# Yamileth Rivero

https://view.genial.ly/6322a217990ade0011e0bbf7/presentation-presentacion-asignatura

# Cristian Fernandez

L03: Readings  
The Window Object  
  The Browser Object Model  
    Collection of properties and methods that contain information about the browser and computer screen. "window" object      
    window.alert(), window.prompt()  
    Keep in mind that most browsers allow users to disable any dialogs from repeatedly appearing, meaning they are not a feature to be relied upon.  
  Browser information  
    Provide information about the user’s browser, and operating system being used.  
    It can be modified by a user to masquerade as a different browser.  
    window.location:  Is an object that contains information about the URL of the current page.   
    window.location.href, window.location.host, window.location.hostname, /port/  
    window.location.pathname: returns a string of the path that follows the domain.  
    window.location.search: returns a string that starts with a “?"  
    window.location.assign('https://www.sitepoint.com/'): can be used to load another resource from a URL provided as a parameter.

  Browser history  
    window.history: can be used to access information about any previously visited pages.  
    window.history.length: shows how many pages have been visited before arriving at the current page.  
    window.history.go(): can be used to go to a specific page, where 0 is the current page.  
    .forward(), .back()  
    window.open(): takes the URL of the page to be opened as its first parameter.  
    window.moveTo(): takes two parameters that are the X and Y coordinates of the screen that the window is to be moved to.  
    window.resizeTo(600,400);

Screen Information  
    window.screen: contains information about the screen the browser is displayed on.  
    Height, availHeight, Width  
    It also allows you to do things like turn off the device’s screen, detect a change in its orientation or lock it in a specific orientation.  
      
  Cookies  
      Browser does not remember anything from one request to another.  
    Cookies can be used for personalizing a user’s browsing experience, storing user preferences, keeping track of user choices (such as a shopping cart), authentication and tracking users.  
    Is starting to be replaced in many cases by the new HTML5 localStorage API as it allows more data to be stored.  
    Cookies can be made persistent ― that is, lasting beyond the browser session ― by adding "; expires=date"

  Timing functions  
    window.setTimeout( () => alert("Time's Up!"), 3000); It should show an alert dialog after three seconds  
    IMPORTANT! The window.setInterval() method works in a similar way to window.setTimeout(), except that it will repeatedly invoke the callback function after every given number of milliseconds.  
    const summon = window.setInterval(chant,1000);  
    setInterval( () => {  
        ......;  
        ......  
        }, 1000/60)  
    To stop this, we can use the window.clearInterval() method and the variable repeat as an argument (this is because the window.setInterval() method returns its ID, so this will be assigned to the variable repeat):

The Content Template element

    The <template> HTML element is a mechanism for holding HTML that is not to be rendered immediately when a page is loaded.  
    template element with the shadowroot attribute is detected by the HTML parser and immediately applied as the shadow root of its parent element.  
    Note that directly using the value of the content property could lead to unexpected behavior  
        const container = document.getElementById("container");  
        const template = document.getElementById("template");  
        function clickHandler(event) {  
        event.target.append(" — Clicked this div");    }  
        const firstClone = template.content.cloneNode(true);  
        firstClone.addEventListener("click", clickHandler);  
        container.appendChild(firstClone);  
        const secondClone = template.content.firstElementChild.cloneNode(true);  
        secondClone.addEventListener("click", clickHandler);  
        container.appendChild(secondClone);  
    firstClone is a DocumentFragment instance, so while it gets appended inside the container as expected, clicking on it does not trigger the click event.  
    secondClone is an HTMLDivElement instance, clicking on it works as one would expect.

L04: Readings

Testing and Debugging  
    
  Errors, Exceptions, and Warnings  
    Incorrect syntax, faulty logic or entered data incorrectly.  
    Programmer errors are our responsibility, we should predict any possible interactions that may throw an error.  
    The program should be designed to prevent the user from making errors.  
    An exception is an error that produces a return value  
    Stack Traces is a sequence of functions or method calls that lead to the point where the error occurred. It will work backward from the point at which the error occurred to identify the origin.  
    If a warning occurs, the JavaScript will continue to run (although possibly incorrectly).

  The Importance of Testing and Debugging  
    The error might give unexpected or incorrect results that nobody spots, or lurk in the background.  
    A ninja programmer should try to make the code fail gracefully, ensure exceptions are caught and dealt with, and code is tested rigorously.

  Strict Mode  
    Writing code in strict mode can also help improve its clarity and speed, and will throw exceptions if any sloppy code.  
    IMPORTANT! Strict mode simply requires this in the first line.  
        'use strict';  
        You can even use strict mode on a per-function basis, and it will be applied only to that function.  
    Lint can be used to test the quality of JavaScript code  
    Linting tools are also useful for enforcing a programming style guide.   
    
  Feature Detection  
    Feature detection guarantees that the method is only called if it actually exists and fails gracefully, without any exceptions being thrown.

  Debugging in the Browser  
    Breakpoints halt the progress of the code and allow us to view the value of different variables.  
    console object that provides a number of methods for logging information and debugging.  
    debugger keyword will create a breakpoint in your code that will pause the execution of the code, the browser's debugging tool will automatically kick in and you'll be able to see the value of the age variable by hovering over it.  
    Remember to remove any references to the debugger command.

  Error Objects  
    All error objects have a number of properties, but they’re often used inconsistently across browsers.  
    
  Throwing Exceptions  
    Are thrown automatically by the JavaScript engine when an error occurs.  
    It is best practice, however, to throw an error object. This can then be caught in a catch block.

  Exception Handling  
    When an exception occurs, the program terminates with an error message.  
    Wrap it in a try/catch block.  
    finally, the keyword will always be executed after the try or catch block.  
    
  Tests  
    The next step is to write some code to make the tests pass.  
    Test-driven development is the practice of writing tests that fail, then writing the code that passes the test, then refactoring the code every time a new feature is implemented.

How Single-Page Applications Work  
    Is a website that re-renders its content in response to navigation actions, most rely on the same browser behavior and native APIs to enable the core functionality.  
    Anyone opening that link will see the same thing as you because the location is always updating  
    The single-page application generally relies on a router.

  Location Primer  
    Properties are important: pathname, hash, and search.

  Route Matching  
    Routes describe the location that they should match. (Static or dynamic)

  How Browsers Handle Locations  
    An array of location entries.  
    The browsing context also keeps track of which entry is currently active.

  The History API  
    The History API was developed to add first-class support for single-page applications.  
    API re-uses the active Document by updating it to reflect the new location.  
      history.pushState(null, '', '/next-location');  
      history.replaceState(null, '', '/replace-location');  
      https://developer.mozilla.org/en-US/docs/Web/API/History\_API  
      https://developer.mozilla.org/en-US/docs/Web/API/Location

The problem with single-page apps

  We over-complicate this  
    Serves all of the code for an entire multi-UI app from a single index.html file.  
    More code to maintain, more complexity to manage, and more things to break.  
    List-maker.com/settings/index.html vs list-maker.com/settings.html.

The HTML5 template element.  
    It's essentially mandatory on the client side if you're writing a single-page application.  
    Is almost always done using templating library or application framework.  
    It provides an easy way to define a reusable fragment of HTML.  
    <template> tag gets parsed just like regular HTML. -scripts, -style.  
    It doesn't load any external resources.  
    It can be accessed as a DocumentFragment.  
    We have to provide an API for manipulating their content.  
    Faster than manipulating an element's innerHTML.  
    The most important thing here is the call to the cloneNode method. It creates an entirely new DocumentFragment instance.  
    You can easily do complex flow-control using loops, conditionals, and even try/catch statements.  
    We can nest HTML templates.  
    5-25% smaller.  
    Can I use?   https://caniuse.com/template.

# Alan Valladares

**The Window Object**

**The Browser Object Model**

* The Browser Object Model (or BOM for short) is a collection of properties and methods that contain information about the browser and computer screen
* Every browser window, tab, popup, frame, and iframe has a window object.
* ES6 made parseInt() and isNaN() methods of the Number object

**Dialogs**

* The most use **alert( ), confirm( ) and prompt( ),** all major browsers support them as methods of the **window** object.

**Browser Information**

* The **Navigator** object contains information about the browser being used
* Its **userAgent** property will return information about the browser and operating system
* The **window.location** property is an object that contains information about the URL of the current page
* The **pathname** property returns a string of the path that follows the domain
* The toString() method returns a string containing the whole URL:

**Controlling Windows**

* A new window can be opened using the **window.open( )**method

**Screen Information**

* The **window.screen** object contains information about the screen the browser is displayed on
* The **availHeight** and **availWidth** can be used to find the height and width of the screen, excluding any operating system menus.
* The **colorDepth** property can be used to find the color bit depth of the user’s monitor, although there are few use cases for doing this other than collecting user statistics.

**The Document Object**

* The write() method simply writes a string of text to the page - document.write('Hello, world!');
* document.write('<h1>Hello, world!</h1>');

**Cookies**

* Cookies are small files that are saved locally on a user’s computer. They were invented by Netscape as a way of getting round HTTP being a stateless protocol.
* Cookies can be used to storing information that can then be retrieved between requests.
* To create a cookie, you assign it to JavaScript’s “cookie jar”, using the document.cookie property, like so: document.cookie = 'name=Superman';<< "name=Superman"

**Timing Functions**

**setTimeout( )**

* The **window** object provides some useful methods for scheduling the execution of a function, and for repeatedly executing functions at regular intervals.
* The **window.setTimeout( )** method accepts a callback to a function as its first parameter and a number of milliseconds as its second parameter

**<template>: The Content Template element**

* The <template> [HTML](https://developer.mozilla.org/en-US/docs/Web/HTML) element is a mechanism for holding [HTML](https://developer.mozilla.org/en-US/docs/Glossary/HTML) that is not to be rendered immediately when a page is loaded but may be instantiated subsequently during runtime using JavaScript.

**Testing and Debugging**

**Errors, Exceptions, and Warnings**

* Errors are caused when something goes wrong in a program.
* They are caused by: System error, programmer error, user error.

**Exceptions**

* An exception is an error that produces a return value that can then be used by the program to deal with the error

**Warnings**

* A warning can occur if there’s an error in the code that isn't enough to cause the program to crash.

**Use Strict**

* Strict mode encourages a better quality of JavaScript to be written that befits a programmer, so its use is recommended.
* Tu use to use it in the line of a JavaScript file: **‘use strict’;**

**Debugging Tools**

* Most modern browsers also have a debugging tool that allows you to set breakpoints in your code that will pause it at certain points.

**try, catch,** and**finally**

* The code inside the catch block will only run if an exception is thrown inside the try block. The error object is automatically passed as a parameter to the catch block.
* Code **try { …code } catch { …code } finally { …code }**

**Jest**

* [Jest](https://facebook.github.io/jest/) is a TDD framework, created by Facebook, that has gained a lot of popularity recently. It makes it easy to create and run tests by providing helper methods for common test assertions.
* Install with **npm install -g jest**

**How Single-Page Applications Work**

* **SPA** = Single Page Application
* Website that re-renders its content without making a request to the server to fetch new HTML.
* **SPA** use **window.location** this allows you to interact with the different parts of the URL without having to parse it yourself.

Diagram

Description automatically generated

**Route Matching**

* Single-page application generally rely on a router.
* These can be static (/about) or dynamic (/album/:id, where the value of :id can be any number of possibilities) paths

**Detecting back/forward button navigation**

* When the back and forward buttons are clicked (as well as when history.go() is called), the browser emits a popstate event. In order to detect these, we can add an event listener to the window object.

Graphical user interface, text, application

Description automatically generated

**The HTML5 <template> element.**

* Reusable fragment of HTML that can be manipulated just like you would the contents of the document itself, but without the overhead of actually updating the DOM or having to compile and parse strings of HTML.
* The HTML5 <template> element is actually very well supported.