L03 READING NOTES

Yamileth Rivero García

Web Frontend Development II

1.- Ch9: The Window Object

- Every JavaScript environment has a **global object**.
- Variables that are created in the global scope are properties of this object
- In a browser environment the **global object** is the **window object**, which represents the browser window that contains a web page.

The Browser Object Model (BOM)

- "**BOM**" 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
- JavaScript can be run in different environments, the BOM only makes sense in a browser environment
    - Other environments (such as Node.js) probably won't have a **window** object, although they will still have a global object; for example, Node.js has an **object** called global.

Going Global

- **Global Variables:** Variables that are created without using the **const**, **let** or **var** keywords. Global variables can be accessed in all parts of the program.
- Global variables are actual properties of a global object
- In a browser environment, the global object is the window object.
- Any global variable created is actually a property of the **window** object

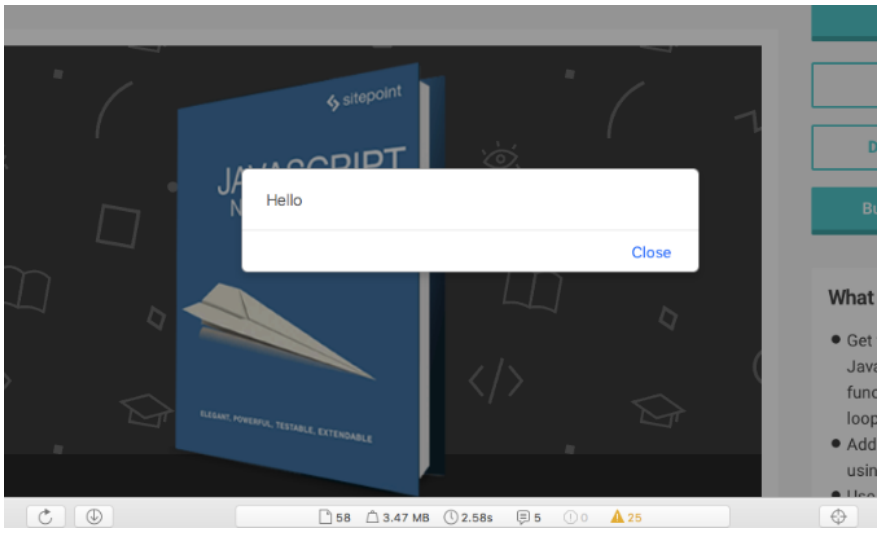
```
x = 6;  // global variable created
<< 6

window.x // same variable can be accessed as a property of the window object
<< 6

// both variables are exactly the same
window.x === x;
<< true
```

- You should refer to global variables **without** using the **window** object

Dialogs

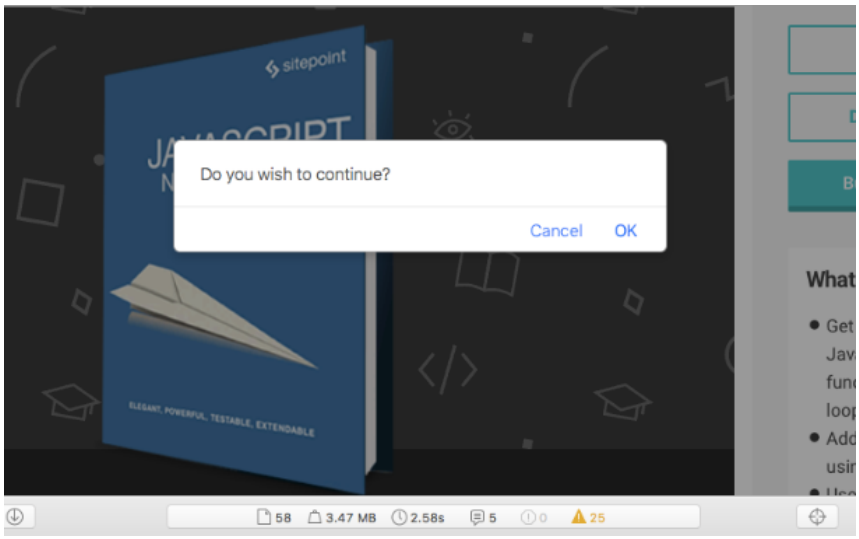
- The window.alert() method will pause the execution of the program and **display a message in a dialog box**. The message is provided as an argument to the method, and **undefined** is always returned:

```
window.alert('Hello');
<< undefined
```



- The window.confirm() method will stop the execution of the program and **display a confirmation dialog** that shows the message provided as an argument, and **giving the options of OK or Cancel**.

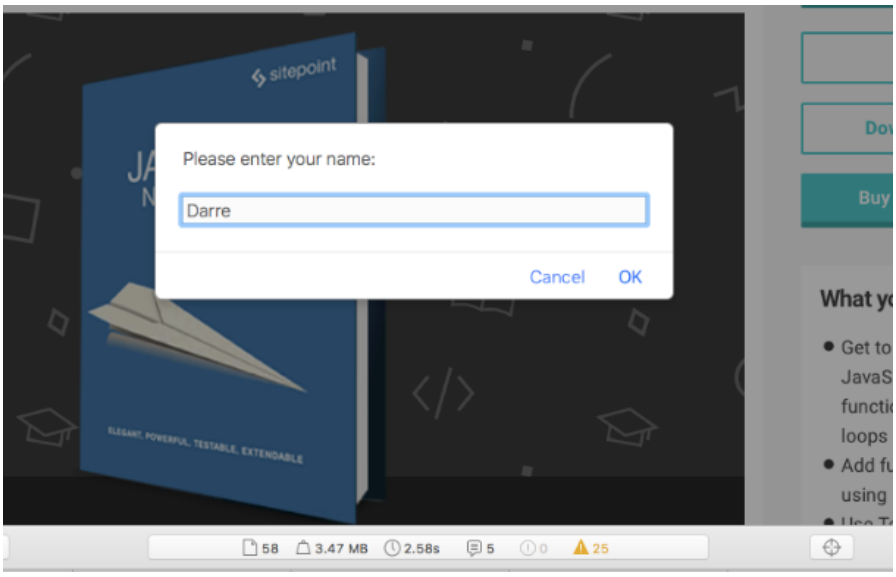
```
window.confirm('Do you wish to continue?');
<< undefined
```



- It returns the boolean values of **true** if the user clicks OK, and **false** if the user clicks Cancel

- The window.prompt() method will stop the execution of the program. It **displays a dialog** that shows a message provided as an argument, **as well as an input field that allows the user to enter text.**

```
window.prompt('Please enter your name:');
```



- This text is then returned as a string when the user clicks OK. If the user clicks Cancel, **null** is returned

I need to be careful when using these methods because everything will stop processing at the point the method is called, until the user clicks OK or Cancel and his can cause problems if the program needs to process something else at the same time.

Browser Information

The **window** object has a number of properties and methods that provide information about the user's browser.

Location

- The window.location property is an object that **contains information about the URL of the current page**.
- It contains a number of properties that provide information about different fragments of the URL.
- The **href** property **returns** the full **URL as** a **string**

```
window.location.href
<< "https://www.sitepoint.com/premium/books/javascript-novice-to-ninja"
```

- The protocol property **returns a string describing the protocol** used (such as **http**, **https**, **pop2**, **ftp** etc.).
- Note that there is a colon **(:)** at the end

```
window.location.protocol
<< "https:"
```

- The host property **returns a string describing the domain** of the current URL **and the port number** (this is often omitted if the default port 80 is used)

```
window.location.host
<< "www.sitepoint.com"
```

- The hostname property **returns a string describing the domain** of the current URL

```
window.location.hostname
<< "www.sitepoint.com"
```

- The port property **returns a string describing the port number**, although it will return an empty string if the port is not explicitly stated in the URL

```
window.location.port
<< ""
```

- The pathname property **returns a string of the path** that follows the domain

```
window.location.pathname
<< "/premium/books/javascript-novice-to-ninja"
```

- The search property **returns a string that starts with a "?"** followed by the query string parameters.
- It returns an empty string if there are no query string parameters. This is what I get when I search for "JavaScript" on SitePoint:

```
window.location.search
<< "?q=javascript&limit=24&offset=0&page=1&
content_types[]=All&slugs[]=all&states[]=available&order="
```

- The hash property **returns a string that starts with a "#"** followed by the fragment identifier.
- It returns an empty string if there is no fragment identifier:

```
window.location.hash
<< ""
```

- The origin property **returns a string that shows the protocol and domain** where the current page originated from
- This property is read-only, so cannot be changed

```
window.location.origin
<< "https://www.sitepoint.com"
```

- The reload() method can be used to force a reload of the current page.
- If it's given a parameter of **true**, it will force the browser to reload the page from the server, instead of using a cached page.

- The assign() method can be used to load another resource from a URL provided as a parameter, for example:

```
window.location.assign('https://www.sitepoint.com/')
```

- The replace() method is almost the same as the assign() method, except the current page will not be stored in the session history, so the user will be unable to navigate back to it using the back button.

- The toString() method returns a string containing the whole URL

```
window.location.toString();
<< "https://www.sitepoint.com/javascript/"
```

The Browser History

- The window.history property can be used to access information about any previously visited pages in the current browser session.

- The window.history.length property shows how many pages have been visited before arriving at the current page.

- The window.history.go() method can be used to go to a specific page, where 0 is the current page

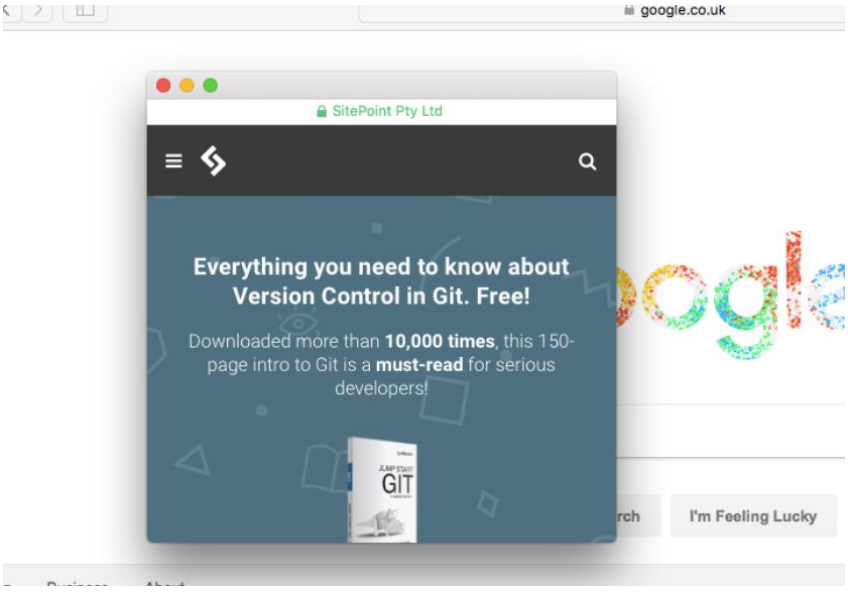
```
window.history.go(1); // goes forward 1 page
window.history.go(0); // reloads the current page
window.history.go(-1); // goes back 1 page
```

- There are also the window.history.forward() and window.history.back() methods that can be used to navigate forwards and backwards by one page respectively, just like using the browser's forward and back buttons.

Controlling Windows

- A new window can be opened using the window.open() method
- This takes the URL of the page to be opened as its first parameter, the window title as its second parameter, and a list of attributes as the third parameter.

```
const popup = window.open('https://sitepoint.com','
SitePoint','width=400,height=400,resizable=yes');
```



- The close() method can be used to close a window, assuming you have a reference to it

```
popup.close();
```

- It is also possible to move a window using the window.moveTo() method.
- This takes two parameters that are the X and Y coordinates of the screen that the window is to be moved to

```
window.moveTo(0,0); // will move the window to the top-left corner of the screen
```

- You can resize a window using the window.resizeTo() method.
- This takes two parameters that specify the width and height of the resized window's dimensions

```
window.resizeTo(600,400);
```

Screen Information

- The window.screen object contains information about the screen the browser is displayed on.
- You can find out the **height** and **width** of the screen in pixels using the height and width properties respectively

```
window.screen.height
<< 1024

window.screen.width
<< 1280
```

- The availHeight and availWidth can be used to find the height and width of the screen, excluding any operating system menus

```
window.screen.availWidth
<< 1280

window.screen.availHeight
<< 995
```

- The colorDepth property can be used to find the color bit depth of the user's monitor

```
window.screen.colorDepth;
<< 24
```

The Document Object

- The write() method simply writes a string of text to the page

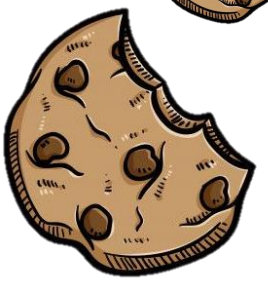
```
document.write('Hello, world!');
```

COOKIES

- Cookies are small files that are saved locally on a user's computer.
- Cookies store information that can then be retrieved between requests.
- Cookies take the form of a text file that contain a list of name/value pairs separated by semicolons. For example, a cookie file might contain the following information:

```
"name=Superman; hero=true; city=Metropolis"
```

Creating Cookies



- To create a cookie, you assign it to JavaScript's "cookie jar", using the document.cookie property

```
document.cookie = 'name=Superman';
<< "name=Superman"
```

- We can add more cookies by assigning them to document.cookie:

```
document.cookie = 'hero=true';
<< "hero=true"

document.cookie = 'city=Metropolis';
<< "city=Metropolis"
```

Changing Cookie Values

- A cookie's value can be changed by reassigning it to **document.cookie** using the same name but a different value.

```
document.cookie = 'name=Batman'
<< "name=Batman"
document.cookie = 'city=Gotham'
<< "city=Gotham"
```

Reading Cookies

- To see the current contents of the cookie jar, simply enter **document.cookie**

```
document.cookie:
<< "name=Batman; hero=true; city=Gotham"
```

Cookie Expiry Dates

- Cookies are session cookies by default. This means **they are deleted once a browser session is finished** (when the user closes the browser tab or window)
- Cookies can be made persistent — that is, lasting beyond the browser session — by adding **"; expires=date"** to the end of the cookie when it's set, where date is a date value in the UTC String format Day, **DD-Mon-YYYY HH:MM:SS GMT**

```
const expiryDate = new Date();
const tomorrow = expiryDate.getTime() + 1000 * 60 * 60 * 24;
expiryDate.setTime(tomorrow);

document.cookie = `name=Batman; expires=${ expiryDate.toUTCString()}`;
```

- An alternative is to set the **max-age** value

```
document.cookie = 'name=Batman; max-age=86400' // 86400 secs = 1 day
```

2.- MDN 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.

Think of a template as a content fragment that is being stored for subsequent use in the document

Attributes

- The only standard attributes that the template element supports are the **global attributes**.

Avoiding DocumentFragment pitfall

- A **DocumentFragment** is not a valid target for various events, as such it is often preferable to clone or refer to the elements within it.