**JAVASCRIPT**

Linguaggio di programmazione he permette di rendere i siti web interattivi e dinamici. Grazie a javascrpt possiamo cambiare il contenuto di una pagina, creare cookie, validare form, aggiungere animazioni, reagire alle azioni degli utenti, ecc..

I browser supportano 3 diversi modi per aggiungere javascript al nostro file html:

* In line
* Embedded (all’interno di un tag script che possiamo inserire alla fine del tag body o all’interno del tag head.
* Tramite foglio esterno (con un tag script e con un attributo src con il pt per il file main.js --> (per il SOC questo è il miglior modo)

Il browser può eseguire del linguaggio javascript tramite la console raggiungibile tramite inspect. Ciò serve per controllare il nostro codice e fare debugging. Per scrivere qualcosa in console usiamo il comando **console.log** che stamperà al suo interno qualcosa.

TIPI TI DATO

**Primitivi: leggeri e immutabili** (possono conservare valori costituiti da un solo valore), per capire di che tipo è un dato possiamo utilizzare l’operatore typeof:

* + Number-->JavaScript non distingue le differenze tra i numeri (positivi, negativi, interi, ecc..). Per riportare a numero da stringa si utilizza La funzione **Number** o **parseFloat** (parseFloat ignora i caratteri non validi, es: si inserisce 22df parsefloat ignora “df” stampando 22, mentre Number darà NaN (Not a number).
  + String--> sono contenuti testuali assegnabili a una variabile. Le stringhe le possiamo dichiarare con ‘ ‘, “ ”, ` ` (backtick o apici greci, alt+ 96, che sono importanti per la **string interpolation** -->processo di evaluation di una stringa contenente una o più variabili, ` Il valore di a è ${a} `). Mentre non volessimo usare questo metodo si può fare la **concatenazione di stringhe** utilizzando l’operatore **+**, “Il valore di a è” + ” ”+ a (“ “ è lo spazio). **.toLowerCase** trasforma una stringa in minuscolo, **.toUpperCase** in maiuscolo. **.replace(“vacchionome”, “nuovonome”)** sostituisce il nome della stringa con un altro. **.split** divide la stringa in base al separatore che noi gli passiamo, restituendoci un array.
  + boolean --> è un tipo di dato che può assumere come valore solo **true** o **false**. Con la funzione Boolean possiamo verificare se un espressione o variabile è vera (es:console.log(Boolean( 10 > 3 )) //true). In javascript tutto ciò che ha un valore è **truthy**, mentre tutto ciò che ne è privo è **falsy**. In javascript un valore truthy è un valore che viene considerato true quando viene incontarto in un contesto booleano.
  + undefined --> è il valore che viene assegnato a una variabile quando è stata dichiarata ma non ancora inizializzata.
  + null --> rappresenta un variabile vuota al cui interno però dovrebbe esserci un oggetto, si inizializza con valore null. Però in questo modo ci da nella console valore object. Quindi null è un dato di tipo primitivo ma di tipo object: null è una primitiva fondamentale.
  + Symbol --> sono dei token che servono come identificativi univoci e l’unico modo per crearli è la funzione Symbol(). Ogni vlta che creiamo un simbolo, javascript crea un valore unico che rende la loro unicità sempre garantita.

**Strutturali: pesanti e mutabili**

* + Object
  + array

**Type cohercion** --> può essere di 2 tipi: implicita o esplicita. *Esplicita* quando siamo noi ad utilizzare dei metodi per cambiare un tipo di dato in un altro (es: **.toString** per trasformare da numero a stringa), *implicita* quando javascript internamente converte un tipo di dato in un altro.

VARIABILE

Ogni linguaggio di programmazione utilizza il concetto di **variabile**, ovvero come una sorta di cassetto con un nome che conservano informazioni o dati. Una variabile è composta da un nome e da un valore, ovviamente essendo variabili il valore può cambiare nel corso del tempo, infatti esse hanno anche uno spazio temporale.

DICHIARAZIONE VARIABILE

Per *dichiarare* una variabile in javascript si possono usare due keyword: **let** e **var**. Var ci permette di dichiarare una variabile con lo stesso nome senza darci errori, quindi ci permette di sovrascriverla, mentre let no. È meglio usare let, perché è sia più efficiente sia ci permette di avere una sicurezza nel caso in cui in un codice molto lung dichiariamo una variabile con lo stesso nome senza accorgercene.

Ci sono delle convenzioni sui nomi per la dichiarazione di variabili: il primo carattere deve essere una lettera, un \_ o un $. Dopo si può usare qualsiasi carattere o simbolo, ad eccezione dello spazio. I nomi delle variabili sono case sensitive.

INIZIALIZZARE LA VARIABILE

Si può assegnarle un valore iniziale: un numero, una stringa o un arrey (struttura di dati formata da più componenti ordinate dello stesso tipo). Dopo aver inizializzato una nuova variabile, in javascript, è sempre necessario dargli un nuovo valore o addirittura cambiarne la tipologia. Si può decidere di fare decidere all’utente il valore della variabile con il comando **prompt (“”)**.

COSTANTI

Sono essenzialmente delle variabili ma il loro valore resta costante dopo l’inizializzazione., che avviene con la keyword **const**. Le regole sono le stesse delle variabili ma è convenzione dichiararle in maiuscolo. Le costanti devono essere dichiarate e inizializzate nello stesso momento.

OPERATORI

In javascript sono presenti molti operatori aritmetici che permettono di manipolare i numeri.

Operazioni aritmetiche di base:

* **Addizione**--> console.log(a + b)
* **Sottrazione**--> console.log(a - b)
* **Divisione**-->console.log(a / b)
* **Moltiplicazion**e--> console.log(a \* b)

Operazioni avanzate:

* **Potenza**--> console.log(a \*\* b)
* **Operazione modulo** (ci restituisce il resto di una divisione, lo utilizziamo per capire se un numero è pari o dispari o se uno è divisibile per un altro)--> console.log(b % a)
* **Operatori di aumento e decremento unitario**--> aumentano o diminuiscono di un solo valore il vlaore della varibile. **--** **++**
* **Operatori di incremento e decremento finito**-->let num = 4;

incremento: num += 2 // 6 num \*= 2 //12 decremento: num -= 2 //10 num /= 2 //5 (è come sscrivere num \* 3 //12)

* **Operatori di confronto**-->  **==** (**uguaglianza semplice**), **===** (**strettamente uguale**--> oltre al valore controlla anche il tipo, quindi non permette la type cohercion), **!=** (**diverso**), **!==** (**strettamente diverso**), **>** ,**<**, ,**>=**, **<=**
* **Operatori logici o booleani** --> **&& (and)**, **|| (or)**, **!** (**bang operato**r:dà il contrario, come il **not**)
* **typeof**--> serve a capire il tipo di dato

CONDIZIONI

Ci servono per eseguire un blocco di codice solo se una determinata condizione è verificata.

Keyword **if** seguita dalle parentesi tonde con dentro la condizione, dopo di che si ha un blocco di codice compreso tra parentesi graffe.

If (condizione) {

Console.log (‘eseguo il codice’);

}

La condizione if può essere seguita da **else** o **else if**

In un susseguirsi di else if, appena javascript trova la prima condizione vera non controlla più le successive.

**Switch** --> serve per condensare una serie di if else

switch(variabile) {

**case** ‘valore stringa’:

console.log(‘rosso’);

**break**;

**case** (...

...

...

...

**default** :

Console.log (‘ testo ‘);}

CICLI

Sono dei costrutti che ci permettono di ripetere un determinato blocco di codice n. volte.

**For**-->costrutto che ci permette di ripetere il flusso di codice un numero determinato di volte.

si dichiara e inizializza la variabile, condizione, comportamento della variabile

Let num = 10;

for (let i = 0; i < num; i++) { //stampare i numeri da 1 a 9

Console.log(i);

}

**While**--> si differenzia dal for perché non abbiamo la certezza di quante volte il nostro ciclo debba venir eseguito. (fai questo codice finché non si avvera la condizione)

While(condition) {

Flusso di codice --> mentre la condizione è vera fai qualcosa

}

**Do while**-->si differenzia da while perché esegue il codice almeno una volta, anche se la condizione non è verificata

Do {

console.log(condizione);

Condizione = Math.random();

} while (condizione > 0.5);

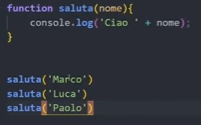
FUNZIONI

Ci servono per incapsulare un blocco di codice che utilizzeremo tante volte all’interno del nostro programma.

DICHAIRAZIONE FUNZIONE

**function** nome.funzione(parametri){

Console.log(“”)

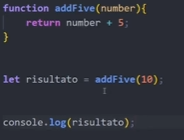
}

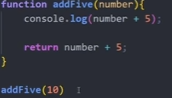
Per renderla generica nei parametri si può inserire un nome, così che una volta richiamata gli si può assegnare il valore che preferiamo.

Si possono ovviamente inserire più parametri, separati dalla virgola. es: (nome , cognome)

RICHIAMARE LA FUNZIONE

nome.funzione ();

**Return**--> istruzione che ci permette di tornare a un valore che è stato valorizzato all’interno della funzione.

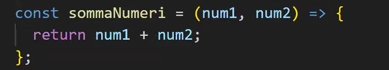


ARROW FUNCTION (di solito si dichiarano con const)

Const utente = () => {

}

La freccia possiede un return implicito.



**=**



ARRAY

Tipo di dato strutturale.

Lista ordinata di elementi

let array = [ 1, “ciao”, null, NaN ];

Console.log(array[2]);

Dato che il linguaggio di Javascript è 0 based, quindi il 2 si riferisce nell’esempio a null.

Per sovrascrivere un elemento della lista--> array[2] = “pollo”;

**.lenght** si riferisce a quanti elementi ci sono nell’array (lunghezza dell’array)

**.unshift** aggiungi all’inizio un elemento

**.push** ti aggiunge alla fine un elemento alla lista che decidi

**.shift** elimina il primo elemento

**.pop** elimina l’ultimo elemento

**.join("")** riunisce i caratteri in una nuova stringa

**.reverse()** inverte l’ordine degli elementi dell’array

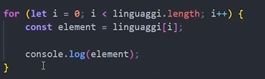
Pop e shift oltre a eliminare, restituiscono l’elemento tolto.

Se non mi serve salvare i valori in una variabile si può scrivere solo la funzione.

Per concatenare un array con un altro array si utilizza **.concat.**



//aggiungi questi due elementi



Per ciclare un array si utilizza il ciclo *for*.



Un altro modo per ciclare è il metodo .**forEach**