

# Workshop: Web Scraping 2



Abril, 2020

# Apresentação

# Sobre a Curso-R



**Athos Damiani**  
*Curso-R*  
*Bacharel em*  
*Estatística*



**William Amorim**  
*Curso-R*  
*Doutor em*  
*Estatística*



**Fernando Corrêa**  
*Curso-R*  
*Mestrando em*  
*Estatística*



**Julio Trecenti**  
*Curso-R, Terranova,*  
*ABJ, Conre*  
*Doutorando em*  
*Estatística*

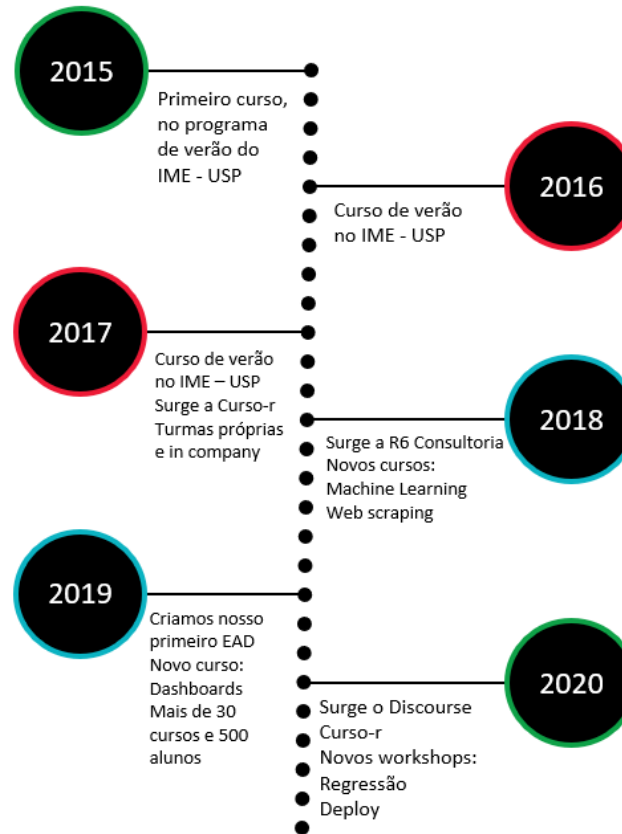


**Daniel Falbel**  
*Curso-R e RStudio*  
*Bacharel em*  
*Estatística*

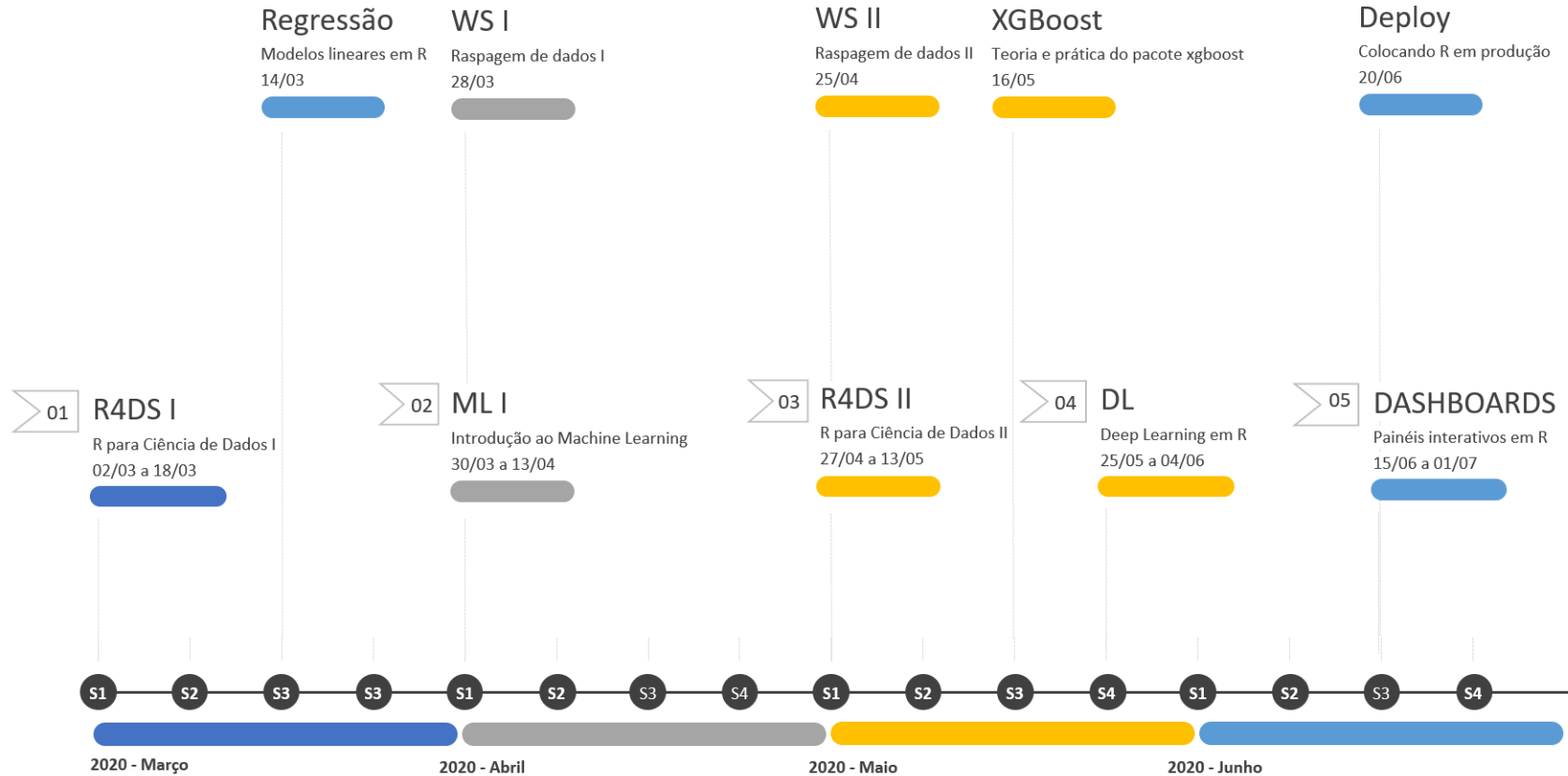


**Caio Lente**  
*Curso-R, Terranova,*  
*ABJ, Mestrando em*  
*ciências da*  
*computação*

# Linha do tempo



# Nossos cursos



# Este curso

- HTML + CSS + XPath
- Iteração + tratamento de erros + agendamento
- Paralelização + futuros
- Selenium + WebDriver + PhantomJS
- SSL + Viewstate + ...
- Exercícios variados no estilo MUY:
  1. (eu) Conceitos/explicações
  2. (nós) Exemplo
  3. (vocês!) Exercício

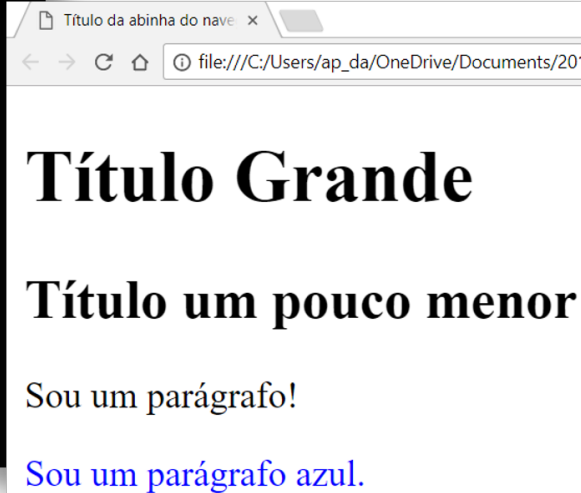
# XPath

# Introdução ao HTML

Exemplo.html no editor de texto

```
1 <!DOCTYPE html>
2
3 <head>
4   <meta charset = latin1>
5   <title>Título da abinha do navegador</title>
6 </head>
7
8 <body>
9   <h1>Título Grande</h1>
10
11   <h2>Título um pouco menor</h2>
12
13   <p>Sou um parágrafo!</p>
14
15   <p style='color: blue;'>Sou um parágrafo azul.</p>
16
17 </body>
18 </html>
```

Exemplo.html no navegador



- HTML (*Hypertext Markup Language*) é uma linguagem de markup cujo uso é a criação de páginas web. Por trás de todo site há pelo menos um arquivo `.html`

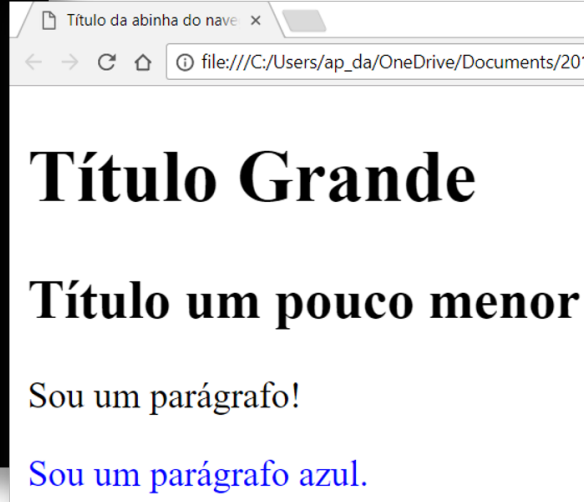


# Introdução ao HTML

Exemplo.html no editor de texto

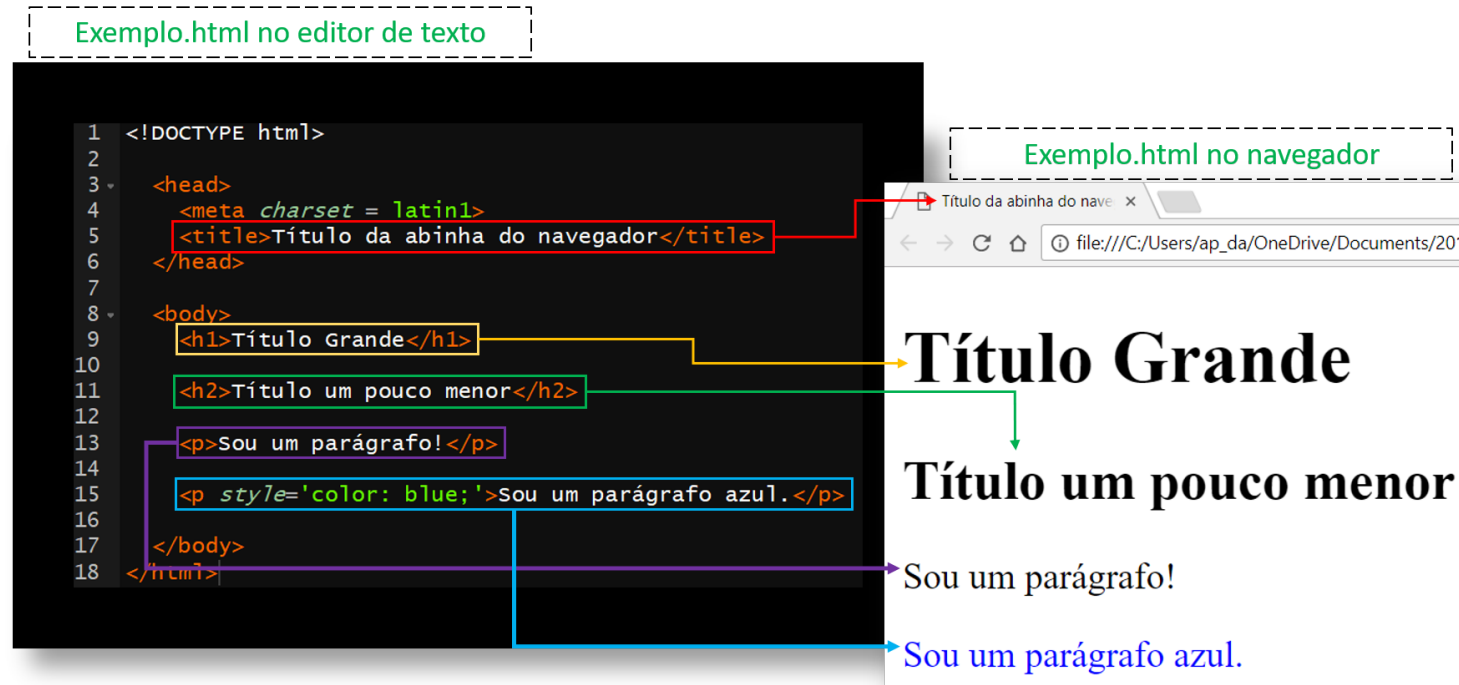
```
1 <!DOCTYPE html>
2
3 <head>
4   <meta charset = latin1>
5   <title>Título da abinha do navegador</title>
6 </head>
7
8 <body>
9   <h1>Título Grande</h1>
10
11   <h2>Título um pouco menor</h2>
12
13   <p>Sou um parágrafo!</p>
14
15   <p style='color: blue;'>Sou um parágrafo azul.</p>
16
17 </body>
18 </html>
```

Exemplo.html no navegador



- Todo arquivo HTML pode ser dividido em seções que definirão diferentes aspectos da página: **head** são "metadados", enquanto **body** é o corpo da página

# Introdução ao HTML

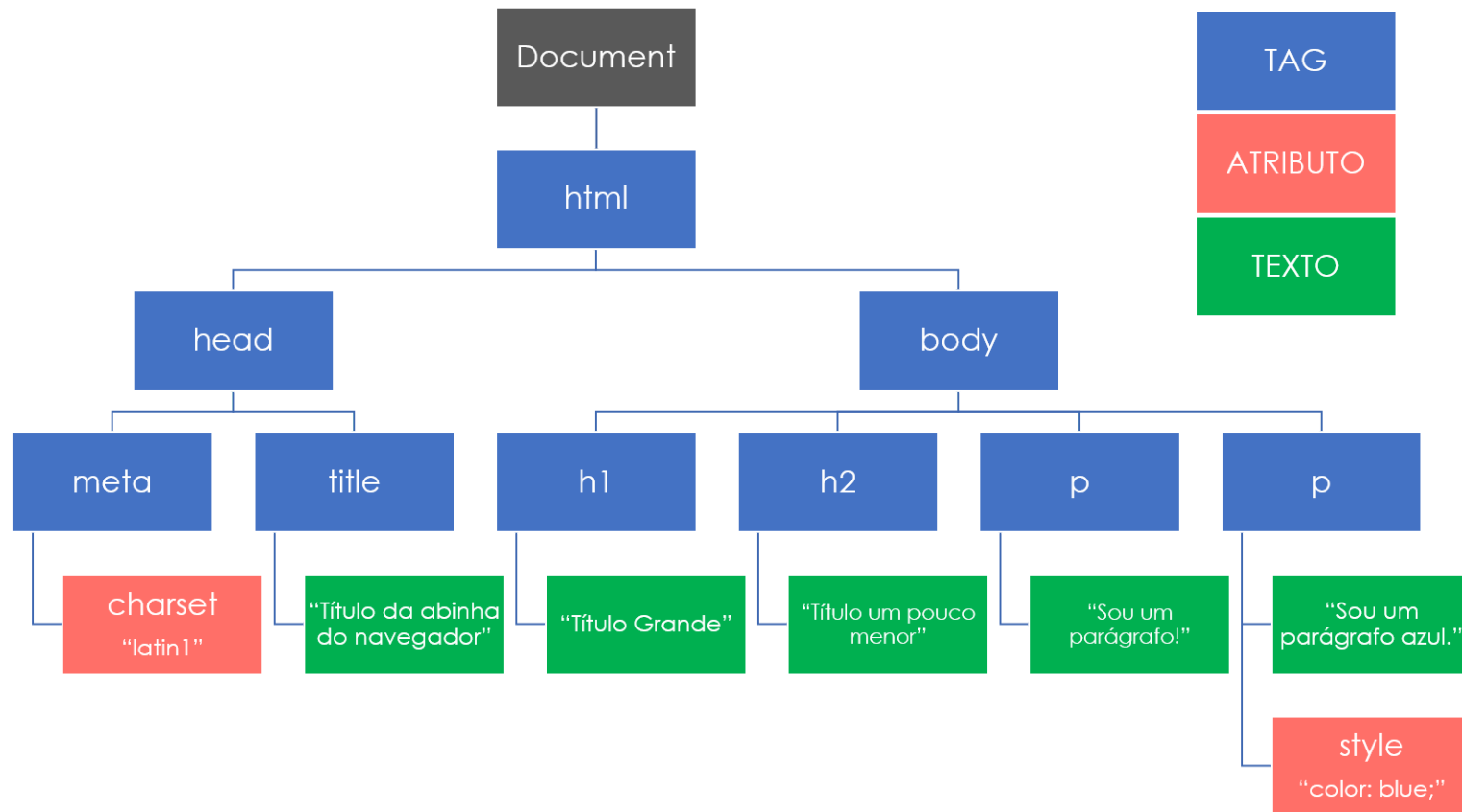


*Tags* (head, body, h1, p, ...) demarcam as seções e sub-seções, enquanto atributos (charset, style, ...) mudam como essas seções são renderizadas pelo navegador

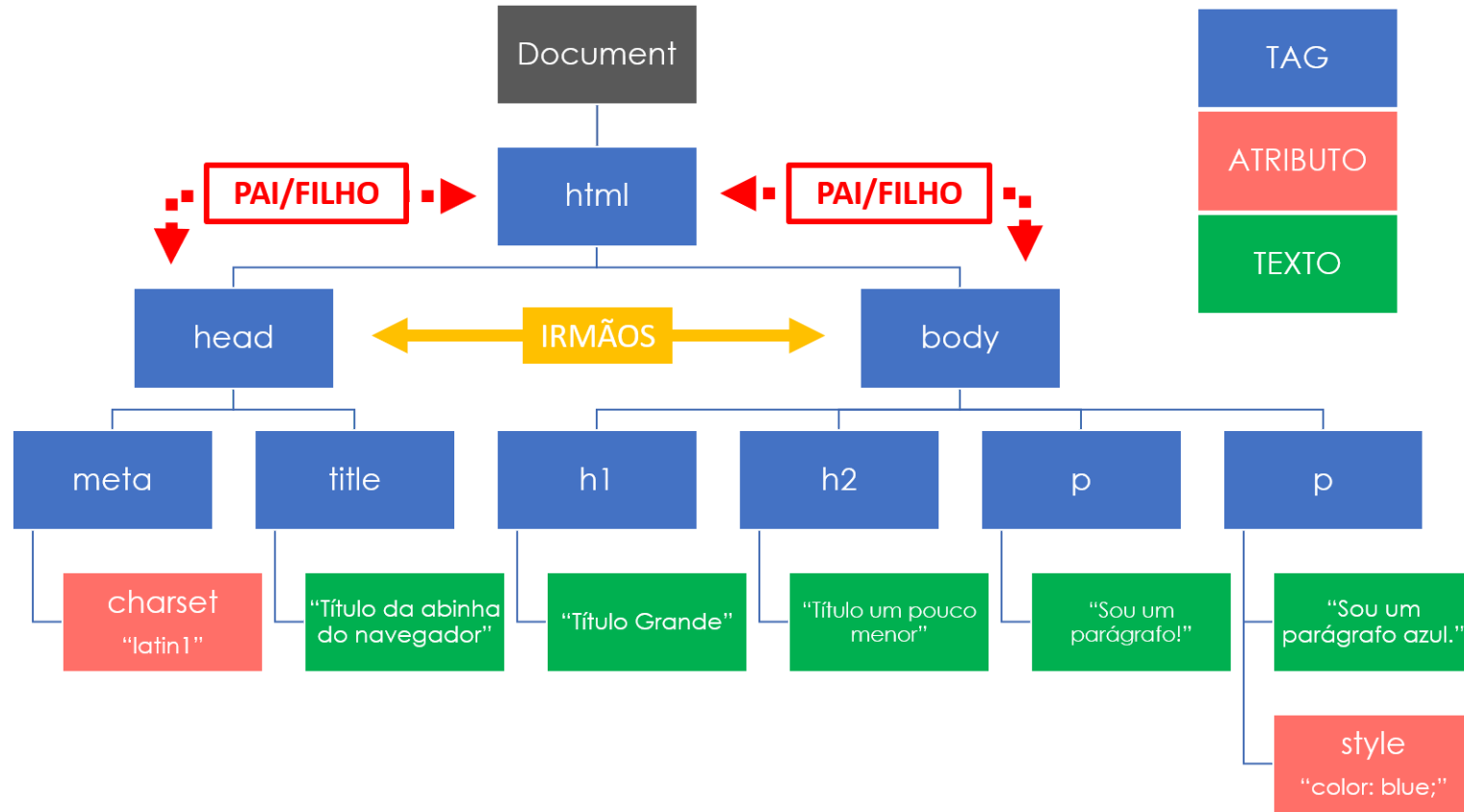
# Introdução ao HTML

- Todo HTML se transforma em um **DOM** (document object model) dentro do navegador.
- Um DOM pode ser representado como uma árvore em que cada *node* é:
  - ou um **atributo**
  - ou um **texto**
  - ou uma **tag**
  - ou um **comentário**
- Utiliza-se a relação de pai/filho/irmão entre os nós.
- Para descrever a estrutura de um DOM, usa-se uma linguagem de markup chamada XML (*Extensible Markup Language*) que define regras para a codificação de um documento.

# Introdução ao HTML



# Introdução ao HTML



# XPath - XML Path Language

- Exemplo: coletando todas as tags <p> (parágrafos)

```
library(xml2)

# Ler o HTML
html <- read_html("static/html_exemplo.html")

# Coletar todos os nodes com a tag <p>
nodes <- xml_find_all(html, "//p")

# Extrair o texto contido em cada um dos nodes
text <- xml_text(nodes)
text
```

```
#> [1] "Sou um parágrafo!"      "Sou um parágrafo azul."
```

# XPath - XML Path Language

- Com `xml_attrs()` podemos extrair todos os atributos de um node:

```
xml_attrs(nodes)
```

```
#> [[1]]  
#> named character(0)  
#>  
#> [[2]]  
#>           style  
#> "color: blue;"
```

```
xml_attr(nodes, "style")
```

```
#> [1] NA           "color: blue;"
```

# XPath - XML Path Language

- Já com `xml_children()`, `xml_parents()` e `xml_siblings()` podemos acessar a estrutura de parentesco dos nós:

```
heads <- xml_find_all(html, "head")
xml_siblings(heads)
```

```
#> {xml_nodeset (1)}
#> [1] <body>\n      <h1>Título Grande</h1>\n      \n      <h2>Título um pouco menor</ ...
```

```
xml_children(heads)
```

```
#> {xml_nodeset (3)}
#> [1] <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n
#> [2] <meta charset="latin1">\n
#> [3] <title>Título da abinha do navegador</title>
```



# CSS

- CSS (Cascading Style Sheets) descrevem como os elementos HTML devem se apresentar na tela. Ele é responsável pela aparência da página.

```
<p style='color: blue;'>Sou um parágrafo azul.</p>
```

- O atributo `style` é uma das maneiras de mexer na aparência utilizando CSS. No exemplo,
  - `color` é uma **property** do CSS e
  - `blue` é um **value** do CSS.
- Para associar esses pares **properties/values** aos elementos de um DOM, existe uma ferramenta chamada **CSS selectors**. Assim como fazemos com XML, podemos usar esses seletores (através do pacote `rvest`) para extrair os nós de uma página HTML.

# CSS

- Abaixo vemos um `.html` e um `.css` que é usado para estilizar o primeiro. Se os nós indicados forem encontrados pelos seletores do CSS, então eles sofrerão as mudanças indicadas.

# Seletores CSS vs. XPath

- A grande vantagem do XPath é permitir que acessemos os filhos, pais e irmãos de um nó. De fato os seletores CSS são mais simples, mas eles também são mais limitados.
- O bom é que se tivermos os seletores CSS, podemos transformá-los sem muita dificuldade em um query XPath:
  - Seletor de tag: `p` = `//p`
  - Seletor de classe: `.azul` = `//*[@class='azul']`
  - Seletor de id: `#meu-p-favorito` = `//*[@id='meu-p-favorito']`
- Além disso, a maior parte das ferramentas que utilizaremos ao longo do processo trabalham preferencialmente com XPath.

# Seletores CSS vs. XPath

```
html <- read_html("static/html_exemplo_css_a_parte.html")
xml_find_all(html, "//p")
```

```
#> {xml_nodeset (3)}
#> [1] <p>Sou um parágrafo normal.</p>
#> [2] <p class="azul">Sou um parágrafo azul.</p>
#> [3] <p id="meu-p-favorito" class="azul">Sou um parágrafo azul e negrito.</p>
```

```
xml_find_all(html, "//*[@class='azul']")
```

```
#> {xml_nodeset (2)}
#> [1] <p class="azul">Sou um parágrafo azul.</p>
#> [2] <p id="meu-p-favorito" class="azul">Sou um parágrafo azul e negrito.</p>
```

# Seletores CSS vs. XPath

```
rvest::html_nodes(html, ".azul")
```

```
#> {xml_nodeset (2)}  
#> [1] <p class="azul">Sou um parágrafo azul.</p>  
#> [2] <p id="meu-p-favorito" class="azul">Sou um parágrafo azul e negrito.</p>
```

- Note que `//p` indica que estamos fazendo uma busca na tag `p`, enquanto `//*` indica que estamos fazendo uma busca em qualquer tag.

# Exercício (eu)

Acesse o site de buscas DuckDuckGo.com. Baixe a página de buscas. Dica: use a função `http::GET()`.

```
library(httr)
GET("https://duckduckgo.com")
```

```
#> Response [https://duckduckgo.com/]
#>   Date: 2020-04-24 23:24
#>   Status: 200
#>   Content-Type: text/html; charset=UTF-8
#>   Size: 5.42 kB
#> <!DOCTYPE html>
#> <!--[if IEMobile 7 ]> <html lang="en_US" class="no-js iem7"> <![endif]-->
#> <!--[if lt IE 7]> <html class="ie6 lt-ie10 lt-ie9 lt-ie8 lt-ie7 no-js" lang="...
#> <!--[if IE 7]> <html class="ie7 lt-ie10 lt-ie9 lt-ie8 no-js" lang="en_US">...
#> <!--[if IE 8]> <html class="ie8 lt-ie10 lt-ie9 no-js" lang="en_US"> <![end...
#> <!--[if IE 9]> <html class="ie9 lt-ie10 no-js" lang="en_US"> <![endif]-->
#> <!--[if (gte IE 9)|(gt IEMobile 7)|!(IEMobile)|!(IE)]><!--><html class="no-js...
#>
```

# Exercício (nós)

Examine o código-fonte da página para encontrar o elemento correspondente à caixa de busca e copie o seu XPath pelo navegador. Esse XPath é apropriado? Por que?

```
//*[@id="search_form_input_homepage"]
```

Uma forma mais simples talvez fosse

```
//input[@name="q"]
```

# Exercício (vocês)

Examine a aba de rede do inspetor do seu navegador. Quando você faz uma busca, qual requisição corresponde à mesma? Reproduza essa requisição com o pacote `httr`. Dica: use o parâmetro `query` para enviar a requisição correta.

```
GET("https://duckduckgo.com", query = list(q = "R", t = "h_"))
```

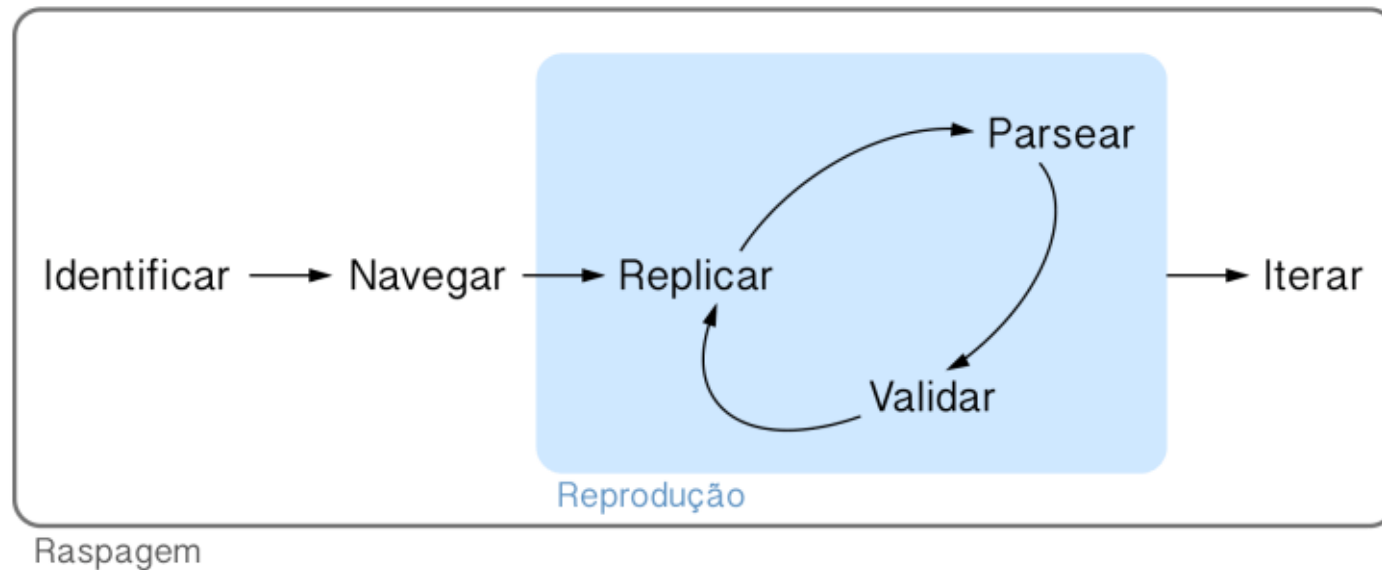
```
#> Response [https://duckduckgo.com/?q=R&t=h_]
#>   Date: 2020-04-24 23:24
#>   Status: 200
#>   Content-Type: text/html; charset=UTF-8
#>   Size: 26.2 kB
```



Iteração

# O fluxo do web scraping

- Sempre que fazemos um web scraper é bom seguir um fluxo definido
- Por enquanto já foram apresentados elementos da maior parte do passo-a-passo, mas nada foi dito sobre a iteração



# Por que iterar?

- Dificilmente queremos fazer uma tarefa de web scraping uma vez só (senão bastaria baixar a página uma vez e raspá-la)
- Podemos querer baixar muitas páginas de uma vez ou uma página a cada certo tempo
- Iteração, tratamento de erros e automatização passam a ser relevantes
  - O pacote `purrr` nos ajudará a iterar
  - O pacote `purrr` retornará para tratar qualquer erro que possa aparecer
  - O pacote `cronR` nos ajudará a agendar a execução de scripts
- Se você estiver interessado em aprender mais, veja nosso curso de Deploy

# Iterar

- Iteração é um padrão de programação extremamente comum que pode ser altamente abreviado

```
nums <- 1:10
resp <- c()
for (i in seq_along(nums)) {
  resp <- c(resp, nums[i] + 1)
}
resp
```

```
#> [1] 2 3 4 5 6 7 8 9 10 11
```

```
library(purrr)
map_dbl(nums, ~.x + 1)
```

```
#> [1] 2 3 4 5 6 7 8 9 10 11
```

# A função map

- A função `map()` recebe um vetor ou uma lista de entrada e aplica uma função em cada elemento do mesmo
- Podemos especificar o formato da saída com a família de funções `map_***()`
- A função pode ser declarada externamente, internamente ou através de um *lambda*

```
soma_um <- function(x) {  
  x + 1  
}  
  
map(nums, soma_um)  
map(nums, function(x) x + 1)  
map(nums, ~.x + 1)
```

# Utilidade do map

- Se tivermos uma lista de URLs, podemos iterar facilmente em todos sem abrir mão da sintaxe maravilhosa do Tidyverse

```
urls <- c(
  "https://en.wikipedia.org/wiki/R_language",
  "https://en.wikipedia.org/wiki/Python_(programming_language)"
)

urls %>%
  map(read_html) %>%
  map(xml_find_first, "//h1") %>%
  map_chr(xml_text)
```

```
#> [1] "R (programming language)"      "Python (programming language)"
```

# Tratando problemas

- Ao repetir uma tarefa múltiplas vezes, não podemos garantir que toda execução funcione
- O R já possui o `try()` e o `tryCatch()`, mas o `purrr` facilita ainda mais o trabalho

```
maybe_read_html <- possibly(read_html, NULL)  
read_html("https://errado.que")
```

```
#> Error in open.connection(x, "rb"): Could not resolve host: errado.que
```

```
maybe_read_html("https://errado.que")
```

```
#> NULL
```

# Agendamento

- Infelizmente o pacote mais comum (`cronR`) não funciona no Windows, nele é necessário usar o `taskscheduleR`

```
library(cronR)
cmd <- cron_rscript("CAMINHO PARA SCRIPT")

cron_add(cmd, "daily", "00:00")
cron_add(cmd, "daily", "14:20", days_of_week = c(0, 3, 5))
cron_add(cmd, "monthly", "10:30", days_of_month = "first")
cron_add(cmd, '@reboot')
```

- Também é possível utilizar uma interface interativa no RStudio em **Addins > Schedule R scripts**



# Exercício (eu)

Na página da Wikipédia, encontrar o objeto correspondente à tabela lateral de informações. Pegar apenas os elementos correspondentes a links.

```
links <- "https://en.wikipedia.org/wiki/R_language" %>%  
  read_html() %>%  
  xml_find_all("//table[@class='infobox vevent']/a")  
  
head(links)
```

```
#> {xml_nodeset (6)}  
#> [1] <a href="/wiki/File:R_logo.svg" class="image"><img alt="R Terminal.pn ...  
#> [3] <a href="/wiki/Programming_paradigm" title="Programming paradigm">Paradig ...  
#> [4] <a href="/wiki/Multi-paradigm_programming_language" class="mw-redirect" t ...  
#> [5] <a href="/wiki/Array_programming" title="Array programming">Array</a>  
#> [6] <a href="/wiki/Object-oriented_programming" title="Object-oriented progra ...
```

# Exercício (nós)

Extrair todos os URLs dos links e completá-los com o resto do caminho da Wikipédia. Continuar usando apenas *pipes*.

```
urls <- "https://en.wikipedia.org/wiki/R_language" %>%  
  read_html() %>%  
  xml_find_all("//table[@class='infobox vevent']/a") %>%  
  xml_attr("href") %>%  
  paste0("https://en.wikipedia.org", .)  
  
head(urls)
```

```
#> [1] "https://en.wikipedia.org/wiki/File:R_logo.svg"  
#> [2] "https://en.wikipedia.org/wiki/File:R_Terminal.png"  
#> [3] "https://en.wikipedia.org/wiki/Programming_paradigm"  
#> [4] "https://en.wikipedia.org/wiki/Multi-paradigm_programming_language"  
#> [5] "https://en.wikipedia.org/wiki/Array_programming"  
#> [6] "https://en.wikipedia.org/wiki/Object-oriented_programming"
```

# Exercício (vocês)

Baixar todas as páginas da Wikipédia. Dicas: use `possibly()` para impedir erros quando o URL for inválido; procure saber sobre a função `map2()` para iterar em mais de uma lista; salve os arquivos com `GET(..., write_disk(path))`.

```
paths <- paste0(dir, seq_along(urls), ".html")

maybe_get <- function(url, path) {
  possibly(GET, NULL)(url, write_disk(path))
}

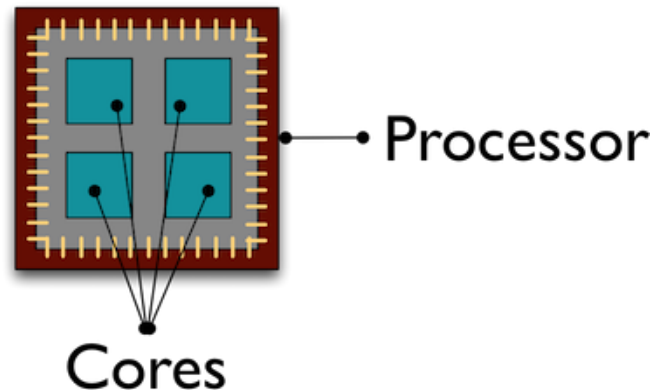
out <- map2(urls, paths, maybe_get)
length(compact(out))
```

```
#> [1] 32
```

Paralelismo

# O que isso significa?

- Antigamente, computadores eram capazes de executar apenas uma sequência de comandos por vez
- Avanços tecnológicos permitiram que o processador fosse capaz de fazer "malabarismo" com diversos processos
- Paralelismo (ou multiprocessamento) chegou apenas com os primeiros *dual-core*



# Em mais detalhes

- A unidade de processamento central pode ter mais de um **núcleo** (*multicore*)
- Um **processo** é composto por uma sequência de comandos ou tarefas
- Cada núcleo consegue executar apenas um **comando** por vez
- Os comandos de um processo podem ser interrompidos para que sejam executados os de outro (*multitasking*)
- O computador pode executar várias tarefas simultaneamente escalonando os comandos para seus diferentes núcleos (*multithreading*)
- Muitos computadores possuem **núcleos virtuais**, permitindo dois comandos por vez em cada núcleo (*hyperthreading*)

# Exemplo mínimo

O pacote `parallel` já vem instalado junto com o R e consegue rodar comandos paralelamente tanto no Windows quanto em outros sistemas. Por padrão, ele quebra a tarefa em 2.

```
library(parallel)
library(microbenchmark)

microbenchmark(
  seq = map_dbl(1:10000, function(x) x + 1),
  par = mclapply(1:10000, function(x) x + 1)
)
```

```
#> Unit: milliseconds
#>   expr      min       lq     mean  median      uq      max neval
#>   seq  8.630773 13.79190 17.49511 17.28082 20.08025 46.08841   100
#>   par 14.398487 28.10556 32.34163 33.18523 38.11062 58.92204   100
```

# Futuros

- O pacote `future` expande o pacote `parallel`, permitindo o descolamento de tarefas da sessão principal
  - Ele pode operar em 2 níveis: *multicore* e *multisession*
- Em cima do `future`, foi construído o `furrr` com o objetivo de emular a sintaxe do `purrr` para processamento paralelo
- Diferentemente do `parallel`, o `future` é capaz de descobrir sozinho o número de núcleos virtuais do computador

```
library(future)
availableCores()
```

```
#> system
#>      8
```



# Exercício (eu)

Estabelecer um plano de execução paralela com a função `plan()`. Entender a diferença entre todos os planos disponíveis.

```
plan(multiprocess)
```

- `sequential`: não executa em paralelo, útil para testes
- `multicore`: mais eficiente, não funciona no Windows nem dentro do RStudio
- `multisession`: abre novas sessões do R, mais pesado para o computador
- `multiprocess`: escolhe o melhor entre `multicore` e `multisession`

# Exercício (nós)

Criar uma função que retorna o primeiro parágrafo de uma página da Wikipédia dado o fim de seu URL (como `/wiki/R_language`). Dicas: textos são denotados pela *tag* `<p>` em HTML; pule o elemento de classe `"mw-empty-elt"`.

```
download_wiki <- function(url) {  
  url %>%  
    paste0("https://en.wikipedia.org", .) %>%  
    read_html() %>%  
    xml_find_first("//p[not(@class='mw-empty-elt')]") %>%  
    xml_text()  
}
```

# Exercício (vocês)

Executar a função anterior em paralelo para todas as páginas baixadas no exercício de iteração. Dicas: utilize `future_map()` do pacote `furrr`; não se esqueça do `possibly()`!

```
library(furrr)

prgs <- "https://en.wikipedia.org/wiki/R_language" %>%
  read_html() %>%
  xml_find_all("//table[@class='infobox vevent']/a") %>%
  xml_attr("href") %>%
  future_map(possibly(download_wiki, ""))

prgs[[3]]
```

```
#> [1] "Programming paradigms are a way to classify programming languages based on the
```

Selenium

# O que é Selenium?

- Selenium é uma ferramenta que permite **automatizar um navegador!**
- Suporta alguns *backends* diferentes: PhantomJS, Firefox, Chrome, etc.
- Diferentemente do web scraping normal, não precisamos nos preocupar com nenhuma requisição HTTP
  - O Selenium literalmente cria um navegador invisível para o qual você pode passar as **ações** a serem tomadas
  - Por ser uma sessão interativa, não há dificuldades em exibir conteúdo dinâmico
  - Não é necessário compreender o *networking* do site: tudo é *headless*

# Por que não usá-lo sempre?

- Vantagens:
  - Fácil de entender
  - Permite raspar dados dinâmicos
  - Permite *screen shots*
- Desvantagens:
  - Lento e de difícil paralelização
  - Bastante sensível
  - R Selenium está **completamente quebrado**

# WebDriver

- Não existe uma diferença real entre "Selenium" e "WebDriver"
  - O nome correto da ferramenta é Selenium WebDriver
- A diferença está no R: pacotes `RSelenium` e `webdriver`
  - `RSelenium` essencialmente não funciona
  - `webdriver` foi feito pela própria RStudio para resolver o problema
- O `webdriver` funciona somente com o PhantomJS, mas isso não é necessariamente um problema
- Instalar é fácil, fazer funcionar é mais ainda

# PhantomJS

- O PhantomJS é um navegador *headless* baseado em JavaScript feito especificamente para interação automatizada com páginas da web

```
library(webdriver)
# webdriver::install_phantomjs()

pjs <- run_phantomjs()
pjs
```

```
#> $process
#> PROCESS 'phantomjs', running, pid 157429.
#>
#> $port
#> [1] 3297
```

```
ses <- Session$new(port = pjs$port)
```



# Exemplo mínimo

```
ses$go("https://google.com")  
ses$takeScreenshot(file = arq)
```



# Elementos

- `ses$findElement()` retorna um elemento da página dado um seletor ou XPath para o mesmo
  - É uma função embutida na sessão (assim como `takeScreenshot()` e `go()`)
- `elem$click()` clica em um elemento, enquanto `elem$sendKeys()` "envia" uma tecla para o elemento
  - São funções embutidas no elemento retornado por `findElement()`
  - A lista `key` contém uma série de teclas que podem ser enviadas (como ENTER, etc.)
  - Ao invés de `elem$sendKeys()` podemos usar `elem$setValue()` para escrever um texto no elemento caso isso seja possível

# Exercício (eu)

Encontrar a página de Fundos de Investimento da XP. Criar uma sessão `webdriver` para ir até esta página.

```
xp <- paste0(  
  "https://institucional.xpi.com.br/investimentos/",  
  "fundos-de-investimento",  
  "/lista-de-fundos-de-investimento.aspx"  
)  
ses$go(xp)  
ses$takeScreenshot(file = arq)
```

# Exercício (eu)

MINHA CONTA

investimentos

[SOBRE A XP](#) [COMECE A INVESTIR](#) [INVESTIMENTOS](#) [OPERE NA BOLSA](#) [ATENDIMENTO](#)

ABRA SUA CONTA  
On-line. Rápido. Grátis.

Você está em: Investimentos

## FUNDOS DE INVESTIMENTO

[TODOS](#) [INTERNACIONAL](#) [RENTA FIXA](#) [MULTIMERCADOS](#) [AÇÕES](#) [CAMBIAL](#)

LEGENDA: menos risco mais risco

Risco  
Selecione

Classificação  
Selecione

Aplicação mínima  
Selecione

Cota resgate  
Selecione

Buscar Fundo

Conheça a classificação Morningstar®.

Classificação Morningstar®	Fundo	Aplic. mín.	Taxa adm. a.a.	Cota resg.	Rent. Mês	Rent. Ano	Rent. 12M	Detalhes	Aplicar
-	Bradesco Asset FIC FIRF CP LP Plus	R\$ 10.000,00	0,25%	D+6	N/D	N/D	N/D		
	CA Indosuez DI Master FI Renda Fixa Referenciado DI LP	R\$ 1.000,00	0,30%	D+0	-1,32%	-2,19%	0,96%		

# Exercício (nós)

Fazer a sessão webdriver clicar na aba "Internacional" no topo da página.

```
elem <- ses$findElement(xpath = '//a[@href="#referenciado"]')  
elem$click()  
ses$takeScreenshot(file = arq)
```

# Exercício (nós)

MINHA CONTA [LOGIN](#)

investimentos

[SOBRE A XP](#) [COMECE A INVESTIR](#) [INVESTIMENTOS](#) [OPERE NA BOLSA](#) [ATENDIMENTO](#)

[ABRA SUA CONTA](#)  
On-line. Rápido. Grátis.

Você está em: Investimentos

## FUNDOS DE INVESTIMENTO

[TODOS](#) [INTERNACIONAL](#) [RENTA FIXA](#) [MULTIMERCADOS](#) [AÇÕES](#) [CAMBIAL](#)

LEGENDA: menos risco mais risco

Risco  
Selecione

Classificação  
Selecione

Aplicação mínima  
Selecione

Cota resgate  
Selecione

Buscar Fundo

Conheça a classificação Morningstar®.

Classificação Morningstar®	Fundo	Aplic. mín.	Taxa adm. a.a.	Cota resg.	Rent. Mês	Rent. Ano	Rent. 12M	Detalhes	Aplicar
-	Claritas Global High Yield FIM IE	R\$ 25.000,00	0,87%	D+1	3,39%	-10,62%	-4,98%		<a href="#">INVESTIR</a>
-	PIMCO Income FIC FIM IE	R\$ 25.000,00	0,93%	D+1	2,62%	-6,28%	-1,13%		<a href="#">INVESTIR</a>
-	Schroder Liquid Alternatives IE FIM	R\$ 25.000,00	1,00%	D+7	2,31%	-5,80%	-2,03%		<a href="#">INVESTIR</a>

# Exercício (vocês)

Filtrar apenas os fundos de alto risco. Dica: podemos selecionar um elemento de uma lista com as setas do teclado (analisar `key`) ou podemos obter a estrutura da lista de seleções.

```
tab <- "//div[@id='tableReferenciadoRisk']"  
opt <- paste0(tab, "/span/select/option[@value='5']")  
  
elem <- ses$findElement(xpath = tab)  
elem$click()  
Sys.sleep(2)  
elem <- ses$findElement(xpath = opt)  
elem$click()  
  
ses$takeScreenshot(file = arq)
```

# Exercício (vocês)

MINHA CONTA

investimentos

[SOBRE A XP](#) [COMEÇE A INVESTIR](#) [INVESTIMENTOS](#) [OPERE NA BOLSA](#) [ATENDIMENTO](#)

ABRA SUA CONTA  
On-line. Rápido. Grátis.

Você está em: Investimentos

## FUNDOS DE INVESTIMENTO

[TODOS](#) [INTERNACIONAL](#) [RENTA FIXA](#) [MULTIMERCADOS](#) [AÇÕES](#) [CAMBIAL](#)

LEGENDA: menos risco mais risco

Risco  
Risco Alto

Classificação  
Selecione

Aplicação mínima  
Selecione

Cota resgate  
Selecione

Buscar Fundo

Conheça a classificação Morningstar®.

Classificação Morningstar®	Fundo	Aplic. min.	Taxa adm. a.a.	Cota resg.	Rent. Mês	Rent. Ano	Rent. 12M	Detalhes	Aplicar
-	AXA WF US High Yield Bonds Advisory FIC FIM IE CP	R\$ 5.000,00	0,60%	D+1	N/D	N/D	N/D		
-	Geo Empresas Globais em Reais FIC FIA IE	R\$ 25.000,00	2,00%	D+60	2,60%	-26,07%	-19,78%		
-	Geo Empresas Globais FIC FIA IE	R\$ 25.000,00	2,00%	D+60	8,89%	0,36%	12,35%		



# Quer mais?

- O PhantomJS, apesar de muito capaz, ainda não consegue exibir todo o conteúdo dinâmico de uma página
- Para solucionar esse problema, é necessário usar o RSelenium com um navegador de verdade como backend
  - Nem sempre a instalação do RSelenium funciona e em alguns sistemas operacionais há outras dependências
  - A documentação do RSelenium está atrasadas, dificultando qualquer pesquisa
  - O método sugerido para utilizar navegadores externos depende do Docker, um programa sem relação com o R
- Não use RSelenium caso não seja estritamente necessário!

# Demonstração

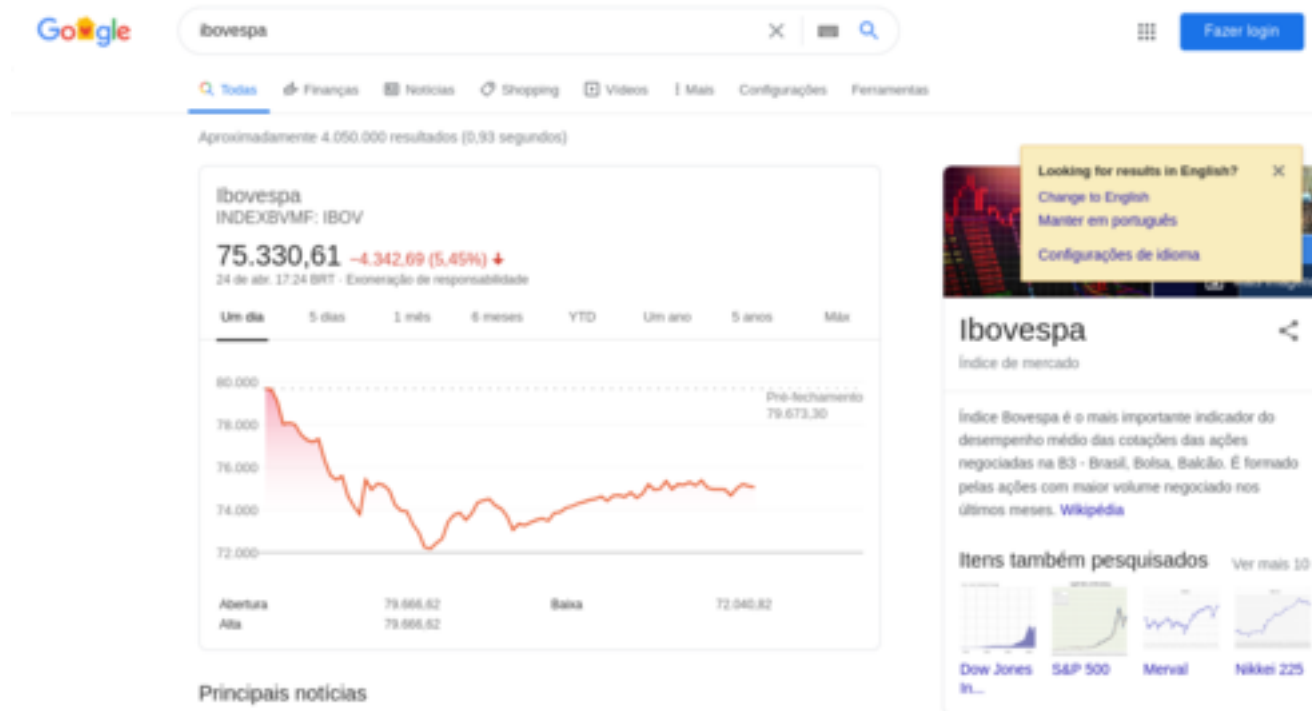
- As funções do `RSelenium` são parecidas com as do `webdriver`, mas envolvem um mais esforço
- No exemplo abaixo, o `RSelenium` abre uma aba do Firefox no meu computador e executa todos os comandos ao vivo nela

```
library(RSelenium)
drv <- rsDriver(browser = "firefox", verbose = FALSE)

drv$client$navigate("https://google.com")
elem <- drv$client$findElement("xpath", "//input[@name='q']")
elem$sendKeysToElement(list("ibovespa", key = "enter"))

Sys.sleep(2)
drv$client$screenshot(file = arq)
```

# Demonstração



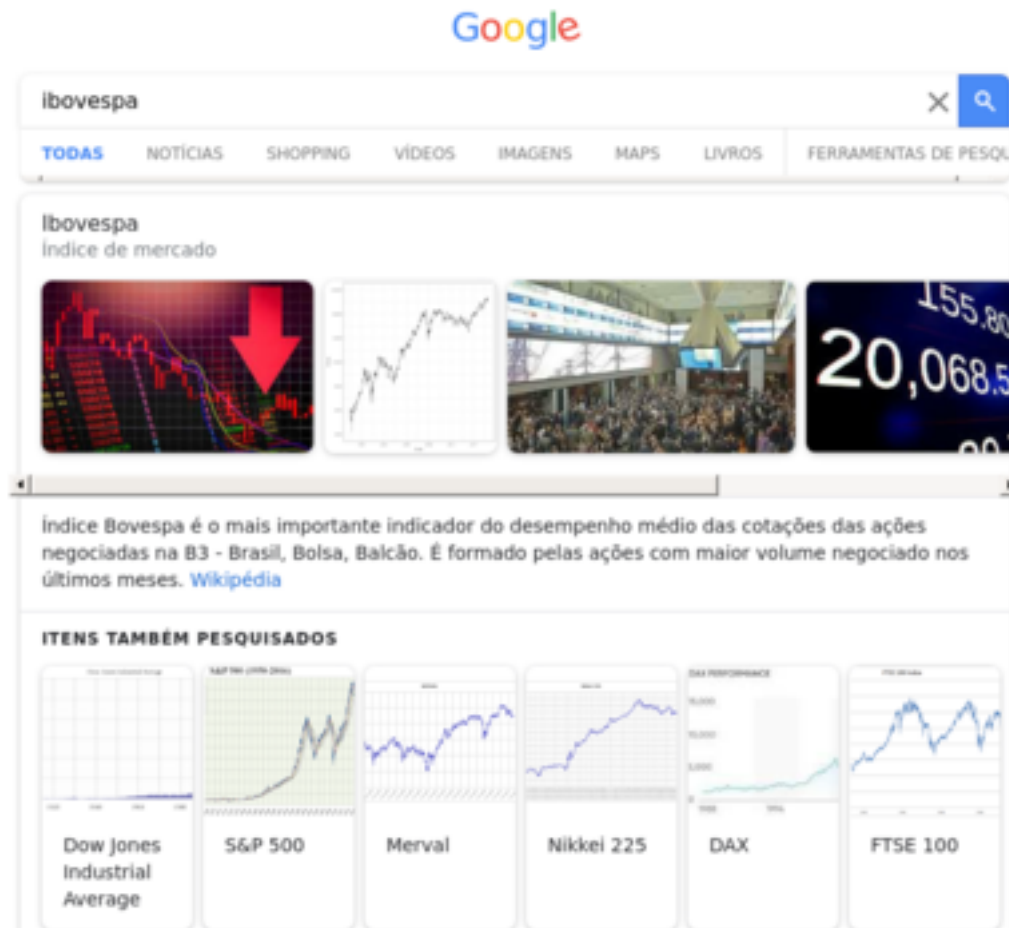
# Mas com o webdriver...

- Note a presença do gráfico interativo na imagem anterior, isso não é possível com o `webdriver`
- Pelas limitações do PhantomJS, nem todo elemento dinâmico pode ser renderizado na tela
  - É possível usar o `webdriver` com Docker também, mas nesse caso é melhor recorrer ao `RSelenium`

```
ses$go("https://google.com")
elem <- ses$findElement(xpath = "//input[@name='q']")
elem$sendKeys("ibovespa", key$enter)

Sys.sleep(2)
ses$takeScreenshot(file = arq)
```

# Mas com o webdriver...



# Miscelânea

# O DEJT

- O Diário Eletrônico da Justiça do Trabalho é publicado (quase) todo dia e contém todas as movimentações de vários tribunais do trabalho
- O nosso objetivo é baixar o PDF do diário para um dia específico

The screenshot shows the DEJT website interface. On the left is a sidebar menu with categories: Pesquisas (Diários, Matérias Publicadas, Feriados, Aferir Autenticidade de Diários), Acesso Restrito (Efetuar Login), and Serviços (Esqueceu Sua Senha?, Fale Conosco, Legislação, Manuais, Ocorrências). The main area is titled 'Pesquisar Diários' and contains a search form. The form has a header 'Informe os argumentos de pesquisa e clique em Pesquisar'. Below this are four fields: 'Data de Início' (17/04/2020), 'Data de Fim' (17/04/2020), 'Tipo de Caderno' (dropdown menu showing '[Todos]'), and 'Órgão' (dropdown menu showing '[Todos]'). There are 'Limpar' and 'F9-Pesquisar' buttons on the right. At the bottom, a footer contains contact information for the Conselho Superior da Justiça do Trabalho and session details: 'Hora login: 18:40' and 'Tempo sessão: 30.0m'.

**DEJT** Diário Eletrônico da Justiça do Trabalho

Pesquisas

- Diários
- Matérias Publicadas
- Feriados
- Aferir Autenticidade de Diários

Acesso Restrito

- Efetuar Login

Serviços

- Esqueceu Sua Senha?
- Fale Conosco
- Legislação
- Manuais
- Ocorrências

Pesquisar Diários

Informe os argumentos de pesquisa e clique em Pesquisar

Data de Início: 17/04/2020

Data de Fim: 17/04/2020

Tipo de Caderno: [Todos]

Órgão: [Todos]

Limpar F9-Pesquisar

Conselho Superior da Justiça do Trabalho - Endereço: SAFS - Qd 8, Conj A, Sala A5.71  
CEP 70.070-943 - Tel.: (61) 3043-4005 / 3043-3804  
dej-6.8.3-2

Hora login: 18:40 Tempo sessão: 30.0m

# A busca

- Para buscar os diários de um dia, basta entrar no site e escolher o dia correspondente
- Analisando o POST, vemos que não há muita dificuldade em reproduzir a requisição

```
▼ Form data
corpo:formulario:dataIni: "17/04/2020"
corpo:formulario:dataFim: "17/04/2020"
corpo:formulario:tipoCaderno: ""
corpo:formulario:tribunal: ""
corpo:formulario:ordenacaoPlc: ""
navDe: ""
detCorrPlc: ""
tabCorrPlc: ""
detCorrPlcPaginado: ""
exibeEdDocPlc: ""
indExcDetPlc: ""
corpo:formulario:alertaAlteracaoPlc: ""
org.apache.myfaces.trinidad.faces.FORM: "corpo:formulario"
_noJavaScript: "false"
javax.faces.ViewState: "-l-m1ev0wvfU"
source: "corpo:formulario:botaoAcaoPesquisar"
```



# Requisição da busca

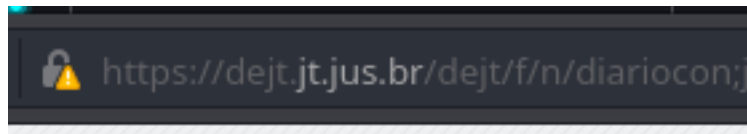
```
body <- list(  
  "corpo:formulario:dataIni" = "17/04/2020",  
  "corpo:formulario:dataFim" = "17/04/2020",  
  "corpo:formulario:tipoCaderno" = "",  
  "corpo:formulario:tribunal" = "",  
  "corpo:formulario:ordenacaoPlc" = "",  
  "navDe" = "",  
  "detCorrPlc" = "",  
  "tabCorrPlc" = "",  
  "detCorrPlcPaginado" = "",  
  "exibeEdDocPlc" = "",  
  "indExcDetPlc" = "",  
  "org.apache.myfaces.trinidad.faces.FORM" = "corpo:formulario",  
  "_noJavaScript" = "false",  
  "javax.faces.ViewState" = "!-mlev0wvfu",  
  "source" = "corpo:formulario:botaoAcaoPesquisar"  
)
```

# Reproduzindo a busca

```
url <- "https://dejt.jt.jus.br/dejt/f/n/diariocon"  
POST(url, body = body)
```

#> Error in curl::curl\_fetch\_memory(url, handle = handle): SSL certificate problem: un

- A primeira tentativa não funcionou por causa de um erro no certificado SSL
- SSL é o nome antigo para TLS (*Transport Layer Security*), uma ferramenta que permite que exista o HTTPS
  - HTTPS é a versão segura e criptografada do HTTP, o protocolo da *web*
- Essencialmente, isso quer dizer que a página do DEJT tem vulnerabilidade



# Reproduzindo a busca 2

```
resp <- POST(  
  url, body = body,  
  config(ssl_verifypeer = FALSE)  
)  
length(resp$content)
```

#> [1] 16946

- Apesar de a consulta ter funcionado, o tamanho da resposta (16kB) não está nem próximo do esperado
- Isso ocorre porque a página espera um POST do tipo formulário
  - A única diferença é o formato em que os dados serão enviados ao servidor

Content-Type: application/x-www-form-urlencoded

# Reproduzindo a busca 3

```
resp <- POST(  
  url, body = body, encode = "form",  
  config(ssl_verifypeer = FALSE)  
)  
length(resp$content)
```

```
#> [1] 16946
```

- Ainda sem sucesso, precisamos começar a conferir o formulário
- Se sairmos da página e tentarmos copiar o POST de novo, notaremos que o Viewstate mudou

```
javax.faces.ViewState: "l-m1ev0wvfs"
```

# Viewstate

- O JSF ViewState é um campo escondido que carrega consigo informações sobre o estado de uma sessão do navegador
- Ele é capaz de indicar para a próxima página acessada o que tinha acontecido até aquele momento
- No nosso caso o Viewstate é relevante apenas porque muitas páginas dependem dele para funcionar

```
# Procurando o Viewstate na página anterior à busca
viewstate <- url %>%
  GET(config(ssl_verifypeer = FALSE)) %>%
  content() %>%
  xml_find_first('//input[@name="javax.faces.ViewState"]') %>%
  xml_attr("value")
```

# Reproduzindo a busca 4

```
body$javax.faces.ViewState <- viewstate
resp <- POST(
  url, body = body, encode = "form",
  config(ssl_verifypeer = FALSE)
)
length(resp$content)
```

#> [1] 47286

- Agora que a resposta está correta, podemos continuar a raspagem da página sem mais problemas
- Basta identificar como baixar cada um dos PDFs

```
<button class="bt_af_commandButton" type="button" onclick="submitForm('corpo:formulario',1,
{source:'corpo:formulario:plcLogicaItens:0:j_id132'});return false;"> event
```

# Encontrando o PDF

- Assim como outras páginas feitas com Java, grande parte dos elementos interativos do DEJT possui um identificador único
- No nosso caso, precisamos encontrar o ID dos PDFs antes de poder baixá-los

```
jid <- resp %>%  
  read_html() %>%  
  xml_find_all("//button") %>%  
  xml_attr("onclick") %>%  
  stringr::str_extract("(?<=plcLogicaItens:0:)j_id[0-9]+") %>%  
  extract(!is.na(.))
```

# Requisição do PDF

```
body2 <- list(  
  "corpo:formulario:dataIni" = "17/04/2020",  
  "corpo:formulario:dataFim" = "17/04/2020",  
  "corpo:formulario:tipoCaderno" = "",  
  "corpo:formulario:tribunal" = "",  
  "corpo:formulario:ordenacaoPlc" = "",  
  "navDe" = "",  
  "detCorrPlc" = "",  
  "tabCorrPlc" = "",  
  "detCorrPlcPaginado" = "",  
  "exibeEdDocPlc" = "",  
  "indExcDetPlc" = "",  
  "org.apache.myfaces.trinidad.faces.FORM" = "corpo:formulario",  
  "_noJavaScript" = "false",  
  "javax.faces.ViewState" = viewstate,  
  "source" = paste0("corpo:formulario:plcLogicaItens:0:", jid)  
)
```



# Baixando o PDF

- Baixar um PDF funciona da mesma forma que baixar um HTML para o disco do computador, basta utilizar a função `write_disk()`

```
POST(  
    url, body = body2,  
    write_disk(arq, TRUE),  
    config(ssl_verifypeer = FALSE)  
)
```

```
#> Response [https://dejt.jt.jus.br/dejt/f/n/diariocon]  
#>   Date: 2020-04-24 23:26  
#>   Status: 200  
#>   Content-Type: application/pdf  
#>   Size: 9.46 MB  
#> <ON DISK>   /home/clente/Downloads/teste.pdf
```

# Fim

(Parabéns!)

# Feedback

<https://forms.gle/rPjT9GiY4T5ZFFEE9>