

Getting Data from the Web with R

Part 7: Getting Data via Web Forms

Gaston Sanchez

April-May 2014

Content licensed under CC BY-NC-SA 4.0



Readme

License:

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
<http://creativecommons.org/licenses/by-nc-sa/4.0/>

You are free to:

- Share** — copy and redistribute the material
- Adapt** — rebuild and transform the material

Under the following conditions:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made.

NonCommercial — You may not use this work for commercial purposes.

Share Alike — If you remix, transform, or build upon this work, you must distribute your contributions under the same license to this one.

Lectures Menu

Slide Decks

1. Introduction
2. Reading files from the Web
3. Basics of XML and HTML
4. Parsing XML / HTML content
5. Handling JSON data
6. HTTP basics and the RCurl package
7. **Getting data via Web Forms**
8. Getting data via Web services

Data via Web Forms

Goal

Web Forms ...

The goal of the present slides is to show you **how to get data via web forms** with R

Synopsis

In a nutshell

We'll cover the following topics:

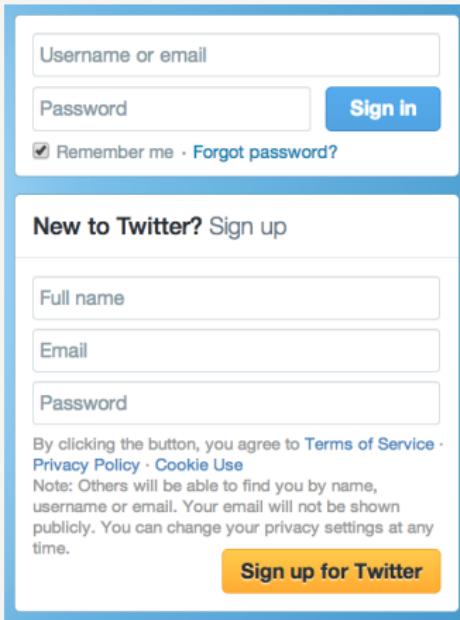
- ▶ Basics of HTML Forms
- ▶ HTTP requests in Web forms
- ▶ R package "RHTMLForms"
- ▶ How to use "RCurl" to query HTML Forms
- ▶ Case Studies

Some References

- ▶ XML and Web Technologies for Data Sciences with R
by Deb Nolan and Duncan Temple Lang
- ▶ HTML Forms W3C Tutorial
<http://www.w3.org/TR/html401/interact/forms.html>
- ▶ R Package "RHTMLForms"
<http://www.omegahat.org/RHTMLForms/>

HTML Forms

Examples of HTML Forms



The Twitter sign-up form features a blue header bar with a white 'Sign in' button. Below it is a 'Remember me' checkbox and a 'Forgot password?' link. The main section has fields for 'Full name', 'Email', and 'Password'. A note at the bottom states: 'By clicking the button, you agree to our Terms of Service · Privacy Policy · Cookie Use'. It also includes a note about privacy settings and a 'Sign up for Twitter' button.

Username or email

Password

Remember me · [Forgot password?](#)

New to Twitter? Sign up

Full name

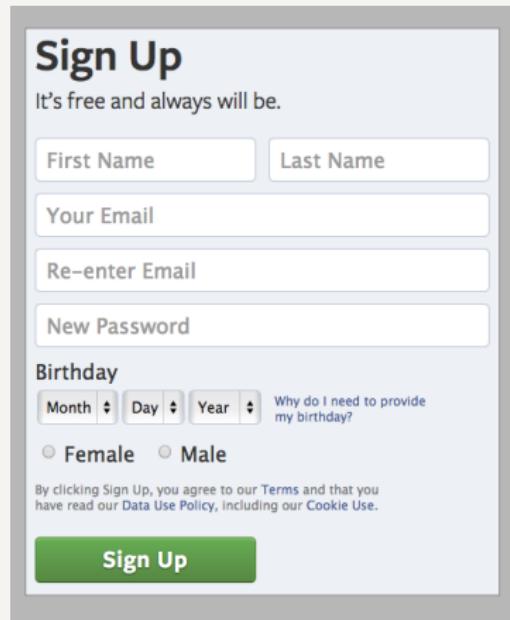
Email

Password

By clicking the button, you agree to our [Terms of Service](#) · [Privacy Policy](#) · [Cookie Use](#)

Note: Others will be able to find you by name, username or email. Your email will not be shown publicly. You can change your privacy settings at any time.

Sign up for Twitter



The sign-up form has a grey header bar with the word 'Sign Up' in large letters. Below it is a sub-header: 'It's free and always will be.' The main section contains fields for 'First Name' and 'Last Name', followed by 'Your Email', 'Re-enter Email', and 'New Password'. There is a 'Birthday' section with dropdown menus for Month, Day, and Year, and a link to explain why birthday is needed. At the bottom are gender selection buttons ('Female' and 'Male'), a note about agreeing to terms and conditions, and a large green 'Sign Up' button.

Sign Up

It's free and always will be.

First Name

Last Name

Your Email

Re-enter Email

New Password

Birthday

Month Day Year

Why do I need to provide my birthday?

Female Male

By clicking Sign Up, you agree to our [Terms](#) and that you have read our [Data Use Policy](#), including our [Cookie Use](#).

Sign Up

Examples of HTML Forms

Settings



General Labels Inbox Accounts and Import Filters Forwarding and POP/IMAP Chat Web Clips Labs Offline Themes Multiple inboxes

Language: Gmail display language: English (US) ▾
Change language settings for other Google products
Show all language options

Phone numbers: Default country code: United States ▾

Maximum page size: Show 50 ▾ conversations per page
Show 250 ▾ contacts per page

Images: Always display external images - Learn more
 Ask before displaying external images

Default reply behavior:
[Learn more](#)
 Reply
 Reply all

Default text style:
(Use the 'Remove Formatting' button on the toolbar to reset the default text style)

Sans Serif	TT	A	I
This is what your body text will look like.			

Conversation View:
(sets whether emails of the same topic are grouped together)
 Conversation view on
 Conversation view off

Email via Google+:
[Learn more](#)
Who can email you via your Google+ profile? Anyone on Google+ ▾
If people who aren't in your circles send you email this way, you must agree before they can send you more.

HTML Forms?

HTML Forms?

We use HTML Forms practically everyday:

- ▶ when we sign in to our email accounts
- ▶ when we make transactions online
- ▶ when we post comments
- ▶ when we fill online surveys
- ▶ *etc*

HTML Forms

HTML Forms (*aka Web Forms*):

- ▶ are used to gather information from the user
- ▶ transmit or *submit* data to a server
- ▶ add interactivity to websites

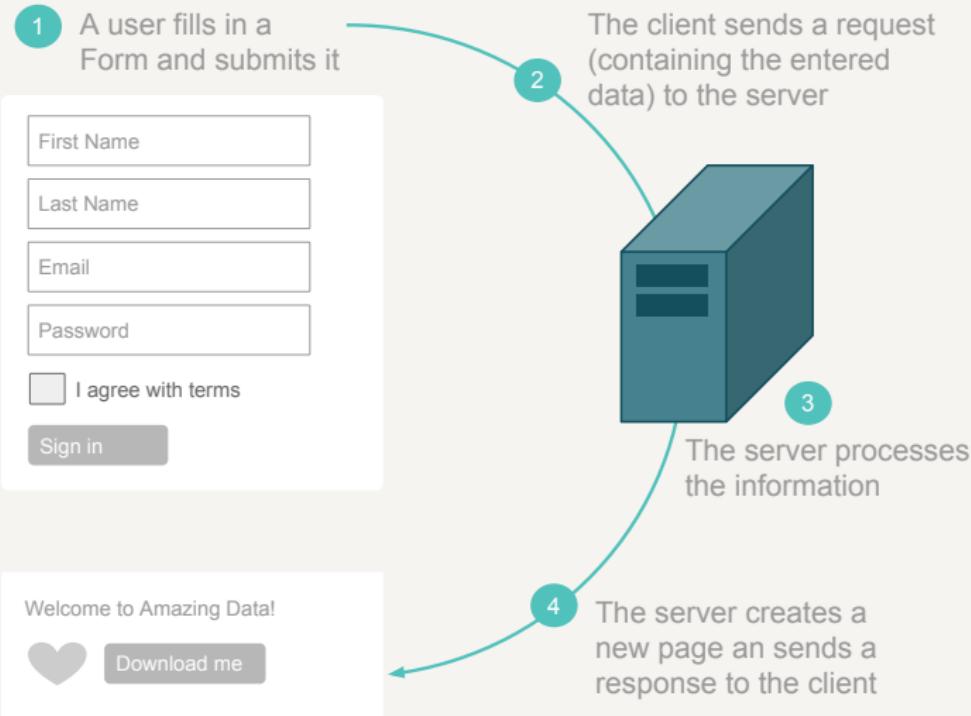
Importance

Why should we care?

Besides allowing websites to provide a way for interacting with users, HTML Forms play a prominent role regarding data on the Web.

Admittedly, **large amounts of data on the Web are accessible via HTML Forms**. Hence the importance for being familiar with the basic concepts around Web Forms.

How Forms Work



HTML Forms

Broadly speaking ...

Many websites have Forms that the visitors must use in order to provide some kind of input information.

This information is taken by the browser —ie the client— which submits a request to the server.

Based on the data submitted, the server takes an appropriate action and produces a customized response.

Basics of HTML Forms

HTML Form Element

Form Characteristics

- ▶ A Form is defined with an html `<form>` element.
- ▶ A `<form>` element has 2 main attributes: `action` and `method`
- ▶ A `<form>` element is a container for all the content of the form

Conceptual form

```
<form action="..." method="...>  
    [The input elements go here]  
</form>
```

Form Action and Method

Form Attributes

An HTML Form has 2 main attributes: **action** and **method**

- ▶ the **action** attribute points to the server side script that handles the form submission
- ▶ the **method** defines the type of HTTP request that is used to submit the information: GET or POST

HTML Forms

Form Controls

- ▶ The way Forms allow to capture input data is by using **form controls**
- ▶ Form controls live inside the `<form>` element
- ▶ There are several types of form controls (eg text boxes, select menus, checkboxes, radio buttons, submit buttons)
- ▶ Some of the more common controls are:
 - ▶ `<input>` (there are various classes of inputs)
 - ▶ `<textarea>`
 - ▶ `<select>`
 - ▶ `<option>`
 - ▶ `<button>`

Some HTML Form Controls

Control	Description
<input type="text">	text box
<input type="password">	password box
<input type="radio">	radio button
<input type="checkbox">	checkbox
<input type="submit">	submit button
<input type="reset">	reset button
<input type="hidden">	hidden field
<input type="button">	button
<select>	selection
<option>	selection
<textarea>	text area

Appearance of some Form Controls

Text input

Password input

Text area

Radio buttons

- small
- medium
- large

Checkboxes

- economy
- business
- first

Drop-down boxes

Continent ▾

- Africa
- America
- Asia
- Europe
- Oceania

Image buttons



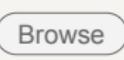
Sign in

Submitting buttons



Subscribe

File upload



Browse



Upload

About Form Controls

Form Control Names

The job of a form control is to gather one piece of information from the user. In order to uniquely identify the collected piece of data, controls have a **name** attribute.

Control examples: Username and Password

```
<form method="get" action="/bin/process">
    Enter your name: <input type="text" name="username"><br />
    Enter your password: <input type="password" name="password"><br />
</form>
```

HTML Form in a Website

[Camping and Hiking](#) > [Backpacks](#) > [Backpacking Packs](#)



ZOOM

ARROYO RED



Osprey Aether 70 Pack

Slip the Osprey Aether 70 pack onto your back and enjoy the comfort of a custom, heat-moldable hipbelt and a superb, lightweight design for your backcountry adventures.

\$289.95

★★★★★ (44)

Item # 846410

REI Members get back an estimated
\$ 29.00 on this item as part of
their annual member refund.

Choose Your Color/Size ▾

* Color/Size is out of stock but can be backordered.

1 Quantity

add to cart

find in store

[add to wish list](#)

THE REI DIFFERENCE | **100%** SATISFACTION GUARANTEED | GEAR & ADVICE YOU CAN TRUST

This item ships for free! [Learn more](#)

HTML Form Components

The image shows a user interface for selecting a product. At the top is a dropdown menu with the placeholder "Choose Your Color/Size". Below it is a note: "* Color/Size is out of stock but can be backordered." To the right of the note is a text input field containing the number "1", which is highlighted with a yellow border. Below the input field is the label "Quantity". At the bottom are three buttons: a red button with a shopping cart icon labeled "add to cart", a dark grey button with a store icon labeled "find in store", and a light grey button with a heart icon labeled "add to wish list". Three arrows point to these elements from the left: a teal arrow points to the dropdown, a gold arrow points to the quantity input, and a red arrow points to the "add to cart" button.

form element

```
<form id="productSelectForm2" name="additem" action="/AddItemToCartController" method="post">
```

selection element

```
<select id="sku" class="required prodLabel" name="sku" size="1"> ... </select>
```

input text element

```
<input id="quantity" class="required" type="text" name="quantity" maxlength="4" value="1">
```

button element

```
<button id="cartFormButton" class="medium primary button" type="submit"> ... </button>
```

```
</form>
```

Submitting Forms

Form Submission

Submission Request

Once the user fills in a Form and hits the submit button —or any other type of submit action— the browser gathers the input data and sends a request to the web browser.

HTTP Request Methods

There are 2 *requesting* mechanisms for submitting Web forms:

- ▶ using an **HTTP GET** request
- ▶ using an **HTTP POST** request

Form Submission

HTTP Request Methods

The type of request —GET or POST— depends on the **method** specified in the Web Form.

Attribute **method**

Remember that the HTTP request method is specified with the attribute **method** inside the `<form>` element.

Forms and HTTP Requests

HTTP methods in forms

The way the browser gathers the input data is by collecting the **name** and **value** of each of the controls. We say that the browser packs them into "**name=value**" pairs.

Both methods, GET and POST, take the set of name=value pairs to parametrize the request.

The difference between GET and POST is in **how the name=value pairs are delivered to the Web server**.

Toy Example: Coffee Form

Choose your coffee:

Beverage	▼
Espresso	
Americano	
Cappuccino	
Latte	
Mocha	

What Size?



small



medium



large

Decaf:



no



yes

```
<form method="get | post" action="/bin/process">
```

```
Choose your coffee: <br />
```

```
<select name="beverage">
```

```
    <option value="esp">Espresso</option>
    <option value="ame">Americano</option>
    <option value="cap">Cappuccino</option>
    <option value="lat">Latte</option>
    <option value="moc">Mocha</option>
</select> <br />
```

What size?

```
<input type="radio" name="size" value="small" />small
<input type="radio" name="size" value="medium" />medium
<input type="radio" name="size" value="large" />large<br />
```

Decaf:

```
<input type="checkbox" name="decaf" value="no" />no
<input type="checkbox" name="decaf" value="yes" />yes
</form>
```

Encoding of Name-Value Pairs

Name-Value Pairs

The browser gathers the **name** and **value** of each of the control fields forming **name=value** pairs:

- ▶ beverage=esp
- ▶ size=small
- ▶ decaf=no

Query String

The **name=value** pairs are concatenated together into a **query string** using "**&**" as a field separator:

beverage=esp&size=small&decaf=no

Query String and Submit Methods

GET or POST

The way the query string is handled depends on the request method:

- ▶ If GET is used, name-value pairs are appended to the requested URL
- ▶ If POST is used, the name-value pairs are sent as part of the request body

Query String in GET and POST



GET Method

GET method

With GET, the parameters are appended to the action and included as part of the URL in the HTTP request.

Query string appended to URL

```
GET /bin/process?beverage=esp&size=small&decaf=no HTTP/1.1
Host: 127.0.0.1:8000
Accept-Language: en-us
```

POST Method

POST method

With POST, the parameters are sent in the body of the HTTP request and the URL is simply the action

Query string as part of the request body

```
POST /bin/process HTTP/1.1
```

```
Host: 127.0.0.1:8000
```

```
Accept-Language: en-us
```

```
beverage=esp&size=small&decaf=no
```

R package "RHTMLForms"

Package RHTMLForms

RHTMLForms

There are several approaches to get data via Web Forms in R. One very nice way to do that is by using the package "RHTMLForms" (by Duncan Temple Lang)

What for?

"RHTMLForms" is an ad-hoc package that allows us to interact with Web Forms in a simple manner without having to worry about all the overwhelming details behind the HTTP requests, the control fields, and the query strings.

Installing RHTMLForms

Package Source

At the moment of this writing, "RHTMLForms" is NOT available on CRAN. Instead, the package must be downloaded from:

<http://www.omegahat.net/RHTMLForms/>

For these slides I'm using the package version 0.6-0 from the source file —*tarball*— **RHTMLForms_0.6-0.tar.gz**

Once you downloaded the package tar.gz file, you can proceed with the installation in R.

Installing RHTMLForms

Installation

Let's assume the *tarball* file **RHTMLForms_0.6-0.tar.gz** is in your Downloads directory.

To install "RHTMLForms" in R type the following in your console:

```
# installing RHTMLForms
install.packages("~/Downloads/RHTMLForms_0.6-0.tar.gz",
                 repos = NULL, type = "source")
```

Working with HTML Forms

Preliminaries

Getting data via HTML Forms requires you to understand the type of Forms you are interacting with:

- ▶ what method the Forms use (GET or POST)
- ▶ what control inputs they have (buttons, text, etc)
- ▶ what type of output is produced by the Server

Working with HTML Forms

Preliminaries

There's a *workflow procedure* that we should follow to take full advantage of "RHTMLForms":

1. we start by **exploring the website** that contains the Forms (how many forms, what structure they have, etc)
2. we get a **description** of the Forms we are interested in
3. we **figure out what inputs** we need to pass to the Forms
4. we **submit** the necessary requests
5. we **process** the documents produced by the Server

Using RHTMLForms

Let's start with the Google search form



Google Search Form

Structure of the Google Search Form

If you open your *web inspection tool* or see the source code of Google's Search homepage, you should be able to identify the HTML form:

```
<form action="/search" id="f" method="get">
  <div id="hf"></div>
  <div id="fkbx">
    <input id="q" jsaction="mousedown:ntp.fkbxclk" aria-hidden="true"
      autocomplete="off" name="q" tabindex="-1" style="opacity: 0;">
    <div id="fkbx_crt"></div>
    <div id="fkbx-spch" tabindex="1" aria-label="Search by voice"
      style="display: block;"></div>
    <div id="fkbx-hspch" tabindex="1" aria-label="Listening for "Ok Google"">
    </div>
    <div id="fkbx-hht">Say "Ok Google"</div>
  </div>
</form>
```

getHTMLFormDescription()

The main function to *explore* the Forms in a website is **getHTMLFormDescription()** which allows us to process an HTML document and get a description of its Forms:

```
# load RHTMLForms
library(RHTMLForms)

# google url
google = "http://www.google.com"

# get forms
google_forms = getHTMLFormDescription(google)

google_forms

## $f
## HTML Form: http://www.google.com/search
## q: [ ]
```

```
# put the form in a separate object
gform = google_forms$f

# object attributes
attributes(gform)

## $names
## [1] "formAttributes" "elements"      "url"
##
## $class
## [1] "HTMLFormDescription"
```

Note that the gform is of class "HTMLFormDescription"

In this case there's only one form:
q (which is a query search)

getHTMLFormDescription()

We can obtain a number of descriptive features from an object of class "HTMLFormDescription":

```
# form attributes  
gform$formAttributes
```

```
##                      action  
## "http://www.google.com/search"  
##                      method  
##                      "get"  
## attr(),"class")  
## [1] "HTMLFormAttributes"
```

```
# form url  
gform$url
```

```
##                      action  
## "http://www.google.com/search"
```

```
# form elements  
gform$elements
```

```
## $ie  
## ie: ISO-8859-1  
##  
## $hl  
## hl: en  
##  
## $source  
## source: hp  
##  
## $biw  
## biw:  
##  
## $bih  
## bih:  
##  
## $q  
## q: [ ]  
##  
## $gbv  
## gbv: 1  
##  
## attr(),"class")  
## [1] "HTMLFormElementsList"
```

Working with getHTMLFormDescription()

What does getHTMLFormDescription() do?

- ▶ parses an HTML document and processes each <form> element
- ▶ returns a list containing the identified <form>'s
- ▶ each list element is of class "HTMLFormDescription"
- ▶ determines the URL to which we should submit the form
- ▶ identifies the method for submitting the form (GET, POST)

For each "HTMLFormDescription" element we can check

- ▶ \$formAttributes
- ▶ \$elements
- ▶ \$url

`createFunction()`

`createFunction()`

There's a very handy function called `createFunction()` that we can use on each of the elements "HTMLFormDescription"

`createFunction()` generates an R function for **form submission**. Together with the package "RCurl", `createFunction()` allows us to specify values for each of the Form parameters, and obtain the result from the Web server.

getHTMLFormDescription()

createFunction() allows us to create an R function to generate our own *Google Search Form Submission*.

```
# create submitting function for google search Form  
google_submit = createFunction(gform)
```

With the created function, and using "RCurl", we can submit a search Form. Then we can parse the obtained outcome with one of the functions from "XML":

```
# load RCurl  
library(RCurl)  
  
# submit a search of the term "R project"  
r_project = google_submit("R project")  
  
# we can then parse the content  
library(XML)  
r_project = htmlParse(r_project)
```

Case Study: National Parks



PLEASE LOG IN OR SIGN UP

Find an Office National Parks Magazine



Search...

GO

Donate



About Us

Protecting Our Parks

Exploring Our Parks

News

Get Involved

Ways to Give

About the National Parks | Find a Park | Insider's Guide | Ten Most Visited Parks | Slideshows | Travel with NPCA

Home > Parks

1 2 54

Find a National Park

BROWSE ALPHABETICALLY: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

BY STATE:



BY NAME:

park name

BY LOCATION:

Select Location

BY CATEGORY:

Select Category

BY THEME:

Select Theme [View All Park Pages](#)

GET INVOLVED

e-News Sign Up

Become a Member

• \$25 • \$50 • \$100

National Parks Conservation Association

NPCA

The website of the *National Parks Conservation Association* (NPCA) has a page that allows us to find a National Park:

<http://www.nPCA.org/parks/>

What for

Let's use "RHTMLForms" to get information and descriptions about the forms available in the *Find a National Park* page.

Finding a National Park

Let's start with `getHTMLFormDescription()` to get the list of form descriptions:

```
# npca url
npca = "http://www.npca.org/parks"

# get descriptions of forms
park_forms = getHTMLFormDescription(npca)

# how many forms
length(park_forms)
```

```
## [1] 10
```

We are interested in the Forms that have to do with *Finding a National Park*

Inspecting NPCA Forms

The first three Forms are NOT the ones we want:

```
# 1st form (not interesting)
park_forms[[1]]

## HTML Form: /search-results.html
## q: [ Search... ]

# 2nd form (not interesting)
park_forms[[2]]

## HTML Form: http://npca.convio.net/site/Survey
## cons_email: [ ]

# 3rd form (not interesting)
park_forms[[3]]

## HTML Form: http://my.nPCA.org/site/Donation2?df_id=1333&Level=1716&1333.donation=form1&s_subsrc=sidebar_
## donation_amount: on
```

Inspecting NPCA Forms

The 4th Form does have to do with *Finding a National Park*

```
# 4th form (now we're talking)
park_forms[[4]]

## HTML Form: /exploring-our-parks/parks/search.jsp
## query: [ park name ]

# park name form
park_name = park_forms[[4]]

# form attributes
park_name$formAttributes

##                                     action
## "/exploring-our-parks/parks/search.jsp"
##                                     method
##                                     "get"
## attr(,"class")
## [1] "HTMLFormAttributes"
```

This is a **query** input that uses a GET method

Park State Form

Let's get the 5th form in a separate object before creating a submission form function:

```
# 5th form (park state form)
park_state = park_forms[[5]]
park_state$elements

## $state
## state: [ ] , al, ak, as, az, ar, ca, co, ct, de, dc, fl, ga, gu, hi, id, il, in, ia, ks, ky, la, me, m
## attr(,"class")
## [1] "HTMLFormElementsList"

park_state$formAttributes

##                                     action
## "/exploring-our-parks/parks/park-list.html"
##                                     method
##                                     "get"
## attr(,"class")
## [1] "HTMLFormAttributes"
```

Park State Form

Here's how to create a submission form function for the Park State Form:

```
require(RCurl)

# create 'form submission' function for park state
park_state_fun = createFunction(park_state)

# submission for parks in California
cal_parks = park_state_fun("California")
```

Finally, we can use the `getHTMLLinks()` function from "XML" to extract all the links:

```
require(XML)

# then we can extract the links
cal_park_links = getHTMLLinks(cal_parks)
```