Treviso JS Meetup:

Cattive abitudini e Linee guida

Anti-patterns

(ovvero situazioni da evitare)

Dichiarare variabili senza il "var"

Da evitare!

```
name="Paperino";
```

```
function stampaPaperino() {
  name ="Paperino";
  console.log(name);
}
```

Dichiarando variabili senza il var, automaticamente andiamo a definirle come Globali.

Corretto!

```
var name="Paperino";
```

```
function stampaPaperino() {
  var name ="Paperino";
  console.log(name);
}
```

Utilizzando il "var", lo scope (ambiente dove vive la variabile) diventa quello della funzione in cui è definita.

```
var a = b = 0;
```

Da evitare!

In questo caso l'interprete riscrive il codice in questo modo:

```
b = 0;
var a = b;
```

quindi "b" è globale.

Corretto!

```
var a = 0,
    b = 0;
```

Questo pattern prevede di dichiarare tutte le variabili utilizzando un solo "var".

Sparpagliare variabili all'interno di una funzione

Da evitare!

```
function plutoAiutaPaperino() {
    var pluto = new Pluto();
    pluto.corre();
    pluto.mangia();
    ...
    var paperino = new Paperino();
    pluto.aiuta(paperino);
}
```

"pluto" e "paperino" sono dichiarate in due momenti separati all'interno della funzione.

Corretto!

```
function plutoAiutaPaperino() {
    var pluto = new Pluto(),
        paperino = new Paperino();

    pluto.corre();
    pluto.mangia();
    pluto.aiuta(paperino);
}
```

Più leggibile e evitiamo problemi legati all' "hoisting".

For loops scritti male

Da evitare!

```
for(i = 0; i < list.length; i++) {
    // do something
};
```

"i" in questo caso è globale. Se "list" è un array particolarmente grande il dover recuperare ad ogni iterazione il valore di "list.length" rallenta il codice.

Corretto!

```
for(var i = 0, len = list.length; i < len; i++) {
    // do something
};
```

"list.length" in questo caso viene "cachato" nella variabile "len". (http://jsperf.com/loops)

Eval is evil

Da evitare!

```
eval('var i = "Welcome Hackers!"; alert(i);');
```

"eval" esegue il codice javascript inserito in una stringa.

Usare i costruttori predefiniti al posto della sintassi "literal".

Da evitare!

```
var disney = new Array("Pippo","Topolino","Paperino");
```

Qualsiasi cosa in Javascript è un Object, il linguaggio mette a disposizione dei costruttori che ereditano da Object implementando nuove funzionalità. "new Array()" ad esempio crea un oggetto di tipo Array con delle funzionalità come "pop","push","shift", etc.

Corretto!

```
var disney = ["Pippo","Topolino","Paperino"];
```

Questa sintassi è molto più pulita e condivisa da gran parte dei linguaggi di programmazione.

Inquinamento Globale

E' buona norma evitare di definire variabili o funzioni nello "scope globale" perchè sono accessibili e modificabili in qualsiasi punto dell'applicazione.

C'è il rischio che vengano accidentalmente sovrascritte da altri programmatori o plugin esterni e rendono il codice difficile da mantenere.

Da evitare!

```
<script>
var pluto = new Pluto(),
    plutoHouse = new DogHouse({color:red,material:'wood', size: [100,

function loadPlutoHouse() {
    return pluto.load(house);
}
</script>
```

Tutte le variabili definite nel frammento di codice sopra-riportato sono globali perchè non sono definite all'interno di una funzione.

Corretto!

```
<script>
// IIFE = Immediately-Invoked Function Expression
(function(document, window, undefined){
    var pluto = new Pluto(),
        plutoHouse = new DogHouse({color:red,material:'wood', size: [

    function loadPlutoHouse() {
        return pluto.load(house);
    }
})(document,window);
</script>
```

Usare una funzione che viene definita e subito invocata crea uno "scope isolato". Variabili e funzioni esistono solo all'interno dell' IIFE.

Esempio in jQuery

```
<script>
// IIFE = Immediately-Invoked Function Expression
(function($,document, window){
    //code immediately executed goes here
    $(function(){
        //code executed on document ready goes here
    });
})(jQuery, document,window, undefined);
</script>
```

jQuery Anti-patterns

Evviva i costruttori!

Da evitare!

```
$('button.confirm').on('click', function() {
    // Do it once
    $('.modal').modal();

    // And once more
    $('.modal').addClass('active');

    // And again for good measure
    $('modal').css(...);
});
```

Ogni volta che usiamo $('...') viene creato un nuovo oggetto jQuery parsando il DOM e cercando l'elemento corrispondente. Assolutamente poco performante.

Soluzione 1: Chaining

```
$('button.confirm').on('click', function() {
    $('.modal')
        .modal()
        .addClass('active')
        .css(...);
});
```

jQuery ci permette di "concatenare" più funzioni al medesimo oggetto $ perchè ogni funzione ritorna l'instanza dell'elemento DOM jquerizzato.

Soluzione 2: Caching

```
var $buttonConfirm = $('button.confirm'),
    $modal = $('.modal');
    $background = $('.bg-overlay');

$buttonConfirm.on('click', function() {
    $modal
        .modal()
        .addClass('active')
        .css(...);

    $background.removeClass('hidden');

});
```

Salvare il riferimento all'oggetto jQuery per poi riusarlo nel codice è la soluzione ottimale. In particolare all'interno dei listener evitiamo di dover "attraversare" il DOM ogni volta che si scatena l'evento a cui l'ascoltatore è associato.


L'inferno delle CallBack

Da evitare!

```
$buttonConfirm.on('click', function() {
    $overlay.fadeIn(400, function(){
        $.ajax('/some/modal/content',{},function(data){
            $overlay.html(data).slideUp(300, function() {
                ....
            })
        })
    })
});
```

Annidare callback porta a codice difficile da capire e mantenere.

Un pò meglio

```
$buttonConfirm.on('click', showOverlay);

function showOverlay() {
     $overlay.fadeIn(400, getOverlayDataFromServer);
}

function getOverlayDataFromServer() {
    $.ajax('/some/modal/content',{}, updateOverlayContent);
}

function updateOverlayContent(data) {
    $overlay.html(data).slideUp(300, function() {
        ....
    })
}
```

Portare le funzioni all'esterno aiuta la leggibilità.

Deferred Objects

```
$buttonConfirm.on('click', function(e){
    $.when(showOverlay())
     .then(getOverlayDataFromServer())
     .done(updateOverlayContent(data))
     .fail(function(){ alert("Something went wrong!") });
});

function showOverlay() {
    // $.fn.fadeIn has a promise() method defined so it works like a
    return $overlay.fadeIn(400);
}

function getOverlayDataFromServer() {
    return $.ajax('/some/modal/content');
}
```

Alcuni medoti jQuery ritornano direttamente un Deferred object, è possibile creare anche "azioni" custom che utilizzano $.Deferred.

Linee guida

(Tools per scrivere Javascript pulito)

'use strict';

```
'use strict';
```

```
function myFunction() {
    'use strict';
    //...
}
```

Con questa dicitura abilitiamo la modalità "Strict Mode": per alcuni sbagli (o cattive abitudini) normalmente accettati dall'interprete JS vengono sollevati errori.

Usare un linter JS: JsHint

Può essere usato sia come plugin nell'editor di testo sia all'interno del vostro task manager (Grunt/Gulp).

Esempio

Scegliere una StyleGuide e usare JSCS

Ogni programmatore ha le sue abitudini, giuste o sbagliate che siano.

Scegliere una **styleguide** e condividerla con il team di progetto è fontamentale per avere un codice omogeneo e "stilisticamente" pulito.

- AirBnB
- Google
- jQuery
- Idiomatic.js

JSCS