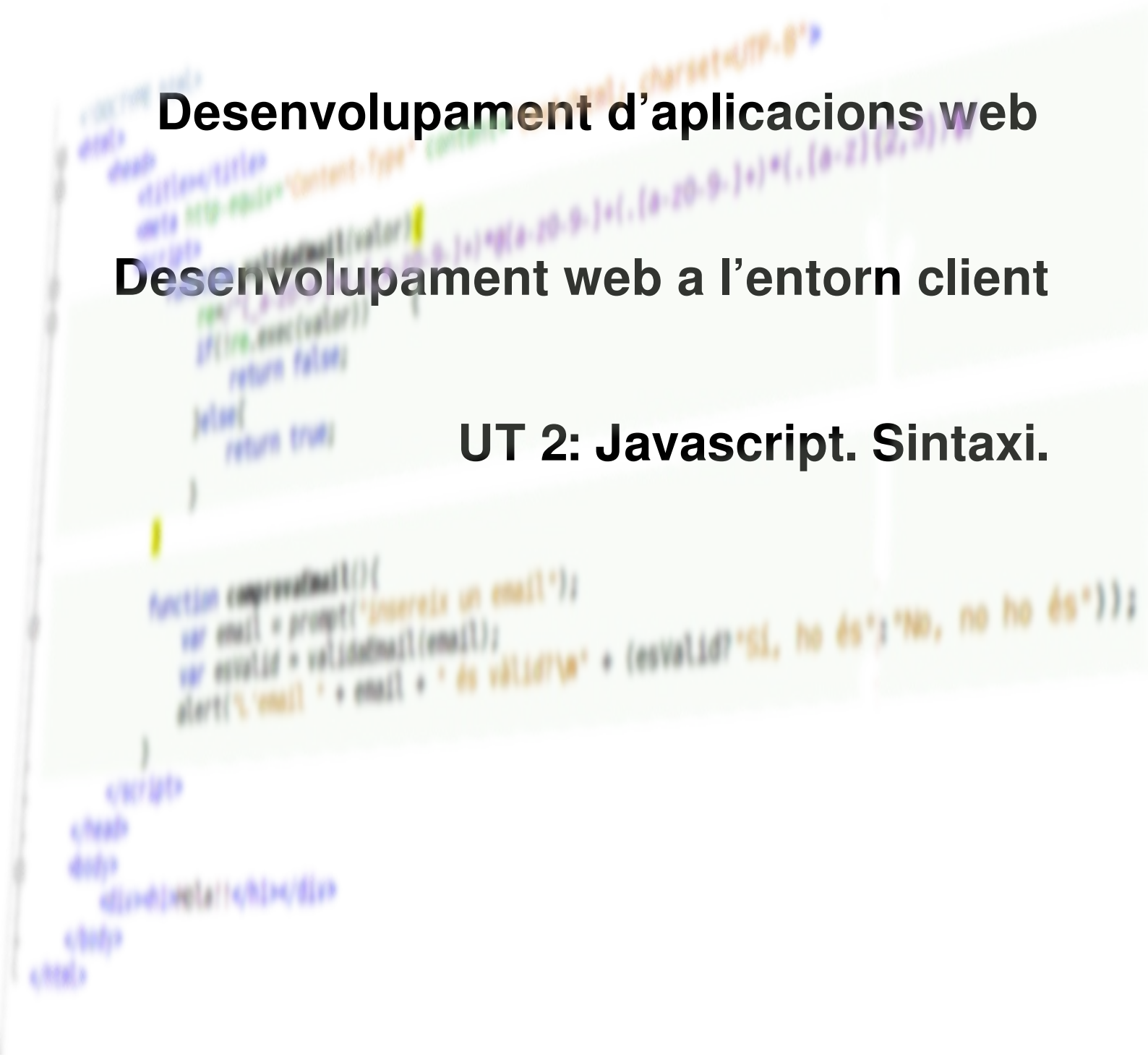
PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Desenvolupament web a l'entorn client

UT 2: Javascript. Sintaxi.



Joan Pons Tugores

Índex de continguts

Sintaxi del llenguatge Javascript....3
Modes de Javascript....3
Variables....5
Tipus de variables....6
Conversions de tipus....7
3 tipus de càstings:....8
Visibilitat de les variables....9
Operadors....11
Unaris....11
Booleans....11
Aritmètics....12
Relacionals....12
Equivalència....12
Asignació....12
Spread....13
Desestructuració....14
Operador ternari ?:....15
typeof....15
Plantilles de cadenes....16
Control de flux....17
Condicional....17
Iteracions....17
Altres....19
Funcions i propietats bàsiques de Javascript....20
Funcions per a cadenes de text....20
Funcions per a nombres....20
Funcions definides per l'usuari....21

Sintaxi del llenguatge Javascript

La versió estàndard de JavaScript es diu ECMAScript. La majoria de navegadors segueixen aquest estàndard.

JavaScript és un llenguatge de propòsit general que va néixer, com ja hem dit, pera realitzar tasques senzilles al costat del client, com validacions de formularis, per exemple.

La seva sintaxi és molt semblant a la de C i a la de Java. Per cert, encara que ho pugui semblar, Javascript i Java no tenen gaires coses en comú. Com a característiques generals podem comentar que:

- El espais en blanc s'ignoren, els que sobren volem dir.
- Es distingeixen majúscules i minúscules.
- No és necessari acabar les sentències amb ; encara que és recomanable
- Els blocs de codi es tanquen amb { }
- Feblement tipat.

Modes de Javascript

Inicialment el Javascript era un llenguatge interpretat, cada instrucció de codi font es transformava a codi executable cada vegada que es trobava al codi. Si una instrucció es trobava dins d'una iteració que es repetia 10 vegades, es feia aquest procés 10 vegades.

Això permetia coses com per exemple no declarar les variables. En trobar un identificador nou es creava una variable nova. Si el desenvolupador s'equivocava teclejant el nom d'una variable, javascript no donava error, sinó que creava una nova variable.

Amb el pas del temps els motors de Javascript dels navegadors s'han anat complicant per millorar l'execució del codi, i avui en dia la majoria compila el codi, el que permet incloure optimitzacions.

Per obligar al desenvolupador a seguir unes "bones pràctiques" de programació, facilitar la tasca dels motors i millorar el temps d'execució dels scripts, amb Ecmascript 5 va aparèixer l'*strict mode*. En aquest mode, moltes coses que es permetien abans, com no declarar les variables, ara donen error.

Característiques "noves" del llenguatge, com les classes o els mòduls, i molts frameworks actuals, com Vue o React, utilitzen l'strict mode. **És el que utilitzarem en aquest curs**.

Per cert, el javascript original es sol anomenar *sloppy mode*, mode descuidat.

Per utilitzar el mode estricte, a cada script hem de posar a l'inici

```
'use strict';
```

Variables

JavaScript, al contrari que Java, és un llenguatge feblement tipat, això vol dir que **les variables no es defineixen de cap tipus en concret**, sinó que són del tipus del valor que els assignam. Una mateixa variable pot contenir un valor numèric i més tard una cadena de caràcters.

Les variables locals a una funció es declaren amb les paraules reservades **var, i** preferiblement, **let i const** (per constants)

```
let nom='Joan';
const x=12;
nom=4;
x=5; //Error
```

En *sloppy mode* no és necessari declarar-les. La primera vegada que l'interpret es troba una assignació a una variable, si no la té creada, crea en aquell moment una **variable global**. Per tant, si ens equivocam teclejant el nom de la variable no dona error, sinó que en crea una de nova. En canvi, si la primera vegada que es troba una variable és en una operació que intenta recuperar el seu valor, dona error.

Per saber si una variable està declarada:

```
if (typeof nomVariable != 'undefined'){ … }
```

El nom de les variables es coneix com a identificador i ha de seguir les següents normes:

- No pot començar per un número
- Només pot estar format per lletres, números i els símbols $ i _

Tipus de variables

undefined: tipus d'una variable sense cap valor (sense inicialitzar) o no declarada.

null: variable o objecte no definit. És un valor que s'ha d'assignar explícitament

Boolean: true o false.

Number: pot contenir valors enters de 32 bits o valors en punt flotant de 64 bits.

```
let num = 98;
let num = 070; // Octal. En decimal : 56
let num = 0xAB; // Hexadecimal. En decimal : 171
```

String: pot ser delimitat per cometes simples o dobles.

```
let primer = "primer"
let segon = 'segon'
```

Es poden utilitzar les expressions següents:

\t(Tabulador) \n(Nova línia) \" (Cometes dobles) \' (Simples) \\(Barra)

Arrays: Són objectes que contenen diferents valors, indexats, amb l'índex que comença per zero. Per accedir als seus valors utilitzam elnom de la variable seguit de l'índex entre claudàtors.

```
let personatges=['Bob', 'Patricio', 'Calamardo'];
console.log(personatges[1]);
```

Objectes:Ja ho veurem més endavant, però podem dir que un objecte és un tipus de dades que conté distintes propietats i mètodes. Es defineixen amb claus (o amb funcions costructores i classes) i cada propietat especifica el seu nom

```
let personatge={nom:'Bob', edat:5};
console.log(personatge.nom);
console.log(personatge[nom]);
```

Conversions de tipus

El Javascript intenta sempre realitzar les instruccions escrites. Si a una operació hi ha dos tipus de dades diferents, automàticament intenta fer la conversió d'un d'ells abans de fallar.

Per exemple si es troba amb l'operació 2+'2' el resultat serà '22': Converteix el 2 numèric a cadena de caràcters i fa la concatenació.

Per evitar confusions d'aquest tipus podem fer les conversions explícitament:

- Conversió cap a String: booleans, numbers i string tenen el mètode .toString() per convertir valors a string.

```
let dada=true;
true.toString(); ens retorna "true"
dada.toString(); ens retorna "true"
dada=212
dada.toString(): ens torna "212"
```

Una altra manera és concatenant una cadena buida al valor que volguem convertir:

```
dada+"";
```

- Conversió cap a number: dos mètodes, parseInt() i parseFloat()

```
var num = parseInt("123");      //retorna 123
var num = parseFloat("8.45"); //retorna 8.45 (punt per decimals!!)
var num = parseInt("blue");     //retorna NaN
```

Alerta amb el parseInt i la base.

3 tipus de càstings:

- Boolean(value): converteix el valor donat a booleà. Retorna true quan el valor és un string com a mínim d'un caràcter, un nombre distint de 0 o un objecte.
- Number (value): fa el càsting a number, ja sigui enter o float. **Millor emprar el parseInt o el parseFloat**.
- String(value): fa el casting del valor a string

A la pràctica no se solen emprar aquests càstings, ja que l'intèrpret de javascript ho fa tot sol.

Visibilitat de les variables

- Qualsevol variable definida fora d'una funció és una **variable global**. L'àmbit global es refereix a **qualsevol script que hi hagi definit o enllaçat a la pàgina**, no només a l'script on està definida. **No hi hauria d'haver variables globals.**
- Una variable definida sense utilitzar var, ni let ni const dins una funció és una variable global. **No hi hauria d'haver variables globals.**
- Un variable definida utilitzant **var** dins una funció és **local a la funció**, no és pot utilitzar fora d'ella.
- Una variable declarada amb **let** o **const** és local al bloc on està definida, per exemple dins d'una iteració o d'un condicional.

Per assegurar-nos que una variable està definida podem utilitzar:

```
if (typeof  variable != "undefined") {
  vol dir que està definida i té un valor assignat
}
```

Evidentment, l'ordre en que s'inclouen els scripts o es criden les funcions o es declaren les variables és important. JavaScript és un llenguatge interpretat. Fins que la línia on es defineix una variable no s'interpreta, aquella variable no està definida.

```
function doi(){
        var a=3; // local a tota la funció;
        b="hola"; //global a la pàgina. No es permés en mode estricte
        If (a<2) {
                let c=12.5; //Només visible dins l'if
                var d="aa"; //visible a tota la funció
        }
        console.log(c); // error. undefined
        console.log(d); // Correcte.
}
```

Operadors

Unaris

- Prefixe increment/decrement: incrementen o decrementen en una unitat un valor numèric. Si estan dins una expressió o com a argument a una funció primer modifiquen i després torna el valor ja modificat.

```
var num = 10;
alert(--num);   //mostra 9
alert(++num);   //mostra 10
```

- Sufixe increment/decrement: també incrementen o decrementen en una unitat un valor numèric. Si estan dins una expressió o com a argument a una funció primer tornen el valor de la variable i després el modifiquen.

```
var num = 10;
console.log("Prefixe --: "+num--);   //mostra 10. En acabar num val 9
console.log("num: "+num); //El valor actual de num
console.log("Prefixe ++: "+num++);   //mostra 9. En acabar num val 10
console.log("num: "+num); //El valor actual de num
```

Booleans

- !: NOT lògic → var o = (!condició);
- &&: AND lògic. Per vàries condicions no avalua a partir d'un valor false. → var o = (cond1 && cond2 && cond3)
- ||: OR lògic. Per vàries condicions no avalua a partir d'un valor true. → var o = (cond1 || cond2 || cond3)

Aritmètics

- * multiplicació → var o = 58 * 59;
- / divisió → var o = 58 / 59;
- + suma → var o = 58 + 59;
- - resta → var o = 58 – 59;
- % mòdul → var o = 58 % 59;

Relacionals

Tots tornen un valor booleà

- > major
- < menor
- >= major o igual
- <= menor o igual

Equivalència

- == Retorna true si el valor d'ambdós operands són iguals 2=="2" → true
- === Retorna true si ambdós operands són del mateix tipus i valor 2==="2" → false
- != Retorna true si i només si els valors dels dos operands no són iguals.
- !== Retorna true si el valor no és el mateix o el tipus tampoc. 2!=="2" → true

Asignació

= a=2 b=a

Spread

spread: l'operador ... permet que un element iterable(un array o cadena) sigui expandit on s'esperin zero o més arguments, o si és un objecte, on s'esperin zero o mes parells clau valor per objectes.

```
function sum(x, y, z) {
  return x + y + z;
}
const numbers = [1, 2, 3];
console.log(sum(...numbers)); // output: 6
```

Es pot utilitzar per incloure el contingut d'un array dins d'un altre array:

```
const numbers = [1, 2, 3];
const more=[0, ...numbers, 4, 5, 6]
```

O per copiar el contingut d'un array a un altre:

```
const numbers = [1, 2, 3];
const copia=[ ...numbers]
```

O per concatenar arrays:

```
let primer = [1, 2, 3];
const segon=[4, 5, 6];
primer=[...primer, ...segon];
```

Amb objectes, es pot utilitzar per copiar-los o modificar-los.

```
const original = { validat: true, actiu: false }
const copia = { ...original }
```

El codi anterior crea un objecte nou amb les propietats i els valors de l'objecte original.

En crides a funcions el podem utilitzar per assignar els valors de l'array o l'objecte als paràmetres de la funció:

```
function suma(a, b, c) {
  return a + b + c;
}
const valors=[2, 4, 6];
console.log(suma(...valors));
```

Desestructuració

Ens permet assignar propietats d'objectes o valors d'arrays a variables.

En el cas dels arrays, utilitzam claudàtors, [ ], per tancar les variables:

```
let personatges=['Bob','Patricio', 'Calamardo'];
let [p1, p2, p3, p4]=personatges;
console.log(p1+" "+p2+" "+p3+" "+p4);
```

les variables dins dels [ ] del let agafen els valors de les posicions de l'array, per ordre. Si hi ha menys variable que valors a l'array, els que sobren s'ignoren. Si hi ha més variables que valors a l'array, les variables que sobren tenen el valor *undefined*.

En el cas dels objectes utilitzam claus { }:

```
const note = {
  id: 1,
  title: 'My first note',
  date: '01/01/1970',
}
const { id, title, date } = note;
```

Per defecte cream variables amb el mateix noms que les propietats dels objectes. Si volem canviar el nom de les variables, podem utilitzar els dos punts:

```
const { id: noteId, title, date } = note
```

Ara tendrem les variables noteId, title i date.

Operador ternari ?:

És la forma reduïda de l'if else. Té la forma

(condicio) ? Expressió si es vertadera : Expressió si és falsa

Si la condició és certa torna el valor de la primera expressió i si és falsa el de la segona.

```
const a=3;
let resultat=(a<5)?'Menor':"Major";
//resultat serà 'Menor'
```

typeof

És un operador unari que ens torna el tipus de la variable o expressió que té com a operand.

```
let num=5;
console.log(typeof num); // number
num='5';
console.log(typeof num); // string
```

Plantilles de cadenes

ES6 ens permeten incorporar el valor de variables dins d'una cadena sense haver de concatenar. Hem de fer dues coses:

1. Tancar la cadena amb ` (L'accent, no la cometa simple)
2. Dins la cadena hem de posar ${nom variable}

```
let num=5;
console.log(`Quantitat: ${num} euros`);

//Per consola apareixerà Quantitat: 5 euros
```

Control de flux

Condicional

Permet executar codi depenent del valor d'una expressió booleana. Quan el valor sigui true s'executaran les instruccions.

```
if(expressióBooleana){
  instruccionsSiVertader;
}
```

La clàusula *else* ens permet determinar quines instruccions s'executaran si l'expressió booleana dona false.

```
if(expressióBooleana){
  instruccionsSiVertader;
} else {
  instruccionsSiFals;
}
```

Iteracions

do{ } while (expressióBooleanaCerta) Es repeteixen les instruccions mentre la condició sigui certa. S'executen com a mínim una vegada.

```
do{
  instruccions;
}while( expressioBooleanaCerta );
```

while (expressióBooleanaCerta) {...} Es repeteixen les instruccions mentre la condició sigui certa. Pot ser que mai s'executin.

```
while(expressioBooleanaCerta){
  instruccions;
}
```

for(inicialització; expressióBooleanaCertaPerIterar; actualització)

S'inicialitza la variable que controla la iteració. Es comprova la condició. Si és certa s'executen les instruccions. S'actualitza la variable que controla la iteració. Es comprova la condició. Si és certa s'executen les instruccions...

```
for(var index=0; index <10; index++){
  document.write("<p>"+index+"</p>");
}
```

for(index in col·lecció) Es repeteixen les instruccions per a cada índex associat a un element que es trobi dins la col·lecció o per a cada nom de cada propietat de l'objecte. Inclou les posicions sense valor assignat. Per exemple,

```
let diesFeiners=["Dilluns", "Dimarts", "Dimecres", "Dijous", "Divendres"];
for(let index in diesFeiners){
  alert(diesFeiners[index]);
}
```

for(index of col·lecció) Es repeteixen les instruccions per a cada element que es trobi dins la col·lecció o per a cada propietat de l'objecte. S'exclouen les posicions sense valor assignat, *undefined*. Per exemple,

```
let diesFeiners=["Dilluns", "Dimarts", "Dimecres", "Dijous", "Divendres"];
for(let valor of diesFeiners){
  alert(valor);
}
```

continue: interromp l'iteració actual i segueix amb la següent.

break: interromp l'execució de la iteració.

Altres

switch: permet seleccionar quin bloc de codi s'executa depenent del valor que torni una expressió. No cal que sigui una expressió numèrica. Si no hi posam els break s'executarà el bloc amb el valor de l'expressió i tots els següents.

```
switch (expressio) {
  case value1: accions:
          break;
  case value2: accions:
          break;
  default: accions;
}
```

Comentaris: com en c i java:

// comentaris d'una línia

/* comentaris

d'un bloc

de línies */

Blocs de codi: com ja hem dit abans es determinen amb claus { }

Funcions i propietats bàsiques de Javascript

Funcions per a cadenes de text

- **length**: torna la longitud de la cadena "hola".length
- **concat**: concatena cadenes de text "hola ".concat("Joan");
- **+** concatena dues cadenes de text "hola "+"Joan"
- **toUpperCase / toLowerCase:** converteixen la cadena a majúscules/minúscules.
- **charAt(posició)**: torna el caràcter que ocupa la posició que indicam.
- **indexOf(caràcter)**: torna la posició de la primera aparició del caràcter dins la cadena.
- **substring(inici, final)**: torna una subcadena formada per els caràcters des de la posició inici fins a la posició final.
- **split(separador)**: torna un array de cadenes de text obtingut separant la cadena original per el separador.

Funcions per a nombres

- **NaN**: valor que torna l'interpret quan s'intenta avaluar com a numèrica una expressió que no ho és.
- **isNaN**: funció que torna true si el paràmetre que li passam no és un valor numèric.
- **Number.isInteger**(x): Torna true si x conté un valor sencer.
- **Infinity**: valor infinit, per exemple el retornat per una divisió per zero.
- **Number.isFinite**(x): true si x té un valor finit.
- **toFixed(posicions):** torna el valor de la variable amb les posicions decimals que li indicam: numero=12.348 numero.toFixed(2) => 12.35

Funcions definides per l'usuari

Per definir una funció hem de seguir la plantilla:

```
function nomDeLaFuncio(paràmetres separats per comes){
  instruccions;
  return valor; //opcional
}
```

Com es pot veure no s'ha d'indicar si la funció torna cap valor o no ni, evidentment, de quin tipus és el valor que torna. Si n'ha de tornar li posam el return i si no, no. El return és la darrera instrucció que s'executa. Dins una funció hi pot haver els que facin falta.

Per definir els paràmetres de la funció, basta posar els seus noms. Per exemple:

```
function suma(primerSumand, segonSumand){
  var resultat=primerSumand+segonSumand;
  return resultat;
}
```

Per cridar una funció basta posar el seu nom, els parèntesis, i si duu arguments, els posam entre els parèntesis.

Javascript **permet la recursivitat**, és a dir, que una funció es cridi a si mateixa, **però no la sobrecàrrega** de funcions, és a dir, l'existència de múltiples funcions amb el mateix nom i distint nombre i/o tipus de paràmetres.