PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Desenvolupament web a l'entorn client

UT 3.3: Objectes predefinits.

Joan Pons Tugores

Índex de continguts

Javascript. Llenguatge basat en objectes.

Un llenguatge basat en objectes és aquell que inclou el concepte d'objectes, però no el de classes ni tot el relacionat amb elles com pugui ser l'herència.

Els llenguatges orientats a objectes, com Java, es basen en els conceptes de classe i instància. La classe defineix les característiques que compartiran els seus objectes, i una instància o objecte és un cas particular d'aquesta classe.

Un llenguatge basat en objectes, o basat en prototipus, com era inicialment JavaScript, no fa aquesta distinció, simplement té objectes. Un objecte no pertany a cap classe, però té propietats (el que serien atributs en Java) i funcions (mètodes).

A partir del ECMAScript15 s'inclouen els conceptes de classe i herència al llenguatge.

Podem crear els objectes amb la paraula clau *new*.

JavaScript disposa d'una sèrie d'objectes predefinits, que seran els que veurem en aquest tema.

Array

Un array és una variable que permet guardar múltiples valors indexats. Podem accedir individualment a un dels valors utilitzant aquest índex.

En JavaScript, **els arrays poden contenir qualsevol tipus de valor**. El seu tamany és dinàmic, de forma que si he creat un array de 10 posicions i assignam un valor a la posició 124, l'array s'expandeix fins a arribar a aquesta posició.

Definició

Podem definir un nou array utilitzant *new*. Té diversos constructors:

```
let valors = new Array()   //Crea un array, sense cap longitud específica.
let valors = new Array(10)   //Crea un array de, inicialment, 10 posicions.
let valors= new Array(2, "hola", true, 45.34); //Crea l'array i li assigna els valors.
```

Per motius d'eficiència es recomana no utilitzar new, sinó

```
let valors=[]; //Crea l'array buid.
let valors= [2, "hola", true, 45.34]; //Crea l'array i li assigna els valors. Claudàtors, no claus.
```

Assignació de valors

Es fa amb la notació tradicional:

```
valors[3]='valor';
```

La primera posició de l'array és la zero.

Particularitat: si l'index que posam és major que la longitud de l'array no passa res, expandeix l'array fins que tengui aquesta posició.

```
let valors = [2, "hola", true, 45.34];
valors[25] = 12.23;
```

La darrera instrucció crea les posicions amb index 4 a 25 i assigna a aquesta última el valor 12.23.

Les posicions 4 a 24 tenen el valor *undefined*.

Accés a una posició

Es realitza a través de l'índex:

```
let b=valors[index];
```

Si *index* és major del permès, longitud-1, evidentment no dona error, sinó que torna *undefined*. El mateix passa si *index* està dins el rang correcte, però a aquella posició encara no hem guardat cap valor.

Altres operacions amb arrays

Recorregut d'un array

Podem recórrer totes les posicions d'un array amb un for de dues formes diferents:

- Recórrer totes les posicions de l'array, tornant *undefined* per a les que no tenen cap valor assignat.

```
for(let index=0; index<vector.length; index++) { ... }
```

- Obtenir els índexs de les posicions amb valors assignats, no els de les posicions *undefined*. S'ha d'anar alerta.

```
for(let index in vector) { ... }
```

- Obtenir els valors de totes les posicions de l'array, inclosos els *undefined.*

```
for(let valor of vector){ ... }
```

Aplicar una funció a tots els elements de l'array.

*forEach é*s un «mètode» de l'array. Com a paràmetre rep la funció que hem d'aplicar. Aquesta funció s'executarà una vegada amb cada element distint d'*undefined*.

```
dades.forEach(numero => console.log(numero));
```

Concatenar arrays

Torna un array nou amb els valors o l'array que rep com a paràmetre afegits al final de l'array que executa aquest mètode.

```
Let valors = [1, 2, 3, 4];
let concatenat = valors.concat(llista de valors separada per comes || array)
let concatenat = valors.concat(5, 6, [7, 8, 9]);
//concatenat===[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Unir els elements

Podem unir tots els elements que formen l'array en una única cadena de text. Podem establir quina cadena posarem entre element i element.

```
let paraules = [“Hola”, “Joan”];
let frase = paraules.join(“ “); //frase=”Hola Joan”
```

Invertir l'array

Modifica l'array col·locant els elements en l'ordre invers al que estaven. Torna una referència a l'array.

```
let paraules = [“Hola”, “Joan”];
let girades = paraules.reverse();
//paraules=[“Joan”, “Hola”]
//girades=[“Joan”, “Hola”]
```

Afegir i eliminar elements

- **pop** Elimina i torna l'últim element de l'array. La longitud disminueix.

```
let valors = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const x = valors.pop();
// x=10, valors=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- **shift** Elimina i torna el primer element de l'array. La longitud disminueix.

```
const y = valors.shift();
// y=0, valors=[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- **push**(valor) Afegeix el valor al final de l'array. La longitud augmenta.

```
valors.push(y);
// valors=[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

- **unshift**(valor) Afegeix el valor a l'inici de l'array. La longitud augmenta.

```
valors.unshift(x);
// valors=[10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

- **splice**(index, quantitat, elements) Elimina quantitat elements començant per la posició index. Si apareix la part opcional elements, els insereix a partir d'index.

```
const z = valors.splice(3, 2);
//z=[3, 4]    valors=[10, 1, 2, 5, 6, 7, 8, 9, 0]
```

- **slice**(inici, final) Torna un array amb els elements indicats. Si no posam final, torna des d'inici fins al darrer element de l'array.

```
const w = valors.slice(3, 2);
//w=[5, 6]    valors=[10, 1, 2, 5, 6, 7, 8, 9, 0]
```

Ordenar els elements

- **sort**(funcio): funció és opcional. Ordena els elements de l'array. Si no especificam cap funció d'ordenació, els ordena alfabèticament. Si aquest ordre no ens va bé, per exemple per ordenar valors numèrics, li hem de passar una funció amb les següents característiques:
  - La funció compararà dos objectes.
  - Ha de tenir dos paràmetres, els dos objectes a comparar
  - Tornarà 0 si son iguals, un valor positiu si s'han d'intercanviar de posició dins l'array, i un negatiu si no ho han de fer.

Per exemple, una funció per ordenar de forma descendent un array amb valors numèrics podria ser la següent:

```
function ordena(a, b){
	if (a<b){
		return 1;
	}else if (b<a){
		return -1;
	}else{
		return 0;
	}
}
```

Array.map

Aquesta funció rep com a paràmetre una altra funció i l'aplica a cada un dels elements de l'array. Torna un array nou amb els nous valors. La funció ha de fer *return* del nou valor.

La funció rep tres paràmetres: un element, el seu índex dins l'array i l'array complet, encara que normalment només s'utilitzi l'element.

```
let nombres = [1, 4, 9];
let arrels = nombres.map(Math.sqrt);
// arrels ara val [1, 2, 3], nombres encara val [1, 4, 9]
```

Podem utilitzar una funció nostra. *map* passa a la funció tres arguments: el valor actual, l'índex actual i l'array complet. Podem utilitzar una anònima

```
let nombres = [1, 4, 9];
let dobles = nombres.map(function(num) {
  return num * 2;
});
// dobles ara val [2, 8, 18]. nombres encara val [1, 4, 9]

let triples = nombres.map(num => num * 3);

// triples ara val [3, 12, 27]. nombres encara val [1, 4, 9]
```

O podem utilitzar una funció amb nom

```
function doble(num){
        return num*2;
}
let dobles=nombres.map(doble);
```

Array.filter

Aquest mètode té com a paràmetre una funció que implementa el filtre que volem aplicar a tots els elements de l'array i torna un array nou amb els elements que han passat aquest filtre.

La funció rep tres arguments, l'element, l'índex i l'array, però normalment només farà falta l'element. Ha de tornar true si l'element passa el filtre i false si no el passa. Només es crida la funció per posicions de l'array que tenguin valors assignats.

```
let dades = [10, 5, 20, 15, 30];
let majors=dades.filter( valor =>  valor >= 15);
```

majors contendrà els valors [20, 15, 30]

Array.reduce

Aquest mètode redueix l'array a un sol valor aplicant-li la funció que rep com a paràmetre. Aquesta funció rep quatre arguments: l'element anterior, l'element actual, l'índex i l'array, però normalment només faran falta els dos primers. La funció ha de fer return del nou valor.

reduce té dos arguments, el primer és la funció, i el segon, opcional, el valor inicial. Si s'especifica aquest argument, la primera crida a la funció serà amb aquest valor com a element anterior i amb l'element 0 com a element inicial. Si no s'especifica, l'element anterior és l'element 0 i l'actual 1.

```
let dades = [1, 2, 3, 4];
let total = dades.reduce((anterior, actual) => anterior + actual);
console.log(total);
```

total tendrà el valor 10. La funció es cridarà quatre vegades. *elementAnterior* tendrà el valor 0 a la primera execució, i anirà acumulant les sumes a les següents crides.

Si volem obtenir el producte dels elements de l'array, necessitarem especificar el valor inicial com a 1, sinó el resultat sempre seria zero:

```
let dades = [1, 2, 3, 4];
let total = dades.reduce((anterior, actual) => anterior * actual, 1);
console.log(total);
```

Per exemple, es pot utilitzar *reduce* per aplanar un array multidimensional.

```
const original = [[0, 1], [2, 3], [4, 5]];
let flattened = original.reduce((a, b) => a.concat(b), []);
console.table(flattened);
```

Array.from

Torna un array amb els elements de la col·lecció, que pot ser un altre array, un conjunt, un mapa, ... És un mètode *static*, s'ha d'utilitzar amb *Array*.

```
let mapa= new Map(...);
let dades=Array.from(mapa);
```

Array.includes

Torna un booleà indicant si el valor rebut al paràmetre es troba a l'array. Té un segon paràmetre opcional que permet especificar la posició a partir de la qual cerca el valor.

let valors=[1,2,3,4,5,6,7];

```
console.log(valors.includes(3)); //true
console.log(valors.includes(3, 4)); //false, a partir de la posició 4 no hi ha
                                    cap 3.
```

Més informació

https://developer.mozilla.org/ca/docs/Web/JavaScript/Referencia/Objectes_globals/Array

http://www.w3schools.com/jsref/jsref_obj_array.asp

Date

L'objecte Date ens permet representar i manipular valors relacionats amb dates i temps:

```
var data = new Date();
```

Dins data hi guardarem la data i l'hora a la que s'executi aquesta instrucció. Internament es guarda com el nombre de mil·lisegons que han passat des del 1/1/1970

A part del constructor que ja hem vist, en té un altre que té com a paràmetre el nombre de mil·lisegons

```
var data = new Date(10000000000000); // "Sat Nov 20 2286 18:46:40 GMT+0100"
```

Afortunadament també en té tres més:

```
var data = new Date(2009, 5, 1); // 1 de juny de 2009 (00:00:00)
var data = new Date(2009, 5, 1, 19, 29, 39); // 1 de juny de 2009 (19:29:39)
```

El format és any, mes dia, hora, minuts, segons. Els mesos van des del 0 (gener) fins a l'11 (desembre). En canvi els dies van des de l'1 fins al 31.

```
var data=new Date("11/25/2012"); //En format americà
```

Mètodes de Date

- getTime() Torna els milisegons des de 1 de gener de 1970.
- getMonth() Torna el més de la data, 0 per gener i 11 per desembre.
- getFullYear() Torna l'any com un número de 4 xifres.
- getDate() Torna el número el dia del mes.

- getDay() Torna el dia de la setmana, 0, per diumenge i 6 per dissabte.
- getHours(), getMinutes(), getSeconds(), getMilliseconds() Tornen respectivament les hores, minuts, segons y mil·lisegons de la hora guardada al Date.

Cada un d'aquests mètodes get té el seu set corresponent per a establir el valor de la propietat.

- toLocaleString(), toLocaleDateString(), toLocaleTimeString(): Tornen la data i el temps en el format local definit al client. Pot variar d'un client a un altre.

Mètodes estàtics de Date

- Date.now(): Torna els mil·lisegons de la data i hora actuals
- Date.parse(cadena) Intenta convertir la cadena de text a data. La torna en mil·lisegons. No es recomana degut a les inconsistències en les implementacions dels diferents navegadors.

Més informació

https://developer.mozilla.org/ca/docs/Web/JavaScript/Referencia/Objectes_globals/Date

http://www.w3schools.com/jsref/jsref_obj_date.asp

Math

L'objecte Math permet utilitzar constants i realitzar operacions matemàtiques. Totes les funcions i propietats són estàtiques.

Moltes funcions tenen una precisió que depèn de la implementació, per la qual cosa diferents navegadors, o el mateix navegador en diferents arquitectures pot donar resultats diferents.

Constants

- Math.E
- Math.PI
- Math.SQRT2 (Arrel quadrada de 2)
- ...

Mètodes

- Math.abs(x) valor absolut de x
- Math.random valor aleatori entre 0 i 1
- Math.floor(x), Math.ceil(x), Math.round(x) l'enter immediatament més petit, més gros o mes proper al paràmetre.
- Math.max(...x), Math.min(...x) tornen el major i el menor respectivament dels arguments que reben.
- Math.pow(x,y) torna x^y.
- Math.sqrt(x) torna l'arrel quadrada de x.
- ...

Més informació

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

http://www.w3schools.com/jsref/jsref_obj_math.asp

String

L'objecte *String* s'utilitza per a manipular la cadena de text que guarda. Ja la varem veure al tema anterior.

Propietats

- length: Torna la longitud de la cadena de text.

Mètodes

- .charAt(index): Torna el caràcter que ocupa la posició *índex* dins la cadena que executa el mètode.
- .charCodeAt(index) torna el valor unicode del caràcter de la posició índex
- .fromCharCode(codi, codi, ...) Converteix els codis unicode a caràcter.
- .includes(String): Torna true si la cadena que executa el mètode conté la que rep de paràmetre.
- .startsWith(String), .endsWith(String): Torna true si la cadena que executa el mètode comença, acaba, amb la que rep el paràmetre.
- .indexOf(string), .search(String): Tornen l'index on comença la cadena que rep el paràmetre dins la que executa el mètode. Torna -1 si no la troba. La diferència entre els dos mètodes és que search admet expressions regulars.
- .split(separador): Torna un array on cada posició és una de les parts de la cadena obtingudes separant-la per l'expressió regular que rep el paràmetre.

- HTML Wrapper Methods Mètodes que emboliquen la cadena de text dins una etiqueta HTML
  - link(url) Torna la cadena dins una etiqueta <a> amb href=url

```
const str = "CIFP Pau Casesnoves";
let enllaç = str.link("http://www.paucasesnovescifp.cat")
<a href="http://www.paucasesnovescifp.cat">CIFP Pau Casesnoves</a>
```

L'hem d'assignar a qualque element de la pàgina per que es vegi:

```
document.getElementById("mostrador").innerHTML=enllaç;
```

- ...

Més informació

https://developer.mozilla.org/ca/docs/Web/JavaScript/Referencia/Objectes_globals/String

http://www.w3schools.com/jsref/jsref_obj_string.asp

Global

L'objecte Global proporciona propietats i mètodes globals que utilitzen els altres objectes.

Propietats

- Infinity
- NaN
- undefined

Mètodes

- IsNaN
- parseInt(), parseFloat()
- eval(cadena). Avalua la cadena com a text JavaScript i l'executa. Pot ser perillós.
- encodeURI(cadena) Torna la cadena codificada com a uri transformant caràcters com &, ?, " ",
- ...

Més informació

http://www.w3schools.com/jsref/jsref_obj_global.asp

Function

L'objecte *Function* ens proporciona un constructor per a crear funcions. Es pot utilitzat per a generar una funció en temps d'execució.

Mètodes

- Constructor: Function(arg1, arg2, ... , argN)

  Tots els arguments es passen com a cadenes de caràcters.

  El darrer argument és el codi de la funció tal qual l'escriuríem en una funció normal, només que el tancam dins una cadena.

  Els altres arguments són els noms dels paràmetres que tendrà la funció.

```
<script>
var funcio=new Function("a", "b", "return a+b;");
var resultat=funcio(3,2);
document.write("Resultat: "+resultat);
</script>
```

Set

L'objecte Set manté un conjunt, una col·lecció de valors sense repeticions. Per avaluar les repeticions s'utilitza ===. Aquests valors poden ser iterats segons l'ordre d'inserció.

Constructor: new Set([objecte iterable])

Un objecte iterable pot ser un Array, String, Set, Map,

Propietats

- size: el nombre d'elements del conjunt

Mètodes

- add(valor) afegeix el valor al conjunt, Torna el conjunt.
- clear(): Elimina tots els valors del conjunt.
- has(valor): Torna un booleà indicant si el valor es troba o no al conjunt.
- delete(valor): Elimina el valor del conjunt. Torna el resultat que tornaria *has(valor)* abans de l'eliminació.
- values(): Torna un iterador amb els valors del conjunt, per exemple, per utilitzar amb un for of.

Recorregut dels valors d'un conjunt

Podem utilitzar tant el conjunt com el resultat de values() dins d'un for.

```
let conjunt = new Set([6, 5, 4, 3, 2, 1]);
for (let valor of conjunt.values()) {
        document.write("<p>" + valor + "</p>")
}
```

Encara que no sigui necessari

```
for (let valor of conjunt) {
        document.write("<p>" + valor + "</p>")
}
```

També utilitzant el forEach

```
conjunt.forEach(x => console.log(x));
```

Més informació

https://developer.mozilla.org/ca/docs/Web/JavaScript/Referencia/Objectes_globals/Set

Map

L'objecte *Map* relaciona claus amb valors. Qualsevol valor es pot fer servir com a clau o com a valor.

No hi pot haver claus repetides.

- Constructor: new Map([objecte iterable])

  Un objecte iterable pot ser un Array, String, Set, Map,

Propietats

- Size: el nombre d'elements del mapa(parells clau-valor).

Mètodes

- set(clau, valor) afegeix el mapatge clau-valor al mapa. Torna el mapa. Si la clau ja existeix assigna el nou valor a la clau i elimina l'anterior valor associat a ella. El mateix valor si que pot estar associat a diferents claus(encara que no tengui gaire sentit fer-ho).
- get(clau): Torna el valor associat a la clau o undefined si no hi ha aquesta relació.
- clear(): Elimina tots els valors del conjunt.
- has(clau): Torna un booleà indicant si el valor es troba o no al mapa.
- delete(clau): Elimina el valor del mapa. Torna el resultat que tornaria *has(valor)* abans de l'eliminació.
- values(): Torna un iterador amb els valors del mapa.
- keys(): Torna un iterador amb les claus del mapa.

- entries(): Torna un iterador que conté un array de dues posicions [clau, valor] per a cada element del mapa.

Recorregut dels valors d'un mapa

Podem utilitzar tant el mapa com el resultat de *values*() dins d'un *for of.* Si utilitzam el mapa per a cada entrada ens tornarà un array amb dos valors.

```
myMap.set(0, "zero");
myMap.set(1, "un");
for (let entrada of myMap)
  console.log(entrada[0] + " = " + entrada[1]);
```

Podem utilitzar la desestructuració per facilitar el recorregut

```
for (let [key, value] of myMap)   console.log(key + " = " + value);
```

Si utilitzam *values* ens tornarà els valors. I si utilitzam *keys*, només les claus.

```
for (let value of myMap.values()) console.log(value);
```

I si utilitzam *keys*, només les claus.

```
for (var value of myMap.keys()) console.log(key);
```

Més informació

https://developer.mozilla.org/ca/docs/Web/JavaScript/Referencia/Objectes_globals/Map