Javascript

Notes by Barry

Introduzione a javascript

Javascript è un linguaggio di programmazione ad alto livello, interpretao, è ampiamente utilizzato per lo sviluppo di siti web, applicazioni web e mobile, e applicazioni lato server.

Introduzione a javascript

Ora andrò ad esaminare il linguagggio, JavaScript è un linguaggio di programmazione:

- ad alto livello;
- dinamico;
- dinamicamente tipizzato;
- debolmente tipizzato;
- interpretato.

Ad alto livello

Javascript fornisce concetti generali che consentono di trascurare i dettagli tecnici dell'esecuzione del programma. Gestisce automaticamente la memoria tramite un garbage collector,consentendoti di concentrarti sul codice senza doverti preoccupare troppo della gestione della memoria come avviene in altri linguaggi come C e C++ con i puntatori.
Inoltre, offre una vasta gamma di strutture che ti consentono dimanipolare variabili e oggetti di grande importanza.

Dinamico

Javascript è un linguaggio di programmazione dinamico che si differenzia dai linguaggi di programmazione statici perché esegue molte operazioni durante il runtime anziché durante la fase di compilazione. Ciò comporta una serie di vantaggi e svantaggi e si traduce in importanti caratteristiche come la tipizzazione dinamica, il binding dinamico, la riflessione, la programmazione funzionale, la modifica delle proprietà di un oggetto durante l'esecuzione, la chiusura e altro ancora.

Dinamicamente tipizzato

In Javascirpt il tipo di dato di una variabile non è vincolato, il che significa che è possibile assegnare a una variabile qualsiasi tipo di dato, indipendentemente dal tipo di dato precedentemente contenuto.

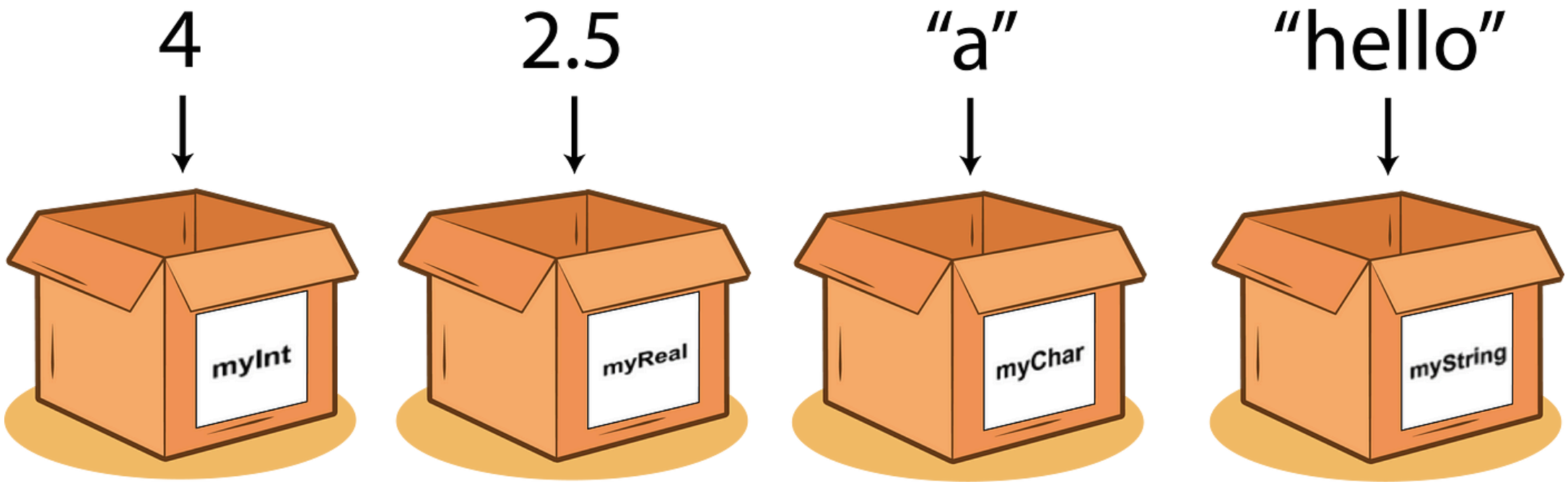
Debolmente tipizzato

in opposizione alla tipizzazione forte, nei linguaggi debolmente tipizzati il tipo di oggetto non è fisso.

Interpretato

Javascript è un linguaggio interpretato, che vuol dire che non necessita di una fase di compilazione prima dell'esecuzione del programma, al contrario di C++ per esempio. In pratica, per una questione di performance, i browser compilano JavaScript prima di eseguirlo senza bisogno di nessun un altro step.

Variabili



Variabili

Una variabile è un oggetto che rappresenta una posizione di memoria in cui è possibile memorizzare e recuperare un valore. Una variabile ha un nome univoco che viene utilizzato per identificarla nel programma e un tipo di dato che definisce il tipo di valore che può essere memorizzato al suo interno.

Naming delle variabili

Ogni variabile ha un nome univoco che la identifica nel contesto del programma, rispettando le regole di denominazione specifiche del linguaggio di programmazione utilizzato. Il nome della variabile è ciò che permette di fare riferimento ad essa nel codice.

Dichiarazione

In javascript le variabili vengono dichiarate con le parole chiave **var**, **let** e **const**.

var

Con var, la variabile può essere dichiarata senza assegnare un valore iniziale e può essere successivamente assegnata con un valore.

```
var = variabelName;
```

var

Tuttavia la parola chiave var è deprecata, è stata la modalità di dichiarazione delle variabili in JavaScript ed è stata utilizzata nelle versioni precedenti dello standard emascript.

let

La parola chiave let viene utilizzata per dichiarare una variabile con uno scope di blocco, ovvero è una variabile il cui accesso è limitato al blocco di codice in cui è stata dichiarata.

```
let variabelName="sono una variabile";
```

const

La parola chiave const viene utilizzata per dichiarare una variabile con un valore costante. Una volta assegnato un valore alla variabile const, quel valore non può essere modificato successivamente.

```
const pi = 3.14;
```

Dati primitivi

I dati primitivi sono un insieme di valori sul quale sono definite delle operazioni, e in js includono:

Number

Number: il tipo di dato numerico è generico in javascript, ed è rappresentato da :

- numeri interi;
- numeri deciamili;
- $]-\infty,+\infty[$

Stringhe

Rappresentano una sequenza di caratteri e vengono dellimitati da apici singoli o doppi.
Ad esempio:

```
let str = " Hello world";
let str2 = 'Hello world'
```

Booleani

Rappresentano un valore di verità possono essere true o false.

```
let isTrue = true;
let isFalse = false;
```

Null

Rappresenta l'assenza intenzionale di un valore e può essere assegnato solo esplicitamente.

```
let nullValue = null;
```

Undefined

Indica che una variabile è stata dichiarata, ma non gli è stato assegnato un valore.

```
let undefValue ;
```

Operatori

Gli operatori sono simboli speciali o sequenze di simboli, utilizzati per eseguire operazioni sugli operandi, che possono essere valori o variabili. Gli operatori possono essere classificati in diverse categorie a seconda del tipo di operazioni che eseguono. Alcune delle categorie comuni degli operatori includono:

Operatori aritmetici

Gli operatori artmetici in JavaScript comprendono:

- somma $+$;
- sottrazione$-$;
- moltiplicazione $-$;
- divisione $\div$;
- esponenzialen^x ;
- modulo $\%$;
- decremento$--$;

Operatori aritmetici

```
let a = 3;
let b= 2;
let result;
result= a + b; // 5
result= a - b; // 1
result= a * b; // 6
result= a /b; // 1,5
result= a ** b; //9
result= a % b; // 1
result= a++; //3
result= a-- //2
```

Operatori di confronto

Un operatore di confronto confronta due operandi e restituisce un valore booleano. Gli operatori di confronto JavaScript sono i seguenti:

- uguale $(==)$;
- disuguaglianza $(\neq)$;
- uguaglianza stretta $(===)$;
- disuguaglianza stretta $(\neq\neq)$;

operatori

- strettamente maggiore $(>)$;
- maggiore o uguale $(\geq)$;
- strettamente minore $(<)$;
- minore o uguale $(\leq)$.

operatori

```
let a = 10;
let b = 12;
let c = 10;
a == c; //true
a != b; //true
10 == "10" //true
10 == ="10" //false
10 !== "10" //true
a > b // false
b < c // true
a >= b // true
```

Operatori di assegnamento

L'operatore di assegnamento è rappresentato dal simbolo e viene utilizzato per assegnare un valore a una variabile. Gli operatori di assegnamento JavaScript sono i seguenti:

- assegnamento $(=)$;
- assegnamento di addizione e concatenazione $(+=)$;
- assegnamento di sottrazione $(-=)$;

operatori

- assegnamento di moltiplicazione $(* =)$;
- assegnamento di divisione $(/ =)$;
- assegnamento di modulo $(\% =)$;

operatori

```
let numero = 5;
numero += 3;
console.log(numero); // Output: 8
let numero = 10;
numero -= 4;
console.log(numero); // Output: 6
let numero = 3;
numero *= 5;
console.log(numero); // Output: 15
let numero = 20;
numero /= 4;
console.log(numero); // Output: 5
let numero = 17;
numero %= 5;
console.log(numero); // Output: 2
let testo = "Hello";
testo += " world";
console.log(testo); // Output: "Hello world"
```

Operatori di logici

Gli operatori logici sono strumenti utilizzati per combinare espressioni booleane al fine di valutarne la verità. Essi si basano sull'algebra di Bool e sulla logica matematica per effettuare tali operazioni. In JavaScript, i principali operatori logici sono AND (&&), OR (||) e NOT (!). L'operatore AND (&&) restituisce true solo se entrambe le espressioni booleane che combina sono true. Ad esempio,nell'espressione "A && B", entrambe le variabili A e B devono essere true affinché l'intera espressione sia valutata come true.

operatori

L'operatore OR (||) restituisce true se almeno una delle espressioni booleane combinate è true. In altre parole, nell'espressione "A || B", se una delle variabili A o B è true, l'intera espressione sarà valutata come true. L'operatore NOT (!) è un operatore unario che inverte il valore di verità di un'espressione booleana. Ad esempio, se "A" è true, allora "$!A$" sarà false e viceversa.

operatori

Prima di utilizzare questi operatori all'interno di un programma JavaScript, è utile comprendere il loro funzionamento matematico e come vengono applicati nell'algebra di Bool e nella logica matematica. Questo aiuterà a utilizzarli correttamente per valutare le espressioni booleane e ottenere i risultati desiderati.

Algebra booleana

L'algebra booleana è un'area della matematica e dell'informatica che si occupa dello studio delle operazioni e delle proprietà degli oggetti logici. ale algebra `e definita su un insieme di due soli elementi vero (V) e falso (F).

Predicati logici

I predicati logici sono espressioni o proposizioni che coinvolgono variabili e operatori. Consideriamo come predicati gli operatori confornto. Poichè Quando viene utilizzato un operatore di confronto, viene effettuato un test logico sulla relazione tra i due valori. Il risultato di tale test logico può essere considerato come un predicato logico che afferma una proposizione di verità o falsità.

operatori

Ad esempio, l'espressione $3 < 5$ utilizza l'operatore di confronto minore per confrontare il valore con il valore , Il risultato di questo confronto è vero.

Connettivi logici

I connettivi logici sono strumenti utilizzati per combinare proposizioni o espressioni logiche al fine di valutare la loro verità. Se P e Q sono due predicati possiamo creare nuovi predicati trammite tre operatori fondamentali:

- $\vee$ **or**;
- $\wedge$ **and**;
- $\neg$ **not**;

Regole dei conettivi logici

- $P \vee Q$ è vero se almeno uno tra P e Q è vero, altrimenti è falso;
- $P \wedge Q$ è vero se P e sono $$veri, altrimenti
- $\neg P$ è vero se P è falso
- $\neg P$ è falso se P è vero

Operatori logici in javascript

Dopo aver visto matematicamente la logica booleana, vediamo la sintassi in javascript:

```
let a = true;
let b = false;
console.log(a && b); // Output: false
let x = true;
let y = false;
console.log(x || y); // Output: true
let z = false;
console.log(!z); // Output: true
```

Conditional statement

I conditional statement sono i punti decisionali del codice. Questo costrutto di controllo del flusso del programma offre la possibilità di eseguire determinati blocchi di codice solo se una determinata condizione è vera.

if statment

L'istruzione if è l'istruzione di controllo che consente all'iterprete di prendere decisioni o, più precisamente, di eseguire istruzioni in modo condizionale. L'istruzione è seguita da una condizione racchiusa tra parentesi tonde (). Questa condizione può essere qualsiasi espressione che può essere valutata come true o false.

Conditional statement

Se la condizione è valutata come true, il blocco di codice all'interno delle parentesi graffe che segue l'istruzione "if" viene eseguito. Se la condizione è valutata come false, il blocco di codice viene saltato e l'esecuzione continua con il codice successivo dopo il blocco "if".

```
let numero = 1;
if (numero > 0) {
    console.log("Il numero è positivo.");
}
```

Conditional statement

In questo esempio, la condizione numero> 0 viene valutata. Se il valore della variabile numero è maggiore di zero, allora la condizione è verà è verrà stampato a schermo:

```
Il numero è positivo.
```

L'isruzione else

L'istruzione else è un'opzione facoltativa che può essere utilizzata insieme all'istruzione if"per gestire un caso alternativo quando la condizione dell'istruzione "if" è valutata come falsa. L'istruzione else viene eseguita solo se la condizione dell'istruzione if è valutata come falsa, altrimenti il blocco di codice all'interno dell'istruzione else viene saltato.

Conditional statement

```
let numero = -1;
if (numero > 0) {
    console.log("Il numero è positivo.");
    } else {
        console.log("Il numero è negativo o zero.");
        }
```

In questo esempio, se il valore della variabile numero è maggiore di zero, la condizione dell'istruzione viene valutata come falsa viene eseguito il blocco di codice all'interno dell'istruzione else , ovvero:

```
Il numero è negativo o zero
```

Operatore ternario

L'operatore ternario , è un'operatore che permette di scrivere un'istruzione condizionale in modo più compatto rispetto all'istruzione if else. L'operatore ternario è rappresentato dal simbolo ? e viene utilizzato per valutare una condizione e restituire un valore in base al risultato della valutazione.

```
condizione ? valore_se_vera : valore_se_falsa;
```

Conditional statement

Quando l'operatore ternario vien utilizzato, la condizione valutata se la condizione è vera, se la condizione è vera viene estituito il valore_se_vera, altrimenti viene restituito il valore_se_falsa.

```
let numero = 10;
let risultato = (numero > 0) ? "Il numero è positivo" : "Il numero è negativo o zero";
console.log(risultato);
```

Come per l'esempio visto in precedenza la condizione numero> 0 viene valutata vera. è verrà stampato a schermo:

```
Il numero è positivo.
```

Conditional statement

Lo Switch statement è un costrutto di controllo del flusso che permette di eseguire diverse azioni a seconda del valore di una variabile o di un'espressione.
Viene utilizzato per semplificare la gestione di molteplici casi o condizioni e può essere un'alternativa più leggibile rispetto a una serie di istruzioni if else.

Conditional statement

```
let giorno = "lunedi";
switch (giorno) {
case "lunedi":
console.log("Oggi è lunedì.");
break;
case "martedi":
console.log("Oggi è martedì.");
break;
case "mercoledi":
console.log("Oggi è mercoledì.");
break;
case "giovedi":
console.log("Oggi è giovedì.");
break;
case "venerdi":
console.log("Oggi è venerdì.");
break;
default:
console.log("Oggi non è un giorno della settimana.");
}
```

Conditional statement

In questo esempio, viene definita una variabile giorno con il valore lunedì. Lo statment viene utilizzato per confrontare il valore di giorno con i casi definiti. Poiché giorno corrisponde al caso "lunedi", viene eseguito il blocco di codice corrispondente, stampando sulla console:

```
Oggi è lunedì.
```

Cicli

I cicli vengono utilizzati per eseguire attività ripetute in base a una condizione. Le condizioni generalmente restituiscono valori di verità. Un ciclo continuerà a funzionare fino a quando la condizione da noi definita ritorna il valore di verità falso.

Ciclo for

I cicli for sono comunemente usati per eseguire un blocco di codice un determinato numero di volte.

Il ciclo for è costituito da tre espressioni facoltative, seguite da un blocco di codice.

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

Cicli

Il ciclo for è costituito da tre espressioni facoltative, seguite da un blocco di codice:

1. **Inizializzazione**: Eseguita all'inizio del ciclo, solitamente per dichiarare e inizializzare una variabile di controllo.
2. **Condizione**: Valutata all'inizio di ogni iterazione. Se vera, il blocco di codice viene eseguito. Se falsa, il ciclo termina.
3. **Incremento/decremento**: Eseguito alla fine di ogni iterazione per modificare il valore della variabile di controllo.

Cicli

```
// Calcola la somma dei numeri da 1 a 10
let sum = 0;
for (let i = 1; i <= 10; i++) {
    sum += i;
}
console.log("La somma dei numeri da 1 a 10 è: " + sum);
```

Questo esempio mostra un ciclo for che calcola la somma dei numeri da 1 a 10. Include:

1. Inizializzazione

- `let i = 1`: Inizializza la variabile di controllo `i`;
- Viene eseguita solo una volta all'inizio del ciclo;
- Può includere la dichiarazione della variabile `(let i)` o usare una variabile esistente;
  Può inizializzare più variabili: `for (let i = 0, j = 10; ...)`.

In questo caso, iniziamo da 1 perché vogliamo sommare i numeri da 1 a 10.

2. Condizione

- `i <= 10`: Condizione che determina se il ciclo continua
- Valutata all'inizio di ogni iterazione
- Se true, il blocco di codice viene eseguito
- Se false, il ciclo termina

Questa condizione assicura che sommiamo solo i numeri da 1 a 10.

3. Incremento

- `i++`: Incrementa la variabile di controllo dopo ogni iterazione.
- Eseguito alla fine di ogni iterazione, prima di valutare nuovamente la condizione.

L'incremento `i++` assicura che passiamo attraverso tutti i numeri da 1 a 10.

4. Corpo del ciclo e Sommatoria

- `sum += i`: Aggiunge il valore corrente di `i` a sum ad ogni iterazione.
- Rappresenta la sommatoria :

$$\sum_{i=1}^{10} i$$

Cicli

- La sommatoria calcola la somma di una sequenza di numeri
- In questo caso, calcola la somma dei primi 10 numeri naturali
- Il risultato di questa sommatoria è:

$$\frac{n(n+1)}{2} = \frac{10(10+1)}{2} = 55$$

While

Il ciclo while viene utilizzato quando la condizione di ripetizione non è nota in anticipo e deve essere verificata all'inizio di ogni iterazione. Come per il ciclo for valuta la condizione se restituisce il valore di verità vero il codice nel blocco delle istruzioni viene eseguito, altrimenti il ciclo termina.

Cicli

```
let somma = 0;
let numero = 1;

while (numero <= 10) {
    somma += numero;
    console.log(`Aggiunto ${numero}. La somma parziale è: ${somma}`);
    numero++;
}

console.log(`La somma totale dei numeri da 1 a 10 è: ${somma}`);
```

do while

Il ciclo do while è simile al ciclo while, ma la condizione viene valutata alla fine di ogni iterazione.
Ovvero il blocco di codice viene sempre eseguito almeno una volta.

```
let i = 0;
do {
console.log(i);
i++;
} while (i <= 5);
```

Invariante del ciclo

L'invariante del ciclo, in questo caso, è che la variabile sum contenga la somma corretta dei numeri da 1 a i. All'inizio del ciclo, quando i è uguale a 1, la variabile sum è uguale a 0. Durante ogni iterazione successiva, il valore di i viene aggiunto a sum, mantenendo l'invariante. Alla fine dell'esecuzione del ciclo, avremo eseguito l'operazione sum += i per ogni i compreso tra 1 e 10. Quindi, la variabile sum conterrà la somma dei numeri da 1 a 10.

Funzioni

Una funzione è un blocco di codice che può essere chiamata o eseguita per svolgere una specifica operazione o compito. Le funzioni consentono di organizzare il codice in unità modulari e riutilizzabili, facilitando la scrittura, la comprensione e la manutenzione del software.

```
function nomeFunzione(parametro){
//copro della funzione
//codice da eseguire
// valore di ritorno
}
```

Funzioni

```
function calcolaQuadrato(numero) {
let quadrato = numero * numero;
return quadrato;
}
let numeroDaCalcolare = 5;
let risultato = calcolaQuadrato(numeroDaCalcolare);
console.log(risultato);
```

Nell'esempio , abbiamo definito una funzione chiamata calcolaQuadrato che accetta un parametro numero. All'interno del corpo della funzione, abbiamo dichiarato una variabile locale chiamata quadrato e abbiamo calcolato il quadrato del numero moltiplicandolo per se stesso. Infine, la funzione restituisce il valore del quadrato.

Arrow function

Le arrow function sono un tipo di sintassi introdotta in ECMAScript 6 per definire funzioni in JavaScript. Sono una forma concisa e compatta per dichiarare funzioni anonime.

```
let calcolaQuadrato = (numero) => {
let quadrato = numero * numero;
return quadrato;
}
let numeroDaCalcolare = 5 ;s
let risultato = calcolaQuadrato(numeroDaCalcolare);
console.log(risultato);
```

Funzioni ricorsive

La ricorsione è una tecnica di programmazione in cui una funzione si chiama su istanze di problema più piccole per risolvere il problema originale. L'intuizione è semplice: per salire una scala con i gradini, si sale un gradino e si è ridotto il problema a quello di salire una scala (più corta) con $n - 1$ gradini.

Funzioni

Nello sviluppo di una funzione ricorsiva, ci sono tre elementi fondamentali da considerare. Innanzitutto, è essenziale definire un caso ricorsivo, che consiste nel descrivere il problema come una composizione di sotto-problemi più piccoli. Ogni sotto-problema deve essere un'istanza dello stesso problema originale, ma applicato a un input di dimensioni ridotte. In questa fase, la funzione chiamerà se stessa per risolvere ciascuno di questi sotto-problemi.

Il secondo elemento cruciale è la definizione del caso base. Questo determina quando la ricorsione deve fermarsi. È necessario identificare la dimensione del problema per la quale la soluzione è già nota e non è più necessario ricorrere alla ricorsione. Questo passaggio è fondamentale per evitare che la funzione si chiami infinitamente, garantendo così la terminazione del processo ricorsivo.

Infine, il terzo ingrediente consiste nel combinare le soluzioni dei sotto-problemi per risolvere il problema originale. Questo passaggio richiede di integrare i risultati ottenuti dalle chiamate ricorsive in modo da costruire la soluzione complessiva del problema iniziale.

caso base

Il caso base è cruciale per il corretto funzionamento di una funzione ricorsiva, in quanto definisce quando la ricorsione deve terminare. Senza un caso base appropriato, la funzione rischierebbe di chiamare se stessa indefinitamente.

Esecuzione step by step

Per comprendere al meglio come funziona la ricorsione, è utile esaminare l'esecuzione passo dopo passo di una funzione ricorsiva. Un esempio efficace è la funzione fattoriale. Quando chiamiamo la funzione fattoriale con un input, diciamo 5, la funzione inizia una serie di chiamate ricorsive.

Funzione fattoriale

La funzione fattoriale è un esempio di funzione che può essere implementata in modo ricorsivo. Matematicamente, il fattoriale di un numero intero non negativo $n \geq 0$, indicato con $n!$, è definito come il prodotto di tutti i numeri interi positivi da 1 a n

Definizione matematica:

- $n \cdot n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$ per $n > 0$
- $0! = 1$
- È possibile dimostrare che $0! = 1$ utilizzando la definzione di fattoriale:

$$n! = n(n-1)!$$

$$1! = 1(1-1)! = 0!$$

Definizione ricorsiva:

$$n! = \begin{cases} n \cdot (n-1)! \text{ if } n > 0 \\ 1 \text{ if } n = 0 \end{cases}$$

```
function fattoriale(n){
    if(n==0) return 1;
    return n * fattoriale(n-1);
}
```

Funzioni

Ad ogni chiamata, la funzione verifica se ha raggiunto il caso base (che per il fattoriale è quando l'input è 0 o 1). Se non è così, effettua una nuova chiamata ricorsiva con un input ridotto. Questo processo continua finché non si raggiunge il caso base.
Nel nostro esempio, la sequenza di chiamate sarebbe:
factorial(4) chiama factorial(3)
factorial(3) chiama factorial(2)
factorial(2) chiama factorial(1)
factorial(1) raggiunge il caso base e restituisce 1

Funzioni

A questo punto, le chiamate iniziano a "svolgersi" nell'ordine inverso:
factorial(2) moltiplica il risultato di factorial(1) per 2 e restituisce 2
factorial(3) moltiplica il risultato di factorial(2) per 3 e restituisce 6
factorial(4) moltiplica il risultato di factorial(3) per 4 e restituisce 24

Questo processo di chiamate ricorsive seguite dal "riavvolgimento" delle chiamate è fondamentale per comprendere come funziona la ricorsione. Ogni chiamata ricorsiva crea un nuovo contesto di esecuzione, che viene mantenuto nello stack di chiamata finché la funzione non restituisce un valore.

Oggetti

Un oggetto è una raccolta non ordinata di proprietà, ognuna delle quali ha una chiave (nome) e un valore. I nomi delle proprietà possono essere una stringa o un Symbol().Un oggetto è una raccolta non ordinata di proprietà, ognuna delle
quali ha una chiave (nome) e un valore. I nomi delle proprietà possono essere una stringa o un Symbol().

Oggetti

Letterali degli oggetti

I letterali degli oggetti in JavaScript consentono di creare oggetti in modo conciso utilizzando una notazione specifica. Una proprietà ha una chiave (conosciuta anche come chiave o “identificatore”) prima dei due punti ":", ed un valore alla sua destra.

```
let persona = {
  nome: "Mario",
  età: 30
 };
```

Dizionario

Il termine dizionario in JavaScript si riferisce all'uso di un oggetto come una collezione di coppie chiave-valore, dove le chiavi sono stringhe e i valori possono essere qualsiasi tipo di dato.

```
let persona = {
"nome completo": "Mario Rossi",
età: 30
};
console.log(persona["nome completo"]);
```

In questo caso, la chiave "nome completo" è composta da più parole e viene inclusa tra le parentesi quadre per indicare che è una chiave valida. Per accedere a questa proprietà, puoi utilizzare la notazione delle parentesi quadre.

Riferimento e copia

Una delle principali differenze tra oggetti e primitivi è la modalità di memorizzazione e copia. Gli oggetti vengono memorizzati e copiati "per riferimento", mentre i primitivi come stringhe, numeri e booleani vengono sempre copiati "per valore".

Oggetti

Esempio di oggetti primitivi:

```
let obj = { nome: "Alice", età: 25 };
```

la variabile obj contiene un riferimento all'oggetto che ha le proprietà "nome" e "età". L'oggetto stesso è memorizzato altrove in memoria, mentre la variabile obj funge da punto di accesso per raggiungere e manipolare l'oggetto.

Metodi

Gli oggetti hanno una serie di metodi incorporati che possono essere utilizzati per manipolare e interagire con i dati all'interno degli oggetti stessi.

Metodo this

Il metodo this in JavaScript si riferisce all'oggetto corrente su cui è stato chiamato il metodo. Può essere utilizzato all'interno di un oggetto per accedere alle sue proprietà e metodi.

```
const obj = {
    name: "John",
    greet: function() {
        console.log("Hello," + this.name + "!");
    }
};

obj.greet();
```

Metodo toSting()

Il metodo toString() in JavaScript è una funzione incorporata che viene utilizzata per convertire un oggetto in una stringa. Il metodo toString() restituisce una rappresentazione testuale dell'oggetto.

```
const obj = { name: 'Alice', age: 25 };
console.log(obj.toString());
```

Array

Un array è una struttura dati che memorizza una collezione di elementi dello stesso tipo in un'unica struttura. Gli array sono utilizzati per rappresentare sequenze di dati, dove ogni elemento è identificato da un indice.

Memorizzazione Contigua

Gli array sono caratterizzati dalla memorizzazione contigua degli elementi. Questo significa che tutti gli elementi dell'array sono memorizzati in posizioni di memoria adiacenti e sequenziali.
Ecco i dettagli principali:

- Accesso Diretto.
- Efficienza nella Memorizzazione.
- Ridimensionamento.

Accesso Diretto

Grazie alla memorizzazione contigua, ogni elemento dell'array può essere accesso direttamente in tempo costante $O(1)$. Questo è possibile perché, conoscendo l'indice dell'elemento e la dimensione di ogni elemento, l'indirizzo di memoria dell'elemento può essere calcolato direttamente.

Efficienza nella Memorizzazione

Poiché tutti gli elementi sono memorizzati in un blocco continuo di memoria, le operazioni di accesso e aggiornamento degli elementi sono molto rapide e non richiedono operazioni aggiuntive di navigazione tra strutture di memoria disparate.

Ridimensionamento

Se un vettore deve essere ridimensionato ad esempio, per aggiungere più elementi di quelli previsti inizialmente, il sistema deve allocare un nuovo blocco di memoria contigua, copiare gli elementi esistenti e aggiornare i riferimenti al nuovo blocco. Questo può comportare un costo di prestazioni temporaneo durante l'operazione di ridimensionamento.

Array

Esempio:

```
const array = [10, 20, 30, 40, 50]; // Definizione dell'array
console.log(array[0]); // Accesso al primo elemento: 10
console.log(array[1]); // Accesso al secondo elemento: 20
console.log(array[2]); // Accesso al terzo elemento: 30
```

Metodi

Gli array in JavaScript sono strutture dati molto potenti e versatili, e JavaScript fornisce una serie di metodi integrati che facilitano l'esecuzione di operazioni comuni su di essi. Questi metodi includono funzioni per l'aggiunta, la rimozione, la modifica e la trasformazione degli elementi all'interno di un array.

Premessa

Sebbene i metodi degli array offerti da JavaScript siano estremamente utili e semplifichino molte operazioni comuni, è fondamentale comprendere come funzionano questi metodi a livello di algoritmi. Questo non solo ti aiuterà a utilizzare i metodi in modo più efficace, ma ti fornirà anche una base solida per implementare soluzioni personalizzate e ottimizzare le prestazioni del tuo codice.