Estàndards de codificació - JavaScript

toc_collapse=0; Taula de continguts

El següent estàndard de codificació està basat en els estàndards de llibreries de javascript com jQuery o YUI. L'objectiu de l'estàndard de la DiBa no es només la llegibilitat, l'homogenitat i la facilitat de lectura del codi, també està pensat per permetre la minificació, agregació i compressió del codi.

1. Compressió del codi

En javascript es molt habitual agregar i minificar tot el codi javascript en els entorns de producció per millorar-ne notablement el rendiment, les estadístiques diuen que la millora del rendiment de la minificació és d'un 40%-60% sense memòria cau i d'un 20% si es carregada des de la memòria cau del navegador.

Hi ha moltes eines que et permeten comprimir el codi automàticament però no tot el codi pot ser minificat automàticament, és per aquest motiu que cal seguir l'estàndard de codificació per permetre la minificació i compressió automàtica del codi, sobretot en alguns punts vitals com el correcte ús del punt i coma i les claus o parèntesis.

Com a eina automatitzada nosaltres utilitzem principalment YUI compressor (http://developer.yahoo.com/yui/compressor/) [1] perquè és la llibreria que s'utilitza per comprimir el jQuery (llibreria de javascript recomanada en la nostra plataforma) i AlloyUI (llibreria de javascript del Liferay).

2. Sagnia

Per identar el codi utilitza dos espais en blanc, no tabuladors.

3. Concatenació de cadenes

Utilitzeu sempre un espai entre el + i les parts concatenades per millorar la llegibilitat.

```
var string = 'Foo' + bar;
string = bar + 'foo';
string = bar() + 'foo';
string = 'foo' + 'bar';
```

Quan s'utilitza l'operador d'assignació de concatenació ('+='), utilitzar un espai a cada costat:

```
var string += 'Foo';
string += bar;
string += baz();
```

4. CamelCasing

A diferència de les variables i funcions definides en PHP, les variables i funcions Javascript han de ser lowerCamelCased. La primera lletra de cada variable o funció ha d'estar en minúscules, mentre que la primera lletra de paraules que segueixen han de ser capitalitzada. No ha d'haver-hi subguions entre les paraules. Per exemple: laMevaFuncio.

5. Punt i coma

JavaScript permet utilitzar qualsevol expressió com una instrucció i utilitza el punt i coma de forma opcional per marcar el final de la instrucció. No obstant això, la no utilització del punt i coma en aquests casos pot emmascarar alguns errors i també impedirà l'agregació del javascript. Per tant, totes les declaracions han de tancar-se amb un ; de forma general, tot i que podem establir excepcions en els següents casos:

for, if, function, switch, try, while

Sempre que la declaració de la funció no s'assigni a una variable:

```
var laMevaFuncio = function (context) {
  // Codi...
};
```

i no s'utilitzi el while amb un do:

```
do {
  // Codi...
} while (condition);
```

Llavors també caldrà el punt i coma final.

Nota: També pot adoptar-se una postura més preventiva i utilitzar el punt i coma com a tancament de qualsevol declaració per evitar tenir presents les excepcions.

6. Estructures de control

Aquestes inclouen if, for, while, switch, etc.

6.1. if

```
if (condition1 || condition2) {
  action1();
}
else if (condition3 && condition4) {
  action2();
}
else {
  defaultAction();
}
```

Les sentències de control han de tenir un espai entre la paraula clau de control i el parèntesi d'obertura, per distingir-les de les crides a funcions.

Es recomana utilitzar sempre claus, fins i tot en situacions en les que són tècnicament opcionals. Tenir-les augmenta la llegibilitat i disminueix la probabilitat d'errors de lògica quan s'estan introduint noves línies al codi.

6.2. switch

```
switch (condition) {
  case 1:
    action1();
    break;

  case 2:
    action2();
    break;

  default:
    defaultAction();
}
```

6.3. try

```
try {
  // Codi...
}
catch (error) {
  // Control d'errors...
}
finally {
  // Codi de tancament...
}
```

6.4. for in

El "for in" et permet recórrer els noms de totes les propietats d'un objecte. Per desgràcia, tots els membres heretats del prototip també s'inclouran a la llista. Això té el desavantatge que si t'interessen les dades també

t'apareixeran tots els mètodes disponibles a la llista. Per evitar això cada "for in" ha de ser embolicat en una sentència if que faci de filtre. Es pot seleccionar un determinat tipus o rang de valors o es poden excloure a les funcions o es poden excloure les propietats del prototip. Per exemple:

```
for (var variable in object) if (filter) {
  // Codi...
}
```

També pots utilitzar el mètode hasOwnProperty per distingir els autèntics membres d'un objecte:

```
for (var variable in object) if (object.hasOwnProperty(variable)) {
  // Codi...
}
```

7. Funcions i mètodes

Les funcions i mètodes s'han d'anomenar utilitzant lowerCamelCase. També han de ser cridades sense espais entre el nom de la funció i el primer parèntesis i amb un punt i coma al final:

```
foobar = foo(bar, baz, quux);
```

Si una funció és anònima, ha d'haver un espai entre la paraula funció i el parèntesi de l'esquerra. Si l'espai s'omet, pot semblar que "function" és el nom de la funció.

```
div.onclick = function (e) {
  return false;
};
```

Tota funció ha de tornar sempre un valor i els valors per defecte han d'anar al final de la llista d'arguments. Intentem que totes les funcions tornin sempre un valor significatiu.

```
function missatgeDivertit(camp) {
  alert("Aquesta funció fa aparèixer un missatge divertit.");
  return camp;
}
```

8. Variables

Totes les variables han de ser declarades amb var abans del seu ús, i només s'han de declarar una sola vegada.

Les variables no s'han de definir en l'àmbit global, intentem definir-les en un àmbit local de la funció costi el que costi. Totes les variables han de ser declarades al principi d'una funció.

9. Arrays

Les matrius s'han de construïr amb un espai a cada cantó de l'assignació i una coma+espai per separar cada element :

```
unArray = ['hola', 'món'];
```

Si la línia s'estén més de 80 caràcters, cada element s'ha de trencar en la seva pròpia línia, i amb sangria d'un nivell (dos espais).

10. Ubicació del codi

El codi JavaScript no ha de ser incorporat en el codi HTML si és possible, ja que contribueix significativament al pes de la pàgina sense poder utilitzar la memòria cau, ni la compressió, ni la càrrega en paral·lel.

11. with

La comanda with te per finalitat oferir una drecera per accedir als membres d'objectes imbrincats, per exemple podem fer el següent (no recomanat):

```
with (foo.bar.foobar) {
  var abc = true;
  var xyz = true;
}
```

No obstant això, és impossible saber mirant només el codi anterior que estem modificant realment. Estem modificant la funció foo.bar.fooba [2] o estem modificar unes variables globals abc i xyz definides com a globals en algun altre lloc? Per tant, per facilitar la legibilitat es recomana utilitzar la versió més explícita:

```
foo.bar.foobar.abc = true;
foo.bar.foobar.xyz = true;
```

o si realment voleu utilitzar una abreviatura, utilitzem la següent forma:

```
var o = foo.bar.foobar;
o.abc = true;
o.xyz = true;
```

12. Operadors

Cal tenir molt persent que els operadors == i != fan la conversió de tipus abans de comparar amb JavaScript. Això pot emmascarar molts errors i provoca que comparacions com aquesta tornin un true: ' \t\r\n' == 0

Així que quan comparis utilitza els operadors === i !== que no fan cap conversió de tipus.

13. Codi inabastable

Per prevenir el codi inabastable un return, un break, un continue o un throw ha de ser seguit per un } o un case o un default .

14. Expresions literals

Sempre que puguis utilitza les experions literals enlloc de l'operador new, per exemple:

Utilitza {} enlloc de Object()

Eviteu utilitzar les formes new Number, new String, new Boolean

Sobre l'ús de les formes new Number, new String, new Boolean hi ha molts problems associats i moltes sutileses que no sempre es tenen presents i fan que no es recomani el seu ús. Podria semblar que és el mateix fer varNum= 0; que fer varNum = new Number(0); però no és així! Per això es recomana no utilitzar la segona forma, pel seu comportament diferenciat que pot induir a errors. Veïem-ne un exemple:

```
var literalNum = 0;
var objectNum = new Number(0);
if (literalNum) { } // false perquè 0 is un valor fals, no serà executat.
if (objectNum) { }  // true perquè objectNum existeix com a objecte, serà executat.
if (objectNum.valueOf()) { } // false perquè el valor de objectNum és 0, no serà executat.
```

15. No utilitzis eval

eval() és el mal en el món del javascript, exigeix al navegador crear un entorn de programació totalment nou (com crear una nova pàgina web), la importació de totes les variables de l'àmbit actual, executar l'script, llençar el garbage collector, etc... a més, el codi no pot desar-se en memòria cau i dificulta la depuració.

16. Typeof

Quan s'utilitza un typeof, no utilitzeu el parèntesis. El següent és l'estàndard de codificació correcta:

```
if (typeof myVariable == 'string') {
  // Codi...
}
```

17. Modificar el DOM

Si afegiu elements HTML a la DOM no utilitzeu document.createElement(). Per compatibilitat entre navegadors es recomana utilitzar l'equivalent de jQuery.

URL d'origen: https://comunitatdstsc.diba.cat/wiki/estandards-de-codificacio-javascript

Enllaços:
[1] http://developer.yahoo.com/yui/compressor/)
[2] http://foo.bar.fooba