

# Corso Enaip

## Rimini 24/02/2023

# Chi sono



Giuseppe Trisciuoglio

CTO e Enterprise Architect

Classe '83

Lavoro nel IT dal 2003

I miei contatti sono:

email: [giuseppe@adriasonline.it](mailto:giuseppe@adriasonline.it)

telegram: eurotrip11



## Le 3 regole del gioco

- Webcam accesa
- Mente accesa
- Ascolto attivo

# Agenda

- Defer vs Async
- Conversione (Casting) in Javascript
- Controllo di flusso (if/else e for/while)
- Introduzione alle funzioni

## Async e Defer per caricare in modo efficace JavaScript

Le dipendenze Javascript sono per loro natura **risorse di blocco**, più precisamente **parser blocking**. Parser blocking significa che al momento in cui il parser del browser inizia la lettura del codice HTML ed incontra uno script, **sospende** tutte le operazioni di lettura e rendering per poterne eseguire il codice.

Una pagina HTML ha solitamente più dipendenza JS, queste continue interruzioni possono rallentare **in modo considerevole la velocità caricamento e rendering** della pagina stessa. Se in più lo script è esterno al sito web, il momento di stallo sarà allungato dai tempi di download della risorsa.

## <script async>

- L'attributo async evita di interrompere il rendering dell'HTML durante il download dello <script>, che di fatto avviene in parallelo
- Lo <script> viene eseguito subito dopo il download
- Il parsing della pagina viene messo in pausa soltanto mentre lo <script> viene eseguito

## <script defer>

- L'attributo defer evita di interrompere il rendering dell'HTML durante il download dello <script>, che di fatto avviene in parallelo
- Lo <script> viene eseguito subito dopo il parsing della pagina HTML
- Il parsing della pagina non viene mai messo in pausa

# Async e Defer per caricare in modo efficace JavaScript

`<script>`







`<script async>`



`<script defer>`



-  HTML Parsing
-  HTML Parsing Paused
-  Script Download
-  Script Execution



# Quale attributo è meglio usare?

In genere **si desidera utilizzare async**, ove possibile, come seconda scelta defer ed infine nessun attributo. Ecco alcune regole generali da seguire:

- Se lo `<script>` è modulare e non si basa su altri `<script>`, sarebbe meglio utilizzare `async`
- Se lo `<script>` si basa su altri `<script>` o viene invocato da un altro `<script>`, sarebbe meglio utilizzare `defer`
- Se lo `<script>` è di piccole dimensioni ed è invocato da uno `<script>` asincrono, sarebbe meglio utilizzare uno `<script>` in linea senza attributi posto sopra gli `<script>` asincroni

# Conversione (Casting) in Javascript

Le variabili JavaScript **possono contenere valori di qualsiasi tipo** e cambiare senza problemi il tipo di dato contenuto. Questa flessibilità libera lo sviluppatore dal dover dichiarare esplicitamente il tipo di dato che può contenere una variabile, scaricando però **sull'engine JavaScript** il compito di dover prendere delle **decisioni** in determinate situazioni ed effettuare delle conversioni implicite.

# Conversione (Casting) in Javascript

```
var x = "10";  
var y = "2";  
var z = x * y;  
console.log(z);
```

```
var x = "10";  
var y = x + 2;  
console.log(y);
```

# Conversioni implicite per il tipo Boolean

Tipo	Valore booleano
undefined	<b>false</b>
null	<b>false</b>
numero	<b>false</b> se 0 o NaN, <b>true</b> in tutti gli altri casi
stringa	<b>false</b> se stringa vuota, <b>true</b> in tutti gli altri casi

# Esempio

```
var x = 2;  
var z = x%2 ? "dispari" : "pari";  
console.log(z);
```

# Conversioni implicite per il tipo Numerico

Tipo	Valore numerico
undefined	<b>NaN</b>
null	<b>0</b>
boolean	<b>false</b> se 0, <b>true</b> se 1
stringa	<b>intero, decimale, zero o NaN</b> in base alla specifica stringa

# Esempio

```
var x = "10";  
var y = "10.05";  
var z = "123abc";  
var z = "uno" * 2;  
console.log(z);
```

# Conversioni implicite per il tipo Stringa

Tipo	Valore stringa
undefined	<b>“undefined”</b>
null	<b>“null”</b>
boolean	<b>“false”</b> se false, <b>“true”</b> se true
numerico	<b>“NaN”</b> se NaN, <b>“Infinity”</b> se Infinity la stringa che rappresenta il numero negli altri casi



# Operatori polimorfi

Di fronte ad operatori polimorfi, cioè operatori che prevedono operandi di tipo diverso, la situazione è un po' **più complessa**. Esempi di operatori polimorfi sono la somma e la concatenazione di stringhe, dato che utilizzano lo stesso simbolo +, oppure gli operatori relazionali ( $\leq$ ,  $\geq$ ,  $<$ ,  $>$ ).

In presenza di espressioni come quelle che abbiamo evidenziato all'inizio di questa lezione, **l'engine JavaScript** si trova a dover fare una scelta ben precisa per stabilire verso quale tipo di dato convertire gli operandi.

# Operatori polimorfi

```
var z = "12" + 2;  
console.log(z);  
var x = true + null;  
console.log(x);
```

# Operatori polimorfi

```
var x = y || "";  
var z = h || 1;  
"12" > 10  
true > null  
"true" > "null"
```

# Operatori di confronto

Per quanto riguarda gli operatori di confronto `==` e `!=` viene presa in considerazione la seguente regola: *se entrambi gli operatori sono stringhe allora viene effettuato un confronto tra stringhe, altrimenti si esegue un confronto tra numeri*; unica eccezione è **`null == undefined`** che è vera per definizione.

Un discorso a parte vale per gli operatori di **uguaglianza e disuguaglianza stretta** (**`===` e `!==`**). Questi operatori confrontano gli operandi senza effettuare alcuna conversione. Quindi due espressioni vengono considerate uguali soltanto **se sono dello stesso tipo ed rappresentano effettivamente lo stesso valore**.

# Evitare le conversioni implicite

Le conversioni implicite di JavaScript sono **spesso fonte di bug**, in particolar modo quando abbiamo a che fare con espressioni complesse. Il modo migliore per non incappare in questi potenziali bug è evitare le conversioni implicite. Ad esempio, è preferibile confrontare espressioni e valori dello stesso tipo ed utilizzare **sempre gli operatori di uguaglianza e disuguaglianza stretta** invece di quelli ordinari, in modo da essere sicuri di quello che si sta confrontando.



# Funzioni parseInt e parseFloat

```
parseInt("12")    // 12
parseInt("12abc")  // 12
parseInt("a12bc")  // NaN
parseInt("12.5")   // 12
parseInt("12", 8)  // il valore di 12 nel sistema di numerazione ottale (base 8), cioè 10

parseFloat("12")   //12
parseFloat("12.5") //12.5
```

# L'Operatore typeof

In qualsiasi momento, comunque, **si può verificare il tipo della variabile** (o di un valore oppure di un letterale) tramite l'operatore **typeof**.

```
var prova = new Function();  
var numero = 1;  
var carattere = "Salve";  
console.log(typeof prova);      // ritorna "function"  
console.log(typeof numero);     // ritorna "number"  
console.log(typeof carattere);  // ritorna "string"
```

# Controllo di flusso



# Istruzioni condizionali - IF

Le istruzioni condizionali sono una categoria di istruzioni che consentono di eseguire **blocchi di codice** alternativi in base ad una **condizione**.

Possiamo trovare l'istruzione if in tre forme:

- if semplice;
- if con alternativa (if..else);
- if a cascata (if..else if...else).

# Istruzioni condizionali - IF

```
if (condizione) {  
    // istruzioni  
}  
  
if (condizione) {  
    // istruzioni1  
} else {  
    // istruzioni2  
}  
  
if (condizione1) {  
    // istruzioni1  
} else if (condizione2) {  
    // istruzioni2  
} else {  
    // istruzioni3  
}
```

# Istruzioni condizionali - Switch

Quando siamo di fronte a diverse alternative, anche un **if a cascata** può risultare **difficile da leggere**. In questi casi possiamo ricorrere all'istruzione switch.

# Istruzioni condizionali - Switch

```
switch (espressione) {  
    case espressione1:  
        istruzioni1;  
        break;  
    case espressione2:  
        istruzioni2;  
        break;  
    // ...  
    default:  
        istruzioni4;  
        break;  
}
```

# Istruzioni iterative

Un'altra categoria di istruzioni comunemente usata nei linguaggi di programmazione è rappresentata dalle iterazioni o cicli. JavaScript prevede le classiche istruzioni di iterazione come **while** e **for**.

## Istruzioni iterative: While e Do/While

```
while (condizione) {  
    // istruzioni  
}
```

```
do {  
    // istruzioni  
}
```

```
while (condizione)
```

## Istruzioni iterative: For

```
var quantita = [1,2,3,4,5];  
for (var i=0; i< quantita.length; i++){  
    console.log(quantita[i]);  
}
```

## Istruzioni iterative: For

```
var quantita = [1, 2, 3, 4, 5];  
var totale = 0;  
var indice;  
for (indice in quantita) {  
    totale = totale + quantita[indice];  
}  
  
var valore;  
for (valore of quantita) {  
    totale = totale + valore;  
}
```



# Continue e Break

```
var x=0;
while (true) {
    console.log(x);
    // condizione di uscita
    if (x > 100) break;
    x++;
}

var x = 0;
while (x < 10) {
    x++;
    if (x > 3) continue;
    // se x è maggiore di 3, questa istruzione non viene più eseguita
    console.log(x);
}
```

# Esercitazione 1

Scrivere un blocco di codice Javascript che stampi a console i primi dieci interi pari compresi tra 20 e 0, partendo da 20. **Importante non bisogna usare delle costanti, ma solo le istruzioni if e for.**

Per eseguire il codice potete utilizzare il tool disponibile online all'indirizzo:

<https://www.programiz.com/javascript/online-compiler/>



# Esercitazione 2

Scrivere un blocco di codice Javascript che stampi a console i primi dieci numeri interi, escluso lo zero, in lingua italiana.

Per eseguire il codice potete utilizzare il tool disponibile online all'indirizzo:

<https://www.programiz.com/javascript/online-compiler/>



# Esercitazione 3

Scrivere un blocco di codice Javascript che dato un numero stampi la tabellina corrispondente.

Per eseguire il codice potete utilizzare il tool disponibile online all'indirizzo:

<https://www.programiz.com/javascript/online-compiler/>



# Esercitazione 4

Scrivi un blocco di codice che dato un array di numeri, calcoli la media dei valori e restituisca in output la media e tutti i valori minori della media.

Esempio:

Input:  $a = [3, 5, 10, 2, 8]$

Output: media = 5.6, valori minori =  $[3, 5, 2]$

Per eseguire il codice potete utilizzare il tool disponibile online all'indirizzo:

<https://www.programiz.com/javascript/online-compiler/>



# Introduzione alle Funzioni

# Funzioni in Javascript

*“Una funzione è un insieme di istruzioni racchiuse in un blocco di codice, che può essere contraddistinto da un nome, può accettare argomenti o parametri di ingresso e restituire valori.”*

A questo punto, secondo questa prima definizione, l'utilizzo di una funzione all'interno di uno script prevede due fasi:

- una fase di definizione o dichiarazione della funzione in cui si assegna un nome ad un blocco di codice;
- una fase di invocazione o chiamata in cui il blocco di codice viene eseguito.

# Funzioni in Javascript

- **function** nome(parametri) { blocco istruzioni }
- **var** nome = **function**(parametri) { blocco istruzioni }
- **function**(parametri) { blocco istruzioni }
- **var** nome = **new Function**("par1", "body")
- Le funzioni possono essere "annidate"
- Le funzioni sono "dati": possono essere assegnate



# Funzioni in Javascript - L'istruzione return

Nel corpo della funzione può essere presente l'**istruzione return** che consente di terminare e restituire un valore al codice che l'ha chiamata. Questo ci consente di **assegnare ad una variabile il valore restituito** da una funzione o utilizzare una funzione all'interno di una espressione.

# Funzioni in Javascript - Esempi

```
function somma() {  
    var z = 10 + 1;  
    return z;  
}  
  
var risultato = somma();  
  
function somma(x, y) {  
    var z = x + y;  
    return z;  
}  
  
function somma() {  
    var z = 0;  
    var i;  
    for (i in arguments) {  
        z = z + arguments[i];  
    }  
    return z;  
}
```

# Funzioni in Javascript - Esempi

```
function somma(x = 0, y = 0) {  
    var z = x + y;  
    return z;  
}
```

```
function mostraMessaggio(messaggio) {  
    alert(messaggio);  
}
```

```
function mostraMessaggio(messaggio) {  
    alert(messaggio);  
    return;  
}
```

# Variabili e ambito (scope)

- Le variabili vengono dichiarate con l'istruzione var
- l'ambito (scope) di una variabile definita in un qualsiasi punto del codice è globale
- una variabile definita in una funzione è locale
- una variabile locale con lo stesso nome di una globale “nasconde” quella globale

## Variabili e ambito (scope)

```
var x = 10;  
var y;  
function incrementa(){  
    var z = 5;  
    x = x + z;  
}  
incrementa();  
y = x + 1;
```

# Variabili e ambito (scope)

```
var scope = "global";
```

```
function test() { var scope = "local"; console.log(scope); }
```

```
test(); console.log(scope);
```

Cosa succede se ometto l'istruzione `var` all'interno della variabile?

# HOISTING

La dichiarazione di una variabile con l'istruzione var viene sempre “innalzata” all'inizio dello scope corrente

```
var scope = "global";  
function test() {  
    console.log(scope); // undefined  
    var scope = "local";  
    console.log(scope); // "local"  
}
```

# Dichiarazione implicita

Se non dichiariamo una variabile con l'istruzione **var**, la **dichiarazione è implicita** e farà sempre riferimento allo scope globale.

```
function test() {  
  scope = "local";  
  console.log(scope); // "local"  
}  
  
test();  
console.log(scope); // "local"
```



# Strict mode

Se non dichiariamo una variabile con l'istruzione **var**, la **dichiarazione è implicita** e farà sempre riferimento allo scope globale.

**"use strict";**

Aggiungendo questa riga di codice all'inizio dello script o di una funzione modifica il comportamento di Javascript in modo che segnali come errore:

- la creazione implicita di variabili globali
- l'assegnazione a variabili non modificabili o la cancellazione di oggetti non cancellabili
- la duplicazione di proprietà o di parametri di funzione.

# Funzioni ricorsive

- Una funzione è definita ricorsivamente quando nella sua definizione compare un riferimento a se stessa.
- La ricorsione consiste nella possibilità di definire una funzione in termini di se stessa.
- È basata sul principio di induzione matematica.

# Fattoriale

In matematica, si definisce fattoriale di un numero naturale, il prodotto dei numeri interi positivi minori o uguali a tale numero.

Esempio:  $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$ .

```
function fattoriale(num) {  
  // Se il numero e' minore di 0 lo scarta  
  if (num < 0) {  
    return -1;  
  }  
  // Se il numero e' uguale a 0 il suo fattoriale e' 1 per definizione  
  else if (num == 0) {  
    return 1;  
  }  
  // In tutti gli altri casi richiama  
  else {  
    return (num * fattoriale(num - 1));  
  }  
}
```

# Esercitazione 1

Scrivi una funzione che prenda in input un numero e restituisca TRUE se è un numero primo, FALSE altrimenti.

Scrivi una seconda funzione, che prenda in input un numero N e stampi i primi N numeri primi.

Esempio:

Input:  $n = 5$

Output: true 2 3 5 7 11

Per eseguire il codice potete utilizzare il tool disponibile online all'indirizzo:

<https://www.programiz.com/javascript/online-compiler/>



# Esercitazione 2

Calcolare la somma dei primi N interi.

Esempio:

Input: 5

Output: 15

Per eseguire il codice potete utilizzare il tool disponibile online all'indirizzo:

<https://www.programiz.com/javascript/online-compiler/>



# Esercitazione 3

Scrivi una funzione che prenda in input una stringa e restituisca TRUE se è **palindroma**, FALSE se non lo è. Nel controllo scarta gli spazi e i segni di punteggiatura.

Esempio:

Input: i topi non avevano nipoti

Output: true

Consigli:

Puoi eliminare spazi e segni di punteggiatura usando le espressioni regolari o il metodo del tipo stringa chiamato replace, in questo modo: `str.replace(/\W/g, "")`.

Per eseguire il codice potete utilizzare il tool disponibile online all'indirizzo: <https://www.programiz.com/javascript/online-compiler/>



# Esercitazione 4

Scrivi una funzione che calcoli la vicinanza tra tre numeri: A, B e N, e restituisca:

- 0 Se A e B sono equidistanti da N
- 1 Se B è più vicino a N rispetto ad A
- -1 Se A è più vicino a N rispetto a B

Per eseguire il codice potete utilizzare il tool disponibile online all'indirizzo:

<https://www.programiz.com/javascript/online-compiler/>





Domande?

