================================================================================================

**Tema:** JavaScript

**Modulo:** 02

**Aula:** Online dal vivo

**Giorni:** 26/05/2025 a 30/05/2025

**Professor:** Marika Di Blasio

================================================================================================

================================================================================================

**JavaScript**

Creato in 1995 da Brendan Eich presso Netscape Communications. Originariamente chiamato Mocha, poi LiveScript, infine JavaScript, era progettato per rendere le pagine web più interattive.

================================================================================================

**PRIMO ASSAGGIO DI CODICE**

Il console.log() è una funzione per mostrare messaggi nella console del browser (strumenti per sviluppatori)

“Hello, World!” è una stringa, un testo da visualizzare.

Ogni istruzione JavaScript termina spesso con un punto e virgola ( ; ) anche se non sempre obbligatorio.

================================================================================================

**VARIABILI**

Contenitori per memorizzare dati. Puoi pensare a loro come etichetta che dai a un valore, che è utili per riutilizzare e modificare i dati nel tuo codice. Utili per riutilizzare e modificare i dati nel tuo codice. In JavaScript usiamo parole chiave per dichiarare le variabili:

* let: Per variabili il cui valore può cambiare;
* const: Per variabili il cui valore non dovrebbe cambiare (constanti);
* var: Dobbiamo evitare se possibile, ha un comportamento più complesso.

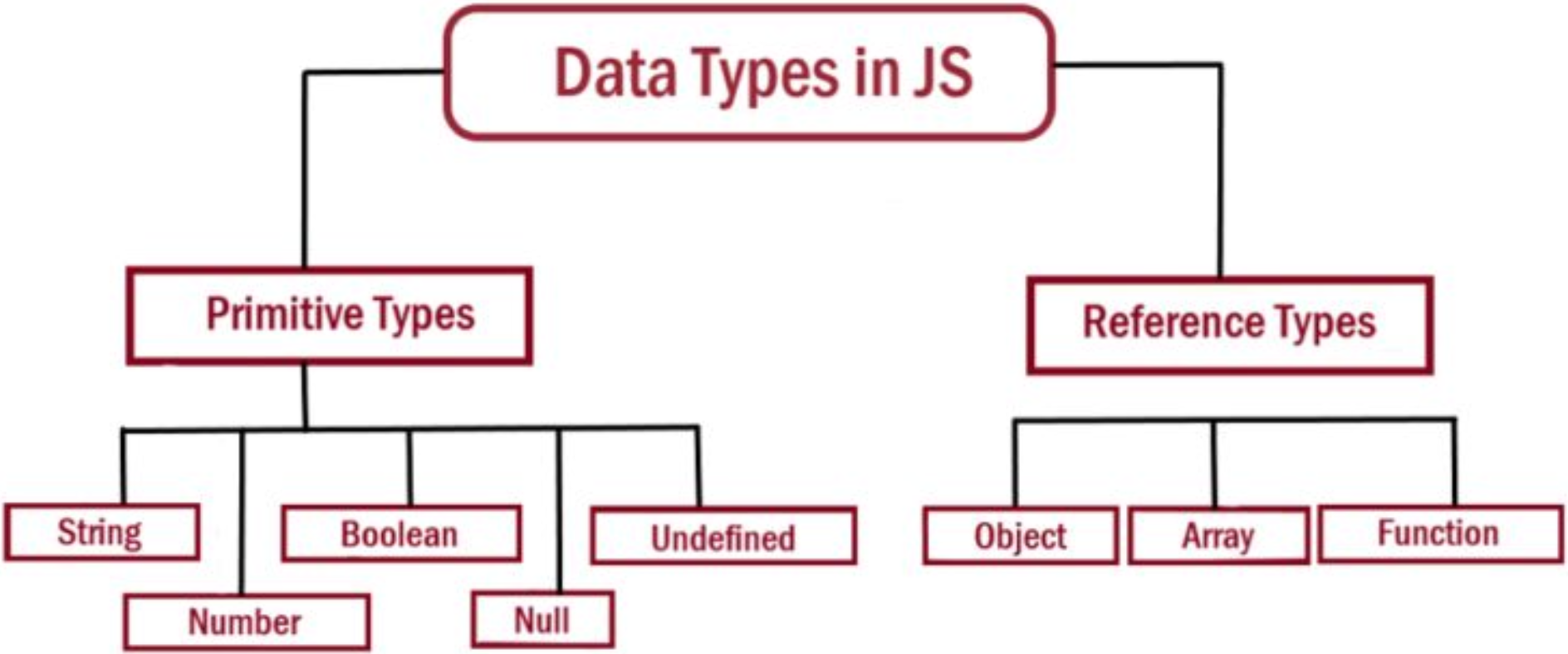
Usiamo l’operatore di assegnazione ( = ) per dare un valore a una variabili.

================================================================================================

**TIPI DI DATI FONDAMENTALI**

Abbiamo diversi tipi di dati:

* Stringa (String): Sequenza di caratteri (testo) racchiusa tra virgolette singole ( ‘ ) o doppie ( “ );
* Numero (Number): Rappresenta valori numerici (interi o decimali);
* Booleano (Boolean): Rappresenta um valore di verità: True / False;
* null: Rappresenta l’assenza intenzionale di un valore
* undefinied: Indica che una variabile è stata dichiarata ma non ha ancora ricevuto un valore;

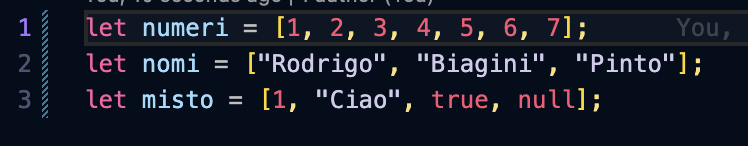


Base dei tipi di variabile.

================================================================================================

**ARRAY**

Una array è una lista ordinata di valori, dove ogni elemento ha un indice, che inizia da 0, può contenere elementi di qualsiasi tipo, anche altre array o oggetti, e si definiscono usando le parentesi quadre []. Se accede agli elementi utilizzando l’indice tra parentese quadri.



Dopo, abbiamo gli oggetti.

================================================================================================

**OGGETTI**

Un oggetto è una collezione di proprietà, dove ogni proprietà ha un nome (chiave) e un valore, dove le chiave sono generalmente stringhe (o simboli) ed i valori possono essere di qualsiasi tipo (primitivo o non primitivo), che si definiscono usando le parentese graffe { }

Se accede alle proprietà usando la notazione a punto ( . ) o la notazione a parentese quadre ( [ ] ).



La notazione a parentesi quadre è utile quando il nome della proprietà è memorizzato in una variabile o contiene spazi o caratteri speciali.

================================================================================================

**FUNZIONI**

Anche le funzioni in JavaScript sono considerate oggetti di “prima classe”, rappresentando blocchi di codice che possono essere definiti e poi invocati (eseguiti) più volte. Possono ricevere parametri (input) e restituire un valore (output).

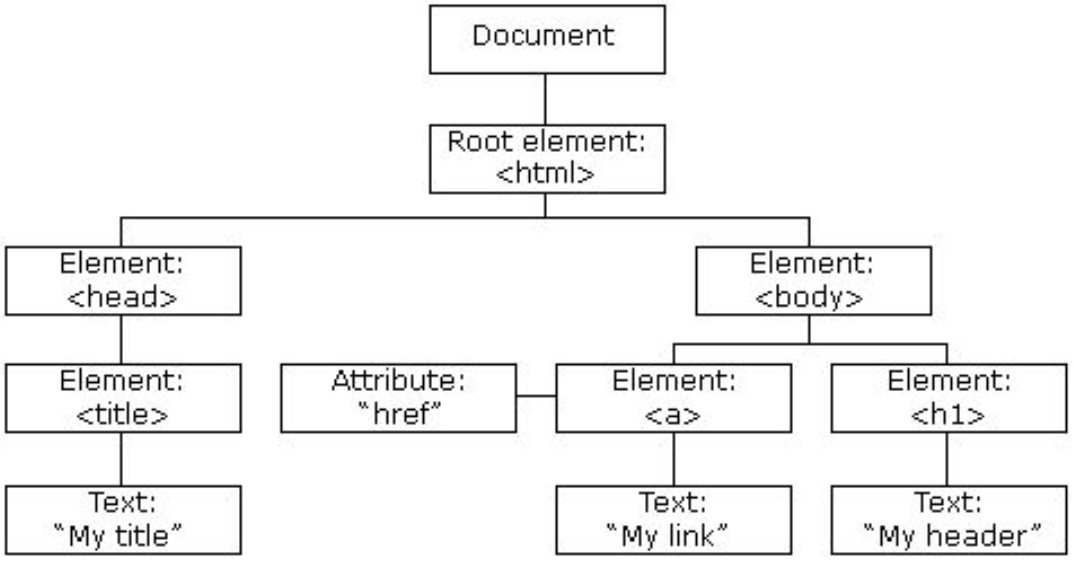


================================================================================================

**DOM DOCUMENT OBJECT MODELO**

Il DOM è la rappresentazione strutturata (ad albero) di un documento HTML;

Ogni elemento HTML, attributo e teste all’interno del documento è un nodo nell’albero del DOM. JavaScript utilizza il DOM per accedere e manipolare gli elementi della pagina web.

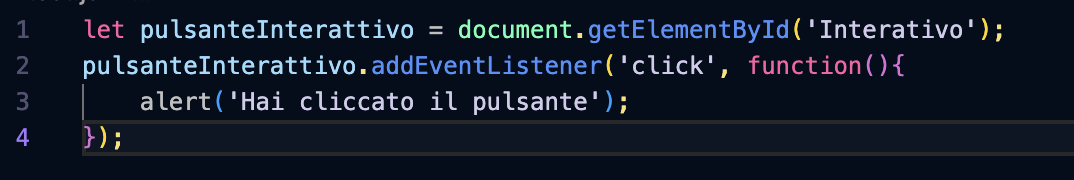


E per accedere agli elementi del DOM, possiamo usare:

* document.getElementByID(‘ID): Seleziona un elemento tramite il suo attributo id (unico);
* document.getElementsByClassName(‘class’): Seleziona tutti gli elementi con una determinata class (restituisce un HTMLCollection);
* document.getElementsByTagName(‘tag’): Seleziona tutti gli elementi con un determinato tag HTML (restituisce un HTMLCollection);
* document.querySelector(‘selettoreCSS’): Seleziona il primo elemento che corrisponde a un selettore CSS (come div, .classe, #id, p, span).
* Document.querySelectorAll(‘selettore CSS’): Seleziona tutti gli elementi che corrispondono a un selettore CSS (restituisce un NodeList).

Il DOM consente di reagire alle azioni dell’utente (click, mouseover, tasti premuti, ecc). Si utilizzano gli eventi listener per “ascoltare” specifici eventi su specifici elementi e eseguire codice in rIsposta.

elemento.addEventListener(‘tipoEvento’, funzioneCallback): Attacca um gestore di eventi all’elemento.



================================================================================================

**MODIFICARE GLI STILI CSS**

La proprietà “elemento.style” consente di accedere e modificare gli stili CSS inline di un elemento, la proprietà elemento.classList fornisce metodi utili per manipolare le classi CSS di un elemento:

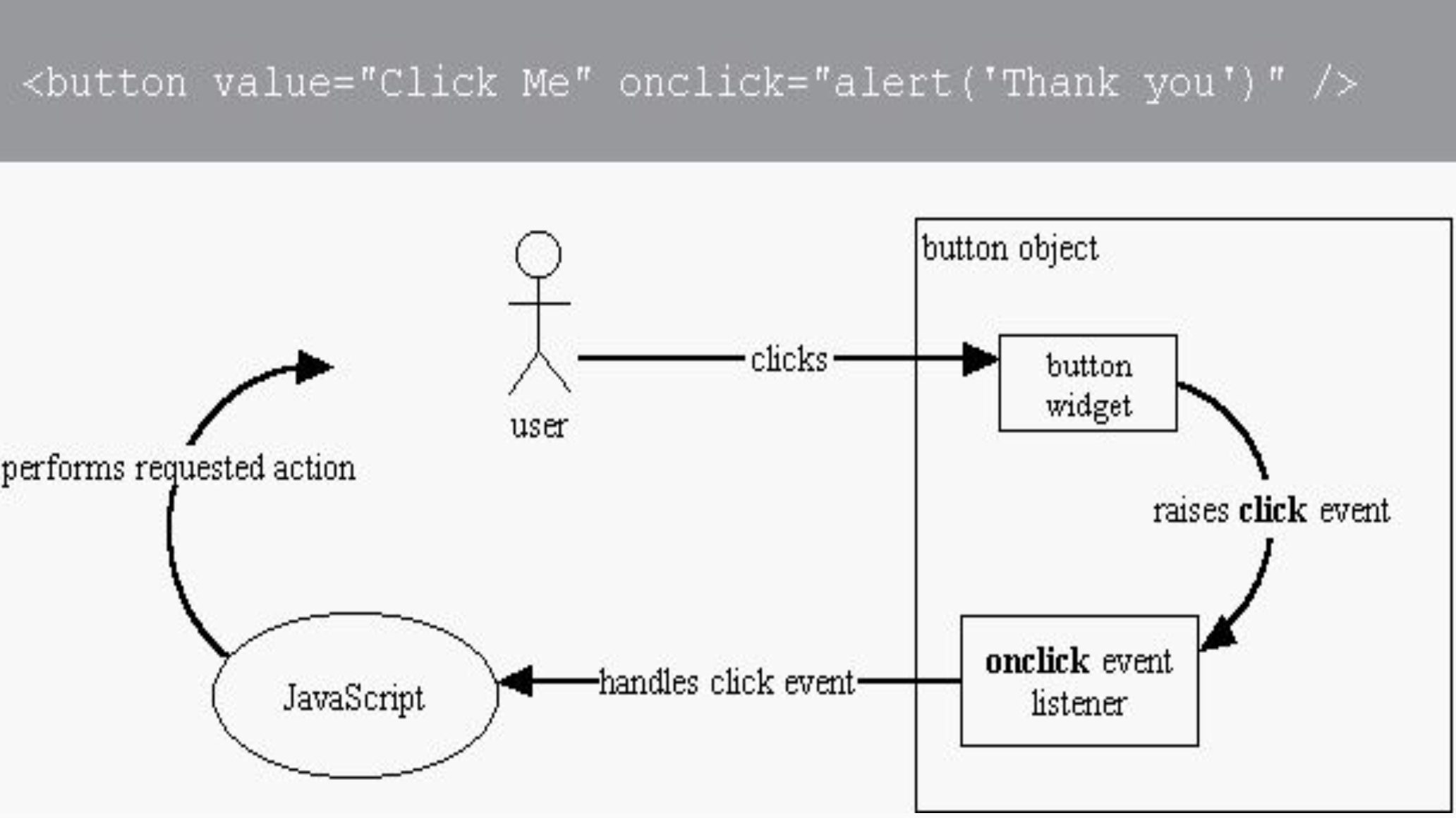
* elemento.classList.add('nuovaClasse'): Aggiunge una o più classi.
* elemento.classList.remove('classeDaRimuovere'): Rimuove una o più classi.
* elemento.classList.toggle('classeDaAlternare'): Aggiunge la classe se non esiste, la rimuove se esiste.
* elemento.classList.contains('classeDaVerificare'): Restituisce true se l'elemento ha la classe specificata.

================================================================================================

**EVENTO**

Un evento è un’azione o un’occorrenza che succede nel browser (o all’interno della pagina web. Può essere causato dall’utente (es. clic del mouse, pressione di un tasto, movimento del mouse) o dal browser stesso (es. caricamento della pagina, ridimensionamento della finestra, errore):

* Click: Clic su un elemento.
* Mouseover: Il puntatore del mouse si sposta sopra un elemento.
* Mouseout: Il puntatore del mouse esce da un elemento.
* Keydown: Un tasto viene premuto.
* Keyup: Un tasto viene rilasciato.
* Submit: Un form viene inviato.
* Change: Il valore di un elemento di input cambia.
* Lood: La pagina o una risorsa (es. immagine) è stata caricata completamente.
* Resize: La finestra del browser è stata ridimensionata.
* Scroll: La barra di scorrimento è stata spostata.



**EVENT LISTENER**

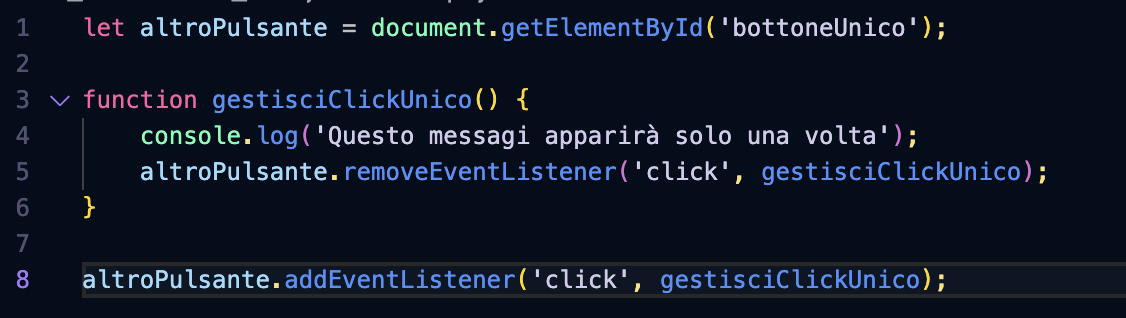
Un event listener è una funzione che “ascolta” un particolare evento su uno specifico elemento HTML. Quando l’evento si verifica, la funzione listener (chiamata anche callback dell’evento o gestore dall’evento= viene eseguita.

* tipoEvento: Una stringa che specifica il tipo di evento da ascoltare (es. ‘Click’, ‘mouseover’, ‘keydown’)
* funzioneCallback: La funzione JavaScript da eseguire quando l’evento si verifica

Quando un evento se verifica e il suo listener viene eseguito, una speciale oggetto evento viene automaticamente passato alla funzione callback come primo argomento. Questo oggetto contiene informazioni dettagliate sull’evento che si è verificato. Proprietà utili dell’oggetto evento:

* type: Il tipo de evento (es. ‘click’, ‘mouseover’)
* target: L’elemento HTML su cui l’evento si è verificato (l’elemento “bersaglio”).
* currentTarget: L’elemento HTML a cui l’event listener è stato effetivamente collegato potrebbe essere diverso da target in caso di event delegation
* clientX, clientY: Le cordinate del puntatore del mouse rispetto alla finestra del browser
* offsetX, offsetY: Le cordinate del puntatore del mouse rispetto all’elemento target
* key: Il tasto premuto (per eventi keydown e keyup).
* keyCode: (deprecato, usare key): Il codice numerico del tasto premuto.
* preventDefault(): Un metodo che impedisce il comportamento predefinito dell’evento (es. impedire l’invio di un form).
* stopPropagation(): Un metodo che impedisce la propagazione dell’evento attraverso il DOM (event bubbling e capturing).
* È importante rimuovere i listener di eventi quando non sono più necessari per evitare memory leak e comportamenti inattesi.
* Si utilizza il metodo removereEventListener(), passando gli stessi argomenti usati per aggiungerlo.

Per rimuovere un listener, la funzione callback passata a removeEventListener deve essere la stessa identica funzione (stessa referenza in memoria) passata a addEventListener. Se si usa una funzione anonima direttamente in addEventListener, non sarà possibile rimuoverla in seguito (a meno di non conservare un riferimento alla funzione).



**EVENT BUBBLING & EVENT CAPTURING**

Event Bubbling (Fase di “bolla”): Quando un evento si verifica su un elemento, si propaga (“risale”) l’albero del DOM, attivando gli event listener degli elementi genitori. Questo è il comportamento predefinito.

Event Capturing (Fase di “cattura”): è una fase opzionale in cui l’evento si propaga “verso il basso” attraverso l’albero del DOM, attivando gli event listener degli elementi antenati prima di raggiungere l’elemento target.

Il terzo argomento opzionale di addEventListener() specifica se il listener deve essere registrato per la fase di capturing (true) o di bubbling (false o omesso).

Usa preventDefault() e stopPropagation() con cautele. Potrebbero interrompere comportamenti predefiniti importanti.

================================================================================================

**OPERATORI**

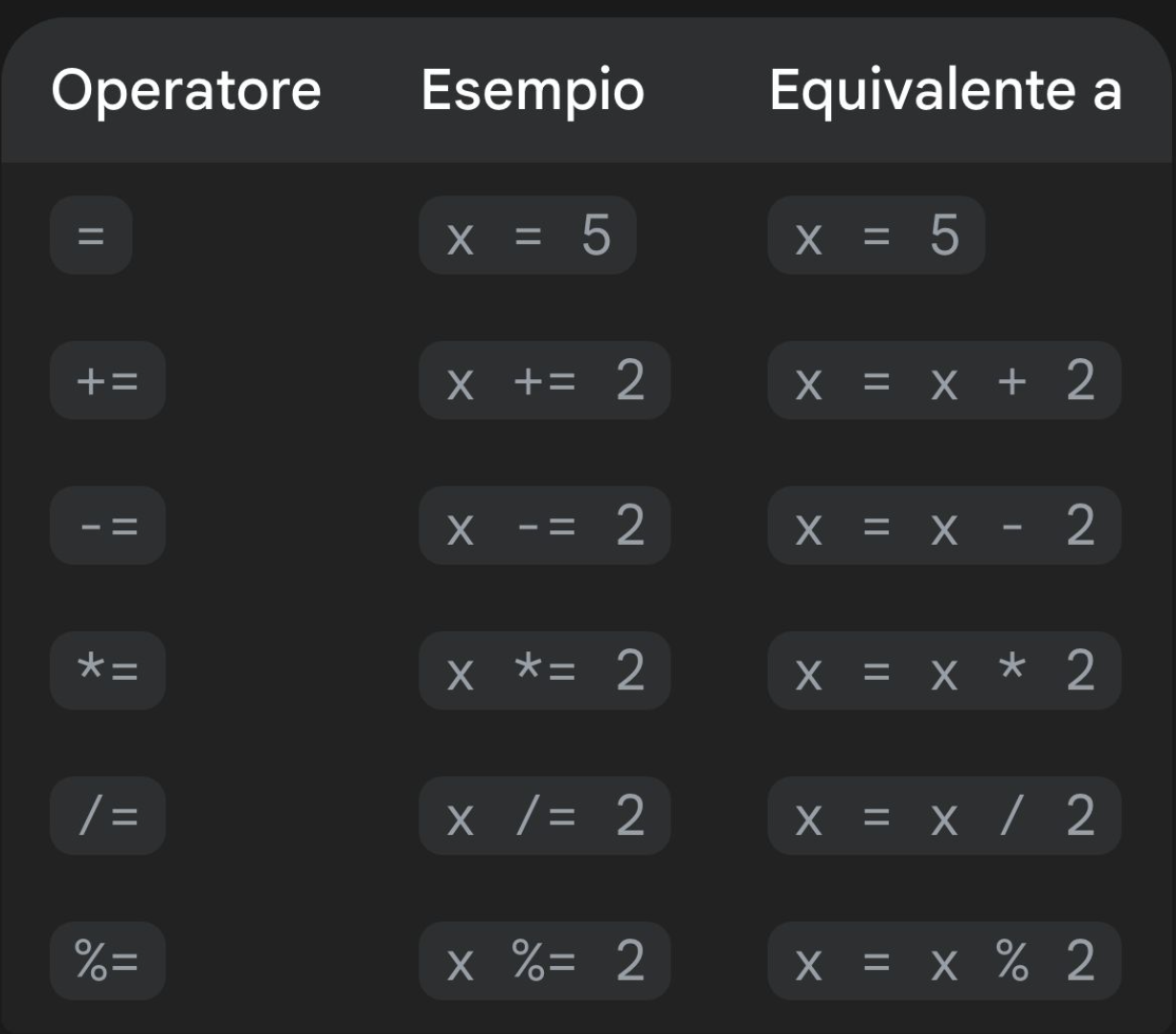
**ARITMETICI**

Eseguono operazioni matematiche.

****

**DI ASSEGNAZIONE**

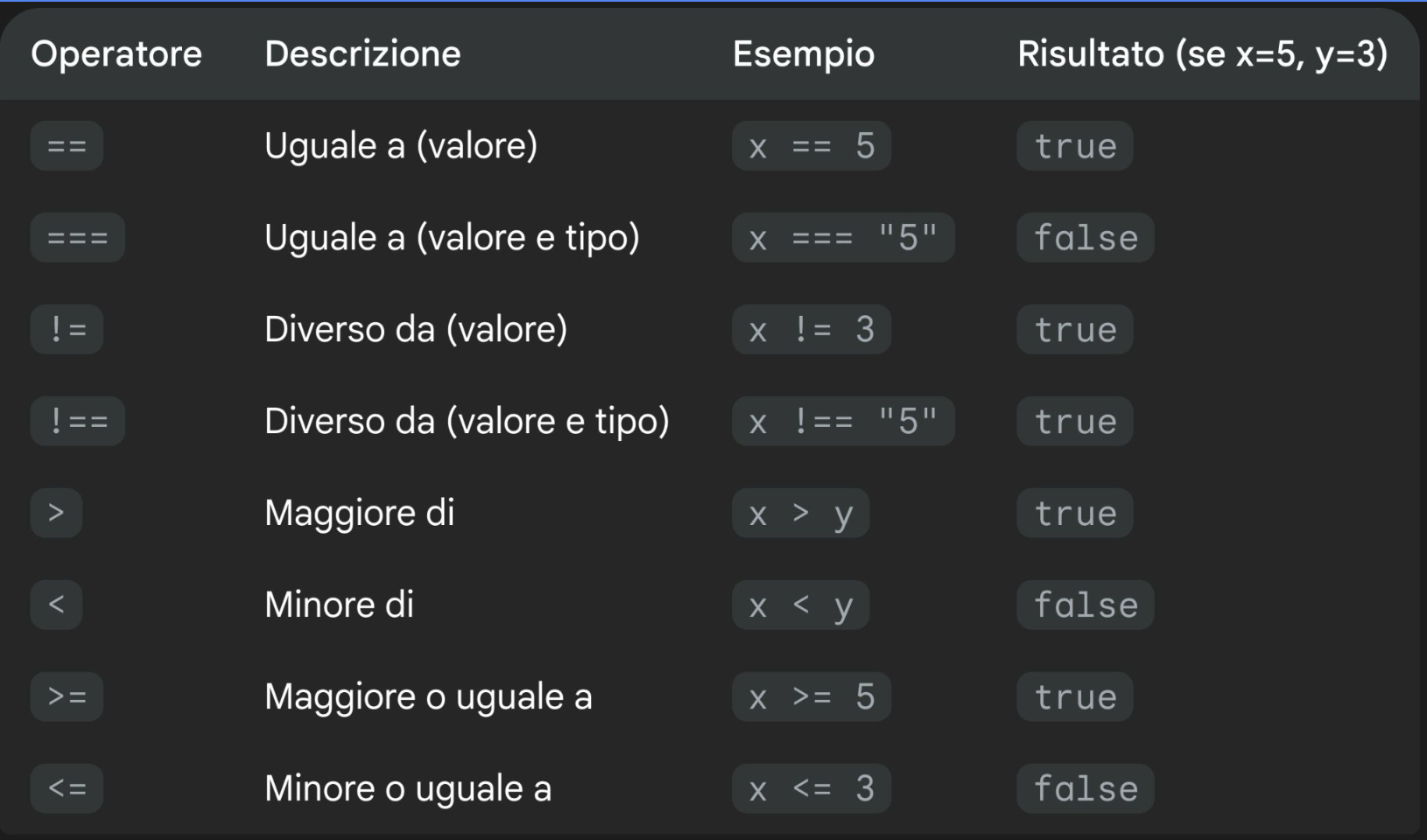
Assegnano valori alle variabili.



**DI CONFRONTO**

Confrontano due valori e restituiscono un valore booleano (true o false)

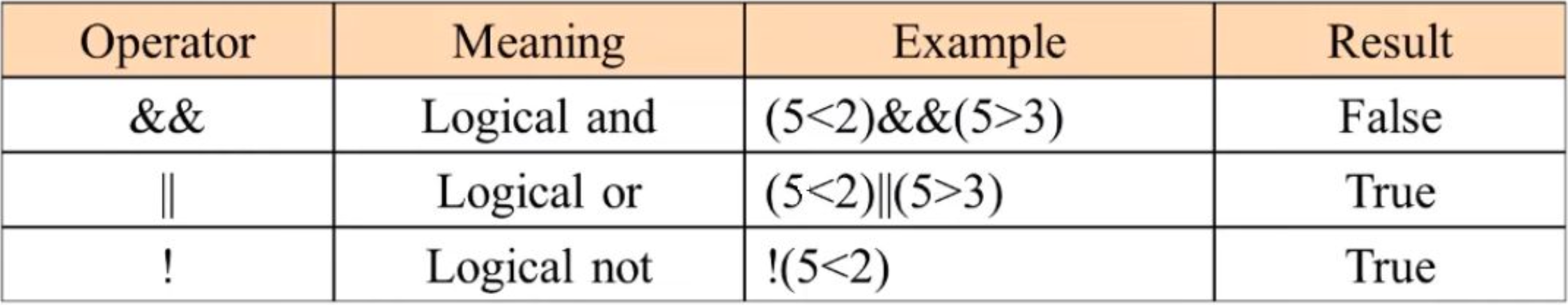
N. B. è generalmente raccomandato usare === e !== per confronti più rigorosi che tengono conto anche del tipo di dato.



================================================================================================

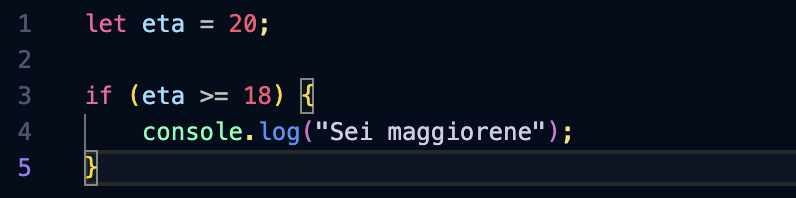
**LOGICI**

Combinano o modificano espressioni booleane.



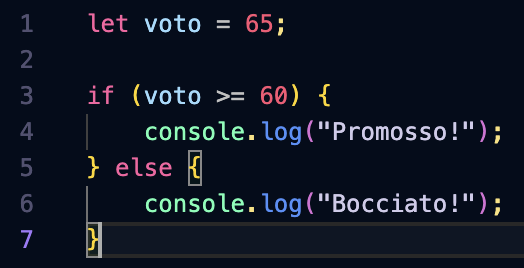
**LOGICA DECISIONALE: L’INSTRUZIONE IF**

L’istruzione if permette di eseguire un blocco di codice solo se una determinata condizione è vera. La condizione all’interno delle parentesi tonde deve essere un’espressione che valuta a true o false.



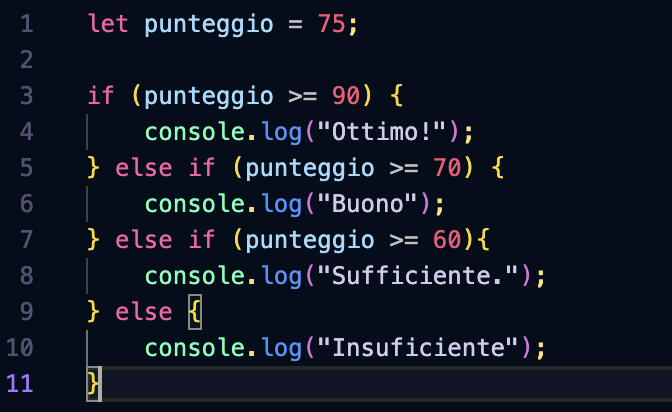
**LOGICA DECISIONALE: L’INSTRUZIONE IF (…) ELSE**

L’istruzione if (…) else permette di eseguire un blocco di codice se la condizione è vera e un altro blocco di codice se la condizione è falsa.



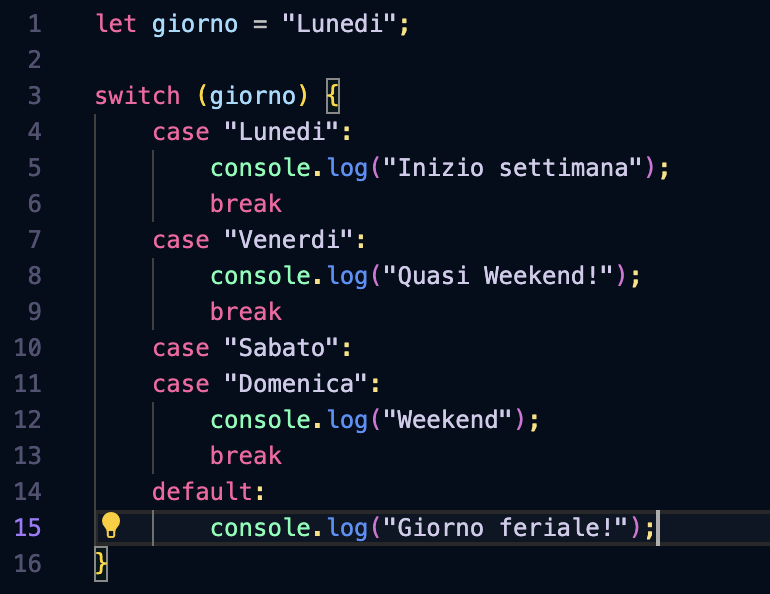
**LOGICA DECISIONALE: L’INSTRUZIONE IF (…) ELSE IS (…) ELSE**

L’istruzione if (…) else permette di eseguire un blocco di codice se la condizione è vera e un altro blocco di codice se la condizione è falsa.



**LA STRUTTURA DI CONTROLLO SWITCH NELL CODICE JAVASCRIPT**

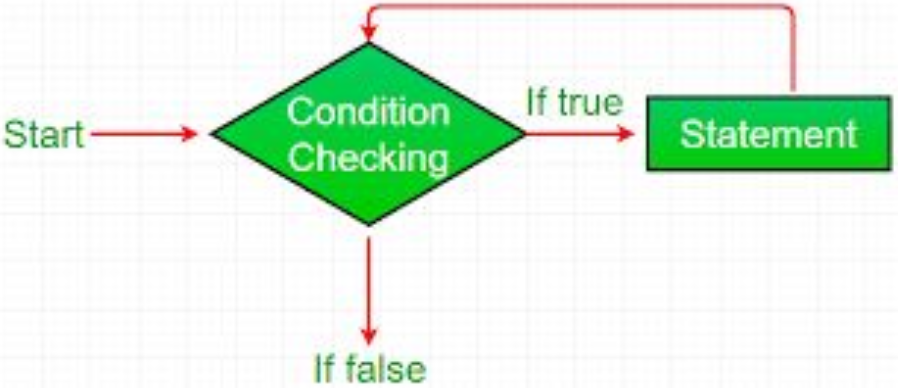
Alternative a “if (…) else” per scelta Multiple. L’istruzione switch valuta un’espressione e confronta il suo valore con diversi casi (case), se un valore dell’espressione corrisponde al valore di un case, viene eseguito il blocco di codice associato a quel case.



**CICLI**

I cicli (o loop) sono strutture di controllo fondamentali nella programmazione che permettono di eseguire ripetutamente um blocco di codice (il “corpo” del ciclo) fino a quando una determinata condizione è vera o falsa, a seconda del tipo di ciclo.

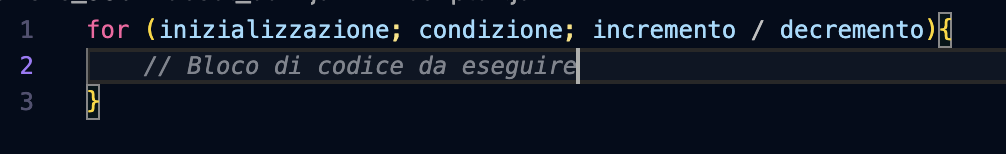
Consentono di automatizzare compiti ripetitivi, evitando di scrivere lo stesso codice più volte.



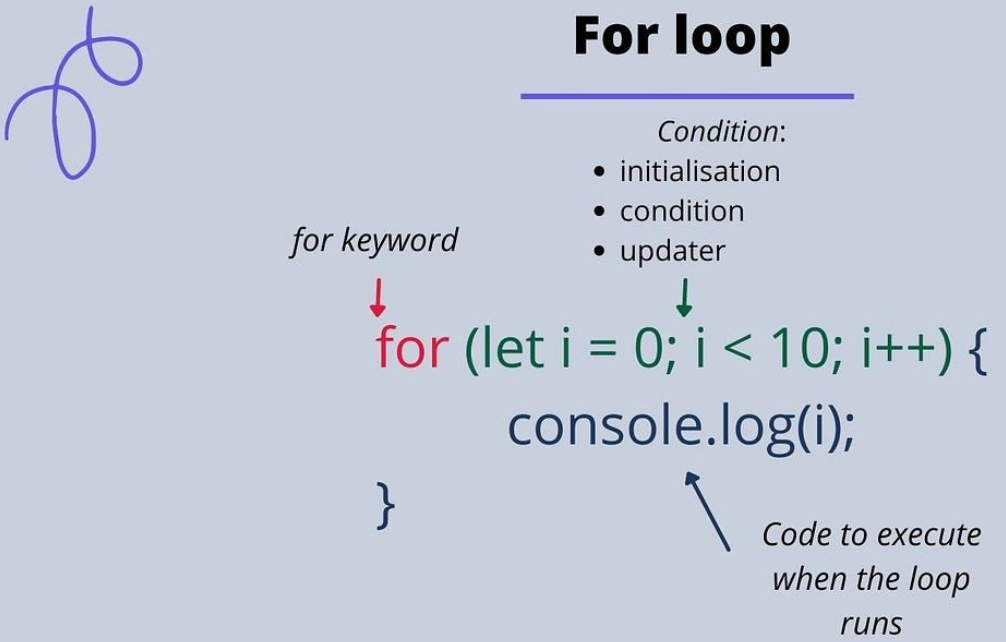
**IL CICLO FOR**

Una struttura di controllo che permette di eseguire un blocco di codice ripetutamente, finchè una condizione specificata è vera. Ideale per quando si conosce in anticipo il numero di iterazioni necessarie.

La sintassi è:

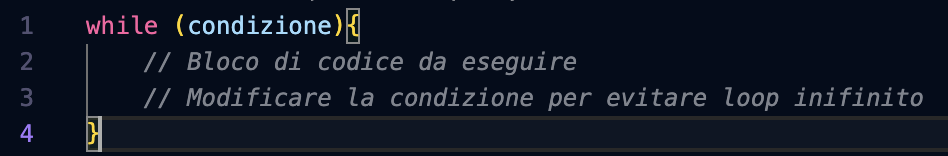


La inizializzazione si dichiara una variabile contatore (es. let i = 0), che viene eseguita una sola volta, all’inizio del ciclo, per doppo la condizione che sarà una espressione booleana valutata prima di ogni iterazione (es. i < 5) e se risulta vero (true), il blocco di codice viene eseguito, se false il ciclo termina in quello punto. O incremento / decremento modifica la variabile contatore dopo ogni iterazione (es. i ++ o i + 2).

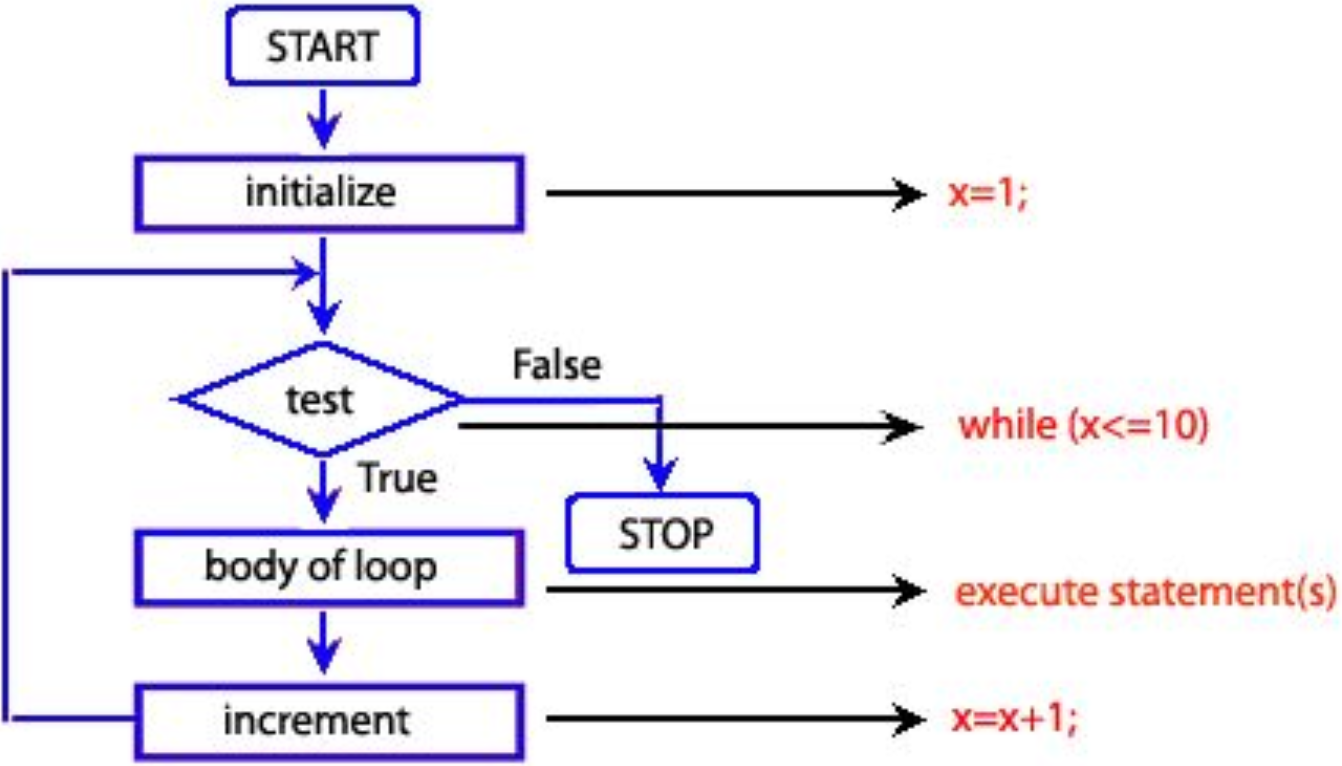


**IL CICLO WHILE**

Il ciclo while è una struttura di controllo che ripete un blocco di codice finché una condizione specificata è vera (true). A differenza del for è utile quando non si conosce a priori ul numero di iterazioni necessarie. La tua sintassi è:



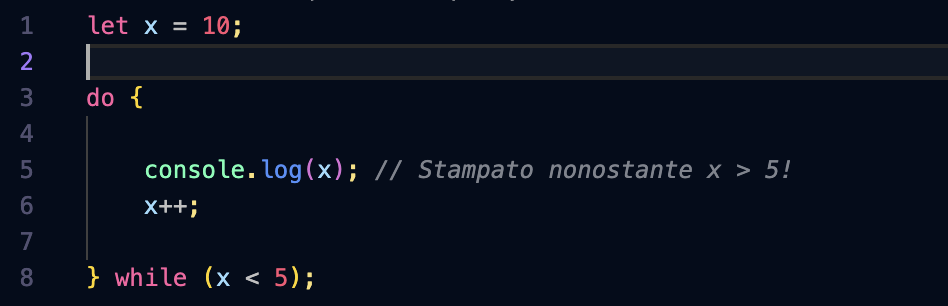
Prima si valuta la condizione, se è true esegue il blocco di codice, se è falsa, il ciclo termina e dopo ogni iterazione, ritorna a controllare la condizione.



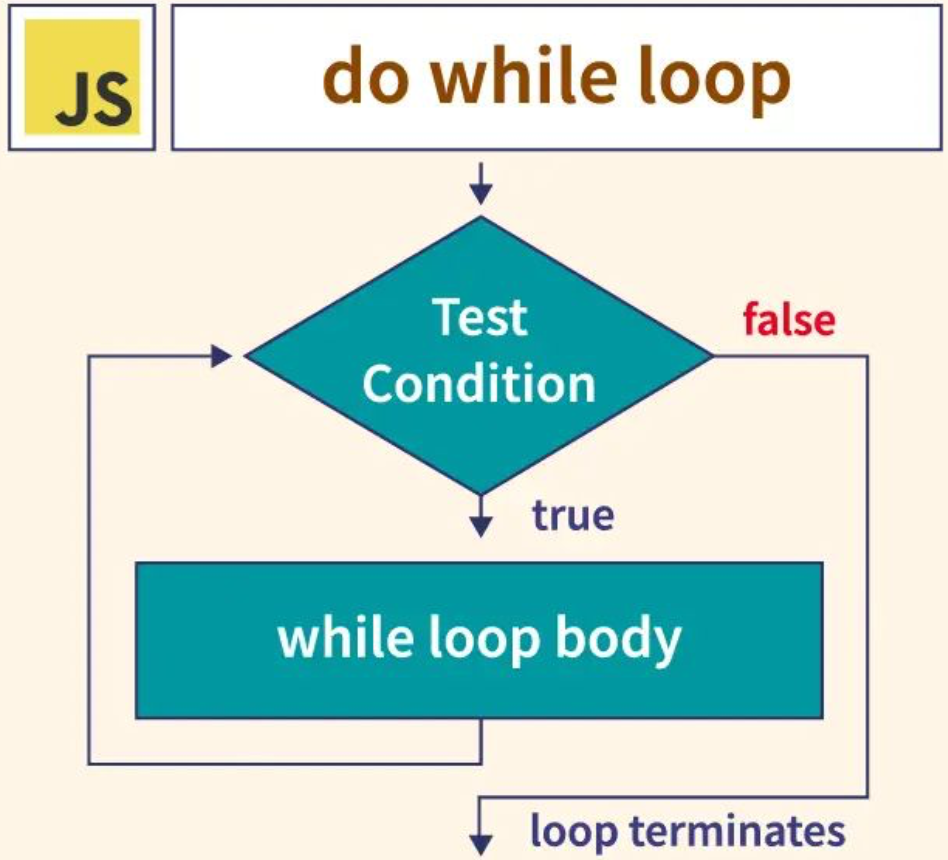
Se la condizione non diventa mai false, il ciclo continua all’infinito, bloccando il programma.

While è ideale quando il numero di iterazioni dipende da una condizione dinamica (es. input utente, valori variabili), quando il ciclo for è meglio quando se itera su un numero prevedibile di elementi (es. array, contatore fisso).

Simile al while, abbiamo il “do (….) while” ma garantisce almeno un’esecuzione del blocco, poiché la condizione è verificata dopo.

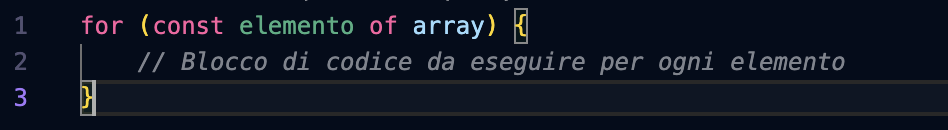


L'immagine qui sotto descrive il ciclo del 'do (...) while':



**ITERARE SU ARRAY CON FOR … OF**

Il ciclo “for (…) of” è un modo semplice e moderno per scorrere gli elementi di un array ( o altri oggetti iterabili) senza dover gestire manualmente un contatore o un indice.



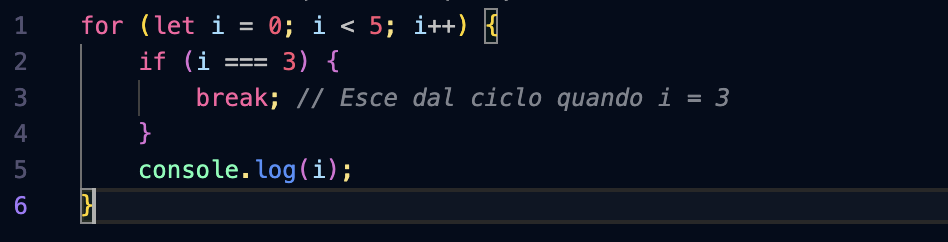
Il elemento variabile che rappresenta l’elemento corrente nell’array (puoi chiamarla come preferisci es. item, value, num), dopo l’array o oggetto iterabile da scorrere e il aotumatico che non serve un contatore (i), o .length il ciclo itera fico alla fine dell’array.

Il metodo for classico richiede un contatore (let i = 0) e l’accesso agli elementi con array[i]. Mentre il forEach è un metodo degli array ma non supporta break o continue, vai a tutti item. For in itera sugli (per array) o chiavi per oggetti non sui valori diretti.

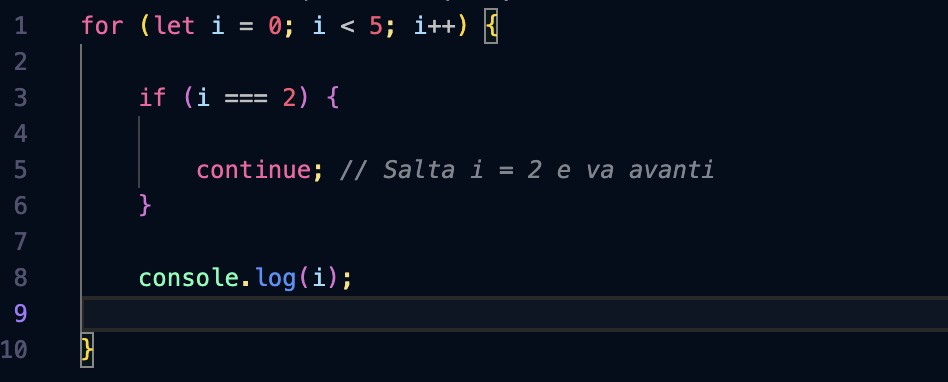
ISTRUZIONI BREAK E CONTINUE NEI CICLI

La istruzioni break e continue servono per controllare il flusso dei cicli (for, while, do (…) while, for (…) of, ecc).

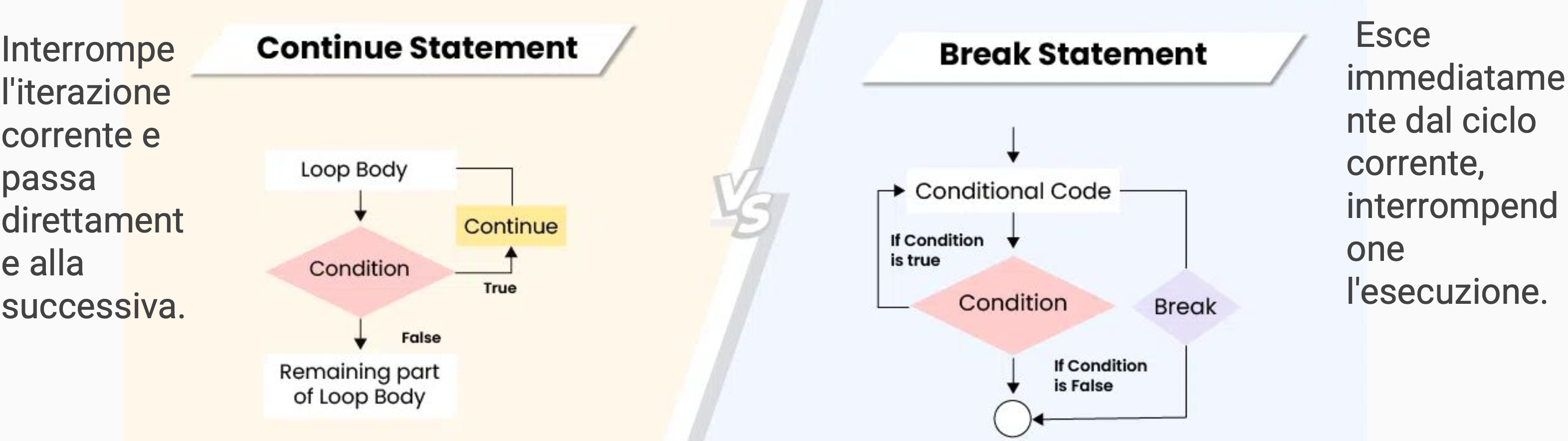
Il break interrompe il ciclo, termina immediatamente il ciclo e esce da esso, è utile quando vuoi fermare l’iterazione una volta trovata una certa condizione, per esempio:



Il continue salta all’iterazione successiva e interrompe l’iterazione corrente e passa alla successiva senza uscire dal ciclo, è utile per saltare certi valori senza dover annidare “if”. Esempio con for:



La differenze chiave sono che il break termina tutto il ciclio per uscire da un loop quando si trova um valore, mentre il continue salita solo l’iterazione corrente, per ignorare certi valori (es. numeri pari).



Aaaaaaa

================================================================================================

**TITOLO**

Testo.

================================================================================================