2. Introducció. Objectes predefinits i objectes del client web

Entre els objectes predefinits per JavaScript hem de distingir entre aquells que són propis de JavaScript com a llenguatge de programació, independentment del context; i aquells que es creen quan el client web (Firefox, Chrome, IE) carrega un document HTML, és a dir, en un context de navegador web.

Els primers representen, bàsicament, cadenes de caràcters, nombres i elements matemàtics i dates, com passa a molts llenguatges de programació. A més, també n'hi ha que representen les expressions regulars, que permeten realitzar amb relativa facilitat operacions de tractament de text amb un grau de complexitat important.

Els segons estan compresos en el BOM (*Browser Object Model*, que pot traduir-se per Model d'Objectes del Navegador). En aquest model s'inclouen objectes que representen els diferents elements del navegador com la finestra, l'historial, l'adreça web i, sobre tot, els documents HTML que s'estan mostrant. Aquests documents presenten una complexitat important i requereixen un sistema d'objectes propi, el DOM (*Document Object Model*, que pot traduir-se per Model d'Objectes del Document).

Tot això suposa llargues llistes de propietats i mètodes. Caldrà estar-hi familiaritzats amb els d'ús més habitual i, també, amb la documentació que tots aquests objectes tenen associada. És bàsic ser capaç d'interpretar-la correctament per poder utilitzar fins i tot elements que no hàgiu utilitzat mai abans.

2.1. Objectes predifinits de JavaScript

Alguns dels objectes predefinits de JavaScript més utilitzats són:

- Objecte `String`: per representar i operar amb cadenes de text.
- Objecte `Number` i `BigInt`: per representar números.
- Objecte `Math`: per representar constants i funcions matemàtiques.
- Objecte `Date`: per representar dates.
- Objecte `RegExp`: per representar expressions regulars.
- Objectes `Array`, `Map` i `Set`: per treballar amb col·leccions d'elements.

Com a objectes que són, tots tenen unes propietats i uns mètodes. És difícil practicar amb totes les propietats i mètodes dels objectes. El que és important és saber cercar dins de la documentació i saber aplicar les propietats i mètodes d'aquests objectes.

Un enllaç directe on podeu trobar la referència de tots els objectes predefinits, les seves propietats i els seus mètodes és: mzl.la/3dkksJe.

2.1.1. L'objecte String

En l'apartat d'"Activitats" del web del mòdul trobareu un exemple de com afegir un nou mètode a un objecte `String`.

```
let strPrimitiva = 'IOC';
let strObjecte = new String(strPrimitiva);

console.log(typeof strPrimitiva); // string
console.log(typeof strObjecte);  // object

console.log(strPrimitiva == strObjecte); //true
console.log(strPrimitiva === strObjecte); //false
```

Tot i que normalment no ens n'haurem de preocupar, podem obtenir resultats sorprenents. Per exemple, si utilitzem la funció de JavaScript `eval()`, que avalua o executa una expressió:

```
let str1 = '2 + 2';    //tipus primitiu de cadena
let str2 = new String('2 + 2'); // objecte String
console.log(eval(str1));        // retorna 4
console.log(eval(str2));        // retorna la cadena "2 + 2"
```

Amb el mètode `valueOf()` podem convertir un objecte al seu tipus primitiu:

```
console.log(eval(str2.valueOf())); //retorna 4
```

Una altra propietat interessant dels objectes `String` és `length` que ens retorna la mida d'una cadena:

```
let str1="IOC";
console.log(str1.length); // retorna 3
```

Mètodes de l'objecte String

Els principals mètodes que farem servir són els següents.

- `charAt(x)`: retorna el caràcter a la posició `x` (començant pel 0).
- `concat(str)`: concatena la cadena `str` a la cadena original.
- `startsWith(str)`: comprova si la cadena comença amb els caràcters o cadena especificats.
- `endsWith(str)`: comprova si la cadena acaba amb els caràcters o cadena especificats.
- `includes(str)`: comprova si la cadena conté els caràcters o cadena especificats.
- `indexOf(str)`: retorna la posició de la primera ocurrència de la cadena que passem com a paràmetre.
- `lastIndexOf(str)`: retorna la posició de l'última ocurrència de la cadena que passem com a paràmetre.
- `match(regexp)`: cerca dins una cadena comparant-la amb l'expressió regular `regexp`, i retorna les coincidències en un *array*.
- `repeat(x)`: retorna una nova cadena que és la repetició de la cadena tantes vegades com el valor del paràmetre `x`.

- `search(str)`: cerca dins de la cadena els caràcters (o expressió regular) `str`, i retorna la posició de la primera ocurrència.
- `slice(x,y)`: extreu una part d'una cadena entre els caràcters `x` i `y`, i retorna una nova cadena.
- `split(car)`: separa una cadena en un *array* de subcadenes, agafant com a llavor del separador el caràcter `car`.
- `substr(x, y)`: extreu caràcters d'una cadena, començant en la posició `x`, i el número de caràcters especificat per `y`.
- `substring(x, y)`: extreu caràcters d'una cadena entre els dos index especificats, `x` i `y`.
- `toLowerCase(str)`: converteix una cadena a minúscules.
- `toUpperCase(str)`: converteix una cadena a majúscules.
- `trim()`: elimina els espais en blanc d'ambdós extrems de la cadena.
- `toString()`: retorna el valor d'un objecte `String`.
- `valueOf()`: retorna el tipus primitiu d'un objecte `String`.

Hi ha altres mètodes que només tenen sentit en el context de la web. Són els *wrapper HTML methods*. Per exemple, el mètode `bold()` retorna la mateixa cadena embolcallat amb els tags `<b>` i `</b>`, de manera que dins d'una pàgina web la cadena es veu com a negreta. Aquests mètodes no són un estàndard, poden tenir un comportament diferent en diversos navegadors i, a més, la majoria d'ells són obsolets. Els principals mètodes d'embolcall HTML són: `big()`, `bold()`, `fontcolor()`, `fontsize()`, `italics()`, `small()`, que embolcallen la cadena amb *tags* perquè es mostri, respectivament: amb font gran, en negreta, d'un color determinat, amb una mida determinada, en cursiva i amb font petita.

Exemple: funció per validar el DNI

En el DNI tenim 8 números i una lletra. Per a cada número només una lletra és vàlida, i la manera de calcular-la la tenim en la funció següent.

S'utilitzen diversos mètodes de `String`: `substr()`, `charAt()` i `toUpperCase()`.

```
function validarDNI(dni) {
  let lletres = "TRWAGMYFPDXBNJZSQVHLCKE";
  let numDni = dni.substr(0, dni.length - 1);

  //la línia següent pot substituir-se per: var lletra_dni=dni.substr(dni.length-1,1)
  let lletraDni = dni.charAt(dni.length - 1).toUpperCase();

  if (lletres.charAt(numDni % 23) === lletraDni) {
    return true;
  }

  return false;
}

console.log(validarDNI("38540343T")); //false
console.log(validarDNI("38540343w")); //true
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/jOWdjmy?editors=0012.

JavaScript compleix amb dos tipus primitius per treballar amb nombres:

- `Number`: per a nombres enters i reals.
- `BigInit`: per treballar amb nombres enters grans.

En aquesta secció ens centrem en el tipus `Number`, que guarda els números en memòria amb una resolució de 64 bits (doble precisió) i coma flotant.

```
let a = 20;
let b = -2.718;
let c = 1e4;
let d = 2.23e-3
```

A part de la notació decimal, podem representar els números en binari, octal o hexadecimal.

```
console.log(0b1010);
console.log(0xFF);
```

I amb el mètode `toString()` podem representar els números en diferents notacions:

```
let num = 76;
console.log(`${num.toString(16)} en hexadecimal`);
console.log(`${num.toString(8)} en octal`);
console.log(`${num.toString(2)} en binari`);
```

`Infinity` és el valor per representar que hem sobrepassat la precisió dels números (que és fins a 15 dígits en la part entera):

```
console.log (5/0); //retorna Infinity
```

Not a Number (`NaN`) és una paraula reservada per indicar que el valor no representa un número. Podem utilitzar la funció global `isNaN()` per saber si l'argument és un número:

```
console.log(3 * "a"); //NaN. Recordeu que 3+"a" retorna "3a"
console.log(isNaN(3 * 4)); //false
console.log(isNaN(3 * "a")); //true
```

De la mateixa manera que passa amb les cadenes, a JavaScript normalment els números són tipus primitius, però també els podem crear com a objectes:

```
let x = 25; //tipus primitiu
let y = new Number(25); //objecte
console.log(typeof x); //number
console.log(typeof y); //object

console.log (x == y); //true
console.log (x === y); //false
```

Podeu veure els exemples anteriors a: codepen.io/ioc-daw-m06/pen/ZEQwdrv?editors=0012.

Mètodes de l'objecte Number

Els principals mètodes de l'objecte `Number` són:

- `isFinite()`: comprova si un valor és un número finit.

`Number.NaN`.

- `toExponential(x)`: representa un número amb la seva notació exponencial. Per exemple, 20,31 a 2.031e+1, que significa 2.031 * 10^1.
- `toFixed(n)`: formata un número amb una precisió decimal de n dígits. Retorna una cadena que és la representació del número.
- `toPrecision(n)`: formata un número a una precisió de n dígits (número total de dígits, incloent la part entera i la decimal).
- `toString()`: converteix un número a una cadena.
- `parseInt()`: converteix un número a enter.
- `parseFloat()`: converteix un número a decimal.
- `valueOf()`: retorna el valor del tipus primitiu del número.

Posem alguns exemples:

```
console.log((18).toString() + (20).toString());

let x = 234.456;
console.log(typeof x); //number
console.log (Number.isInteger(x)); //false
console.log (x.toExponential());
x = x.toFixed(2); // 234.46, arrodoneix de la forma esperada
console.log(x);
console.log(typeof x); //compte! x ara és un string
x = parseFloat(x);
console.log(typeof x); //number. Torna a ser number

x = x.toPrecision(6); //234.460
console.log(x);
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/bGEzPKB?editors=0012.

Paràmetres i valors retornats per funcions

Aquest exemple que acabeu de veure no era trivial. Fixeu-vos que després d'utilitzar toFixed() el resultat és una cadena, i per tant no podem aplicar toPrecision(), que només s'aplica a *Numbers*. Per convertir a Number tampoc puc utilitzar Number.parseFloat(), que només s'aplica a Number, sinó la funció global `parseFloat(x)`. Un cop tenim Number, ja podem aplicar el mètode toPrecision(). Aquesta manera de raonar és habitual en JavaScript i en d'altres llenguatges de programació. Així doncs, quan us trobeu amb un resultat inesperat no heu de treure conclusions precipitades, i heu d'aprendre a trobar i raonar sobre la causa del problema.

2.1.3. L'objecte Math

L'objecte `Math` permet realitzar tasques matemàtiques, com per exemple crear números aleatoris, realitzar funcions trigonomètriques, etc.

Propietats de Math

`Math.E` (número d'Euler, 2.718…), `Math.LN2` (logaritme neperià de 2), `Math.LN10` (logaritme natural de 10), `Math.PI` (número Pi), `Math.SQRT2` (arrel quadrada de 2), i algunes altres més.

L'objecte `Math` és especial en el sentit que no té constructor. No hi ha cap mètode per crear un objecte `Math`. Podem fer-lo servir sense haver de crear-lo.

Per exemple,

```
console.log (Math.PI);
```

Mètodes de l'objecte Math

Amb els següents mètodes de `Math` podem realitzar operacions matemàtiques usuals:

- `abs(x)`: valor absolut.
- `sin(x)`, `cos(x)`, `tan(x)`: funcions trigonomètriques de sinus, cosinus i tangent.
- `asin(x)`, `acos(x)`, `atan(x)`: retorna en radians l'arcsinus, arccosinus, arctangent.
- `round(x)`: arrodoneix x al valor enter més proper.
- `ceil(x)`, `floor(x)`: retorna el mateix número però arrodonit a l'enter més proper cap a dalt (`ceil`) o cap a baix (`floor`).
- `truncx(x)`: elimina la part fraccionària d'un número (i queda només la part entera).
- `exp(x)`: retorna el valor e^x.
- `max(a,b,…)`, `min(a,b,…)`: retorna el valor més gran (o més petit) de la llista de números.
- `pow(x,y)`: retorna x^y.
- `log(x)`: retorna el logaritme natural (base e) de x.
- `random()`: retorna un número aleatori entre 0 (inclòs) i 1 (exclòs).
- `sqrt(x)`: retorna l'arrel quadrada de x.

Quan es programa, sovint és necessari obtenir números aleatoris. Per generar-los, JavaScript utilitza una llavor (*seed*) interna que no és accessible a l'usuari. En el següent exemple es veu un exemple típic de com es poden generar números aleatoris en diferents rangs:

```
// Funció que retorna un número aleatori entre 0 (inclusiu) i 1 (exclusiu)
function getRandom() {
  return Math.random();
}

// Funció que retorna un número aleatori entre min (inclusiu) i max (exclusiu)
function getRandomArbitrary(min, max) {
  return Math.random() * (max - min) + min;
}

// Funció que retorna un número enter aleatori entre min (inclusiu) i max (exclusiu)
function getRandomInt(min, max) {
  return Math.floor(Math.random() * (max - min)) + min;
}

// Funció que retorna un número enter aleatori entre min (inclusiu) i max (inclusiu)
function getRandomIntInclusive(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
```

```
for (let i=0; i<10; i++) {
  console.log(getRandom());
}

console.log('\nNúmero aleatori entre 1 (inclusiu) i 5 (exclusiu)');
for (let i=0; i<10; i++) {
    console.log(getRandomArbitrary(1,5));
}

console.log('\nNúmero enter aleatori entre 1 (inclusiu) i 5 (exclusiu)');
for (let i=0; i<10; i++) {
    console.log(getRandomInt(1,5));
}

console.log('\nNúmero enter aleatori entre 1 (inclusiu) i 5 (inclusiu)');
for (let i=0; i<10; i++) {
    console.log(getRandomIntInclusive(1,5));
}
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/jrBeNW?editors=0012.

2.1.4. L'objecte Date

L'objecte `Date` s'utilitza per treballar amb dates i temps. Per instanciar un objecte tipus `Date` s'utilitza `new`, i admet diferents arguments:

```
let data = new Date(); //data del sistema
console.log(data);
data = new Date(34885453664); //genera un data que representa els mil·lisegons que han pas
console.log(data);
data = new Date('2016/05/23');
console.log(data);
data = new Date(2016,5,23,12,15,24,220);  //any,mes,dia,hora,minuts,segons,mil·lisegons
console.log(data);
```

Treballar amb dates no sempre és fàcil i immediat: haurem de tenir en compte el format de data i el fus horari.

Sigui quin sigui el format en què expressem les dates, aquestes es guarden internament com un número que representa els mil·lisegons que han passat des de l'1 de gener de 1970 (00:00:00). Per exemple, des de la data actual (amb precisió de mil·lisegons) fins aquesta data original han passat:

```
let data = new Date(); //data del sistema
console.log(data.getTime()); //milisegons
console.log(`Des de 1970 han passat ${data.getTime()/1000/3600/24/365} anys aprox`);
```

Podeu veure els exemples a: codepen.io/ioc-daw-m06/pen/jOWdjXG?editors=0012.

Mètodes de l'objecte Date

Hi ha molts mètodes de l'objecte `Date`. En veurem alguns, en especial aquells que permeten representar les dates en el format usual (dd/mm/aaaa):

- `getDay()`: retorna el dia de la setmana (0-6, començant per diumenge).
- `getFullYear()`: retorna l'any (4 dígits).
- `getMonth()`: retorna el mes (0-11).
- `getDate()`: retorna el dia del mes (1-31).
- `getHours()`: retorna l'hora (0-23).

- `getMilliseconds()`: retorna els mil·lisegons (0-999).
- `getUTCDate()`: retorna el dia del mes (1-31), d'acord amb l'horari UTC universal.
- `getUTCDay()`: retorna el dia de la setmana (0-6, començant per diumenge), d'acord amb l'horari UTC universal.
- `getUTCFullYear()`: retorna l'any (4 dígits), d'acord amb l'horari UTC universal.
- `getUTCMonth()`: retorna el mes (0-11), d'acord amb l'horari UTC universal.
- `getUTCHours()`: retorna l'hora (0-23), d'acord amb l'horari UTC universal.
- `getUTCMinutes()`: retorna els minuts (0-59), d'acord amb l'horari UTC universal.
- `getUTCSeconds()`: retorna els segons (0-59), d'acord amb l'horari UTC universal.
- `getUTCMilliseconds()`: retorna els mil·lisegons (0-999), d'acord amb l'horari UTC universal.
- `getTime()`: retorna el nombre de mil·lisegons que han transcorregut des de la data fins l'1 de gener de 1970.
- `now()`: retorna el nombre de mil·lisegons que han transcorregut des de la data del sistema fins l'1 de gener de 1970.
- `parse()`: transforma una cadena de text (representant una data), i retorna el nombre de mil·lisegons transcorreguts des de la data fins l'1 de gener de 1970.
- `setFullYear()`: especifica el dia de l'any.
- `setMonth()`: especifica el mes.
- `setDate()`: especifica el dia del mes.
- `setHours()`: especifica l'hora.
- `setMinutes()`: especifica els minuts.
- `setSeconds()`: especifica els segons.
- `setMilliseconds()`: especifica els mil·lisegons.
- `setTime()`: especifica una data a partir del nombre de mil·lisegons abans o després de l'1 de gener de 1970.
- `setUTCFullYear()`: especifica l'any, d'acord amb l'horari UTC universal.
- `setUTCMonth()`: especifica el mes, d'acord amb l'horari UTC universal.
- `setUTCDate()`: especifica el dia del mes, d'acord amb l'horari UTC universal.
- `setUTCHours()`: especifica l'hora, d'acord amb l'horari UTC universal.
- `setUTCMinutes()`: especifica els minuts, d'acord amb l'horari UTC universal.
- `setUTCSeconds()`: especifica els segons, d'acord amb l'horari UTC universal.
- `setUTCMilliseconds()`: especifica els mil·lisegons, d'acord amb l'horari UTC universal.
- `toDateString()`: converteix la part de la data en una cadena llegible.
- `toLocaleDateString()`: converteix la part de la data en una cadena llegible, utilitzant les convencions locals.
- `toISOString()`: converteix la data a cadena, utilitzant la convenció ISO.
- `toJSON()`: converteix la data a cadena amb format JSON.
- `toTimeString()`: converteix la part del *timestamp* de l'objecte `Date` a cadena.

- `toString()`: converteix l'objecte `Date` a cadena.
- `toLocaleString()`: converteix l'objecte `Date` a cadena, utilitzant les convencions locals.

Diferents exemples per mostrar com s'utilitzen aquests mètodes són:

```
let data = new Date();
let arrayMesos = ['gener', 'febrer', 'març', 'abril', 'maig', 'juny', 'juliol', 'agost', '
var arrayDiesSetmana = ['diumenge', 'dilluns', 'dimarts', 'dimecres', 'dijous', 'divendres
console.log(`Avui és ${arrayDiesSetmana[data.getDay()]}, ${data.getDate()} de ${arrayMesos
```

```
//Per veure el nombre de mil·lisegons des de l'origen dels temps (1 de gener de 1970 00:00
let data = new Date();
console.log(data.getTime());
// i també:
console.log(Date.now());
```

```
let d1 = Date.parse("23 March 2016"); //parse sap interpretar diferents formats de data, p
let d2 = Date.parse("28 May 2016");
console.log(d1);
console.log(d2);
console.log(d1 > d2); //Podem comparar les dates per saber quina és la més gran
```

Per veure la diferència entre utilitzar UTC o no, hem de pensar que aquí a Catalunya estem en el fus horari UTC/GMT +1 hora, però hem de tenir en compte si estem en horari d'estiu o d'hivern. Quan estem a l'estiu s'ha de compensar el fus horari i aleshores és UTC/GMT +2 hora. Per tant, podem veure la diferència entre l'hora local i l'hora absoluta internacional:

```
let data = new Date();
console.log(data.getHours()); //per ex, 14, hora real d'un rellotge local.
console.log(data.getUTCHours()); //per ex, 12, hora UTC
```

Vegeu la diferència entre utilitzar les convencions locals o no:

```
let d = new Date();
//Dia del Treball
d.setDate(1);
d.setMonth(4); //Representa el mes de maig
d.setFullYear(2016);
console.log(d);
console.log(d.toDateString()); //Sun May 01 2016
console.log(d.toLocaleDateString()); //1/5/2016

//Com que toLocaleDateString() retorna una cadena, ja puc utilitzar les funcions de cadena
let arrayData = d.toLocaleDateString().split('/');
console.log(arrayData[0]); //dia
console.log(arrayData[1]); //mes
console.log(arrayData[2]); //any
```

Utilitzant les convencions locals:

```
let d = new Date();
console.log(d.toLocaleString()); // 2/5/2016, 14:44:37
console.log(d.toLocaleDateString()); // 2/5/2016
console.log(d.toLocaleTimeString()); // 14:44:37
```

2.1.5. L'objecte RegExp

Les expressions regulars són objectes de JavaScript que s'utilitzen per descriure un patró de caràcters. Aplicades a les cadenes de text, amb les expressions regulars podem esbrinar si una cadena de text compleix un patró, i realitzar funcions de cerca i reemplaçament.

Les expressions regulars estan explicades amb deteniment en la unitat "Esdeveniments. Manejament de formularis".

En una expressió regular distingim entre el **patró** i els **modificadors**: */pattern/modificadors;*.

```
var patro = /IOC/i
```

En aquest cas, la *i* és un modificador que significa que farem una cerca sense tenir en compte majúscules/minúscules. Hi ha altres modificadors que pots consultar, entre d'altres llocs, al tutorial de la Fundació Mozilla mzl.la/3atdqA2.

Per veure el funcionament bàsic de les expressions regulars veurem els mètodes `test()` i `match()`:

- `test()`: mètode que retorna `true` o `false` si la cadena de text compleix el patró.
- `match()`: mètode de l'objecte `String` que fa una cerca de la cadena contra un patró, i retorna un *array* amb els resultats trobats, o el valor primitiu `null` en cas que no en trobi cap.

```
let patro1 = /ioc/i
let patro2 = /ioc/
let cad="Curs de JavaScript de l'IOC";

console.log(patro1.test(cad));
console.log(patro2.test(cad));

let res = cad.match(patro1);
console.log(res.length); //1, una sola ocurrència
console.log(res[0]); //IOC
```

Dins una expressió regular podem utilitzar *brackets* per expressar un rang de caràcters, i metacaràcters, que tenen un significat especial. Per exemple, anem a veure una expressió regular per comprovar si una cadena compleix amb el format de data dd/mm/aaaa:

```
regexp = /^(3[01]|[12][0-9]|0?[1-9])\/(1[0-2]|0?[1-9])\/([0-9]{2})?[0-9]{2}$/;
console.log(regexp.test("01/01/2016")); //true
console.log(regexp.test("2016/01/01")); //false
console.log(regexp.test("01/01/16")); //true
console.log(regexp.test("1/1/16")); //true
console.log(regexp.test("IOC")); //false
```

- Amb el circumflex (^) estem indicant com ha de començar l'expressió.
- Amb el dòlar ($) estem indicant com ha d'acabar l'expressió.
- Utilitzem la contrabarra (\) per indicar que la barra (/) no és un metacaràcter, sinó un caràcter a tenir en compte.
- El dia del mes pot ser: 30 o 31; del 10 al 29; o també del 1 al 9, ficant o no un 0 al davant.
- El mes pot ser 1-12, ficant o no un zero al davant.

Un altre exemple, en aquest cas per validar un numero de targeta de credit:

```
regexp = /^[0-9]{4}(-|\s)[0-9]{4}(-|\s)[0-9]{4}(-|\s)[0-9]{4}$/;
console.log(regexp.test("3782-8224-6310-0067")); //true
console.log(regexp.test("3782 8224 6310 0067")); //true
console.log(regexp.test("3782*8224*6310*0067")); //false
console.log(regexp.test("37828224630006")); //false
```

Podeu veure els exemples a: codepen.io/ioc-daw-m06/pen/XWXOLvy?editors=0012.

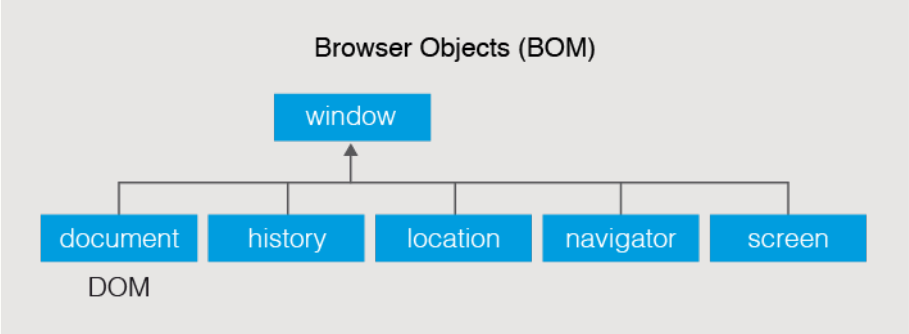
2.2. BOM (Browser Object Model)

L'objectiu del BOM no és només canviar el contingut del document HTML, sinó també realitzar altres manipulacions relacionades amb el navegador i les seves finestres. Per exemple, és possible redimensionar i moure una finestra del navegador, modificar la informació que es mostra a la barra d'estat, moure's per l'historial d'URLs en les quals navegador ha accedit, gestionar les *cookies* (galetes), etc.

El DOM (Document Object Model) està explicat amb més detall a la unitat "Model d'objectes del document".

Quan un client web (el navegador web: Firefox, Chrome, IE) carrega un document HTML, crea uns objectes associats al document, al seu contingut, i altra informació útil. Aquests objectes guarden una jerarquia en què l'objecte `window` és l'arrel, i d'aquí pengen els altres objectes que podem referenciar (vegeu la figura 2.1).

Figura 2.1. Jerarquia d'objectes del BOM



Com es veu a la figura 2.1, cada document HTML crea els següents objectes:

- `window`: representa l'arrel de l'arbre del BOM. Una finestra pot tenir subfinestres (per exemple, *pop-ups*), que seran fills en aquest arbre que representa l'estructura.
- `location`: conté les propietats de la direcció URL del document.
- `history`:conté la informació de les direccions URL que s'han visitat en la sessió actual.
- `document`: conté informació sobre el document actual, com ara el títol, imatges, *links*, taules, formularis que conté, etc. Per accedir a tota aquesta informació tenim el DOM. Les propietats de l'objecte `document` poden ser alhora altres objectes (com ara l'objecte `cookies`).

A més de propietats i mètodes, també hi tenim *events* associats. És a dir, aquests objectes poden respondre a *events* disparats per les accions de l'usuari.

En l'arrel de la jerarquia d'objectes hi ha l'objecte `window`, que representa una finestra oberta en el navegador. En la figura figura 2.1 es pot veure la jerarquia d'objectes que pengen de `window`.

En aquesta jerarquia, els descendents d'un objecte es representen mitjançant propietats de l'objecte. Per exemple, una imatge `img1` és un objecte però també és una propietat de l'objecte `document`, i serà referenciat com a `document.img1`. Per referenciar una propietat s'han de precisar els seus ascendents. De fet, seria `window.document.img1`, però `window`, que és l'arrel, es pot ometre. Alhora aquesta imatge, com a objecte que és, té les seves propietats (típicament `width` i `heigh`): `document.img1.width`.

Com que una imatge forma part del contingut del document HTML més que no pas de les propietats del navegador, s'estudia amb més profunditat en la unitat "Model d'objectes del document".

Un altre exemple: utilitzant la jerarquia podem saber la URL actual de la pàgina que estem visitant: `window.location.href` (o `location.href`).

El codi HTML per poder provar els anteriors exemples és:

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>Objecte window</title>
    </head>
    <body>
        <img src="./muntanya.jpg" name="img1" />
        <script>
            console.log(window.document.img1.width);
            console.log(document.img1.height);
            console.log(window.location.href);
            console.log(location.protocol);
        </script>
    </body>
</html>
```

Evidentment, l'usuari haurà de tenim una imatge *muntanya.jpg* en la mateixa carpeta que el document HTML.

Propietats de window

Les propietats de `window` són:

- `defaultStatus`: és el missatge que es visualitza en la barra d'estat del navegador.
- `frames`: és la matriu de la col·lecció de *frames* que conté una finestra. Per exemple, si volem fer referència al primer *frame* de la finestra: `window.frames[0]`. `frames.length` és el número de *frames* que conté una finestra.
- `parent`: ascendent d'una finestra.
- `self`: fa referència a la finestra actual.
- `status`: missatge que mostra l'estat del client web.
- `top`: fa referència a l'arrel de la jerarquia d'objectes.
- `window`: fa referència a la finestra actual.

- `screenX`, `screenY`: coordenades de la finestra en relació a la pantalla. Vàlid per als navegadors Google Chrome i IE (per a Firefox utilitzar `screenLeft` i `screenTop`).
- `opener`: retorna una referència a la finestra que ha creat la finestra.
- `localStorage`: retorna una referència a l'objecte `localStorage` que s'utilitza per emmagatzemar dades. Al contrari que les galetes, no té data d'expiració.
- `document`, `history`, `location`, `navigator`, `screen`: són propietats de `window` que fan referència als objectes `document`, `history`, `location`, `navigator`, `screen`. Recordeu que formen part d'una estructura jeràrquica.
- `sessionStorage`: retorna un objecte `storage` per emmagatzemar dades en una sessió web.

Les propietats més clares les podem veure en aquest exemple:

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>Objecte window</title>
    </head>
    <body>
        <script>
            console.log(window.innerWidth + '-' + window.innerHeight); //Pots redimensiona
            console.log(window.screenX + '-' + window.screenY); //Si mous la finestra de p
            window.defaultStatus = "informació a la barra d'estat";

            console.log(window.defaultStatus);
            console.log(window.closed);
        </script>
    </body>
</html>
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/JKWmdv.

Mètodes de window

Els principals mètodes de l'objecte `window` són:

- `alert(missatge)`: crea una finestra de diàleg on es mostra el missatge.
- `confirm(missatge)`: crea una finestra de confirmació (OK/Cancel) on es mostra el missatge. Retorna `true` o `false`.
- `prompt(missatge, text)`: crea una finestra de diàleg on es mostra el missatge i permet l'edició. El paràmetre `text` és el valor predeterminat. Igual que `confirm()`, conté Acceptar i Cancelar. Si acceptem, el mètode retorna el valor inserit (o el valor per defecte). Si cancelem, retorna `null`.
- `close()`: només es poden tancar les finestres que també s'han creat amb un script.
- `stop()`: atura la càrrega de la finestra.
- `setTimeout(expressió, msec)`: executa l'expressió que es passa com a argument, un cop han passat `msec` mil·lisegons. S'utilitza per prorrogar l'execució de l'expressió.
- `clearTimeout(timeoutID)`: restaura el compte enrere iniciat per `setTimeout()`.
- `setInterval(expressió, msec)`: executa l'expressió passada com a argument cada `msec` mil·lisegons. S'utilitza per executar l'expressió a intervals regulars de temps.
- `open(url, windowName, params)`: crea una finestra nova, li associa el nom `windowName`, i accedeix a la URL que li hem passat. Li passem un conjunt de paràmetres que

En el següent exemple utilitzem alguns d'aquests mètodes:

Cal tenir en compte que el nom de la finestra no és el títol i, per consegüent, no es mostra a la nova finestra.

```
<html>

<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>Creació d´una finestra</title>
</head>

<body>
  <script>
    let finestra = window.open("http://ioc.xtec.cat", "", "width=300,height=300, left=300,
    setTimeout(function() {
      finestra.close();
    }, 2000);

    setTimeout(function() {
      let nom = prompt("Posa el nom de la finestra:", "Nom");
      crearFinestra(nom);
    }, 2000);


    function crearFinestra(nom) {
      let opcions = "toolbar=0, location=0, directories=0, status=0,";
      opcions += "menubar=0, scrollbars=0, resizable=0, copyhistory=0,";
      opcions += "width=100,height=100";
      window.open("", nom, opcions);
    }
  </script>
</body>

</html>
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/ezvPgG?editors=1002. Us recomanem, però, que proveu l'exercici amb un fitxer propi, ja que és posible que el *prompt* i les noves finestres siguin bloquejades.

2.2.2. L'objecte document

Quan un document HTML es carrega en un navegador web, obtenim un objecte `document`. L'objecte `document` és el node arrel d'un document HTML, i d'ell pengen tots els altres nodes: nodes d'elements, de text, d'atributs, de comentaris.

Recordeu que el `document` és part de l'objecte `window` i, per tant, s'hi pot accedir amb `window.document`.

Propietats de document

Les propietats de `document` són:

La programació amb *cookies* es detalla a l'apartat "Programació amb galetes, finestres i marcs" d'aquesta unitat.

- `lastModified`: data de l'última modificació.
- `cookie`: cadena de caràcters que conté la *cookie* associada al recurs representat per l'objecte.
- Les propietats `anchors`, `forms`, `images`, `links` retornen un objecte amb el qual podem accedir a cadascun dels elements (àncores, formularis, imatges, enllaços) presents en el document.

El següent exemple utilitza algunes d'aquestes propietats:

Les propietats `anchors`, `forms`, `images` i `links` es veuran amb més detall a la unitat "Model d'objectes del document".

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>Objecte document. Propietats</title>
    </head>
    <body>
        <a href="http://ioc.xtec.cat">IOC</a><br />
        <img src="http://ioc.xtec.cat/educacio/templates/ioc/images/ioc_logo.png" /img>
        <script>
            console.log(`location: ${document.location}`);
            console.log(`títol: ${document.title}`);
            console.log(`lastModified: ${document.lastModified}`);
            console.log(document.links[0].href);
            console.log(document.links.item(0).href);
            console.log(document.images[0].src);
        </script>
    </body>
</html>
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/yLeZmRw?editors=1011.

Mètodes de document

Els mètodes més importants de l'objecte `document` són:

- `write()`: escriu codi HTML en el document.
- `writeln()`: idèntic a `write()`, però inserta un retorn de carro al final.
- `open(tipus MIME)`: obre un *buffer* per recollir el que retorna els mètodes `write()` i `writeln()`. Els tipus MIME que es poden utilitzar són: `text/html`, `text/plain`, `text/jpeg`, etc.
- `close()`: tanca el *buffer*.
- `clear()`:esborra el contingut del document.

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>Objecte document. Mètodes</title>
        <script type="text/javascript">
          function ReemplacarContingut () {
                       document.open ("text/html");
                       document.write ("<a href=\"http://web.gencat.cat/ca/inici/\">genca
                       document.write ("<img src=\"http://web.gencat.cat/web/.content/Ima
                       document.close ();
          }
        </script>
```

```
        <img src="http://ioc.xtec.cat/educacio/templates/ioc/images/ioc_logo.png" /img><br
        <button onclick="ReemplacarContingut();">Reemplaçar el contingut del document</but
    </body>
</html>
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/pbeYEK?editors=1000.

2.2.3. L'objecte location

L'objecte `location` conté informació sobre l'actual URL, conté les propietats de la direcció URL del document.

L'objecte `location` és part de l'objecte `window` i, per tant, s'hi pot accedir a través de la propietat `window.location`.

Propietats de Location

Les propietats de l'objecte `location` són:

- `hash`: retorna la part de la URL que representa l'àncora (#). Per ex, *url#IOC*.
- `host`: retorna el *hostname* i el número de port de la URL.
- `hostname`: retorna el *hostname* de la URL.
- `href`: retorna la URL sencera.
- `origin`: retorna el protocol, *hostname* i port de la URL.
- `pathname`: retorna el *path* de la URL.
- `port`: retorna el número de port de la URL.
- `protocol`: retorna el protocol de la URL.
- `search`: retorna la part de *querystring* de la URL. Per ex, *url?search=IOC*.

Mètodes de location

Els mètodes més importants de l'objecte `location` són:

- `assign()`: carrega un nou document.
- `reload()`: recarrega el document actual.
- `replace()`: reemplaça el document actual per un de nou.

Un exemple per veure la major part d'aquestes propietats i mètodes:

Si proveu l'anterior exemple com a fitxer HTML obtindreu `location.protocol: file`. Si el proveu com a pàgina web que ens serveix un servidor web obtindreu `location.protocol: http`.

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>Objecte location. Propietats i Mètodes</title>
```

```
                ,
                function reloadUrl () {
                        document.location.reload();
                }
                function replaceUrl () {
                        document.location.replace('http://www.gencat.cat'); // fixeu-vos q
                }
                console.log(`location.hash: ${location.hash}`);
                console.log(`location.hostname: ${location.hostname}`);
                console.log(`location.href: ${location.href}`);
                console.log(`location.origin: ${location.origin}`);

                console.log(`location.pathname: ${location.pathname}`);
                console.log(`location.port: ${location.port}`);
                console.log(`location.protocol: ${location.protocol}`);
                console.log(`location.search: ${location.search}`);
        </script>
    </head>
    <body>
        <a href="http://ioc.xtec.cat">IOC</a><br />
        <img src="http://ioc.xtec.cat/educacio/templates/ioc/images/ioc_logo.png" /img><br
        <button onclick="assignUrl();">Mètode location.assign</button>
        <button onclick="reloadUrl();">Mètode location.reload</button>
        <button onclick="replaceUrl();">Mètode location.replace</button>
    </body>
</html>
```

Podeu veure aquest exemple en el següent enllaç, però tingueu en compte que a la web de Codepen els botons no funcionaran: codepen.io/ioc-daw-m06/pen/ZEQwgaP?editors=1111.

2.2.4. L'objecte history

L'objecte `history` conté les URLs visitades per l'usuari (en el marc d'una finestra del navegador). L'objecte `history` és part de l'objecte `window`, i s'hi pot accedir a través de la propietat `window.history`.

Propietats d'history

La propietat més important de l'objecte `history` és:

- `length`: retorna el nombre d'URL a la llista de l'històric de navegació.

Mètodes de history

Els mètodes més importants de l'objecte `history` són:

- `back()`: carrega la URL prèvia en la llista de l'històric de navegació.
- `forward()`: carrega la següent URL en la llista de l'històric de navegació.
- `go()`: carrega una URL específica en la llista de l'històric de navegació.

Com a exemple, i sempre dins de la mateixa finestra, podeu navegar per les següents URLs, i finalment introduir la URL del codi que es proposa.

- ca.wikipedia.org/wiki/Manresa
- ca.wikipedia.org/wiki/Balsareny
- ca.wikipedia.org/wiki/Navàs
- ca.wikipedia.org/wiki/Puig-reig
- ca.wikipedia.org/wiki/Gironella
- ca.wikipedia.org/wiki/Berga

```
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>Objecte history. Propietats i Mètodes</title>
    </head>
    <body>
        <button onclick="console.log(`Nombre elements: ${history.length}`);">No
        <button onclick="history.back()">URL anterior</button>
    </body>
</html>
```

2.2.5. L'objecte navigator

L'objecte `navigator` conté informació sobre el navegador web que s'està utilitzant.

Propietats de navigator

Les propietats de `navigator` són:

- `appCodeName`: retorna el codi del navegador, per exemple, Mozilla.
- `appName`: retorna el nom oficial del navegador. Per exemple, en el cas de Mozilla Firefox, retorna *netscape*.
- `appVersion`: la versió del navegador (potser no coincideix amb la versió real del navegador).
- `cookieEnabled`: retorna `true` si les galetes estan habilitades.
- `language`: retorna l'idioma del navegador.
- `onLine`: retorna `true` si el navegador està en línia.
- `platform`: retorna en quina plataforma es va compilar el navegador.
- `product`: retorna el nom del motor del navegador.
- `userAgent`: retorna tota la capçalera *user-agent* que el navegador envia al servidor en el protocol HTTP. Aquest valor l'utilitza el servidor per identificar el client. En aquesta capçalera sí que podem veure la versió real del nostre navegador (per exemple, Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:22.0) Gecko/20100101 Firefox/22.0).

En el següent exemple s'utilitzen les principals propietats de l'objecte `navigator`:

```
console.log(`appCodeName: ${navigator.appCodeName}`);
console.log(`appName: ${navigator.appName}`);
console.log(`appVersion: ${navigator.appVersion}`);
console.log(`cookieEnabled: ${navigator.cookieEnabled}`);
console.log(`language: ${navigator.language}`);
console.log(`onLine: ${navigator.onLine}`);
console.log(`platform: ${navigator.platform}`);
console.log(`product: ${navigator.product}`);
console.log(`userAgent: ${navigator.userAgent}`);
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/wWJOJr?editors=1012.

2.2.6. L'objecte screen

L'objecte `screen` conté informació sobre la pantalla on està obert el navegador del client. Aquesta informació s'extreu a partir de les propietats descrites més avall. En canvi, no té mètodes. No s'ha de confondre l'objecte `screen` amb l'objecte `window`, que representa la finestra del navegador.

Als dissenyadors web sempre els ha preocupat saber quina és la resolució del navegador del client per tal que les pàgines web es vegin bé en els diferents formats de pantalla. Avui en dia, gràcies a *frameworks* com *bootstrap*, l'adaptació de la web a diferents formats de pantalla és molt més fàcil.

- `availHeight`: retorna l'altura de la pantalla.
- `height`: retorna l'altura total de la pantalla. La diferència amb `availHeight` és l'altura de la barra de tasques (depèn del sistema operatiu).
- `availWidth`: retorna l'amplada de la pantalla.
- `width`: retorna l'amplada total de la pantalla.
- `colorDepth`: retorna la resolució de la paleta de colors (en número de bits). Per exemple: 8 significa 256 colors; 24 significa 16 milions de colors.
- `pixelDepth`:retorna la resolució de color (en bits per píxel) de la pantalla.

En el següent exemple s'utilitzen les principals propietats de l'objecte `screen` que acabem de veure:

```
console.log(`availHeight: ${screen.availHeight}`);
console.log(`height: ${screen.height}`);
console.log(`availWidth: ${screen.availWidth}`);
console.log(`width: ${screen.width}`);
console.log(`colorDepth: ${screen.colorDepth}`);
console.log(`pixelDepth: ${screen.pixelDepth}`);
```

Podeu veure l'exemple a: codepen.io/ioc-daw-m06/pen/NWxoQzz?editors=0012.