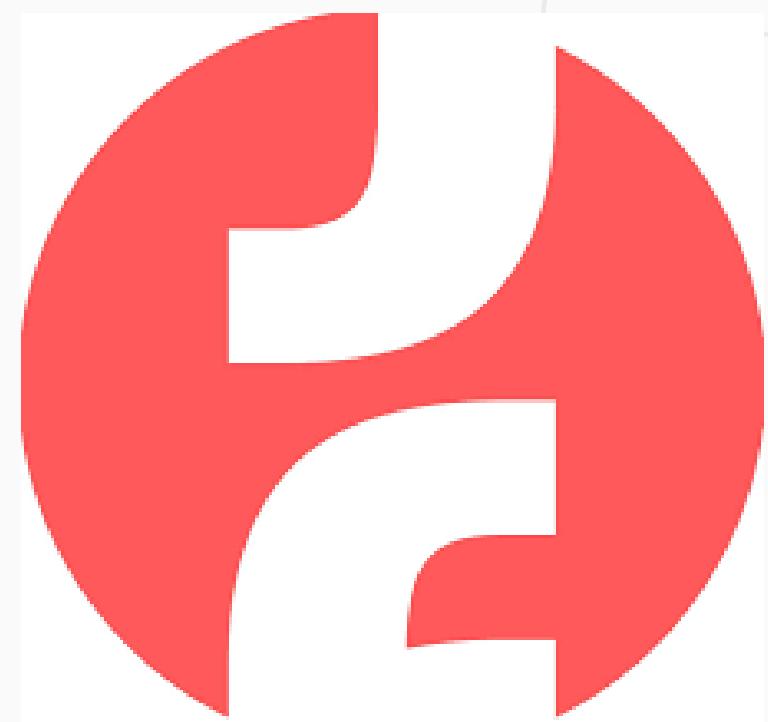


JAVASCRIPT





JAVASCRIPT

È UN **LINGUAGGIO DI PROGRAMMAZIONE**
MULTI PARADIGMA ORIENTATO AGLI EVENTI,
USATO PRINCIPALMENTE PER CREARE SITI
WEB DINAMICI E INTERATTIVI.

È STATO INVENTATO DA BRENDAN EICH NEL
1995. INIZIALMENTE, IL LINGUAGGIO FU
CREATO PER ESSERE UTILIZZATO
ALL'INTERNO DEL BROWSER NETSCAPE
NAVIGATOR, MA BEN PRESTO VENNE
ADOTTATO ANCHE DA ALTRI BROWSER COME
INTERNET EXPLORER E MOZILLA FIREFOX.





JAVASCRIPT



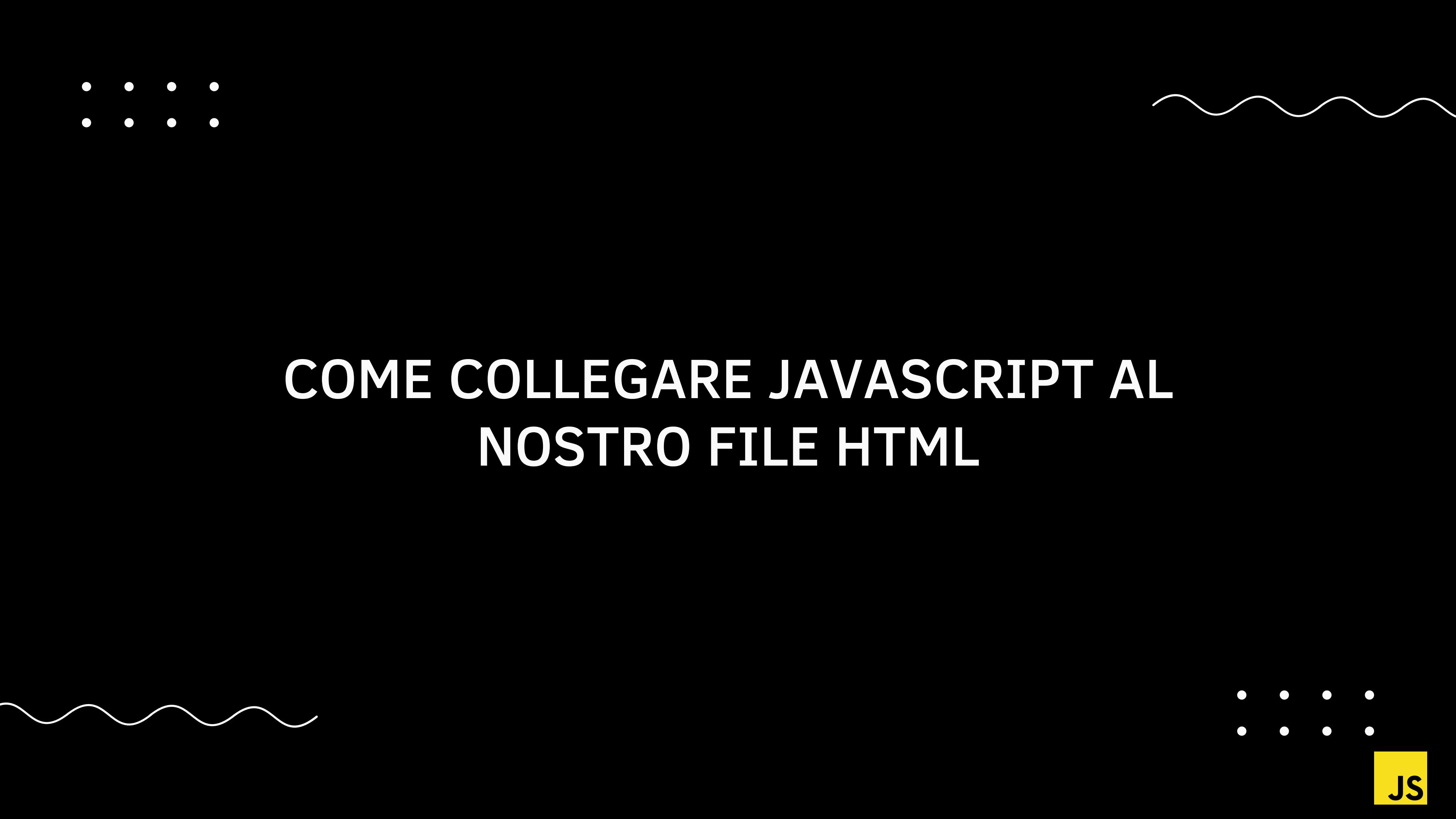
JAVASCRIPT È UN LINGUAGGIO DI SCRIPTING INTERPRETATO, OVVERO IL CODICE SORGENTE VIENE ESEGUITO DIRETTAMENTE DAL BROWSER, SENZA LA NECESSITÀ DI UN COMPILATORE. CIÒ SIGNIFICA CHE IL CODICE PUÒ ESSERE MODIFICATO E TESTATO RAPIDAMENTE, SENZA DOVER RICOMPILARE L'INTERO PROGRAMMA.



JAVASCRIPT

JAVASCRIPT VIENE SPESSO USATO PER CREARE ANIMAZIONI, EFFETTI VISIVI E INTERAZIONI UTENTE, MA PUÒ ANCHE ESSERE UTILIZZATO PER ELABORARE DATI, GESTIRE IL LAYOUT DELLA PAGINA WEB E COMUNICARE CON SERVER ESTERNI.

GRAZIE ALLE SUE NUMEROSE LIBRERIE E FRAMEWORK, JAVASCRIPT È UNO DEI LINGUAGGI DI PROGRAMMAZIONE PIÙ POPOLARI AL MONDO, ED È STATO UTILIZZATO PER CREARE MOLTE DELLE APPLICAZIONI WEB E MOBILE PIÙ FAMOSE.



COME COLLEGARE JAVASCRIPT AL NOSTRO FILE HTML

JS



COME COLLEGARE JAVASCRIPT AL NOSTRO FILE HTML

PER UTILIZZARE JAVASCRIPT IN UNA PAGINA WEB, È NECESSARIO INSERIRE IL CODICE ALL'INTERNO DEL CODICE HTML.

ANALIZZIAMO I MODI PRINCIPALI



COME COLLEGARE JAVASCRIPT AL NOSTRO FILE HTML

INSERIRE IL CODICE JAVASCRIPT DIRETTAMENTE ALL'INTERNO
DELL'ELEMENTO



Esempio

```
<script>  
    // codice JavaScript  
</script>
```



COME COLLEGARE JAVASCRIPT AL NOSTRO FILE HTML

UTILIZZARE L'ATTRIBUTO "SRC" PER INDICARE IL PERCORSO DI UN FILE
JAVASCRIPT ESTERNO



Esempio

```
<script src="percorso/file.js"></script>
```



COME COLLEGARE JAVASCRIPT AL NOSTRO FILE HTML

IN ENTRAMBI I CASI, È IMPORTANTE ASSICURARSI CHE IL CODICE JAVASCRIPT SIA VALIDO E CORRETTO, AL FINE DI EVITARE ERRORI DURANTE L'ESECUZIONE. INOLTRE, È IMPORTANTE POSIZIONARE IL CODICE JAVASCRIPT NELL'ORDINE CORRETTO ALL'INTERNO DELLA PAGINA HTML, PER GARANTIRE CHE TUTTI I COMPONENTI DELLA PAGINA SIANO CARICATI CORRETTAMENTE PRIMA CHE IL CODICE JAVASCRIPT VENGA ESEGUITO.

PROMPT PER GLI INPUT



PROMPT PER GLI INPUT

```
LET PERSON = PROMPT("PLEASE ENTER YOUR NAME");
```

VISUALIZZARE GLI OUTPUT



VISUALIZZARE GLI OUTPUT

QUANDO SI SCRIVE CODICE JAVASCRIPT, È POSSIBILE VISUALIZZARE GLI OUTPUT IN DIVERSI MODI. IL MODO PIÙ SEMPLICE PER VISUALIZZARE GLI OUTPUT È UTILIZZARE IL METODO **CONSOLE.LOG()**. QUESTO METODO PERMETTE DI STAMPARE I VALORI DELLE VARIABILI O LE STRINGHE DI TESTO SULLA CONSOLE DEL BROWSER.

• • • Esempio

```
console.log("Ciao mondo");
```

SINTASSI

JS



SINTASSI

JAVASCRIPT È UN LINGUAGGIO DI SCRIPTING AD INTERPRETAZIONE, IL CHE SIGNIFICA CHE IL CODICE VIENE ESEGUITO LINEA PER LINEA, SENZA LA NECESSITÀ DI COMPILARE IL CODICE PRIMA DELL'ESECUZIONE. QUESTO RENDE LA SCRITTURA E LA MODIFICA DEL CODICE MOLTO PIÙ VELOCI.

IN JAVASCRIPT, UNA RIGA DI CODICE TERMINA CON UN PUNTO E VIRGOLA ";" OPPURE PUÒ ESSERE OMESSO IN ALCUNI CASI. AD ESEMPIO, IL PUNTO E VIRGOLA PUÒ ESSERE OMMESSO ALLA FINE DI UNA ISTRUZIONE SE QUESTA È SEGUITA DA UNA PARENTESI GRAFFA "{", O SE SI TRATTA DELL'ULTIMA ISTRUZIONE IN UN BLOCCO DI CODICE.

VARIABILI E COSTANTI



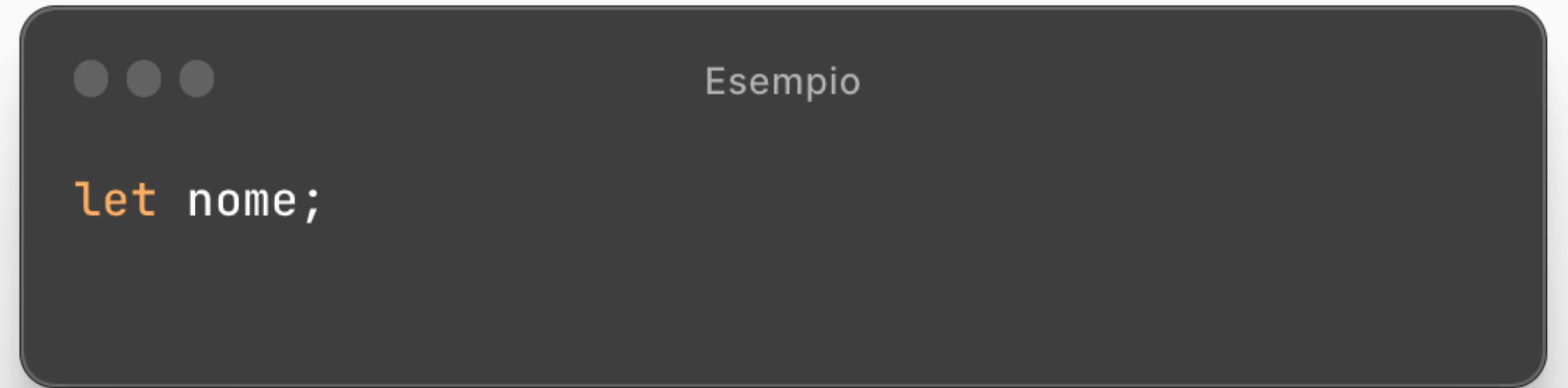
VARIABILI E COSTANTI

LE **VARIABILI** E LE **COSTANTI** SONO ELEMENTI FONDAMENTALI DELLA PROGRAMMAZIONE IN JAVASCRIPT. ESSE CONSENTONO DI IMMAGAZZINARE DATI ALL'INTERNO DI UN PROGRAMMA E DI UTILIZZARLI PER ESEGUIRE OPERAZIONI O EFFETTUARE CALCOLI.



VARIABILI

IN JAVASCRIPT, UNA VARIABILE VIENE DEFINITA UTILIZZANDO LA PAROLA CHIAVE "**VAR**" O "**LET**", SEGUITA DAL NOME DELLA VARIABILE. AD ESEMPIO, PER DICHIARARE UNA VARIABILE CHIAMATA "NOME", SI PUÒ SCRIVERE:



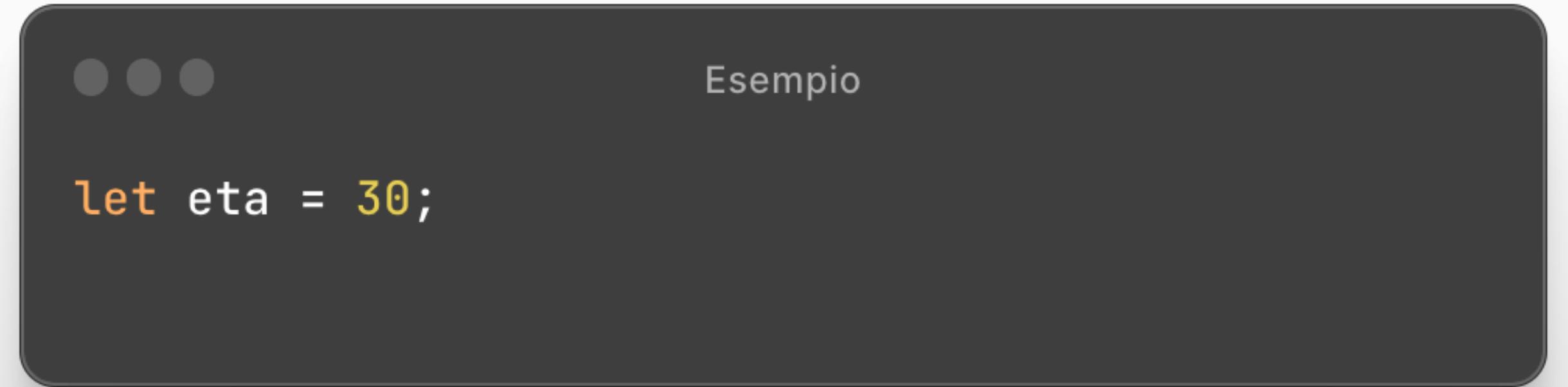
Esempio

```
let nome;
```



VARIABILI

IN ALTERNATIVA, È POSSIBILE INIZIALIZZARE UNA VARIABILE NEL MOMENTO DELLA DICHIARAZIONE, SPECIFICANDO IL VALORE INIZIALE. AD ESEMPIO, PER DICHIARARE UNA VARIABILE "ETA" CON VALORE INIZIALE DI 30, SI PUÒ SCRIVERE:



• • •

Esempio

```
let eta = 30;
```



VARIABILI

È IMPORTANTE NOTARE CHE LE VARIABILI DICHIARATE CON "VAR" E QUELLE DICHIARATE CON "LET" DIFFERISCONO PER IL LORO AMBITO DI VISIBILITÀ.

- LE VARIABILI DICHIARATE CON "VAR" HANNO UN AMBITO DI FUNZIONE, IL CHE SIGNIFICA CHE SONO ACCESSIBILI ALL'INTERNO DELLA FUNZIONE IN CUI SONO STATE DICHIARATE.
- LE VARIABILI DICHIARATE CON "LET" HANNO INVECE UN AMBITO DI BLOCCO, IL CHE SIGNIFICA CHE SONO ACCESSIBILI SOLO ALL'INTERNO DEL BLOCCO DI CODICE IN CUI SONO STATE DICHIARATE.



COSTANTI

LE COSTANTI, AL CONTRARIO DELLE VARIABILI, NON POSSONO ESSERE RIASSEGNAME UNA VOLTA CHE È STATO LORO ASSEGNAATO UN VALORE.

PER DICHIARARE UNA COSTANTE IN JAVASCRIPT, SI UTILIZZA LA PAROLA CHIAVE "**CONST**". AD ESEMPIO, PER DICHIARARE UNA COSTANTE CHIAMATA "PI" CON VALORE PARI A 3.14, SI PUÒ SCRIVERE:

```
const PI = 3.14;
```



VARIABILI E COSTANTI: IN SINTESI

- LE VARIABILI DI TIPO **VAR** HANNO UNO SCOPE GLOBALE, O DI FUNZIONE (LOCALE), MENTRE QUELLE DICHIARATE CON LET E CONST HANNO SCOPE DI BLOCCO (LOCALE).
- LE VARIABILI **VAR** POSSONO ESSERE AGGIORNATE O RI-DICHIARATE
- LE VARIABILI **LET** POSSONO ESSERE AGGIORNATE MA NON RI-DICHIARATE
- LE VARIABILI **CONST** NON POSSONO ESSERE NÉ AGGIORNATE NÉ RI-DICHIARATE.
- LE VARABILI **VAR** SONO INIZIALIZZATE AUTOMATICAMENTE CON **UNDEFINED**, MENTRE LE VARIABILI **LET** E **CONST** NON SONO INIZIALIZZATE.
- LE VARABILI **VAR** E **LET** POSSONO ESSERE DICHIARATE SENZA ESSERE INIZIALIZZATE, CONST DEVE ESSERE INIZIALIZZATO DURANTE LA SUA DICHIARAZIONE.

OPERATORI



OPERATORI: OPERATORI ARITMETICI

GLI OPERATORI SONO SIMBOLI SPECIALI UTILIZZATI PER ESEGUIRE OPERAZIONI MATEMATICHE O LOGICHE SUI DATI IN UN PROGRAMMA. IN JAVASCRIPT, ESISTONO DIVERSI TIPI DI OPERATORI, TRA CUI OPERATORI ARITMETICI, OPERATORI DI CONFRONTO E OPERATORI LOGICI.

GLI **OPERATORI ARITMETICI** SONO UTILIZZATI PER ESEGUIRE OPERAZIONI MATEMATICHE SUI NUMERI. ECCO ALCUNI DEGLI OPERATORI ARITMETICI PIÙ COMUNI IN JAVASCRIPT:

- "+" (SOMMA)
- "-" (SOTTRAZIONE)
- "*" (MOLTIPLICAZIONE)
- "/" (DIVISIONE)
- "%" (RESTO DELLA DIVISIONE)



OPERATORI: OPERATORI ARITMETICI

AD ESEMPIO, PER SOMMARE DUE NUMERI IN JAVASCRIPT, SI PUÒ UTILIZZARE L'OPERATORE "+". AD ESEMPIO:

Esempio

```
let risultato = 5 + 3;
```

IN QUESTO ESEMPIO, LA VARIABILE "RISULTATO" CONTERRÀ IL VALORE 8, CHE È IL RISULTATO DELLA SOMMA TRA 5 E 3.



OPERATORI: OPERATORI DI CONFRONTO

GLI **OPERATORI DI CONFRONTO**, COME IL LORO NOME SUGGERISCE, SONO UTILIZZATI PER CONFRONTARE DUE VALORI. ECCO ALCUNI DEGLI OPERATORI DI CONFRONTO PIÙ COMUNI IN JAVASCRIPT:

- "**==**" (UGUALE A)
- "**!=**" (DIVERSO DA)
- "**>**" (MAGGIORE DI)
- "**<**" (MINORE DI)
- "**>=**" (MAGGIORE O UGUALE A)
- "**<=**" (MINORE O UGUALE A)



OPERATORI: OPERATORI DI CONFRONTO

AD ESEMPIO, PER CONFRONTARE DUE NUMERI IN JAVASCRIPT, SI PUÒ UTILIZZARE L'OPERATORE DI CONFRONTO ">". AD ESEMPIO:

Esempio

```
let maggiore = 5 > 3;
```

IN QUESTO ESEMPIO, LA VARIABILE "MAGGIORE" CONTERRÀ IL VALORE "TRUE", POICHÉ 5 È MAGGIORE DI 3.



OPERATORI: OPERATORI LOGICI

GLI **OPERATORI LOGICI**, INFINE, SONO UTILIZZATI PER ESEGUIRE OPERAZIONI LOGICHE SUI VALORI BOOLEANI (VERI O FALSI). ECCO ALCUNI DEGLI OPERATORI LOGICI PIÙ COMUNI IN JAVASCRIPT:

- "**&&**" (AND LOGICO)
- "**||**" (OR LOGICO)
- "**!**" (NOT LOGICO)



OPERATORI: OPERATORI LOGICI

AD ESEMPIO, PER UTILIZZARE L'OPERATORE "&&" IN JAVASCRIPT, SI PUÒ SCRIVERE:



Esempio

```
let risultato = (5 > 3) && (2 < 4);
```

IN QUESTO ESEMPIO, LA VARIABILE "RISULTATO" CONTERRÀ IL VALORE "TRUE", POICHÉ SIA L'ESPRESSIONE "5 > 3" CHE L'ESPRESSIONE "2 < 4" SONO VERE.

LO SCOPE IN JAVASCRIPT

JS



LO SCOPE IN JAVASCRIPT

IN JAVASCRIPT, LO SCOPE DEFINISCE LA VISIBILITÀ E L'ACCESSIBILITÀ DELLE VARIABILI IN UNA PARTICOLARE PORZIONE DI CODICE. CI SONO DUE TIPI DI SCOPE IN JAVASCRIPT:
LO **SCOPE GLOBALE** E LO **SCOPE LOCALE**.



LO SCOPE IN JAVASCRIPT: SCOPE GLOBALE

LE VARIABILI DEFINITE AL DI FUORI DI QUALSIASI FUNZIONE SONO DEFINITE NELLO SCOPE GLOBALE.

CIÒ SIGNIFICA CHE POSSONO ESSERE ACCESSIBILI E MODIFICABILI DA QUALSIASI PARTE DEL CODICE, INCLUSI LE FUNZIONI. TUTTAVIA, L'USO DI VARIABILI GLOBALI DOVREBBE ESSERE LIMITATO POICHÉ PUÒ CREARE PROBLEMI DI CONFLITTO CON ALTRE PARTI DEL CODICE.



LO SCOPE IN JAVASCRIPT: SCOPE LOCALE

LE VARIABILI DEFINITE ALL'INTERNO DI UN BLOCCO SONO DEFINITE NELLO SCOPE LOCALE.

CIÒ SIGNIFICA CHE SONO ACCESSIBILI SOLO ALL'INTERNO DELLA FUNZIONE STESSA E NON SONO ACCESSIBILI DAL CODICE AL DI FUORI DELLA FUNZIONE. QUESTO AIUTA A PREVENIRE PROBLEMI DI CONFLITTO E MIGLIORA LA SICUREZZA DEL CODICE.



LO SCOPE IN JAVASCRIPT

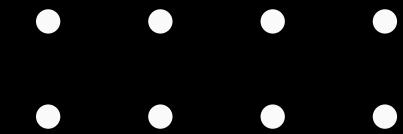
È POSSIBILE ACCEDERE ALLE VARIABILI GLOBALI ALL'INTERNO DI UNA FUNZIONE, MA SE UNA VARIABILE LOCALE HA LO STESSO NOME DI UNA VARIABILE GLOBALE, LA VARIABILE LOCALE AVRÀ LA PRECEDENZA ALL'INTERNO DELLA FUNZIONE.

IN JAVASCRIPT, LO SCOPE È DETERMINATO DALLA POSIZIONE IN CUI UNA VARIABILE VIENE DEFINITA. AD ESEMPIO, UNA VARIABILE DEFINITA ALL'INTERNO DI UN BLOCCO DI CODICE COME UN'ISTRUZIONE CONDIZIONALE O UN CICLO FOR AVRÀ LO SCOPE LIMITATO A QUEL BLOCCO DI CODICE.

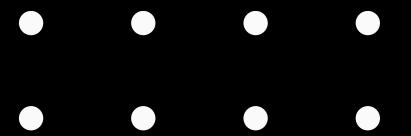


LO SCOPE IN JAVASCRIPT

COMPRENDERE LO SCOPE IN JAVASCRIPT È IMPORTANTE PER SCRIVERE CODICE EFFICIENTE E SICURO. UTILIZZARE VARIABILI GLOBALI CON PARSIMONIA E DEFINIRE LE VARIABILI LOCALI NELLA PORTATA CORRETTA POSSONO AIUTARE A PREVENIRE PROBLEMI DI CONFLITTO E AUMENTARE LA CHIAREZZA E LA LEGGIBILITÀ DEL CODICE.



STRINGHE



JS



STRINGHE

LE STRINGHE SONO UN TIPO DI DATI FONDAMENTALE IN JAVASCRIPT, UTILIZZATE PER RAPPRESENTARE TESTO O SEQUENZE DI CARATTERI.

LE STRINGHE SONO DEFINITE ALL'INTERNO DI VIRGOLETTE SINGOLE (' ') O DOPPIE (" ").

AD ESEMPIO:

Esempio

```
let nome = 'Maria';
let saluto = "Ciao, mondo!";
```



STRINGHE: CONCATENAZIONI

È POSSIBILE UNIRE PIÙ STRINGHE UTILIZZANDO L'OPERATORE DI CONCATENAZIONE (+) O L'OPERATORE DI ASSEGNAZIONE CONCATENATO (+=).



Esempio

```
let nome = 'Maria';
let cognome = 'Rossi';
let nomeCompleto = nome + ' ' + cognome;
console.log(nomeCompleto); // Output: "Maria
Rossi"

let saluto = "Ciao, ";
saluto += nome;
console.log(saluto); // Output: "Ciao, Maria"
```



STRINGHE: LUNGHEZZA

PER DETERMINARE LA LUNGHEZZA DI UNA STRINGA, POSSIAMO UTILIZZARE LA PROPRIETÀ LENGTH.



Esempio

```
let nome = 'Maria';
console.log(nome.length); // Output: 5
```



STRINGHE: ACCESSO AI CARATTERI

IN JAVASCRIPT, I CARATTERI DI UNA STRINGA POSSONO ESSERE ACCESSIBILI UTILIZZANDO LA NOTAZIONE DELLE PARENTESI QUADRE [] E L'INDICE DEL CARATTERE DESIDERATO.
L'INDICE DEL PRIMO CARATTERE DI UNA STRINGA È 0.

• • • Esempio

```
let saluto = "Ciao, mondo!";
console.log(saluto[0]); // Output: "C"
console.log(saluto[5]); // Output: ","
console.log(saluto[saluto.length-1]); // Output: "!"
```



STRINGHE: METODI

LE STRINGHE IN JAVASCRIPT SONO OGGETTI, IL CHE SIGNIFICA CHE POSSONO ESSERE MANIPOLATI CON VARI METODI. ECCO ALCUNI DEI METODI PIÙ COMUNI DELLE STRINGHE IN JAVASCRIPT:

- **LENGTH**: RESTITUISCE LA LUNGHEZZA DELLA STRINGA.
- **TOLOWERCASE()**: RESTITUISCE LA STRINGA IN MINUSCOLO.
- **TOUPPERCASE()**: RESTITUISCE LA STRINGA IN MAIUSCOLO.
- **INDEXOF(SUBSTRING)**: RESTITUISCE L'INDICE DELLA PRIMA OCCORRENZA DI UNA SOTTOSTRINGA ALL'INTERNO DELLA STRINGA, 0 -1 SE LA SOTTOSTRINGA NON VIENE TROVATA.
- **SUBSTRING(START, END)**: RESTITUISCE UNA SOTTOSTRINGA DALLA STRINGA, CON INDICE DI INIZIO E DI FINE SPECIFICATI.
- **REPLACE(OLDVALUE, NEWVALUE)**: SOSTITUISCE LA PRIMA OCCORRENZA DI UNA SOTTOSTRINGA CON UN'ALTRA SOTTOSTRINGA ALL'INTERNO DELLA STRINGA.
- **SPLIT(SEPARATOR)**: SUDDIVIDE LA STRINGA IN UN ARRAY DI SOTTOSTRINGHE IN BASE AL SEPARATORE SPECIFICATO.



STRINGHE: METODI - ESEMPIO

• • •

Esempio

```
let nome = 'Alice';

console.log(nome.length); // stampa 5

let nomeMaiuscolo = nome.toUpperCase();
console.log(nomeMaiuscolo); // stampa "ALICE"

let saluto = 'Ciao, mondo!';
let indiceVirgola = saluto.indexOf(',');
console.log(saluto.substring(0, indiceVirgola)); // stampa "Ciao"

let nuovoSaluto = saluto.replace('mondo', 'Alice');
console.log(nuovoSaluto); // stampa "Ciao, Alice!"

let parole = saluto.split(',');
console.log(parole); // stampa ["Ciao", " mondo!"]
```

I NUMERI IN JAVASCRIPT

JS



NUMERI

IN JAVASCRIPT, I NUMERI POSSONO ESSERE RAPPRESENTATI IN DIVERSI FORMATI, TRA CUI NUMERI INTERI, DECIMALI E NOTAZIONE SCIENTIFICA.

AD ESEMPIO:



Esempio

```
let intero = 42;  
let decimale = 3.14;  
let scientifico = 1e6; // rappresenta 1 milione
```



NUMERI: UTILIZZO CON OPERATORI MATEMATICI



Esempio

```
let x = 10;  
let y = 5;  
  
let somma = x + y; // 15  
let differenza = x - y; // 5  
let prodotto = x * y; // 50  
let quoziente = x / y; // 2  
let resto = x % y; // 0  
let potenza = x ** 2; // 100
```



NUMERI: METODI

IN JAVASCRIPT, CI SONO ANCHE DIVERSI METODI DISPONIBILI PER LAVORARE CON I NUMERI, COME AD ESEMPIO:

- **NUMBERPARSEINT(STRINGA)**: CONVERTE UNA STRINGA IN UN NUMERO INTERO.
- **NUMBERPARSEFLOAT(STRINGA)**: CONVERTE UNA STRINGA IN UN NUMERO DECIMALE.
- **NUMBERISINTEGER(NUMERO)**: RESTITUISCE TRUE SE IL NUMERO È UN INTERO, FALSE ALTRIMENTI.
- **NUMBERISNAN(NUMERO)**: RESTITUISCE TRUE SE IL NUMERO NON È UN NUMERO VALIDO (AD ESEMPIO, NAN), FALSE ALTRIMENTI.

BOOLEANI

JS



BOOLEANI

IN JAVASCRIPT, I VALORI BOOLEANI RAPPRESENTANO LA VERITÀ O LA FALSITÀ DI UN'ESPRESSIONE. CI SONO SOLO DUE VALORI BOOLEANI IN JAVASCRIPT: TRUE E FALSE. QUESTI VALORI SONO SPESSO UTILIZZATI IN CONDIZIONI PER CONTROLLARE IL FLUSSO DEL CODICE.

• • • Esempio

```
let x = 5;
let y = 10;

if (x < y) {
  console.log('x è minore di y');
} else {
  console.log('x è maggiore o uguale a y');
}
```



BOOLEANI

IN JAVASCRIPT, I VALORI BOOLEANI RAPPRESENTANO LA VERITÀ O LA FALSITÀ DI UN'ESPRESSONE. CI SONO SOLO DUE VALORI BOOLEANI IN JAVASCRIPT: TRUE E FALSE. QUESTI VALORI SONO SPESSO UTILIZZATI IN CONDIZIONI PER CONTROLLARE IL FLUSSO DEL CODICE.

...
Esempio

```
let x = 5;
let y = 10;

if (x < y) {
  console.log('x è minore di y');
} else {
  console.log('x è maggiore o uguale a y');
}
```

IN QUESTO ESEMPIO, L'ESPRESSONE X < Y RESTITUISCE TRUE PERCHÉ 5 È INFERIORE A 10. PERTANTO, IL CODICE ALL'INTERNO DEL BLOCCO IF VERRÀ ESEGUITO E LA STRINGA "X È MINORE DI Y" VERRÀ STAMPATA SULLA CONSOLE.



BOOLEANI

CI SONO ANCHE OPERATORI BOOLEANI IN JAVASCRIPT, CHE POSSONO ESSERE UTILIZZATI PER COMBINARE VALORI BOOLEANI. GLI OPERATORI BOOLEANI INCLUDONO:

- **&& (AND)**: RESTITUISCE TRUE SE ENTRAMBE LE ESPRESSIONI SONO TRUE.
- **|| (OR)**: RESTITUISCE TRUE SE ALMENO UNA DELLE DUE ESPRESSIONI È TRUE.
- **! (NOT)**: RESTITUISCE IL VALORE OPPOSTO DELL'ESPRESSIONE.



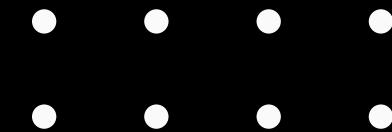
BOOLEANI: ESEMPIO

• • •

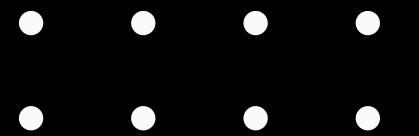
Esempio

```
let a = true;  
let b = false;
```

```
console.log(a && b); // stampa false  
console.log(a || b); // stampa true  
console.log(!b); // stampa true
```



GLI OGGETTI IN JAVASCRIPT





GLI OGGETTI IN JAVASCRIPT

IN JAVASCRIPT, UN **OGGETTO** È UN TIPO DI DATO COMPLESSO CHE RAGGRUPPA DATI E FUNZIONALITÀ CORRELATI IN UNA STRUTTURA. GLI OGGETTI SONO UNA PARTE FONDAMENTALE DEL LINGUAGGIO JAVASCRIPT E SONO UTILIZZATI AMPIAMENTE NEL CODICE JAVASCRIPT MODERNO. SONO COSTITUITI DA PROPRIETÀ E METODI.

LE PROPRIETÀ SONO COME VARIABILI CHE SONO ASSOCiate ALL'OGGETTO E CONTENGONO UN VALORE.

I METODI SONO FUNZIONI CHE SONO ASSOCiate ALL'OGGETTO E POSSONO ESSERE UTILIZZATE PER MANIPOLARE I DATI DELL'OGGETTO O ESEGUIRE OPERAZIONI SPECIFICHE.



GLI OGGETTI IN JAVASCRIPT

UN OGGETTO PUÒ ESSERE CREATO UTILIZZANDO UNA DELLE SEGUENTI SINTASSI:

SINTASSI LETTERALE DELL'OGGETTO:



Esempio

```
var objectName = {propName1: propValue1,  
propName2: propValue2};
```

UTILIZZANDO LA PAROLA CHIAVE "NEW" E IL COSTRUTTORE OBJECT:



Esempio

```
var objectName = new Object();  
objectName.propName1 = propValue1;  
objectName.propName2 = propValue2;
```



GLI OGGETTI IN JAVASCRIPT: ESEMPIO

AD ESEMPIO, PUOI CREARE UN OGGETTO "PERSON" CHE HA PROPRIETÀ COME NOME, COGNOME E ETÀ, OLTRE A METODI:

```
● ● ●          Esempio

var person = {
  firstName: "Mario",
  lastName: "Rossi",
  age: 30,
  greet: function() {
    console.log("Ciao, mi chiamo " + this.firstName + " " +
this.lastName + " e ho " + this.age + " anni.");
  },
  getFullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

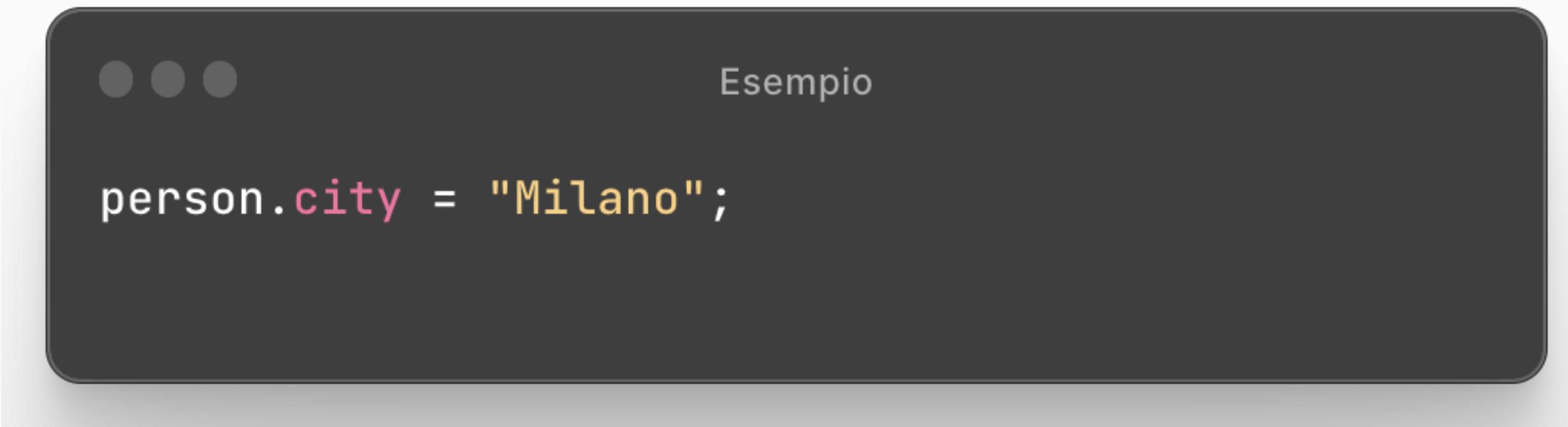
person.greet(); // Output: Ciao, mi chiamo Mario Rossi e ho 30 anni.
console.log(person.getFullName()); // Output: Mario Rossi
```

IN QUESTO ESEMPIO, L'OGGETTO "PERSON" HA TRE PROPRIETÀ: "FIRSTNAME", "LASTNAME" E "AGE", E DUE METODI: "GREET" E "GETFULLNAME". IL METODO "GREET" UTILIZZA LE PROPRIETÀ DELL'OGGETTO PER SALUTARE LA PERSONA, MENTRE IL METODO "GETFULLNAME" RESTITUISCE IL NOME COMPLETO DELLA PERSONA.



GLI OGGETTI IN JAVASCRIPT: ESEMPIO

PUOI ANCHE AGGIUNGERE O RIMUOVERE PROPRIETÀ E METODI DA UN OGGETTO IN QUALSIASI MOMENTO. AD ESEMPIO, PUOI AGGIUNGERE UNA NUOVA PROPRIETÀ ALL'OGGETTO "PERSON" IN QUESTO MODO:

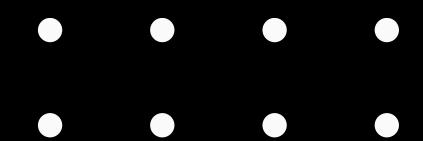
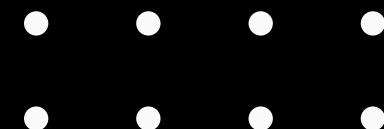


Esempio

```
person.city = "Milano";
```

L'OGGETTO "PERSON" AVRÀ UNA NUOVA PROPRIETÀ "CITY" CON IL VALORE "MILANO".

ARRAY



JS



ARRAY

GLI **ARRAY** IN JAVASCRIPT SONO COLLEZIONI CHE POSSONO CONTENERE PIÙ VALORI DI DATI CORRELATI IN UNA SINGOLA VARIABILE. GLI ARRAY SONO UTILIZZATI PER MEMORIZZARE E MANIPOLARE INSIEMI DI DATI, COME NUMERI, STRINGHE O OGGETTI, IN MODO EFFICACE E ORGANIZZATO. SONO RAPPRESENTATI DA UNA LISTA ORDINATA DI ELEMENTI, CHE POSSONO ESSERE DI QUALSIASI TIPO DI DATO VALIDO IN JAVASCRIPT. OGNI ELEMENTO DELL'ARRAY È IDENTIFICATO DA UN INDICE, CHE RAPPRESENTA LA SUA POSIZIONE NELL'ARRAY. GLI INDICI DELL'ARRAY INIZIANO DA 0, IL CHE SIGNIFICA CHE IL PRIMO ELEMENTO DELL'ARRAY HA UN INDICE DI 0, IL SECONDO ELEMENTO HA UN INDICE DI 1, E COSÌ VIA.



ARRAY

UN ARRAY IN JAVASCRIPT PUÒ ESSERE CREATO UTILIZZANDO LA SEGUENTE SINTASSI:



Esempio

```
var myArray = [element1, element2, element3, ...]
```



ARRAY

AD ESEMPIO, PUOI CREARE UN ARRAY DI NUMERI COME QUESTO:

The screenshot shows a dark-themed code editor window. In the top right corner, there are three small gray dots. To the right of these dots, the word "Esempio" is written in white. Below this, the code `var numbers = [1, 2, 3, 4, 5];` is displayed in white text.

PER ACCEDERE AGLI ELEMENTI DELL'ARRAY, PUOI UTILIZZARE L'INDICE DELL'ELEMENTO. GLI INDICI DEGLI ARRAY IN JAVASCRIPT PARTONO DA ZERO, QUINDI IL PRIMO ELEMENTO DELL'ARRAY HA UN INDICE DI 0, IL SECONDO ELEMENTO HA UN INDICE DI 1, E COSÌ VIA. AD ESEMPIO, PER ACCEDERE AL SECONDO ELEMENTO DELL'ARRAY "NUMBERS", PUOI UTILIZZARE IL SEGUENTE CODICE:

The screenshot shows a dark-themed code editor window. In the top right corner, there are three small gray dots. To the right of these dots, the word "Esempio" is written in white. Below this, the code `var secondElement = numbers[1]; // 2` is displayed in white text.

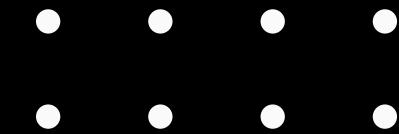


ARRAY

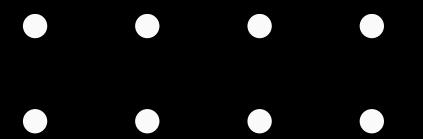
PUOI ANCHE AGGIUNGERE O RIMUOVERE ELEMENTI DALL'ARRAY IN QUALSIASI MOMENTO. AD ESEMPIO, PUOI AGGIUNGERE UN NUOVO ELEMENTO ALL'ARRAY "NUMBERS" IN QUESTO MODO:

The screenshot shows a dark-themed code editor window. In the top right corner, there are three small white dots. To the right of these dots, the word "Esempio" is written in a smaller white font. Below this, the code "numbers.push(6);" is written in a white monospaced font. The background of the code editor is dark gray.

IN QUESTO MODO, L'ARRAY "NUMBERS" AVRÀ UN NUOVO ELEMENTO 6 ALLA FINE DELL'ARRAY. GLI ARRAY IN JAVASCRIPT HANNO ANCHE NUMEROSI METODI INCORPORATI CHE POSSONO ESSERE UTILIZZATI PER MANIPOLARE GLI ELEMENTI DELL'ARRAY. ALCUNI ESEMPI DI METODI INCLUDONO "SLICE", "SPLICE", "CONCAT", "INDEXOF", "MAP", "FILTER", E MOLTI ALTRI.



LE FUNZIONI IN JAVASCRIPT



JS



LE FUNZIONI IN JAVASCRIPT

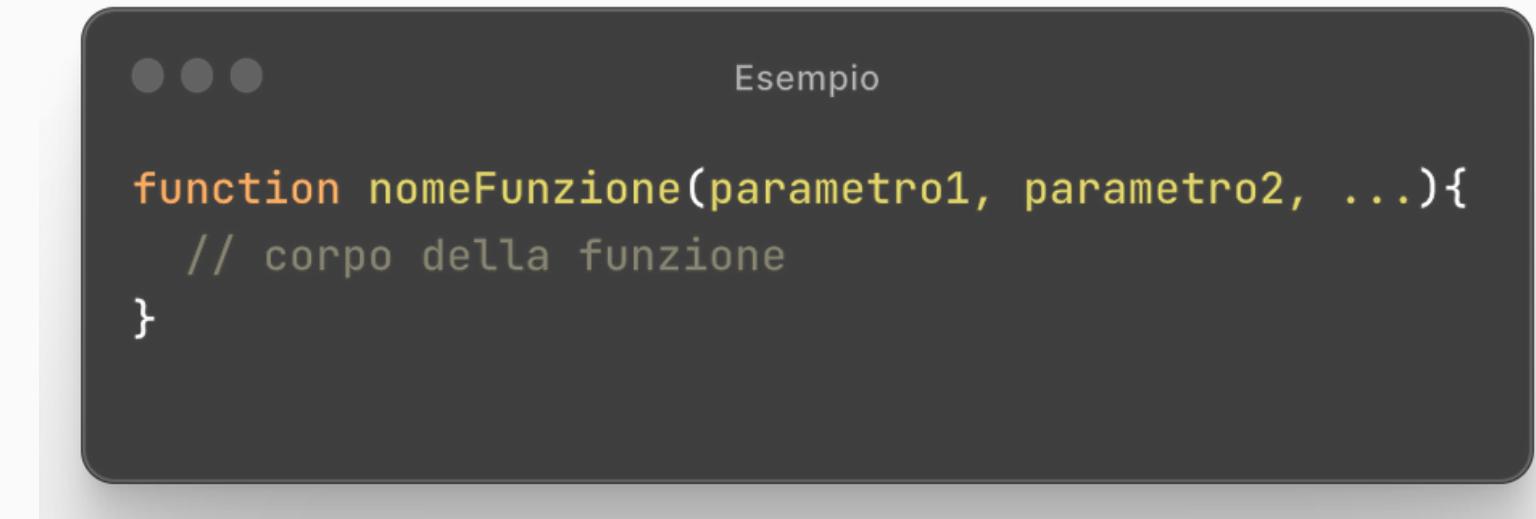
LE **FUNZIONI** IN JAVASCRIPT SONO BLOCCHI DI CODICE CHE ESEGUONO UN'AZIONE SPECIFICA QUANDO VENGONO RICHIAMATI.

LE FUNZIONI SONO UTILIZZATE PER ORGANIZZARE IL CODICE IN MODO CHE POSSA ESSERE RIUTILIZZATO, FACILITANDO LA MANUTENZIONE E LA COMPRENSIONE DEL CODICE.



LE FUNZIONI IN JAVASCRIPT

CI SONO DIVERSI MODI PER DEFINIRE LE FUNZIONI IN JAVASCRIPT. LA SINTASSI PIÙ COMUNE È LA SEGUENTE:



Esempio

```
function nomeFunzione(parametro1, parametro2, ...){  
    // corpo della funzione  
}
```

IN QUESTA SINTASSI, NOMEFUNZIONE È IL NOME DELLA FUNZIONE, MENTRE PARAMETRO1, PARAMETRO2, ECC. SONO I PARAMETRI CHE LA FUNZIONE ACCETTA. IL CORPO DELLA FUNZIONE È IL BLOCCO DI CODICE CHE VERRÀ ESEGUITO QUANDO LA FUNZIONE VIENE RICHIAMATA.



LE FUNZIONI IN JAVASCRIPT

ECCO UN ESEMPIO DI FUNZIONE CHE ACCETTA DUE PARAMETRI E RESTITUISCE LA LORO SOMMA:

```
function somma(a, b) {  
    return a + b;  
}
```

PER RICHIAMARE QUESTA FUNZIONE, È SUFFICIENTE SCRIVERE IL NOME
DELLA FUNZIONE SEGUITO DAI PARAMETRI TRA PARENTESI:

Esempio

```
let risultato = somma(2, 3); // il valore di risultato sarà 5
```



LE FUNZIONI IN JAVASCRIPT

LE FUNZIONI POSSONO ANCHE ESSERE ASSEGNAME A VARIABILI, COME IN QUESTO ESEMPIO:

Esempio

```
let quadrato = function(x) {
    return x * x;
};

let risultato = quadrato(3); // il valore di risultato sarà 9
```

N QUESTO CASO, LA FUNZIONE VIENE ASSEGNATA ALLA VARIABILE "QUADRATO".



LE FUNZIONI IN JAVASCRIPT

LE FUNZIONI POSSONO ANCHE ESSERE DEFINITE COME ESPRESSIONI DI FRECCIA, CHE È UNA SINTASSI PIÙ BREVE PER DEFINIRE FUNZIONI ANONIME. ECCO UN ESEMPIO:

The screenshot shows a dark-themed code editor window. In the top right corner, there is a small white box containing three dots and the word "Esempio". The main area contains the following JavaScript code:

```
let cubo = x => x * x * x;  
  
let risultato = cubo(2); // il valore di risultato sarà 8
```

IN QUESTO CASO, LA FUNZIONE È DEFINITA COME UNA SINGOLA
ESPRESSIONE DOPO LA FRECCIA =>.

LA FUNZIONE ACCETTA UN PARAMETRO X E RESTITUISCE IL SUO CUBO.



LE FUNZIONI IN JAVASCRIPT

LE FUNZIONI POSSONO ANCHE ACCEDERE ALLE VARIABILI DEFINITE AL DI FUORI DELLA FUNZIONE STESSA. ECCO UN ESEMPIO:

Esempio

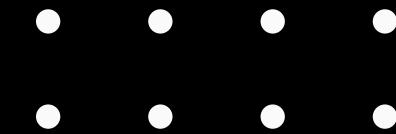
```
let valoreIniziale = 0;

function incrementa() {
    valoreIniziale++;
    console.log(valoreIniziale);
}

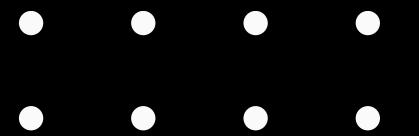
incrementa(); // stampa "1"
incrementa(); // stampa "2"
```

IN QUESTO CASO, LA VARIABILE VALOREINIZIALE VIENE DEFINITA AL DI FUORI DELLA FUNZIONE INCREMENTA.

TUTTAVIA, LA FUNZIONE PUÒ COMUNQUE ACCEDERE E MODIFICARE QUESTA VARIABILE



BLOCCHI CONDIZIONALI





BLOCCHI CONDIZIONALI

I BLOCCHI CONDIZIONALI **IF-ELSE** E **SWITCH** SONO DUE STRUMENTI FONDAMENTALI DI CONTROLLO DEL FLUSSO IN JAVASCRIPT CHE CONSENTONO DI ESEGUIRE CODICE IN BASE A DETERMINATE CONDIZIONI.



BLOCCHI CONDIZIONALI: IF-ELSE

IL BLOCCO CONDIZIONALE **IF-ELSE** VIENE UTILIZZATO PER ESEGUIRE UN BLOCCO DI CODICE IN BASE AL VERIFICARSI O meno DI UNA CONDIZIONE.

ECCO UN ESEMPIO:

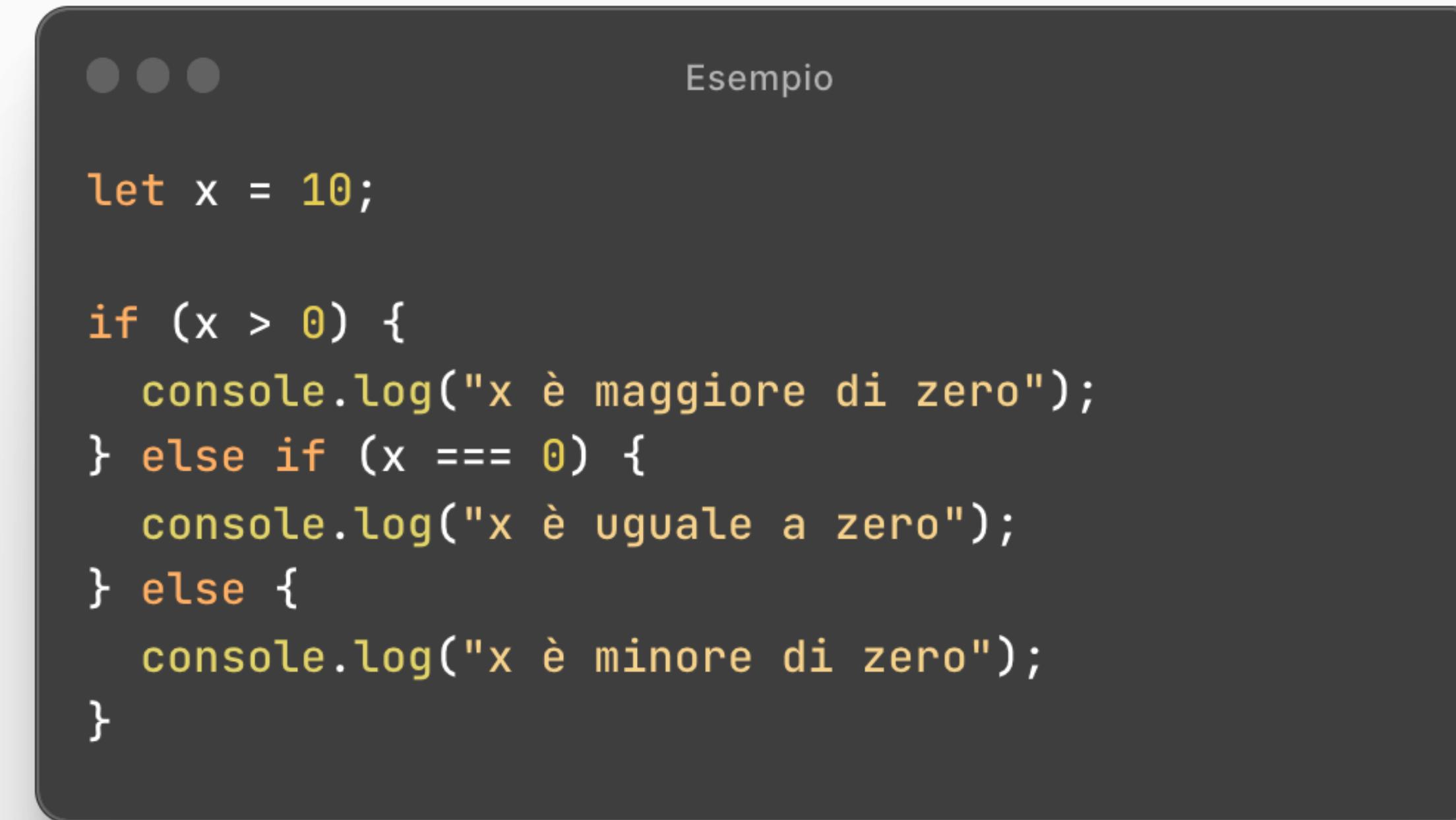
```
...
Esempio
let x = 10;
if (x > 0) {
  console.log("x è maggiore di zero");
} else {
  console.log("x è minore o uguale a zero");
}
```

IN QUESTO CASO, IL BLOCCO DI CODICE DENTRO L'IF VIENE ESEGUITO SOLO SE LA CONDIZIONE TRA PARENTESI È VERA, ALTRIMENTI VIENE ESEGUITO IL BLOCCO DI CODICE DENTRO L'ELSE.



BLOCCHI CONDIZIONALI: IF-ELSE

L'IF-ELSE PUÒ ANCHE ESSERE UTILIZZATO CON CONDIZIONI ANNIDATE,
COME IN QUESTO ESEMPIO:



Esempio

```
let x = 10;

if (x > 0) {
  console.log("x è maggiore di zero");
} else if (x === 0) {
  console.log("x è uguale a zero");
} else {
  console.log("x è minore di zero");
}
```



BLOCCHI CONDIZIONALI: SWITCH

IL BLOCCO CONDIZIONALE **SWITCH** È UN COSTRUTTO DI CONTROLLO DEL FLUSSO DI ESECUZIONE IN JAVASCRIPT, CHE CONSENTE DI ESEGUIRE UN BLOCCO DI CODICE DIVERSO IN BASE AL VALORE DI UNA ESPRESSIONE.



Esempio

```
switch (espressione) {  
    case valore1:  
        // blocco di codice da eseguire se  
        l'espressione è uguale a valore1  
        break;  
    case valore2:  
        // blocco di codice da eseguire se  
        l'espressione è uguale a valore2  
        break;  
    ...  
    default:  
        // blocco di codice da eseguire se  
        l'espressione non corrisponde a nessun valore  
        break;  
}
```

IN QUESTA SINTASSI, ESPRESSIONE È UN'ESPRESSONE JAVASCRIPT IL CUI VALORE VIENE CONFRONTATO CON I VARI VALORE1, VALORE2, ECC.

QUANDO SI TROVA UN VALORE CORRISPONDENTE, IL BLOCCO DI CODICE ASSOCIAZIO VIENE ESEGUITO, E L'ESECUZIONE PASSA AL BLOCCO SUCCESSIVO DOPO LA PAROLA CHIAVE BREAK.

SE NESSUN VALORE CORRISPONDE ALL'ESPRESSONE, IL BLOCCO DEFAULT VIENE ESEGUITO.



BLOCCHI CONDIZIONALI: SWITCH

ECCO UN ESEMPIO DI BLOCCO
SWITCH CHE DETERMINA IL
GIORNO DELLA SETTIMANA IN
BASE A UN NUMERO:

```
● ● ● Esempio

let giorno = 3;
let nomeGiorno;

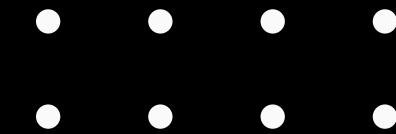
switch (giorno) {
  case 1:
    nomeGiorno = "Lunedì";
    break;
  case 2:
    nomeGiorno = "Martedì";
    break;
  case 3:
    nomeGiorno = "Mercoledì";
    break;
  case 4:
    nomeGiorno = "Giovedì";
    break;
  case 5:
    nomeGiorno = "Venerdì";
    break;
  case 6:
    nomeGiorno = "Sabato";
    break;
  case 7:
    nomeGiorno = "Domenica";
    break;
  default:
    nomeGiorno = "Valore non valido";
    break;
}

console.log(nomeGiorno); // stampa "Mercoledì"
```

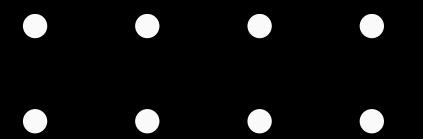


BLOCCHI CONDIZIONALI: SWITCH

IN CONCLUSIONE, IL BLOCCO CONDIZIONALE SWITCH È UN COSTRUTTO DI CONTROLLO DEL FLUSSO DI ESECUZIONE CHE PERMETTE DI ESEGUIRE UN BLOCCO DI CODICE DIVERSO IN BASE AL VALORE DI UN'ESPRESSione. LA SINTASSI DI BASE CONSISTE IN UNA SERIE DI CASI (CASE) SEGUiti DA UN BLOCCO DI CODICE DA ESEGUIRE E DALLA PAROLA CHIAVE BREAK. SE NESSUN CASO CORRISPONDE ALL'ESPRESSione, VIENE ESEGUITO IL BLOCCO DEFAULT.



IL CICLO "FOR" IN JAVASCRIPT



JS



IL CICLO “FOR” IN JAVASCRIPT

I CICLI **FOR** IN JAVASCRIPT SONO UN COSTRUTTO FONDAMENTALE PER ITERARE SU UN INSIEME DI VALORI O ESEGUIRE UN BLOCCO DI CODICE UN CERTO NUMERO DI VOLTE.

ESISTONO DIVERSE VARIANTI DI CICLI FOR IN JAVASCRIPT, MA LA SINTASSI DI BASE È LA SEGUENTE:



Esempio

```
for (inizializzazione; condizione; incremento) {  
    // blocco di codice da eseguire  
}
```



IL CICLO “FOR” IN JAVASCRIPT

• • •

Esempio

```
for (inizializzazione; condizione; espressioneIncremento) {  
    // blocco di codice da eseguire ripetutamente  
}
```

IN QUESTA SINTASSI, LA INIZIALIZZAZIONE È UNA ESPRESSIONE CHE VIENE ESEGUITA UNA SOLA VOLTA ALL'INIZIO DEL CICLO, PER INIZIALIZZARE LA VARIABILE DI CONTROLLO. LA CONDIZIONE È UNA ESPRESSIONE BOOLEANA CHE VIENE VALUTATA ALL'INIZIO DI OGNI ITERAZIONE, E SE È VERA IL BLOCCO DI CODICE VIENE ESEGUITO, ALTRIMENTI IL CICLO VIENE TERMINATO. L'ESPRESSI~~O~~NEINCREMENTO È UN'ESPRESSIONE CHE VIENE ESEGUITA ALLA FINE DI OGNI ITERAZIONE, PER MODIFICARE LA VARIABILE DI CONTROLLO.



IL CICLO “FOR” IN JAVASCRIPT

AD ESEMPIO, IL SEGUENTE CICLO FOR STAMPA I NUMERI DA 1 A 5:

The screenshot shows a dark-themed code editor window. In the top right corner, there is a small gray box with three dots and the word "Esempio". Below this, the code is displayed in white text on a dark background:

```
for (let i = 1; i <= 5; i++) {  
    console.log(i);  
}
```

IN QUESTO ESEMPIO, LA VARIABILE `i` VIENE INIZIALIZZATA A 1, E VIENE VALUTATA LA CONDIZIONE `i <= 5` ALL'INIZIO DI OGNI ITERAZIONE. SE LA CONDIZIONE È VERA, IL BLOCCO DI CODICE VIENE ESEGUITO, E ALLA FINE DI OGNI ITERAZIONE LA VARIABILE `i` VIENE INCREMENTATA DI 1.



IL CICLO “FOR” IN JAVASCRIPT

E POSSIBILE ANCHE UTILIZZARE IL CICLO FOR PER ITERARE SU UN ARRAY. IN QUESTO CASO, SI PUÒ UTILIZZARE LA PROPRIETÀ LENGTH DELL'ARRAY PER DEFINIRE LA CONDIZIONE DI USCITA DAL CICLO:

• • • Esempio

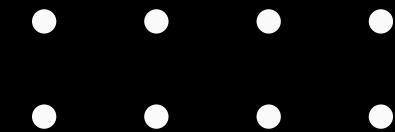
```
let numeri = [1, 2, 3, 4, 5];
for (let i = 0; i < numeri.length; i++) {
    console.log(numeri[i]);
}
```

IN QUESTO ESEMPIO, LA VARIABILE I VIENE INIZIALIZZATA A 0, E LA CONDIZIONE `I < NUMERI.LENGTH` VIENE VALUTATA ALL'INIZIO DI OGNI ITERAZIONE. IL BLOCCO DI CODICE VIENE ESEGUITO PER OGNI ELEMENTO DELL'ARRAY NUMERI, UTILIZZANDO L'INDICE I PER ACCEDERE ALL'ELEMENTO CORRENTE.

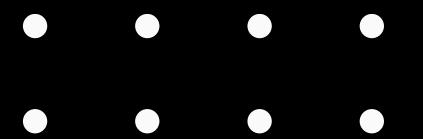


IL CICLO “FOR” IN JAVASCRIPT

IN CONCLUSIONE, IL CICLO FOR IN JAVASCRIPT È UN COSTRUTTO DI CONTROLLO DEL FLUSSO DI ESECUZIONE CHE PERMETTE DI ESEGUIRE UN BLOCCO DI CODICE RIPETUTAMENTE FINO A QUANDO UNA CERTA CONDIZIONE NON VIENE SODDISFATTA. LA SINTASSI DI BASE PREVEDE UN'ESPRESSONE DI INIZIALIZZAZIONE, UNA CONDIZIONE DI USCITA E UN'ESPRESSONE DI INCREMENTO. IL CICLO FOR PUÒ ESSERE UTILIZZATO PER ITERARE SU UN ARRAY O SU QUALESIASI TIPO DI OGGETTO CHE SUPPORTA LA PROPRIETÀ LENGTH.



IL CICLO "WHILE" IN JAVASCRIPT





IL CICLO “WHILE” IN JAVASCRIPT

IL CICLO **WHILE** IN JAVASCRIPT È UN COSTRUTTO DI CONTROLLO DEL FLUSSO DI ESECUZIONE CHE PERMETTE DI ESEGUIRE UN BLOCCO DI CODICE RIPETUTAMENTE FINCHÉ UNA DETERMINATA CONDIZIONE È VERA.

A DIFFERENZA DEL CICLO FOR, IL CICLO WHILE NON HA UN NUMERO PREDETERMINATO DI ITERAZIONI, MA CONTINUA AD ESEGUIRE IL BLOCCO DI CODICE FINCHÉ LA CONDIZIONE SPECIFICATA È VERA.



IL CICLO “WHILE” IN JAVASCRIPT

LA SINTASSI GENERALE DEL CICLO "WHILE" IN JAVASCRIPT È LA
SEGUENTE:

● ● ● Esempio

```
while (condizione) {  
    // blocco di codice da eseguire finché la condizione è vera  
}
```

LA "CONDIZIONE" È L'ESPRESSIONE VALUTATA IN OGNI ITERAZIONE DEL CICLO. SE LA CONDIZIONE È VERA, IL BLOCCO DI CODICE ALL'INTERNO DELLE PARENTESI GRAFFE VIENE ESEGUITO, ALTRIMENTI IL CICLO SI INTERROMPE E IL CONTROLLO PASSA ALLA SUCCESSIVA ISTRUZIONE NEL PROGRAMMA.



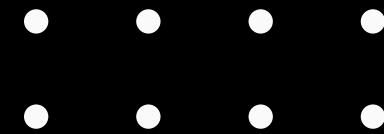
IL CICLO “WHILE” IN JAVASCRIPT

AD ESEMPIO, IL SEGUENTE CODICE UTILIZZA IL CICLO "WHILE" PER STAMPARE I NUMERI DA 1 A 5:

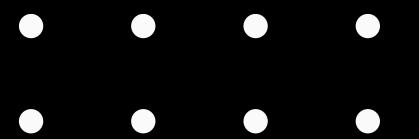
Esempio

```
var i = 1;  
while (i <= 5) {  
    console.log(i);  
    i++;  
}
```

IN QUESTO CASO, LA CONDIZIONE È "I <= 5", QUINDI IL CICLO VIENE ESEGUITO FINCHÉ LA VARIABILE "I" È MINORE O UGUALE A 5. AD OGNI ITERAZIONE DEL CICLO, VIENE STAMPATO IL VALORE DI "I" E POI VIENE INCREMENTATO DI 1.



IL CICLO "DO-WHILE" IN JAVASCRIPT





IL CICLO “DO-WHILE” IN JAVASCRIPT

IL CICLO "DO-WHILE" È SIMILE AL CICLO "WHILE", MA LA DIFFERENZA PRINCIPALE È CHE IL BLOCCO DI CODICE ALL'INTERNO DELLE PARENTESI GRAFFE VIENE ESEGUITO ALMENO UNA VOLTA, ANCHE SE LA CONDIZIONE È FALSA. IN ALTRE PAROLE, IL CICLO "DO-WHILE" ESEGUE IL BLOCCO DI CODICE ALMENO UNA VOLTA E POI VERIFICA LA CONDIZIONE PER DETERMINARE SE DEVE CONTINUARE AD ESEGUIRE IL BLOCCO DI CODICE O MENO.



IL CICLO "DO-WHILE" IN JAVASCRIPT

LA SINTASSI GENERALE DEL CICLO "DO-WHILE" IN JAVASCRIPT È LA SEGUENTE:

• • •

Esempio

```
do {  
    // blocco di codice da eseguire  
} while (condizione);
```

DOVE "CONDIZIONE" È L'ESPRESSONE VALUTATA ALLA FINE DI OGNI ITERAZIONE DEL CICLO. SE LA CONDIZIONE È VERA, IL CICLO RIPETE IL BLOCCO DI CODICE, ALTRIMENTI IL CICLO SI INTERROMPE E IL CONTROLLO PASSA ALLA SUCCESSIVA ISTRUZIONE NEL PROGRAMMA.



IL CICLO “DO-WHILE” IN JAVASCRIPT

AD ESEMPIO, IL SEGUENTE CODICE UTILIZZA IL CICLO "DO-WHILE" PER STAMPARE I NUMERI DA 1 A 5:

The screenshot shows a dark-themed code editor window. In the top right corner, there is a small icon with three dots and a label 'Esempio'. The main area contains the following JavaScript code:

```
var i = 1;
do {
    console.log(i);
    i++;
} while (i <= 5);
```

IN QUESTO CASO, LA CONDIZIONE È "I <= 5", QUINDI IL CICLO VIENE ESEGUITO FINCHÉ LA VARIABILE "I" È MINORE O UGUALE A 5. TUTTAVIA, A DIFFERENZA DEL CICLO "WHILE", IL BLOCCO DI CODICE ALL'INTERNO DELLE PARENTESI GRAFFE VIENE ESEGUITO ALMENO UNA VOLTA, ANCHE SE LA CONDIZIONE È FALSA.



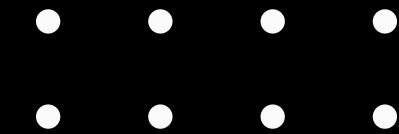
IL CICLO "DO-WHILE" IN JAVASCRIPT

AD ESEMPIO, IL SEGUENTE CODICE UTILIZZA IL CICLO "DO-WHILE" PER STAMPARE I NUMERI DA 1 A 5:

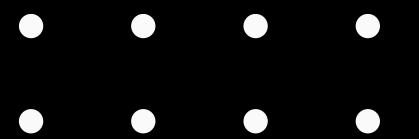
The screenshot shows a dark-themed code editor window. In the top right corner, there are three small circular icons. To the right of the window, the word "Esempio" is written in a light gray font. Inside the window, the following JavaScript code is displayed:

```
var i = 1;
do {
    console.log(i);
    i++;
} while (i <= 5);
```

IN QUESTO CASO, LA CONDIZIONE È "I <= 5", QUINDI IL CICLO VIENE ESEGUITO FINCHÉ LA VARIABILE "I" È MINORE O UGUALE A 5. TUTTAVIA, A DIFFERENZA DEL CICLO "WHILE", IL BLOCCO DI CODICE ALL'INTERNO DELLE PARENTESI GRAFFE VIENE ESEGUITO ALMENO UNA VOLTA, ANCHE SE LA CONDIZIONE È FALSA.



IL BLOCCO "TRY/CATCH"





IL BLOCCO "TRY/CATCH"

IL BLOCCO "**TRY/CATCH**" VIENE UTILIZZATO PER GESTIRE GLI ERRORI IN JAVASCRIPT. IN PARTICOLARE, IL BLOCCO "TRY" CONSENTE DI DEFINIRE UN BLOCCO DI CODICE IN CUI SI TENTA DI ESEGUIRE UN'OPERAZIONE CHE POTREBBE GENERARE UN ERRORE, MENTRE IL BLOCCO "CATCH" CONSENTE DI DEFINIRE UN BLOCCO DI CODICE DA ESEGUIRE IN CASO DI ERRORE.



IL BLOCCO "TRY/CATCH"

LA SINTASSI GENERALE DEL BLOCCO "TRY/CATCH" IN JAVASCRIPT È LA SEGUENTE:

• • •

Esempio

```
try {  
    // blocco di codice in cui si tenta di eseguire  
    un'operazione che potrebbe generare un errore  
} catch (errore) {  
    // blocco di codice da eseguire in caso di errore  
}
```

DOVE "ERRORE" È LA VARIABILE CHE CONTIENE L'OGGETTO ERRORE GENERATO NEL BLOCCO "TRY". SE DURANTE L'ESECUZIONE DEL BLOCCO "TRY" VIENE GENERATO UN ERRORE, IL CONTROLLO PASSA AL BLOCCO "CATCH" E VIENE ESEGUITO IL BLOCCO DI CODICE AL SUO INTERNO. IN QUESTO MODO È POSSIBILE GESTIRE L'ERRORE IN MODO APPROPRIATO E CONTINUARE L'ESECUZIONE DEL PROGRAMMA SENZA CHE SI INTERROMPA DEL TUTTO.



IL BLOCCO "TRY/CATCH"

AD ESEMPIO, IL SEGUENTE CODICE UTILIZZA IL BLOCCO "TRY/CATCH" PER GESTIRE UN'OPERAZIONE DI DIVISIONE PER ZERO:

● ● ●

Esempio

```
try {
  var x = 1/0;
  console.log(x);
} catch (errore) {
  console.log("Errore: " + errore.message);
}
```

IN QUESTO CASO, VIENE TENTATO DI ESEGUIRE UN'OPERAZIONE DI DIVISIONE PER ZERO, CHE GENERA UN ERRORE DI TIPO "DIVISIONE PER ZERO". IL BLOCCO "TRY" ESEGUE COMUNQUE IL CODICE, MA QUANDO VIENE GENERATO L'ERRORE IL CONTROLLO PASSA AL BLOCCO "CATCH". IL BLOCCO "CATCH" STAMPA UN MESSAGGIO DI ERRORE CHE INCLUDE IL MESSAGGIO DELL'ERRORE GENERATO.

I TIPI DI FUNZIONE



I TIPI DI FUNZIONE

IN JAVASCRIPT, CI SONO DIVERSI MODI PER DEFINIRE E UTILIZZARE LE FUNZIONI. ECCO UNA PANORAMICA DEI PRINCIPALI TIPI DI FUNZIONI:



I TIPI DI FUNZIONE

FUNZIONI DEFINITE CON LA PAROLA CHIAVE "FUNCTION": LE FUNZIONI DEFINITE CON LA PAROLA CHIAVE "FUNCTION" SONO IL TIPO PIÙ COMUNE DI FUNZIONI IN JAVASCRIPT. HANNO UNA SINTASSI SIMILE ALLA SEGUENTE:

```
● ● ● Esempio

function nomeFunzione(argomento1, argomento2) {
    // blocco di codice
    return risultato;
}
```

DOVE "NOMEFUNZIONE" È IL NOME DELLA FUNZIONE, "ARGOMENTO1" E "ARGOMENTO2" SONO GLI ARGOMENTI DELLA FUNZIONE (OPZIONALI) E "RISULTATO" È IL VALORE RESTITUITO DALLA FUNZIONE (OPZIONALE).



I TIPI DI FUNZIONE

FUNZIONI ANONIME: LE FUNZIONI ANONIME SONO FUNZIONI SENZA NOME CHE VENGONO ASSEGNAME A UNA VARIABILE O PASSATE COME ARGOMENTO AD UN'ALTRA FUNZIONE. HANNO UNA SINTASSI SIMILE ALLA SEGUENTE:

• • • Esempio

```
var funzioneAnonima = function(argomento1, argomento2) {  
    // blocco di codice  
    return risultato;  
}
```

DOVE "FUNZIONEANONIMA" È IL NOME DELLA VARIABILE A CUI VIENE ASSEGNATA LA FUNZIONE.



I TIPI DI FUNZIONE

ARROW FUNCTIONS: LE ARROW FUNCTIONS SONO UNA SINTASSI PIÙ BREVE PER LE FUNZIONI IN JAVASCRIPT, INTRODOTTE CON ECMASCIRIPT 6. HANNO UNA SINTASSI SIMILE ALLA SEGUENTE:

Esempio

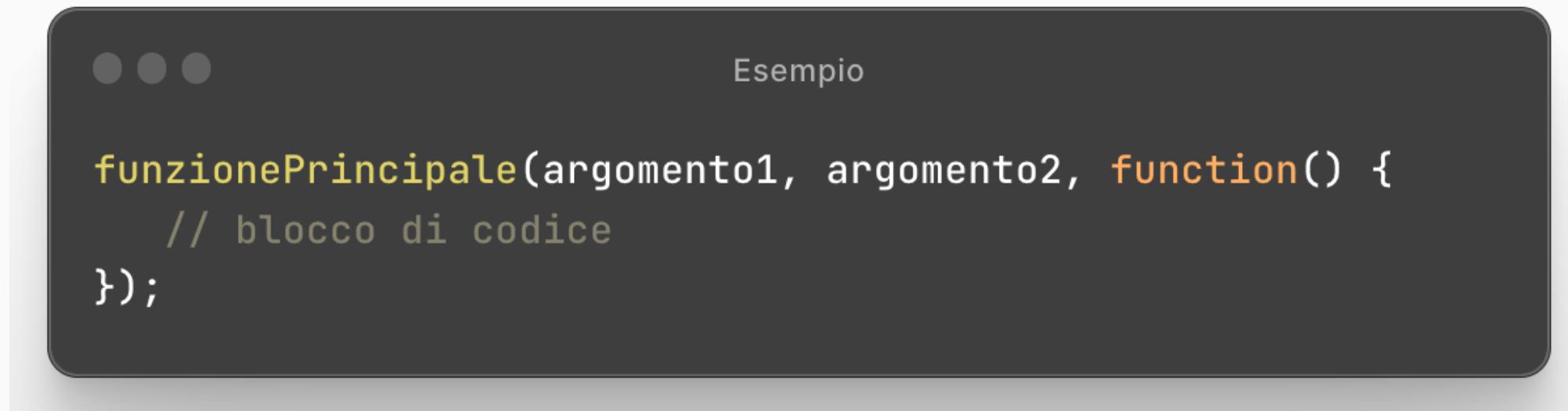
```
var arrowFunction = (argomento1, argomento2) => {  
    // blocco di codice  
    return risultato;  
}
```

DOVE "ARROWFUNCTION" È IL NOME DELLA VARIABILE A CUI VIENE ASSEGNATA LA FUNZIONE. IN QUESTO CASO, LA SINTASSI DELL'ARROW FUNCTION PREVEDE LA FRECCIA " $=>$ " PER SEPARARE GLI ARGOMENTI E IL BLOCCO DI CODICE.



I TIPI DI FUNZIONE

FUNZIONI DI CALLBACK: LE FUNZIONI DI CALLBACK SONO FUNZIONI PASSATE COME ARGOMENTO AD UN'ALTRA FUNZIONE E CHIAMATE DA QUELLA FUNZIONE. SONO SPESSO UTILIZZATE PER GESTIRE EVENTI O PER ESEGUIRE OPERAZIONI ASINCRONE. HANNO UNA SINTASSI SIMILE ALLA SEGUENTE:



Esempio

```
• • •  
funzionePrincipale(argomento1, argomento2, function() {  
    // blocco di codice  
});
```

DOVE LA FUNZIONE DI CALLBACK È DEFINITA COME UNA FUNZIONE ANONIMA ALL'INTERNO DEGLI ARGOMENTI DELLA FUNZIONE PRINCIPALE.

PARAMETRI E CALLBACK



PARAMETRI E CALLBACK

I **PARAMETRI** IN JAVASCRIPT SONO VALORI CHE VENGONO PASSATI AD UNA FUNZIONE QUANDO VIENE CHIAMATA.

UNA FUNZIONE PUÒ ACCETTARE ZERO O PIÙ PARAMETRI, CHE POSSONO ESSERE DI QUALSIASI TIPO DI DATO JAVASCRIPT.

I PARAMETRI SONO SEPARATI DA VIRGOLE ALL'INTERNO DELLE PARENTESI TONDE DELLA DICHIARAZIONE DELLA FUNZIONE.



PARAMETRI E CALLBACK

AD ESEMPIO, QUESTA È LA SINTASSI DI UNA FUNZIONE CHE ACCETTA DUE PARAMETRI:

Esempio

```
function miaFunzione(parametro1, parametro2) {  
    // corpo della funzione  
}
```



PARAMETRI E CALLBACK

LE **CALLBACK**, INVECE, SONO FUNZIONI CHE VENGONO PASSATE COME PARAMETRI AD ALTRE FUNZIONI. LE CALLBACK SONO SPESSO UTILIZZATE IN JAVASCRIPT PER ESEGUIRE AZIONI ASINCRONE O PER GESTIRE EVENTI.

IN QUESTO MODO, LA FUNZIONE CHE RICEVE LA CALLBACK PUÒ CHIAMARLA QUANDO NECESSARIO, O QUANDO SI VERIFICA UN EVENTO SPECIFICO.



PARAMETRI E CALLBACK

AD ESEMPIO, QUESTO È UN ESEMPIO DI FUNZIONE CHE ACCETTA UNA CALLBACK COME PARAMETRO:

```
● ● ● Esempio

function eseguiOperazioneAsync(callback) {
  // Simuliamo un'operazione asincrona che richiede tempo
  setTimeout(function() {
    // Quando l'operazione è completata, chiamiamo la callback
    callback();
  }, 1000);
}

// Chiamiamo la funzione "eseguiOperazioneAsync" passando una
// callback come parametro
eseguiOperazioneAsync(function() {
  console.log('L\'operazione è stata completata con successo!');
});
```

IN QUESTO ESEMPIO, LA FUNZIONE "ESEGUOPERAZIONEASYNC" ACCETTA UNA CALLBACK COME PARAMETRO E LA CHIAMA DOPO UN SECONDO, QUANDO L'OPERAZIONE SIMULATA È STATA COMPLETATA. LA CALLBACK È DEFINITA COME UNA FUNZIONE ANONIMA PASSATA COME PARAMETRO ALLA FUNZIONE "ESEGUOPERAZIONEASYNC". QUANDO LA CALLBACK VIENE CHIAMATA, IL MESSAGGIO "L'OPERAZIONE È STATA COMPLETATA CON SUCCESSO!" VIENE STAMPATO SULLA CONSOLE DEL BROWSER.

CLASSI

⋮ ⋮ ⋮ ⋮

JS



CLASSI

LE **CLASSI** IN JAVASCRIPT SONO UNO STRUMENTO CHE CONSENTE DI CREARE OGGETTI CON PROPRIETÀ E METODI. RAPPRESENTANO UNA NOVITÀ INTRODOTTA CON L'AGGIORNAMENTO DEL LINGUAGGIO ECMASCIRIPT 6.



CLASSI

IN QUESTO ESEMPIO, ABBIAMO DEFINITO UNA CLASSE "PERSONA" CON UN COSTRUTTORE CHE ACCETTA TRE PARAMETRI: NOME, COGNOME ED ETA. ABBIAMO INOLTRE DEFINITO UN METODO "PRESENTATI" CHE LOGGA UNA STRINGA CONTENENTE IL NOME, IL COGNOME E L'ETA DELL'OGGETTO.

Esempio

```
class Persona {  
    constructor(nome, cognome, eta) {  
        this.nome = nome;  
        this.cognome = cognome;  
        this.eta = eta;  
    }  
  
    presentati() {  
        console.log("Ciao, sono " + this.nome + " " +  
this.cognome + " e ho " + this.eta + " anni.");  
    }  
}
```



CLASSI

PER CREARE UNA NUOVA Istanza DELLA CLASSE "PERSONA" UTILIZZIAMO LA SINTASSI "**NEW**":

Esempio

```
let persona1 = new Persona("Mario", "Rossi", 35);
persona1.presentati(); // "Ciao, sono Mario Rossi e ho 35 anni."
```

IN QUESTO ESEMPIO, ABBIAMO CREATO UNA NUOVA Istanza DELLA CLASSE "PERSONA" CHIAMATA "PERSONA1" UTILIZZANDO LA SINTASSI "NEW". ABBIAMO POI CHIAMATO IL METODO "PRESENTATI" DELL'OGGETTO "PERSONA1".



CLASSI

UN ALTRO ESEMPIO POTREBBE ESSERE QUELLO DI CREARE UNA SOTTOCLASSE "STUDENTE" CHE EREDITA DALLA CLASSE "PERSONA" E CHE HA UN METODO "STUDIA" AGGIUNTIVO:

The screenshot shows a dark-themed code editor window with a title bar 'Esempio'. The code is written in JavaScript and defines a class 'Studente' that extends 'Persona'. It includes a constructor that takes four parameters: nome, cognome, eta, and corsoDiStudi. Inside the constructor, it calls 'super()' with the first three parameters and assigns the fourth to 'this.corsoDiStudi'. It also contains a method 'studia()' that logs a message to the console indicating the student is studying their course.

```
class Studente extends Persona {
  constructor(nome, cognome, eta, corsoDiStudi) {
    super(nome, cognome, eta);
    this.corsoDiStudi = corsoDiStudi;
  }

  studia() {
    console.log("Sto studiando " + this.corsoDiStudi);
  }
}
```

IN QUESTO ESEMPIO, ABBIAMO DEFINITO UNA SOTTOCLASSE "STUDENTE" CHE ESTENDE LA CLASSE "PERSONA" UTILIZZANDO LA PAROLA CHIAVE "EXTENDS". ABBIAMO INOLTRE DEFINITO UN COSTRUTTORE CHE ACCETTA QUATTRO PARAMETRI: NOME, COGNOME, ETA E CORSODISTUDI. ABBIAMO POI DEFINITO UN METODO "STUDIA" CHE LOGGA UNA STRINGA CONTENENTE IL CORSO DI STUDI DELL'OGGETTO.



CLASSI

PER CREARE UNA NUOVA Istanza DELLA CLASSE "STUDENTE" UTILIZZIAMO LA SINTASSI "NEW":

• • • Esempio

```
let studente1 = new Studente("Giuseppe", "Verdi", 22, "Ingegneria Meccanica");
studente1.presentati(); // "Ciao, sono Giuseppe Verdi e ho 22 anni."
studente1.studia(); // "Sto studiando Ingegneria Meccanica."
```

IN QUESTO ESEMPIO, ABBIAMO CREATO UNA NUOVA Istanza DELLA SOTTOCLASSE "STUDENTE" CHIAMATA "STUDENTE1" UTILIZZANDO LA SINTASSI "NEW". ABBIAMO POI CHIAMATO IL METODO "PRESENTATI" E IL METODO "STUDIA" DELL'OGGETTO "STUDENTE1". NOTA CHE IL METODO "PRESENTATI" È EREDITATO DALLA CLASSE "PERSONA", MENTRE IL METODO "STUDIA" È DEFINITO NELLA SOTTOCLASSE "STUDENTE".

EREDITARIETÀ

JS



EREDITARIETÀ

L'**EREDITARIETÀ** È UN CONCETTO CHIAVE NELLA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI ED È PRESENTE ANCHE IN JAVASCRIPT. IN JAVASCRIPT, L'EREDITARIETÀ VIENE REALIZZATA ATTRAVERSO IL MECCANISMO DI "PROTOTIPI".

UN OGGETTO PROTOTIPO È UN OGGETTO CHE PUÒ ESSERE UTILIZZATO COME MODELLO PER CREARE ALTRI OGGETTI. OGNI OGGETTO IN JAVASCRIPT HA UN PROTOTIPO DAL QUALE EREDITA PROPRIETÀ E METODI. QUANDO UN OGGETTO CERCA DI ACCEDERE AD UNA PROPRIETÀ O UN METODO CHE NON È PRESENTE AL SUO INTERNO, JAVASCRIPT CERCA DI RISOLVERE LA RICHIESTA CERCANDO LA PROPRIETÀ O IL METODO ALL'INTERNO DEL PROTOTIPO DELL'OGGETTO.



EREDITARIETÀ

JAVASCRIPT SUPPORTA L'EREDITARIETÀ SINGOLA, IL CHE SIGNIFICA CHE UN OGGETTO PUÒ EREDITARE DA UN SOLO OGGETTO PROTOTIPO. PER DEFINIRE UN NUOVO PROTOTIPO IN JAVASCRIPT, POSSIAMO UTILIZZARE LA FUNZIONE COSTRUTTRICE E LA PROPRIETÀ PROTOTYPE.

Esempio

```
function Veicolo(marca, modello, anno) {
    this.marca = marca;
    this.modello = modello;
    this.anno = anno;
}

Veicolo.prototype.descrizione = function() {
    return this.marca + " " + this.modello + " del " + this.anno;
};
```

IN QUESTO ESEMPIO, ABBIAMO DEFINITO LA FUNZIONE COSTRUTTRICE "VEICOLO" CON TRE PARAMETRI (MARCA, MODELLO E ANNO) E ABBIAMO AGGIUNTO UNA PROPRIETÀ "DESCRIZIONE" AL PROTOTIPO "VEICOLO" UTILIZZANDO LA PROPRIETÀ PROTOTYPE. LA PROPRIETÀ "DESCRIZIONE" È UNA FUNZIONE CHE RESTITUISCE UNA STRINGA CONTENENTE LA MARCA, IL MODELLO E L'ANNO DEL VEICOLO.



EREDITARIETÀ

POSSIAMO POI CREARE UN OGGETTO CHE EREDITA DAL PROTOTIPO "VEICOLO" UTILIZZANDO LA SINTASSI "NEW":

Esempio

```
let auto = new Veicolo("Fiat", "Panda", 2020);
console.log(auto.descrizione()); // "Fiat Panda del 2020"
```

IN QUESTO ESEMPIO, ABBIAMO CREATO UN OGGETTO "AUTO" UTILIZZANDO LA SINTASSI "NEW" E LA FUNZIONE COSTRUTTRICE "VEICOLO". ABBIAMO POI CHIAMATO IL METODO "DESCRIZIONE" DELL'OGGETTO "AUTO" PER OTTENERE UNA DESCRIZIONE DEL VEICOLO. POSSIAMO ANCHE DEFINIRE UN NUOVO PROTOTIPO CHE EREDITA DAL PROTOTIPO "VEICOLO" UTILIZZANDO LA FUNZIONE COSTRUTTRICE E LA PROPRIETÀ PROTOTYPE:

Esempio

```
function Moto(marca, modello, anno, cilindrata) {
  Veicolo.call(this, marca, modello, anno);
  this.cilindrata = cilindrata;
}

Moto.prototype = Object.create(Veicolo.prototype);
Moto.prototype.constructor = Moto;

Moto.prototype.descrizione = function() {
  return Veicolo.prototype.descrizione.call(this) + " con
cilindrata " + this.cilindrata;
};
```



EREDITARIETÀ: PROTOTYPE

LA PROPRIETÀ PROTOTYPE IN JAVASCRIPT È UN OGGETTO CHE VIENE UTILIZZATO COME MODELLO PER CREARE NUOVI OGGETTI TRAMITE LA SINTASSI NEW. QUESTA PROPRIETÀ È PRESENTE IN OGNI FUNZIONE IN JAVASCRIPT E PUÒ ESSERE UTILIZZATA PER AGGIUNGERE PROPRIETÀ E METODI CONDIVISI TRA TUTTI GLI OGGETTI CREATI TRAMITE QUELLA FUNZIONE.

QUANDO UNA FUNZIONE VIENE DEFINITA IN JAVASCRIPT, VIENE CREATO AUTOMATICAMENTE UN OGGETTO PROTOTYPE VUOTO PER QUELLA FUNZIONE. POSSIAMO POI AGGIUNGERE PROPRIETÀ E METODI A QUESTO OGGETTO PER CONdividerli TRA TUTTI GLI OGGETTI CREATI TRAMITE LA FUNZIONE.