

Presentation by Uplatz

Contact Us: <https://training.uplatz.com/>

Email: info@uplatz.com

Phone: +44 7836 212635

Table Of Contents:

- Selectors
- Each function
- Attributes
- document-ready event
- Events

Getting Started with JQuery:

Create a file hello.html with the following content:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Hello, World!</title>
```

```
</head>
```

```
<body>
```

```
<div>
```

```
<p id="hello">Some random text</p>
```

```
</div>
```

```
<script src="https://code.jquery.com/jquery-  
2.2.4.min.js"></script>
```

```
<script>
$(document).ready(function() {
$('#hello').text('Hello, World!');
});
</script>
</body>
</html>
```

Loads the jQuery library from the jQuery CDN:

```
<script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
```

- This introduces the \$ global variable, an alias for the jQuery function and namespace.
- Be aware that one of the most common mistakes made when including jQuery is failing to load the library

- BEFORE any other scripts or libraries that may depend on or make use of it.
- Defers a function to be executed when the DOM (Document Object Model) is detected to be "ready" by jQuery:

```
// When the `document` is `ready`, execute this function `...`
```

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

```
// A commonly used shorthand version (behaves the same as the above)
```

```
$(function() { ... });
```

- Once the DOM is ready, jQuery executes the callback function shown above. Inside of our function, there is only one call which does 2 main things:

- Gets the element with the id attribute equal to hello (our selector #hello).
- Using a selector as the passed argument is the core of jQuery's functionality and naming; the entire library essentially evolved from extending **document.querySelector** **AllMDN**.
- Set the text() inside the selected element to Hello, World!.

↓ - Pass a `selector` to `\$` jQuery, returns our element

\$('#hello').text('Hello, World!');

↑ - Set the Text on the element

Avoiding namespace collisions

- Libraries other than jQuery may also use \$ as an alias. This can cause interference between those libraries and jQuery.
- To release \$ for use with other libraries:

jQuery.noConflict();

- After calling this function, \$ is no longer an alias for jQuery.
- However, you can still use the variable jQuery itself to access jQuery functions:
- `jQuery('#hello').text('Hello, World!');`
- Optionally, you can assign a different variable as an alias for jQuery:

`var jqy = jQuery.noConflict();`

jqy('#hello').text('Hello, World!');

- Conversely, to prevent other libraries from interfering with jQuery, you can wrap your jQuery code in an immediately invoked function expression (IIFE) and pass in jQuery as the argument:

```
(function($) {  
  $(document).ready(function() {  
    $('#hello').text('Hello, World!');  
  });  
})(jQuery);
```

Inside this IIFE, \$ is an alias for jQuery only.

- Another simple way to secure jQuery's \$ alias and make sure DOM is ready:


```
jQuery(function( $ ) { // DOM is ready
  // You're now free to use $ alias
  $('#hello').text('Hello, World!');
});
```

To summarize,

- **jQuery.noConflict()** : \$ no longer refers to jQuery, while the variable jQuery does.
- **var jQuery2 = jQuery.noConflict()** - \$ no longer refers to jQuery, while the variable jQuery does and so does the variable jQuery2.
- Now, there exists a third scenario - What if we want jQuery to be available only in jQuery2

Use,

```
var jQuery2 = jQuery.noConflict(true)
```

jQuery Namespace ("jQuery" and "\$")

- jQuery is the starting point for writing any jQuery code.
- It can be used as a function `jQuery(...)` or a variable `jQuery.foo`.
- `$` is an alias for jQuery and the two can usually be interchanged for each other (except where `jQuery.noConflict();` has been used - see Avoiding namespace collisions).

Assuming we have this snippet of HTML -

`<div id="demo_div" class="demo"></div>`

- We might want to use jQuery to add some text content to this div. To do this we could use the `jQuery text()` function.

This could be written using either jQuery or \$. i.e. -

jQuery("#demo_div").text("Demo Text!");

Or -

\$("#demo_div").text("Demo Text!");

➤ **Both will result in the same final HTML -**

<div id="demo_div" class="demo">Demo Text!</div>

- As \$ is more concise than jQuery it is the generally the preferred method of writing jQuery code.
- jQuery uses CSS selectors and in the example above an ID selector was used. For more information on selectors in jQuery see types of selectors.

Loading jQuery via console on a page that does not have it

- Sometimes one has to work with pages that are not using jQuery while most developers are used to have jQuery handy.
- In such situations one can use Chrome Developer Tools console (F12) to manually add jQuery on a loaded page by running following:

```
var j = document.createElement('script');  
j.onload = function(){ jQuery.noConflict(); };  
j.src =  
"https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jqu  
ery.min.js";  
document.getElementsByTagName('head')[0].appendChi  
ld(j);
```

- Copy the full code (or click on the copy icon) and paste it in the <head> or <body> of your html.
- The best practice is to load any external JavaScript libraries at the head tag with the async attribute.
Here is a demonstration:

<!DOCTYPE html>

<html>

<head>

<title>Loading jquery-2.2.4</title>

<script src="https://code.jquery.com/jquery-2.2.4.min.js" async></script>

</head>

<body><p>This page is loaded with jquery.</p>

</body>

</html>

- When using async attribute be conscious as the javascript libraries are then asynchronously loaded and executed as soon as available.
- If two libraries are included where second library is dependent on the first library is this case if second library is loaded and executed before first library then it may throw an error and application may break
- The jQuery Object Every time jQuery is called, by using `$()` or `jQuery()`, internally it is creating a new instance of jQuery.
- This is the source code which shows the new instance:

```
// Define a local copy of jQuery  
jQuery = function( selector, context ) {
```

- `// The jQuery object is actually just the init constructor 'enhanced'`
- `// Need init if jQuery is called (just allow error to be thrown if not included)`

`return new jQuery.fn.init(selector, context);`

- Internally jQuery refers to its prototype as `.fn`, and the style used here of internally instantiating a jQuery object allows for that prototype to be exposed without the explicit use of `new` by the caller.
- In addition to setting up an instance (which is how the jQuery API, such as `.each`, `children`, `filter`, etc. is exposed),
- internally jQuery will also create an array-like structure to match the result of the selector (provided that something other than nothing, undefined, null, or similar was passed as the argument).

- In the case of a single item, this arraylike structure will hold only that item.
- A simple demonstration would be to find an element with an id, and then access the jQuery object to return the underlying DOM element (this will also work when multiple elements are matched or present).


```
var $div = $("#myDiv");
```

```
//populate the jQuery object with the result of the id selector
```

```
var div = $div[0];
```

- //access array-like structure of jQuery object to get the DOM Element

Selectors

- A jQuery selectors selects or finds a DOM (document object model) element in an HTML  document.

- It is used to select HTML elements based on id, name, types, attributes, class and etc.
- It is based on existing CSS selectors.

Overview

- Elements can be selected by jQuery using jQuery Selectors.
- The function returns either an element or a list of elements.

Basic selectors

`$("*")` // All elements

`$("div")` // All <div> elements

`$(".blue")` // All elements with class=blue

`$(".blue.red")` // All elements with class=blue AND class=red

`$(".blue,.red")` // All elements with class=blue OR class=red

`$("#headline")` // The (first) element with id=headline

`$("[href]")` // All elements with an href attribute

`$("[href='example.com']")` // All elements with href=example.com

Relational operators

`$("div span")` // All s that are descendants of a <div>

`$("div > span")` // All s that are a direct child of a <div>

`$("a ~ span")` // All s that are siblings following an <a>

`$("a + span")` // All s that are immediately after an <a>

Types of Selectors

- In jQuery you can select elements in a page using many various properties of the element, including:

Type

Class

ID

Possession of Attribute

Attribute Value

Indexed Selector

Pseudo-state

- If you know CSS selectors you will notice selectors in jQuery are the same (with minor exceptions).
- Take the following HTML for example:

` <!-- 1 -->`

` <!-- 2 -->`

` <!-- 3 -->`

` <!-- 4 -->`

` <!-- 5 -->`

Selecting by Type:

- The following jQuery selector will select all `<a>` elements, including 1, 2, 3 and 4.

`$("a")`

Selecting by Class

- The following jQuery selector will select all elements of class `example` (including non-`a` elements), which are 3, 4 and 5.

`$(".example")`

Selecting by ID

- The following jQuery selector will select the element with the given ID, which is 2.

`$("#second-link")`

Selecting by Possession of Attribute

- The following jQuery selector will select all elements with a defined href attribute, including 1 and 4.

`$("[href]")`

Selecting by Attribute Value

- The following jQuery selector will select all elements where the href attribute exists with a value of index.html, which is just 1.

`$("[href='index.html']")`

Selecting by Indexed Position (Indexed Selector)

- The following jQuery selector will select only 1, the second <a> ie. the second-link because index supplied is 1 like eq(1) (Note that the index starts at 0 hence the second got selected here!).

`$("a:eq(1)")`

Selecting with Indexed Exclusion

- To exclude an element by using its index :not(:eq())
- The following selects <a> elements, except that with the class example, which is 1

`$("a").not(":eq(0)")`

Selecting with Exclusion

- To exclude an element from a selection, use :not()
- The following selects <a> elements, except those with the class example, which are 1 and 2.

`$("a:not(.example)")`

Selecting by Pseudo-state

- You can also select in jQuery using pseudo-states, including :first-child, :last-child, :first-of-type, :last-of-type, etc.
- The following jQuery selector will only select the first <a> element: number 1.

`$("a:first-of-type")`

Combining jQuery selectors

- You can also increase your specificity by combining multiple jQuery selectors; you can combine any number of them or combine all of them.
- You can also select multiple classes, attributes and states at the same time.

`$("a.class1.class2.class3#someID[attr1][attr2='something'][attr3='something']:first-of-type:firstchild")`

- This would select an `<a>` element that:
- Has the following **classes: class1, class2, and class3**
- Has the following **ID: someID**
- Has the following **Attribute: attr1**
- Has the following **Attributes and values: attr2** with value something, attr3 with value something
- Has the following states: first-child and first-of-type
- You can also separate different selectors with a comma:

`$("a, .class1, #someID")`

This would select:

- All `<a>` elements
- All elements that have the class class1
- An element with the id #someID

Child and Sibling selection

- jQuery selectors generally conform to the same conventions as CSS, which allows you to select children and siblings in the same way.
- To select a non-direct child, use a space
- To select a direct child, use a >
- To select an adjacent sibling following the first, use a +
- To select a non-adjacent sibling following the first, use a ~

Wildcard selection

- There might be cases when we want to select all elements but there is not a common property to select upon (class, attribute etc).
- In that case we can use the * selector that simply selects all the elements:

`$('#wrapper *')` // Select all elements inside #wrapper element

Caching Selectors:

- Each time you use a selector in jQuery the DOM is searched for elements that match your query.
- Doing this too often or repeatedly will decrease performance.
- If you refer to a specific selector more than once you should add it to the cache by assigning it to a variable:

```
var nav = $('#navigation');  
nav.show();
```

This would replace:

```
$('#navigation').show();
```

- Caching this selector could prove helpful if your website needs to show/hide this element often.
- If there are multiple elements with the same selector the variable will become an array of these elements:

```
<div class="parent">
```

```
  <div class="child">Child 1</div>
```

```
  <div class="child">Child 2</div>
```

```
</div>
```

```
<script>
```

```
  var children = $('.child');
```

```
  var firstChildText = children[0].text();
```

```
  console.log(firstChildText);
```

```
// output: "Child 1"
```

```
</script>
```

- NOTE: The element has to exist in the DOM at the time of its assignment to a variable.
- If there is no element in the DOM with a class called child you will be storing an empty array in that variable.

```
<div class="parent"></div>
```

```
<script>
```

```
var parent = $('.parent');
```

```
var children = $('.child');
```

```
console.log(children);
```

```
// output: []
```

```
parent.append('<div class="child">Child 1</div>');
```

```
children = $('.child');
```

```
console.log(children[0].text());
```

```
// output: "Child 1"
```

</script>

- Remember to reassign the selector to the variable after adding/removing elements in the DOM with that selector.
- **Note:** When caching selectors, many developers will start the variable name with a \$ to denote that the variable is a jQuery object like so:

```
var $nav = $('#navigation');
```

```
$nav.show();
```

Combining selectors

Consider following DOM Structure

```
<ul class="parentUI">
```

```
<li> Level 1
```

```
<ul class="childUI">
```

```
<li>Level 1-1 <span> Item - 1 </span></li>
```

```
<li>Level 1-1 <span> Item - 2 </span></li>
</ul>
</li>
<li> Level 2
  <ul class="childUI">
    <li>Level 2-1 <span> Item - 1 </span></li>
    <li>Level 2-1 <span> Item - 1 </span></li>
  </ul>
</li></ul>
```

- Descendant and child selectors
- Given a parent - parentUI find its descendants (),

1. Simple \$('parent child')

```
>> $('ul.parentUI li')
```

- This gets all matching descendants of the specified ancestor all levels down.

2. > - \$('parent > child')

```
>> $('ul.parentUl > li')
```

- This finds all matching children (only 1st level down).

3. Context based selector - \$('child','parent')

```
>> $('li','ul.parentUl')
```

- This works same as 1. above.

4. find() - \$('parent').find('child')

```
>> $('ul.parentUl').find('li')
```

- This works same as 1. above.

5. children() - \$('parent').find('child')

```
>> $('ul.parentUl').children('li')
```

This works same as 2. above.

Other combinators

Group Selector : ","

- Select all elements AND all elements AND all elements :

`$('ul, li, span')`

Multiple selector : "" (no character)

- Select all elements with class parentUl :

`$('ul.parentUl')`

Adjacent Sibling Selector : "+"

- Select all elements that are placed immediately after another element:

`$('li + li')`

General Sibling Selector : "~"

- Select all elements that are siblings of other elements:

`$('li ~ li')`

DOM Elements as selectors

- jQuery accepts a wide variety of parameters, and one of them is an actual DOM element.
- Passing a DOM element to jQuery will cause the underlying array-like structure of the jQuery object to hold that element.
- jQuery will detect that the argument is a DOM element by inspecting its `nodeType`.
- The most common use of a DOM element is in callbacks, where the current element is passed to the jQuery constructor in order to gain access to the jQuery API.
- Such as in the `each` callback (note: `each` is an iterator function).

```
$(".elements").each(function(){
```

```
//the current element is bound to `this` internally by  
jQuery when using each
```

```
var currentElement = this;
```

```
//at this point, currentElement (or this) has access to  
the Native API
```

```
//construct a jQuery object with the  
currentElement(this)
```

```
var $currentElement = $(this);
```

```
//now $currentElement has access to the jQuery API  
});
```

HTML strings as selectors

- jQuery accepts a wide variety of parameters as "selectors", and one of them is an HTML string.

Passing an HTML

- string to jQuery will cause the underlying array-like structure of the jQuery object to hold the resulting constructed

HTML.

- jQuery uses regex to determine if the string being passed to the constructor is an HTMLstring, and also that it must

start with <. That regex is defined as `quickExpr = /^(?:\s*(<[\w\W]+>)[^>]* | #([\w-]*))$/` (explanation at regex101.com).

- The most common use of an HTML string as a selector is when sets of DOM elements need to be created in code only, often this is used by libraries for things like Modal popouts.

- For example, a function which returned an anchor tag wrapped in a div as a template

```
function template(href,text){  
  return $("<div><a href='" + href + "'>" + text +  
  "</a></div>");  
}
```

Would return a jQuery object holding

<div>

Google

</div>

if called as `template("google.com","Google")`.

Each function

jQuery each function

HTML:

```
<ul>
```

```
<li>Mango</li>
```

```
<li>Book</li>
```

```
</ul>
```

Script:

```
$( "li" ).each(function( index ) {  
  console.log( index + ": " + $( this ).text() );  
});
```

A message is thus logged for each item in the list:

0: Mango

1: Book

Attributes

Difference between attr() and prop()

- **attr()** gets/sets the HTML attribute using the DOM functions **getAttribute()** and **setAttribute()**.
- **prop()** works by setting the DOM property without changing the attribute.
- In many cases the two are interchangeable, but occasionally one is needed over the other.

To set a checkbox as checked:

- `$('#tosAccept').prop('checked', true);` // using `attr()` won't work properly here
- To remove a property you can use the `removeProp()` method.
- Similarly `removeAttr()` removes attributes.

Get the attribute value of a HTML element

- When a single parameter is passed to the `.attr()` function it returns the value of passed attribute on the selected element.

Syntax:

```
$([selector]).attr([attribute name]);
```

Example:

HTML:

```
<a href="/home">Home</a>
```

jQuery:

```
$('a').attr('href');
```

Fetching data attributes:

- jQuery offers .data() function in order to deal with data attributes.
- .data function returns the value of the data attribute on the selected element.

Syntax:

```
$([selector]).data([attribute name]);
```

Example:

Html:

```
<article data-column="3"></article>
```

jQuery:

```
$("article").data("column")
```

Note:

Setting value of HTML attribute

- If you want to add an attribute to some element you can use the `attr(attributeName, attributeValue)` function.

For example:

```
$('a').attr('title', 'Click me');
```

- This example will add mouseover text "Click me" to all links on the page.

- The same function is used to change attributes' values.

Removing attribute:

- To remove an attribute from an element you can use the function `.removeAttr(attributeName)`.

For example:

`$('#home').removeAttr('title');`

- This will remove title attribute from the element with ID home.

document-ready event

What is document-ready and how should I use it?

- jQuery code is often wrapped in `jQuery(function($){ ... });`
- so that it only runs after the DOM has finished loading.

```
<script type="text/javascript">  
jQuery(function($) {  
  // this will set the div's text to "Hello".  
  $("#myDiv").text("Hello");  
});  
</script>
```

```
<div id="myDiv">Text</div>
```

- This is important because jQuery (and JavaScript generally) cannot select a DOM element that has not been rendered to the page.

```
<script type="text/javascript">  
  // no element with id="myDiv" exists at this point, so  
  $("#myDiv") is an  
  // empty selection, and this will have no effect
```

```
$("#myDiv").text("Hello");
```

```
</script>
```

```
<div id="myDiv">Text</div>
```

- Note that you can alias the jQuery namespace by passing a custom handler into the `.ready()` method.
- This is useful for cases when another JS library is using the same shortened `$` alias as jQuery, which create a conflict.
- To avoid this conflict, you must call `$.noConflict();` - This forcing you to use only the default jQuery namespace
- (Instead of the short `$` alias).
- By passing a custom handler to the `.ready()` handler, you will be able to choose the alias name to use jQuery.

```
$.noConflict();
```

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

```
  // Here we can use '$' as jQuery alias without it  
  conflicting with other
```

```
  // libraries that use the same namespace
```

```
$('#body').append('<div>Hello</div>')
```

```
});
```

```
jQuery( document ).ready(function( jq ) {
```

```
  // Here we use a custom jQuery alias 'jq'
```

```
jq('body').append('<div>Hello</div>')
```

```
});
```

- Rather than simply putting your jQuery code at the bottom of the page, using the `$(document)`.
- ready function ensures that all HTML elements have been rendered and the entire Document Object Model

- (DOM) is ready for JavaScript code to execute
jQuery 2.2.3 and earlier
- These are all equivalent, the code inside the blocks
will run when the document is ready:

```
$(function() {
```

```
  // code
```

```
});
```

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

```
  // code
```

```
});
```

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

```
  // code
```

```
});
```

- Because these are equivalent the first is the recommended form, the following is a version of that with the jQuery keyword instead of the \$ which produce the same results:

```
jQuery(function() {
```

```
// code
```

```
});
```

jQuery 3.0

Notation

As of jQuery 3.0, only this form is recommended:

```
jQuery(function($) {
```

```
// Run when document is ready
```

```
// $ (first argument) will be internal reference to jQuery
```

```
// Never rely on $ being a reference to jQuery in the  
global namespace
```

```
});
```

Asynchronous

- As of jQuery 3.0, the ready handler will always be called asynchronously.
- This means that in the code below, the log 'outside handler' will always be displayed first, regardless whether the document was ready at the point of execution.

```
$(function() {  
  console.log("inside handler");  
});  
console.log("outside handler");
```

> outside handler

> inside handler

Attaching events and manipulating the DOM
inside ready()

Example uses of \$(document).ready():

1. Attaching event handlers

Attach jQuery event handlers

```
$(document).ready(function() {  
    $("button").click(function() {  
        // Code for the click function  
    });  
});
```

➤ Run jQuery code after the page structure is created

```
jQuery(function($) {  
    // set the value of an element.  
    $("#myElement").val("Hello");  
});
```

3. Manipulate the loaded DOM structure

- **For example:** hide a div when the page loads for the first time and show it on the click event of a button

```
$(document).ready(function() {  
    $("#toggleDiv").hide();  
    $("button").click(function() {  
        $("#toggleDiv").show();  
    });  
});
```

Difference between `$(document).ready()` and `$(window).load()`

- `$(window).load()` was deprecated in jQuery version 1.8 (and completely removed from jQuery 3.0) and as such should not be used anymore.

- The reasons for the deprecation are noted on the jQuery page about this event Caveats of the load event when used with images
- A common challenge developers attempt to solve using the `.load()` shortcut is to execute a function when an image (or collection of images) have completely loaded.
- There are several known caveats with this that should be noted.

These are:

- It doesn't work consistently nor reliably cross-browser
- It doesn't fire correctly in WebKit if the image src is set to the same src as before
- It doesn't correctly bubble up the DOM tree

- Can cease to fire for images that already live in the browser's cache
- If you still wish to use **load()** it is documented below:
- **\$(document).ready()** waits until the full DOM is available -- all the elements in the HTML have been parsed and are in the document.
- However, resources such as images may not have fully loaded at this point.
- If it is important to wait until all resources are loaded, **\$(window).load()** and you're aware of the significant limitations of this event then the below can be used instead:

```
$(document).ready(function() {  
    console.log($("#my_large_image").height()); // may  
    be 0 because the image isn't available  
});
```

```
$(window).load(function() {  
  console.log($("#my_large_image").height()); // will be  
  correct  
});
```

Difference between jQuery(fn) and executing your code before </body>

- Using the document-ready event can have small performance drawbacks, with delayed execution of up to ~300ms.
- Sometimes the same behavior can be achieved by execution of code just before the closing </body> tag:

<body>

** world!**

<script>

```
$("#greeting").text("Hello");
```

```
</script>
```

```
</body>
```

- will produce similar behavior but perform sooner than as it does not wait for the document ready event trigger as it does in:

```
<head>
```

```
<script>
```

```
jQuery(function($) {
```

```
$("#greeting").text("Hello");
```

```
});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<span id="greeting"></span> world!
```

</body>

- Emphasis on the fact that first example relies upon your knowledge of your page and placement of the script just prior to the closing </body> tag and specifically after the span tag.

Events

Delegated Events

Let's start with example. Here is a very simple example HTML.

Example HTML

<html>

<head>

</head>

<body>

```
<ul>  
  <li>  
    <a href="some_url/">Link 1</a>  
  </li>  
  <li>  
    <a href="some_url/">Link 2</a>  
  </li>  
  <li>  
    <a href="some_url/">Link 3</a>  
  </li>  
</ul>  
</body>  
</html>
```

- The problem Now in this example, we want to add an event listener to all `<a>` elements.
- The problem is that the list in this example is dynamic. `` elements are added and removed as time passes by.
- However, the page does not refresh between changes, which would allow us to use simple click event listeners to the link objects (i.e. `$('a').click()`).
- The problem we have is how to add events to the `<a>` elements that come and go.

Background information - Event propagation

- Delegated events are only possible because of event propagation (often called event bubbling).
- Any time an event is fired, it will bubble all the way up (to the document root).

- They delegate the handling of an event to a non-changing ancestor element, hence the name "delegated" events.
- So in example above, clicking <a> element link will trigger 'click' event in these elements in this order:

a

li

ul

body

html

document root

Solution

- Knowing what event bubbling does, we can catch one of the wanted events which are propagating up through our HTML.

- A good place for catching it in this example is the `` element, as that element does is not dynamic:

```
$('ul').on('click', 'a', function () {
```

```
  console.log(this.href); // jQuery binds the event  
  function to the targeted DOM element
```

```
  // this way `this` refers to the anchor and not to the list
```

```
  // Whatever you want to do when link is clicked
```

```
});
```

In above:

- We have 'ul' which is the recipient of this event listener
- The first parameter ('click') defines which events we are trying to detect.
- The second parameter ('a') is used to declare where the event needs to originate from (of all child elements under this event listener's recipient, ul,).

- Lastly, the third parameter is the code that is run if first and second parameters' requirements are fulfilled.

In detail how solution works

1. User clicks `<a>` element
2. That triggers click event on `<a>` element.
3. The event start bubbling up towards document root.
4. The event bubbles first to the `` element and then to the `` element.
5. The event listener is run as the `` element has the event listener attached.
6. The event listener first detects the triggering event. The bubbling event is 'click' and the listener has 'click', it is a pass.

.

7. The listener checks tries to match the second parameter ('a') to each item in the bubble chain.

As the last item in the chain is an 'a' this matches the filter and this is a pass too.

8. The code in third parameter is run using the matched item as it's this.

If the function does not include a call to `stopPropagation()`, the event will continue propagating upwards towards the root (document).

- **Note:** If a suitable non-changing ancestor is not available/convenient, you should use document.
- As a habit do not use 'body' for the following reasons:
- body has a bug, to do with styling, that can mean mouse events do not bubble to it.

- This is browser dependant and can happen when the calculated body height is 0 (e.g. when all child elements have absolute positions).
- Mouse events always bubble to document.

document always exists to your script, so you can attach delegated handlers to document outside of a DOMready handler and be certain they will still work

- We can rearticulate our usage of `<Content />` by passing in multiple activities instead of just one:

`<Content activities={activities} />`

- However, if we refresh the view nothing will show up! We need to first update our Content component to accept multiple activities.
- As we learned about previously, JSX is really just JavaScript executed by the browser.
- We can execute JavaScript functions inside the JSX content as it will just get run by the browser like the rest of our JavaScript.
- Let's move our activity item JSX inside of the function of the map function that we'll run over for every item.

```
class Content extends React.Component {  
  render() {  
    const {activities} = this.props; // ES6 destructuring  
    return (  
      <div className="content">  
        <div className="line"></div>  
        { /* Timeline item */ }  
        {activities.map((activity) => {  
          return (  
            <div className="item">  
              <div className="avatar">  
                <img  
                  alt={activity.text}  
                  src={activity.user.avatar} />  
              </div>  
            </div>  
          )  
        })}  
      </div>  
    );  
  }  
}
```

```
{activity.user.name}  
    </div>  
<span className="time">  
    {activity.timestamp}  
</span>  
<p>{activity.text}</p>  
<div className="commentCount">  
    {activity.comments.length}  
</div>  
</div>  
);  
})</div>  
)  
}
```


- Now we can pass any number of activities to our array and the Content component will handle it, however if we leave the component right now, then we'll have a relatively complex component handling both containing and displaying a list of activities

ActivityItem

- Here is where it makes sense to write one more component to contain displaying a single activity item and then rather than building a complex Content component, we can move the responsibility.
- This will also make it easier to test, add functionality, etc.
- Let's update our Content component to display a list of ActivityItem components (we'll create this next).

```
class Content extends React.Component {  
  render() {  
    const {activities} = this.props; // ES6 destructuring  
    return (  
      <div className="content">  
        <div className="line"></div>  
  
        {/* Timeline item */}  
        {activities.map((activity) => (  
          <ActivityItem  
            activity={activity} />  
        ))}</div>  
      )  
    }  
  }  
}
```

- Not only is this much simpler and easier to understand, but it makes testing both components easier.
- With our freshly-minted Content component, let's create the ActivityItem component.
- Since we already have the view created for the ActivityItem, all we need to do is copy it from what was our Content component's template as it's own module.

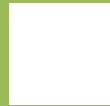
```
class ActivityItem extends React.Component {  
  render() {  
    const {activity} = this.props; // ES6 destructuring  
    return (  
      <div className="item">  
        <div className="avatar">
```

```
<img
    alt={activity.text}
    src={activity.user.avatar} />
    {activity.user.name}
</div>
<span className="time">
    {activity.timestamp}
</span>
<p>{activity.text}</p>
<div className="commentCount">
    {activity.comments.length}
</div>
</div>
))
}
```

Summary:



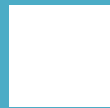
Selectors



Each function



Attributes



document-ready event



Events

Thank You.....

If you have any queries please write to info@uplatz.com".