jQuery

jQuery is a **library**, or set of helpful add-ons, to the **JavaScript** programming language. It may seem counterintuitive to learn how to use a library before learning the actual language, but there are a few good reasons for this.
An HTML document is structured according to the **Document Object Model**, or **DOM**. It's by interacting with the DOM that jQuery is able to access and modify HTML.
The DOM consists of every element on the page, laid out in a hierarchical way that reflects the way the HTML docum

$(document).ready(function() {
  Do something
});
Let's go through it bit by bit.

- $(document) is a jQuery object. The $() is actually a function in disguise; it turns the document into **a jQuery object.**
- .ready() is a type of function; you can think of it as sort of a helper that runs the code inside its parentheses as soon as the HTML document is ready.
- function(){} is the action that .ready() will perform as soon as the HTML document is loaded. (In the above example, the Do something placeholder is where those actions would go.)

Variables are a place for us to store information for use at a later time.
Variables can hold any type of information you work with, so just think of them as containers.
var $p = $('p'); // $p is **just a variable name**

$('div')**.mouseenter**
$('div')**.mouseleave**
$('div')**.click**
 $('div').**slideDown(**'slow'**)**
.fadeTo('fast', 0.25);

We can apply jQuery functions to classes, ids of HTML element (selectors) ; you can apply the function to more than one element:

$(**'p, li, div, ul, .class_name'**).fadeTo('slow', 0); // all elements separated by commas, inside one single quotes, for all selectors

---

The **this** keyword refers to the jQuery object you're currently doing something with. Its complete rules are a little tricky, but the important thing to understand is if you use an **event handler** on an element—that's the fancy name for

actions like .click() and .mouseenter(), since they handle jQuery events—you can call the actual **event** that occurs (such as fadeOut()) on $(this), and the event will *only* affect the element you're currently doing something with (for example, clicking on or mousing over).

```
$(document).ready(function() {
   $('div').click(function() {
      $(this).fadeOut('slow');
   });
});
```

---

var **$h1 = $("<h1>Hello</h1>")**; //we create the $h1 a jQuery object containing a string

---

.append() inserts the specified element as the last child of the target element. **.prepend()** inserts the specified element as the *first* child of the target element. If we have a div of class .info,
$(".info")**.append**(" Stuff! ”);// equivalent $(' Stuff! ')**.appendTo**('.info');
$(".info")**.prepend**(" Stuff!);// $(' Stuff! ’)**.prependTo**('.info');

We can specify where in the DOM we insert an element with the **.before()** and **.after()** functions. The syntax looks like this:
$('target')**.after**('<tag>To add</tag>');
Where 'target' is the element after which you want to add something and the bit between <tag>s is the HTML element you want to add. You can add <h1>s, s, or any other valid HTML you like.

Moving elements around in the DOM is a snap—all we need to do is use the jQuery functions we just learned on existing elements instead of creating new ones.

```
var $paragraph = $("p"); // existing element
$("div").after($paragraph); // Move it!
// Same as:
$("div").after($("p"));
```

1. We can select an element using **$("p")** and assign it to a variable
2. We can move the position in the DOM by using the variable in our after() statement

Note: This does not copy the element from one location to another, it moves the original element effectively saving you from having to delete the original

---

.empty() deletes an element's content and ***all its descendants***. For instance, if you .empty() an 'ol', you'll also remove all its 'li's and their text.
.remove(), not only deletes an element's content, but **deletes the element itself.**

$('p').remove();//removes all p tags from the html

---

jQuery includes two functions, **.addClass()** and **.removeClass(),** that can be used to add or remove a class from an element. This is great if, for example, you have a highlighted class that you want to apply to an element when clicked.
$('selector')**.addClass('className')**;

$('selector')**.toggleClass('className')**; //adds/removes the class

```
$("div").height("100px");
$("div").width("50px");
```

would give all s on the page a height of 100 pixels and a width of 50 pixels.
jQuery also includes a general-purpose .css() function that takes two inputs: the first is the CSS element to alter, and the second is the value to set it to. For example:

```
$("div")css("background-color","#008800");
```

Modifying Content
Finally, we can **update the contents of our HTML elements**—that is, the bit between the opening and closing tags—using the **.html()** and **.val() functions.**
.html() can be used to set the contents of the first element match it finds. For instance,
$('div').html();
will **GET the HTML contents** of the *first* div it finds, and
$('div').html("I love jQuery!");
will **SET the contents of the first div** it finds to "I love jQuery!"
.val() is used to **get** the **value** of **form** elements. For example,
$('input:checkbox:checked').val();
would **get** the **value of the first checked checkbox** that jQuery finds.

```
<form name="checkListForm">
        <input type="text" name="checkListItem"/>
</form>
```

//Get the value from an input
var input = $(**'input[name=checkListItem]').val();** //name=nume_input must be written with no spaces

---

Remove What's Been Clicked
Great job! Finally, we want to be able to check items off our list.
You might think we could do this:

```
$('.item').click(function() {
    $(this).remove();
});
```

and that's not a bad idea. The problem is that it won't work—jQuery looks for all the .items when the DOM is loaded, so by the time your document is ready, it's already decided there are no .items to .remove(), and your code won't work.
For this, we'll need a new event handler: **.on()**. You can think of .on() as a general handler that takes the event, its selector, and an action as inputs. The syntax looks like this:

```
$(document).on('event', 'selector', function() {
    Do something!
});
```

In this case, 'event' will be 'click', 'selector' will be '.item', and the thing we'll want to do is call .remove() on this.

```
    $(document).on('click','.item', function(){
        $(this).remove();
        });
    });
```

https://learn.jquery.com/events/
http://api.jquery.com/category/events/

```
$(document).ready(function() {
    $('button').click(function() {
        var toAdd = $("input[name=message]").val();
        $('#messages').append("<p>"+toAdd+"</p>");
    });
});
```

---

http://learn.jquery.com/effects/
http://api.jquery.com/category/effects/

HOVER

```
$('div').hover(
    function(){
      $(this).addClass('highlight');
   },
   function(){
```

```
    $(this).removeClass('highlight');
  }
);
```

1. We first select the element we want to modify $('div')
2. Secondly notice that **our hover effect is able to take two functions(){} separated by a comma.** The comma is very important!

FOCUS

```
$(document).ready(function(){
   $('input').focus(function(){
      $('input').css('outline-color','#f00');
      $('input').css('outline-style','solid')});
});
```

The **.keydown()** Event
It only works on whatever page element has focus, so you'll need to click on the window containing your div before pressing a key in order for you to see its effects.

```
$(document).ready(function(){
   $('div').keydown(function(){
//We want the whole document object to respond whenever a key is pressed.)
      $('div').animate({left:' += 10px'},500);
      })});
```

This **will take the first div it finds and move it ten pixels** to the right during 500 milliseconds.
//.animate({top:'-=10px'},'fast') moves it 10px down fast

---

new jQuery library: **jQuery UI**.
includes a number of ultra-fancy animations you can use to make your websites do incredible things.

```
$('div').effect('explode');
$('div').effect('bounce', {times:2}, 200);
```

http://jqueryui.com/

```
$("#menu").accordion({collapsible: true, active: false});
```

jQuery UI includes a **.draggable()** function that can make any HTML element draggable—you can click on it and move it anywhere on the page!

```
$(document).ready(function(){
```

```
  $('#car').click(function(){
     $('#car').draggable();
  });
   });
```

where the car is made from css:

```
#top {
   position: relative;
   height: 50px;
   width: 50px;
   border-radius: 5px;
   background-color: #cc0000;
   left: 25px;
}
#bottom {
        position: relative;
        height:25px;
        width: 100px;
        background-color: #cc0000;
        border-top-left-radius: 5px;
        border-top-right-radius: 5px;
        top: -25px;
}
#fwheel {
        position: relative;
        height:20px;
        width:20px;
        border-radius: 100%;
        background-color: black;
        top: -35px;
        left: 5px;
}
#bwheel {
        position: relative;
        height:20px;
        width:20px;
        border-radius: 100%;
        background-color: black;
        top: -55px;
        left: 75px;
}
```

One Resize Fits All

```
$(document).ready(function(){
   $('div').click(function(){
      $('div').resizable();});
   });
```

A Greater Selection

with jQuery UI—**we can also enhance our ordered and unordered lists.**

```
$(document).ready(function(){
   $('ol').selectable();
   });
```

Let's **Sort Things Out** with jQuery UI .sortable() function

```
$(document).ready(function(){
   $('ol').sortable();
   });
```

```
 <div id="menu">
          <h3>Section 1</h3>
          <div>
             <p>I'm the first section!</p>
          </div>
          <!--Add two more sections below!-->
          <h3>Section 2</h3>
           <div>
             <p>I'm the 2 section!</p>
          </div>
          <h3>Section 3</h3>
           <div>
             <p>I'm the3th section!</p>
          </div>
       </div>

$(document).ready(function(){
   $('#menu').accordion();
   });
```

---

window.outerHeight

It's the height of the window on screen, it includes the page and all the visible browser's bars (location, status, bookmarks, window title, borders, ...).

This **not** the same as jQuery's $(window).outerHeight().

window.innerHeight or $(window).height()

It's the height of the viewport that shows the website, just the content, no browser's bars.

document.body.clientHeight or $(document).height()

It's the height of your document shown in the viewport. If it is higher than $(window).height() you get the scrollbars to scroll the document.

screen.availHeight

It's the height the browser's window can have if it is maximized, including the browser's bars. So when the window is maximized, screen.availHeight === window.outerHeight

screen.height

It simply matches the screen's resolution. So on a 1920×1080 screen, screen.height will be 1080.
screen.availHeight is equal to [screen.height + the operating system's bars], like the taskbar on Windows, the dock and menu on OS X, or whatever is fixed on top or bottom of your screen if you're using Linux.

After some digging, I discover that different Android versions (and devices) have different devicePixelRatio values.
If the devicePixelRatio is equal or greater than 1, the actual number of pixels in the screen (for the html page point of view) is given by window.screen.width (or ...height).
But, if the window.screen.width is less than 1 (it happens in some old Android devices), the actual number of pixels becomes: window.screen.width/devicePixelRatio.
So, you just have to cope with this.

```
w = window.screen.width;
h = window.screen.height;

if(window.devicePixelRatio < 1){
  w = window.screen.width/window.devicePixelRatio;
  h = window.screen.height/window.devicePixelRatio;
}

 var seenWidth = window.screen.width;
     var seenHeight = window.screen.innerHeight -200;

if(window.devicePixelRatio < 1){
  seenWidth = window.screen.width/window.devicePixelRatio;
  seenHeight = (window.screen.innerHeight/window.devicePixelRatio) -50;
}
```