PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Desenvolupament web a l'entorn client

UT 3.2: Funcions.

Joan Pons Tugores

Índex de continguts

En aquest apartat veurem alguns aspectes nous de les funcions que segurament no vàreu veure a programació.

Arguments

Quan definim una funció definim quins paràmetres tendrà. En JavaScript ens trobam amb unes quantes peculiaritats:

- No passa res si no assignam un valor a cada paràmetre. Els que quedin sense assignar tendran el valor *undefined*.
- Si assignam més valors dels esperats tampoc passa res. Els que sobren seran ignorats.
- Per tant és impossible la sobrecàrrega de funcions.
- Tota funció inclou una variable local anomenada arguments amb les següents propietats:
  - Es pot utilitzar com un array on cada posició és un dels arguments passats a la funció.
  - length: la quantitat d'arguments passats, no els esperats.
  - No permeses en mode estricte:
    - arguments.callee torna el codi de la funció.
    - arguments.callee.caller torna el codi de la funció que ha cridat a aquesta.

Per altra banda:

- nomFunció.length ens torna la quantitat de paràmetres que espera la funció.
- nomFunció.name ens torna el nom de la funció.

Valors per defecte

A diferència d'altres llenguatges, Javascript admet valors per defecte per als arguments:

```
function prova(nom="Jo Mateix"){
  alert(nom);
}
```

Si cridam la funció amb un valor per el paràmetre utilitzarà el valor que ha rebut:

```
prova("Joan"); //Mostrarà Joan
```

Si cridam la funció sense paràmetres

```
prova(); //Mostrarà Jo Mateix
```

Si tenim paràmetres amb valors per defecte sempre han de ser els últims. Javascript assigna el primer argument al primer paràmetre, el segon al segon paràmetre, ... Si tenc

```
function prova(nom="Jo", llinatges="Mateix", adreça="Ca Nostra"){
  alert(nom);
}
```

No podem posar

```
prova("Joan",,"Joan Miró 23");
```

I si posam

```
prova("Joan","Joan Miró 23");
```

estic donant valors al nom i als llinatges.

Paràmetres Rest

Igual que en Java podem definir un paràmetre que accepti qualsevol quantitat de valors són els anomenats paràmetres *Rest*. Només l'últim paràmetre de la funció pot ser d'aquest tipus.

La principal diferència amb arguments és que mentre arguments conté tots els valors passats a la funció, inclosos els que tenen nom, un paràmetre *Rest* només conté els valors restants, els que no tenen nom.

Es declaren posant ... (tres punts) davant el nom del paràmetre. Dins la funció tendrem un array amb el nom d'aquest paràmetre.

```
function suma(...valors){
  let resultat=0;
  for(let index in valors){
        resultat+=valors[index];
  }
}
```

Funcions assignades a variables

Una variable pot tenir assignada una funció:

```
function suma(a, b){
  return a+b;
}
let operacio=suma; // operacio és un àlias de suma
let resultat=suma(2,3); // resultat guarda el que torni la funció suma
                        //aplicada a 2 i 3
console.log(operacio(5,7)); //Escriu la suma de 5 i 7
```

Per tal que la variable, o l'esdeveniment, tengui assignada la funció i no el seu resultat, no hem d'incloure els parèntesis, a l'assignació només posam el nom de la funció.

A partir del moment que hem fet l'assignació, podem utilitzar el nom de la variable com si fos, realment ara ho és, una funció.

Ja veurem més endavant que aquesta característica del llenguatge serà la que ens permetrà assignar mètodes a objectes.

El més freqüent és assignar les funcions no a variables, sinó a esdeveniments:

```
document.getElementById("boto").onclick=suma;
```

Si com a resposta a l'esdeveniment hem de passar arguments a la funció suma com ho feim?

Resposta: L'apartat següent.

Funcions anònimes

Encara que sembli mentida, el nom d'una funció és opcional. Això ens permet definir una funció sense nom:

```
function(a, b){
  return a+b;
}
```

Aquest codi no ens donaria cap errada, però és una mica inútil. Hem creat la funció però no hi ha manera de cridar-la.

Normalment es declaren funcions anònimes quan s'assigna la funció a una variable o, el més habitual, a un esdeveniment:

```
let suma=function(a, b){
  return a+b;
}
document.getElementById("resultat").innerHTML=suma(2,3);
document.getElementById("boto"2).onclick=function(){
            ...
 };
```

Recursivitat: Una funció anònima no pot ser recursiva.

Si hem de passar arguments a una funció assignada a un esdeveniment: La tancam dins d'una funció anònima:

```
document.getElementById("boto").onclick=function(){
   suma(2,3);
};
```

Nota: Darrera la clau hi ha un punt i coma perquè acaba l'assignació a l'esdeveniment.

Nota: Sempre que es crei una funció per ser assignada única i exclusivament a un esdeveniment és millor fer-la anònima, d'aquesta manera tenim un identificador menys al programa.

```
document.getElementById("boto").onclick=function(){
  alert("Botó pitjat!"):
};
```

Funcions com a paràmetre

Una funció pot rebre com a paràmetre una altre funció i cridar-la utilitzant el nom del paràmetre:

```
function suma(a, b){
  return a+b;
}
function resta(a, b){
  return a-b;
}
function calcula(a, b, operacio){
  document.write(operacio(a,b));
}
calcula(2,3,suma);  //escriurà a la pàgina 5
calcula(8,2,resta); //escriurà a la pàgina 6
```

A la funció *calcula* li podrem passar qualsevol funció. Sigui quina sigui, l'executarà i escriurà el resultat que li torni aquesta funció.

Passar funcions com a paràmetres és molt freqüent en frameworks com Jquery, Angular, React, ... En fer AJAX també ho utilitzareu.

Funcions internes

Dins del cos d'una funció podem definir una altra funció.

```
function grossa(a,b){
        let cadena='Resultat: ';
        function petita(valor){
                alert(cadena+valor);
        }
        petita(a+b);
}
grossa(3,5);
```

Dins la funció grossa hem definit una altra funció, petita, només accessible des de dins la funció grossa.

La funció interna, petita, té accés a totes les variables locals de la funció que la conté. És el que es coneix com a **clausura**.

Serà molt freqüent utilitzar funcions internes, per exemple dis el window.onload per assignar funcions anònimes als esdeveniments.

```
window.onload=function(){
  document.getElementById("boto").onclick=function(){
        alert("Pitjat!!!);
  }
}
```

Clausura

Implicacions de la clausura:

L'espai de noms de la funció externa és accessible des de la funció interna.

L'espai de noms de la funció externa és manté actiu mentre ho estigui la funció interna.

```
function assignaBotons(){
	for(var i=0; i<10; i++){
		document.getElementById("boto"+i).onclick=function (){
			alert(i);
		}
	}
}
```

En executar-se la funció interna s'agafen els valors actuals de les variables de l'externa. Tots els botons de l'exemple anterior mostraran un alert amb el valor 10.

Declarant la variable amb let no passa, ja que per a cada iteració guarda un valor distint.

```
function assignaBotons(){
	for(let i=0; i<10; i++){
		document.getElementById("boto"+i).onclick=function (){
			alert(i);
		}
	}
}
```

Arrow functions

Tenim una altra notació per definir funcions, que funciona pràcticament igual que la tradicional excepte un parell de detalls: No es pot utilitzar per definir mètodes d'objectes, no té *arguments*, no poden ser constructors, ...

```
(a, b) =>{
        return a+b;
}
```

Declara una funció anònima amb dos paràmetres que torna la suma dels seus valors. Es pot assignar a una variable o constant.

```
const suma = (a, b) =>{
        return a+b;
}
```

Si només té un paràmetre no fan falta els parèntesis

```
const negatiu = a =>{
  return -a;
}
```

I si només té un return com a única instrucció no fan falta ni el return ni les claus.

```
const negatiu = a => -a;
```

Per assignar funcions anònimes a esdeveniments també és pot utilitzar:

```
window.onload= () => {
...
}
```

Les *arrow functions* tenen unes quantes limitacions:

- No tenen *arguments*.
- No reben la referència a *this*. Això impedeix per exemple, que siguin mètodes d'una classe.
- No poden ser constructors.