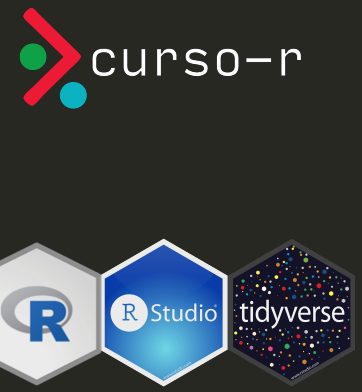
Web Scraping

XPath e CSS



XPath - XML Path Language

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

```
library(xml2)

# Ler o HTML
html <- read_html("img/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</h
```

```
xml_children(heads)
```

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

{rvest}

- Pacote construído sobre `{xml2}` e `{httr}`
- Busca facilitar a vida com alguns helpers
- Permite utilização de CSS path, uma alternativa ao XPath
- Na prática, no entanto, pode ser improdutivo utilizá-lo
- No nosso curso, só vamos utilizar a função `rvest::html_table()`, que transforma o conteúdo de uma tag `<table>` em um `data.frame`.

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.

```
9     <p>Sou um parágrafo normal.</p>
10
11    <p class='azul'>Sou um parágrafo azul.</p>
12
13    <p id="meu-p-favorito" class = 'azul'>Sou um parágrafo azul e negrito.</p>
14
```

No HTML	No CSS
<p>	p
class = ‘azul’	.azul
Id = ‘meu-p-favorito’	#meu-p-favorito

css_exemplo.css

```
1  p {
2    font-size: 25px;
3  }
4
5  .azul {
6    color: blue;
7  }
8
9  #meu-p-favorito {
10   font-weight: bold;
11 }
```

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("img/html_exemplo_css_a_parte.html")
xml_find_all(html, "//p")
```

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

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

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

Seletores CSS vs. XPath

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

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

Exemplo

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

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

```
## Response [https://duckduckgo.com/]
##   Date: 2020-11-16 04:33
##   Status: 200
##   Content-Type: text/html; charset=UTF-8
##   Size: 5.76 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="e
## <!--[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"> <![endi
## <!--[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"
##
## <head>
##     <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
## ...
```

Exemplo

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"]
```

Alternativa com CSS Path

```
#search_form_input_homepage
```

Uma forma mais simples talvez fosse

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

Vamos ao R!