if `vect1` and `vect2` are identical; `FALSE` otherwise

2.3 More advanced manipulations

- Replace Na's:
- one first solution :

```
df %>% filter(code_rub %in% rub_depends) %>% group_by(code_rub, RUB_AXE_ANA_ME, annee) %>% summarise(mtt = sum(resultats)) %>% pivot_wider(names_from=annee, values_from=mtt) %>% mutate_if(is.numeric, ~replace_na(.,0)) %>%
```
- one second solution : `df %>% mutate_all(., ~replace(., is.na(.), 0))`
- Add totals at the bottom of a data frame :
 this is one of the most painful task in R, whereas it seems completely obvious in R for example. why? cause, dataframes are mostly data structures. They are not done to display datas. however, package janitor helps doing this task very easily (thank you!) `library(janitor)` `df %>% adorn_totals(fill=" ")` fill arguments allow to chose what will be displayed when the column is not numbers
- Understand how a function is built with his source code `View(function_name)`
 e.g. `View(adorn_totals)`

2.4 How to explore a data frame?

- check the type of a field with **class**
`class(df$mois) -> date` class returns data type
- `df %>% head()`
- `df %>% tail()`

In both cases, you can specify the number of rows at the top or at the bottom you want to see.

For example `head(3)` or `tail(3)`

Unique values of a field. Two techniques : * `df %>% distinct(mois)` * `unique(df$mois)` The two solutions give the same results. The only difference is that the first solution returns a data frame whereas the second returns a vector.

- Display all rows from a dataframe in dplyr:
 with pipe `print %>% print(n=nrow(.))` `df %>% filter(mois=='2023-06-01') %>% group_by(code_rub) %>% summarise(format(sum(resultats), nsmall=2)) %>% print(n = nrow(.))`

2.5 About stringr package

Stringr is more coherent than base R functions for strings treatments. Stringr functions always begin with prefix **str_** ; the first argument is always the string you want to treat. And then comes the pattern you want to identify.

Most common and useful functions in Stringr :

- `str_detect()` -> returns a logical vector (a vector of TRUE and FALSE)
- `str_subset()`
- `str_view()`
- `str_view_all()`
- `str_replace()`
- `str_replace_all()`
- `str_split()`
- `str_trim()`
- `str_to_lower()`

2.6 About Regex in R

2.6.1 Special characters

- `\\d` stands for **one of any digit 0,1,2, up to 9**
- `\\s` stands for **one** charater whitespace
- The dot “.” **matches any character**
- So, to match a literal dot “.” in regex, we need two backslashes then dot `\\.`
- The star “*” stands for **0 or more** instances of the previous character
- The plus sign “+” stands for **1 or more** instances of the previous character
- The question mark “?” stands for **0 or one** instance of the previous character
- `()` “\\1” capture le groupe de la parenthèse 1 et “\\2” capture le groupe de la parenthèse 2

Separate and extract function are from tidyr package.

In **extract**, you can use regex to split a string.

```
library(dplyr)
library(tidyr)
s <- c("5'6", "6'4")
tab <- data.frame(x = s)

tab %>% separate(x, c("feet", "inches"), sep = "'")
```

```
##      feet inches
## 1      5      6
## 2      6      4
```

```
tab %>% extract(x,c("feet","inches"), regex = "(\\d)'(\\d{1,2})")
```

```
##      feet inches
## 1      5      6
## 2      6      4
```

2.7 Creating samples

```
set.seed(1) sample()
```

2.8 About dates

- Sys.time() from base R returns current date/time.
- Extract year of date (with the field already of type date) with function year from lubridate library(lubridate)
df <- df %>% mutate(annee=year(mois))

2.9 Useful packages or datasets

- gapminder library(gapminder)
data("gapminder")
- Galton dataset Galton dataset is in HistData library
- tidyr : pivot_wider and pivot_longer are from tidyverse (tidyr)

2.10 Useful libraries

- recommendation of HWickham: In shiny, If you want greater control over the output of dataTableOutput(), I highly recommend the reactable package by Greg Lin.
- It maybe possible to extract a table from a pdf with pdftools
Not tested myself library("pdftools") temp_file <- tempfile() url <-
"https://www.pnas.org/action/downloadSupplement?doi=10.1073%
2Fpnas.1510159112&file=pnas.201510159SI.pdf" download.file(url,
temp_file) txt <- pdf_text(temp_file) file.remove(temp_file)

2.11 Advance programming

Expressions are a powerful tool for manipulating and executing R code. Environments record associations between names and values. *Execution environments create a scope for variable names inside of functions.

Chapter 3

Learning Python

- shebang line `#!/usr/bin/env python3`

Chapter 4

Learning Git/Github

- my actual 20230918 bookdown workflow On github server on our repo, we need a .nojekyll file at the root and we need the /docs folder with our files that are generated when launching “build book”. When I am currently working locally, I must open courses_notes_as_a_bookdown.Rproj Then Rstudio opens. Then I open and I write/work in the .Rmd files. Those are not the files that must be committed. So I placed them in my gitignore file with echo command » etc. Once the files are modified, I go in Rstudio right hand corner. I click build/build book. Then I open git bash. I go to /projects/courses_notes_as_a_bookdown.Rproj I think this is the good level to be but to be verified next time (if I need to go down in /docs) I can make a git status. I have to make a git commit -a -m ‘labelmycommit’. Then I have to make a git push. And It is it. If everything has worked well, I can see my up to date book available in my github account.
- three main states each track file can be in one of the three main states: modified/staged/committed another state in untracked!
- git config user.name “my_name”
- git config user.email “me@example.com”
- git config – global user.name “my_name”
-> set the value of the username for all git repos
whereas if “git config” without global you set it up for the current directory
- git init -> when in the directory which you want to set under git control (initialize a new repo)
- git add myfile -> stage myfile (place it in the staging area)
- git commit -m “my message for this commit”

- `git config -l`
- `git status` -> check current state
- three status for tracked files : modified/staged/committed
- in order to visualize all **the commits** (not all the modifications) which were made :
`git log`
- `git add -p` a way to review changes before adding them git will show us which files were not staged and ask us if we want to commit
- `git log -p` gives more informations in a viewer
the -p comes from patch you can see differences line by line you can quit the viewer typing q as with less viewer
- `git log --stat` extra info (how many lines you have added or remove)
- `git show 'commit_id'`
`git show` takes a commit id as a parameter
- `git -stats`
- Admit you modified a file `readme.txt` which is under version control.
You can see the modifications since the previous version with this command line:
`git diff readme.txt`
- Add a file to `.gitignore` in order it is not tracked anymore(?) `echo 01-Learning-R.Rmd > .gitignore` `echo .RData > .gitignore` after modifying `.gitignore` you need to stage (`git add`) and commit (`git commit`) it.
- `git commit -a -m 'message for the commit'` when you want to commit only the modifications **super useful** (a is for only modified files / m is for message)
- `git rm filename`
after this you must commit in order the changes to be taken into account
- `git mv filename` in order to move or rename a file
`git mv old_name new_name` after this need to commit
- `git diff -u`
- `git diff` only shows unstaged changes by default
- `git diff --staged` to see the changes that are staged but not committed
- `git`
- `git checkout "commit_id"` roll back to a previous version

- `git reset` remove from staging area
- `git commit -amend` to modify a commit

Chapter 5

Footnotes and citations

5.1 Footnotes

Footnotes are put inside the square brackets after a caret `^[]`. Like this one ¹.

5.2 Citations

Reference items in your bibliography file(s) using `@key`.

For example, we are using the **bookdown** package [Xie, 2023] (check out the last code chunk in `index.Rmd` to see how this citation key was added) in this sample book, which was built on top of R Markdown and **knitr** [Xie, 2015] (this citation was added manually in an external file `book.bib`). Note that the `.bib` files need to be listed in the `index.Rmd` with the YAML `bibliography` key.

The RStudio Visual Markdown Editor can also make it easier to insert citations: <https://rstudio.github.io/visual-markdown-editing/#/citations>

¹This is a footnote.

Chapter 6

Learning Markdown

In markdown you need to escape twice the backslash in order to display two backslashes

So what you see here, I wrote it with **four** not just three : \\

To write a list, you must write a star * followed by a coma a the beginning of a line. Before the list starts you need a blankline and same at the end of the list otherwise Markdown won't recognize it.

To introduce a return to the ligne, you need not only to type return in Markdown, but also to make the line followed by two spaces.

To make a few words bold you need to surrender it with two stars both sides. It is **bold** gives : It is **bold**

Chapter 7

Learning linux commands

- `git -version`
- `mkdir ->` create a directory
- `cat` to read a file
- or `less` (type `q` in order to exit `less` viewer) why `less`? because previous version of `less` was more :)
- write in a file :
`echo toto et tata > toto.txt`

`echo toto et titi > titi.txt`

- differences between two files:
`diff toto.txt titi.txt`
or `diff -u toto.txt titi.txt`
- `diff -u` is more readable than simple `diff` command.
- Create a diff file:
`diff -u toto.txt titi.txt > change.diff`
- Patch the .diff file:
`patch titi.txt < change.diff`
- Clear the console:
just as in Rstudio `ctrl+l` or typing “clear” and then enter in the console.
both works
- Content of a directory:
`dir` or `ls` : both works.

- Content of a directory including hidden files:
dir -a ls -a from the help of ls : " -a, -all do not ignore entries starting with."
- Add the options l to see rights on the files: ls -la
- Get the help in git bash on windows:
function -help example: ls -help
- Make a file executable:
chmod +x filename
- Open a file with nano :
nano my_file.txt
- Save changes made to a file in nano:
ctrl+o + Enter + ctrl+x
- 'cd -' in order to come back to previous directory
- set a file in executable mode chmod +x file_name
- copy a file cp disk_usage.py disk_usage_original.py
- git config --global user.name "my_name" set the value for all git repos on the pc

Chapter 8

Statistics with R

```
beads <- rep(c("red","blue"), times = c(2,3))
beads
```

```
## [1] "red" "red" "blue" "blue" "blue"
```

```
# pick a bead at random
sample(beads,1)
```

```
## [1] "blue"
```

```
# evaluate the probability of drawing a blue at random
mean(beads == "blue")
```

```
## [1] 0.6
```

```
B <- 10000
```

```
# Nota : if you want the "same random" each time
# you draw the sample, you need to set a seed
set.seed(1)
# for example
```

```
events <- replicate(B,sample(beads,1))
tab <- table(events)
# calculate probability of each events
prop.table(tab)
```

```
## events
##   blue   red
## 0.6028 0.3972
```

```
# note that with this monte carlo simulation we have
# quite the same probability for blue events as with
# mean(beads == "blue")
```

by default, sample samples without replacement. it means, if you try

```
sample(beads,5)
```

```
## [1] "blue" "red"  "red"  "blue" "blue"
```

```
# it works, but if you try
# sample(beads,6)
# it fails because there are only 6 beads
```

so sample by default is equivalent to `sample(beads,5, replace = FALSE)` but we can use sample **with** replacement.

```
# so this is working:
sample(beads,6,replace = TRUE)
```

```
## [1] "blue" "red"  "red"  "red"  "blue" "red"
```

```
# and we can use simply sample with replacement
# just like with replicate
```

```
events2 <- sample(beads,B, replace = TRUE)
tab2 <- table(events2)
prop.table(tab2)
```

```
## events2
##   blue   red
## 0.6026 0.3974
```

```
# the result is almost the same
```

discrete and continuous variables example: discrete : flip of a coin outcome from the roll of a die

web site traffic on a given day > particular discrete variable (no upper bound > poisson) same for the number of people clicking on an add

continuous : BMI (body mass index) of a subject four years after a baseline measurement hypertension status but could be modelled as discrete (1 hypertension/0 no hypertension)

- probability mass function (pmf) for discrete variable > assign a probability for each value they can take 1-must always be larger or equal to 0 (prob is number bet 0 abd 1) 2- the sum of the poss values has to add up to 1
- examples of pmf : binomial, canonical (flipping a cooin)

Bernouli = flip a coin $X = 0$ tails $X = 1$ represents heads

Two broad flavors of inference : * frequency, which uses “long run proportion of times an event occurs in **independent**, identically distributed repetitions”
 * he second is Bayesian in which the probability estimate for a hypothesis is updated as additional evidence is acquired

- If A and B are two **independent** events then the probability of them both occurring is the product of the probabilities. $P(A \& B) = P(A) * P(B)$
- Suppose you rolled the fair die twice. What is the probability of rolling the same number two times in a row? Since we don't care what the outcome of the first roll is, its probability is 1. The second roll of the dice has to match the outcome of the first, so that has a probability of 1/6. The probability of both events occurring is $1 * 1/6$.
- The probability of at least one of two events, A and B, occurring is the sum of their individual probabilities minus the probability of their intersection. $P(A \cup B) = P(A) + P(B) - P(A \& B)$.

Chapter 9

Learning Shiny

```
library(shiny)
```

- run shinyapp shortcut:
ctrl + shift + enter not ctrl + shift + K (this is to knit Rmarkdown's documents)
- To have a look on which html tags are available in Shiny: ?builder
- Ctrl + U to see the html content of a web page > it opens the html code in a new page
- Alt + s + w + s or via the menu Session > setwdir > To source file loc

```
runApp()
```

```
shinyUI()
```

```
shinyServer()
```

9.1 outputs

9.1.1 Text

- renderText() combines the result into a single string, and is usually paired with textOutput()
- renderPrint() prints the result, as if you were in an R console, and is usually paired with verbatimTextOutput()

9.1.2 Tables

`tableOutput()` and `renderTable()` render a static table of data, showing all the data at once.

`dataTableOutput()` and `renderDataTable()` render a dynamic table, showing a fixed number of rows along with controls to change which rows are visible.

9.1.3 Plots

You can display any type of R graphic (base, `ggplot2`, or otherwise) with `plotOutput()` and `renderPlot()`

9.2 Basic reactivity

9.2.1 Server function

9.2.1.1 Input

It must be a simple string that contains only letters, numbers, and underscores (no spaces, dashes, periods, or other special characters allowed!). Name it like you would name a variable in R.

It must be unique. If it's not unique, you'll have no way to refer to this control in your server function!

Most input functions have a second parameter called `label`. This is used to create a human-readable label for the control. Shiny doesn't place any restrictions on this string, but you'll need to carefully think about it to make sure that your app is usable by humans! The third parameter is typically `value`, which, where possible, lets you set the default value. The remaining parameters are unique to the control

`sliderInput()`: If you supply a length-2 numeric vector for the default value of `sliderInput()`, you get a "range" slider with two ends. example : `sliderInput("rng", "Range", value = c(10, 20), min = 0, max = 100)`

Unlike a typical list, input objects are read-only. If you attempt to modify an input inside the server function, you'll get an error.

To read from an input, you must be in a reactive context created by a function like `renderText()` or `reactive()`

9.2.1.2 Output

You always use the output object in concert with a render function, as in the following simple example.

Link toward vers Mastering shiny from Hadley Wickham

Each **render{Type}** function is designed to produce a particular type of output (e.g. text, tables, and plots), and is *often paired* with a **{type}Output** function. For example, in this app, `renderPrint()` is paired with `verbatimTextOutput()` to display a statistical summary with fixed-width (verbatim) text, and `renderTable()` is paired with `tableOutput()` to show the input data in a table.

```
server ui textInput selectInput sliderInput passwordInput textAreaInput
numericInput dateInput dateRangeInput radioButtons checkboxGroupInput
fileInput actionButton
```

```
renderPrint() verbatimTextOutput() renderTable() tableOutput() render-
DataTable() dataTableOutput() renderPlot(res = 96) plotOutput()
renderText() textOutput()
```

```
outputnamexxxOutput("name")inputid_selected selectInput("id_selected",
label)
```

```
example of radioButton animals <- c("dog", "cat", "mouse", "bird", "other", "I
hate animals") radioButtons("animal", "What's your favourite animal?", ani-
mals)
```

Allow the user to select multiple elements `selectInput("state", "What's your favourite state?", state.name, multiple = TRUE)`

There's no way to select multiple values with radio buttons, but there's an alternative that's conceptually similar: `checkboxGroupInput()`.

```
ui <- fluidPage( checkboxGroupInput("animal", "What animals do you like?",
animals) )
```

You create a reactive expression by wrapping a block of code in `reactive({...})` and assigning it to a variable, and you use a reactive expression by calling it **like a function** that is the name of the reactive expression we created followed by parentheses. Note that the `{}` are only required in render functions if need to run multiple lines of code. As you'll learn shortly, you should do as little computation in your render functions as possible, which means you can often omit them

To create a reactive expression, we call `reactive()` and assign the results to a variable. To later use the expression, we call the variable like it's a function.

example:

```
server <- function(input, output, session) { x1 <- reactive(rnorm(inputn1, inputmean1,
input$sd1))
```

Most simple exemple of reactive expression:

```
server <- function(input, output, session) { string <- reactive(paste0("Hello",  
inputname, "!"))outputgreeting <- renderText(string()) }
```

Just like any other R function, when the server function is called it creates a new local environment that is independent of every other invocation of the function. This allows each session to have a unique state, as well as isolating the variables created inside the function.

the order in which reactive code is run is determined only by the reactive graph, not by its layout in the server function.

Chapter 10

Learning html

Install visual studio and live server?

```
#<DOCTYPE! html>
#<html lang="en">

#<\html>
```

ctr + u to see the content of a web page

form tag includes input tag(s) form as actions such as action or method input is a self closing tag input can take many attributes such as value placeholder, readonly required name max min

pour valider le code : <https://validator.w3.org/>

pour valider l'accessibilité : swave mais uniquement avec une url publiée...

tag is for metadata

section of the page that links to other pages or to parts within the page

exemples

```
<input type="text" placeholder="ceci est mon exemple"/>

</form>

<input type="text" value="valeur_ini"/>
  <input type="date":>

</form>
```

```

</main>
<footer>

</footer>

```

- tag img

© to display copyright > greater than < less than to display blank space & display le éesperluette

- tag is for informations at the bottom of the page: warning footer doesn't place the information at the bottom copyrights links to social medias related documents
- more semantics than

Second page or Second page or link to a different location in the page (link with an ID) History section

leads to

the image is the link

blabla to open the link into a new tab

.ext et on vient coller #t= 5,25 > joue la vidéo ou l'audio de la seconde 5 à la seconde 25

10.1 different sorts of inputs

include it into a form

- attributes are numerus value, placeholder, readonly (the field cannot be edited), required (necessary before the input is submitted - permet de s'assurer que toutes les données sont saisies), name (used to identify the control when it is submitted to the server)

=> radio button

<label="you are minor">

drop down (?) Red Blue

Chapter 11

Learning CSS

`h1 {color:}` selector property colon value of the property

- element selectors apply style to all similar elements eg to paragraphs
- class selectors apply style to all elements with a specific class
- id selectors apply style to single elements through the id association
- attribute selectors apply styles to elements with a specific attribute such as all the anchors tags which have the target attribute

examples:

11.1 simple selectors

11.1.1 tag selectors

11.1.2 class selectors (class = a group of elements)

```
.redBackground { background-color : red }
```

11.1.3 id selectors

match a single element with a special ID attribute (each and every html element has a unique id)

This is third h1

11.1.4 universal selector

will match all elements in the page

11.2 CSS combined selectors also called combinators

11.2.1 descendant selector

11.2.2 child selector

apply only to **direct children** not to all descendants

`div > h1 { color:red; }` s'applique seulement aux h1 directement sous un div.
S'il y a un tag intermédiaire (notion de dépendant), ça ne s'appliquera pas.

11.2.3 adjacent sibling selector :

11.2.4 general sibling selector :

`h1 ~ h2 { color:blue; }`

11.3 pseudo class selectors

target specific elements based on their current states such as hovered, click...

11.3.1 **:hover** pseudo class used to style an element when the user hovers over it with a mouse

example Click me

11.3.2 **:active**

when a button is clicked

Click me

11.3.3 :visited

when a link had been visited

Click here

11.3.4 :focus**11.3.5 :nth-child**

to select elements based on their position in the parent element

Apply the style only on the first paragraph

11.3.6 :nth-child(2n) : apply only to even elements**11.3.7 :nth-child(2n+1) : apply only to odd elements****11.3.8 :nth-child(odd) works to****11.3.9 :nth-child(even)****11.3.10 :first-child**

to select the first child element of a parent

11.3.11 :last-child

to select the last child element of a parent

reprendre à pseudo elements après pseudo class selectors part 2

11.4 pseudo elements

to style specified parts of an element you must include it inside your tag eg first letter of a paragraph denoted by two semi-colons

- ::before

```
p:before { content:''; display:block; # ou inline-block width: 20px; height: 20px; background-color: green; border: none; border-radius:50px; }
```

::before

- ::after
- ::first-ligne

11.5 attribute selectors

style elements based on the presence or the value of their attributes denoted by square brackets in combination with other selectors

- example:
`a[href~="http"] { background-color:red; }`

Internal1 Google

- example:

11.6 Css font properties

11.7 Css positions

control the position of an element on a webpage

11.7.1 static

the default position of all html elements

11.7.2 relative

top, bottom, left, right

with bottom, the two elements can overlap

11.7.3 absolute

absolute removes `<style #firstDiv { position:absolute; top:10px; }` position regarding to the body

11.7.4 fixed

position:fixed; even when we are scrolling, the div remains fixed

11.8 Ccss float

allows us to create multi column layout and wrap text around images or other elements allows to position the element on the left or the right of its container float often used in conjunction with clear property which control the behaviour of elements following our floating elements

si on ne veut pas que les paragraphes s'ajustent autour de la photo, on leur ajoute une classe avec la propriété clear:right

11.9 Ccss display

control the type of box used for an html element and how it is displayed on the webpage values for display : Block, Inline and Inline Block

block elements create a new line and take up the full width of their container inline elements flow with text and take up only as much width as necessary inline-block allows to fix the height for a div whereas it is not possible with inline ** span by default has display type inline div by default has display type : block **

if we want to change (to change default behaviour):

11.10 Ccss units

majorly two types one is absolute and the other is relative

11.10.1 relative

depends on the screen size 10px 10vw (viewport width) 10vh (viewport height) 10em 1rem

rem defines size from the root em takes his size from its parents

11.11 Css specify colors

- with color name
- with RGB
- with RGBA (last argument for opacity)
- with #Hexade

11.12 Css box models

basicaly in Css model, everything is a box defined lik this: content area > padding > margin > border

example

Fifth!!! Box model

This is div One

This is div Two

- padding you can define padding-top, right, bottom, lef padding-right:10px or padding:10px 5px 10 px 5px (from top to left)

example: #divTwo { background-color: aqua; border: 1px solid black; padding: 40px 5px 10px 60px ; }

- Property box-sizing:border-box for adjusting

Chapter 12

Keyboard shortcuts

- F12 = save as in MS Office

Bibliography

Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL <http://yihui.org/knitr/>. ISBN 978-1498716963.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*, 2023. URL <https://CRAN.R-project.org/package=bookdown>. R package version 0.35.