ES6 - EcmaScript 2015

ECMAScript 6 (ES6), noto anche come ECMAScript 2015, è una versione importante del linguaggio di programmazione JavaScript. È stato rilasciato nel 2015 e ha introdotto numerose nuove caratteristiche e miglioramenti che hanno reso il codice JavaScript più potente, leggibile ed efficiente. Ecco un'introduzione a ES6 e alcune delle sue principali modifiche:

1. **Variabili con `let` e `const`:** In ES6, sono state introdotte due nuove parole chiave per dichiarare variabili: `let` e `const`. `let` permette la dichiarazione di variabili con ambito di blocco, mentre `const` dichiara variabili immutabili.
2. **Arrow Functions:** Le arrow function (`() => {}`) sono una nuova sintassi per definire funzioni più brevi e leggibili. Sono particolarmente utili per le funzioni anonime e i callback.
3. **Classi:** ES6 ha introdotto una sintassi più chiara e orientata agli oggetti per definire classi e gestire l'ereditarietà. È diventato più simile ad altri linguaggi di programmazione orientata agli oggetti.

4. **Moduli:** ES6 ha aggiunto il supporto nativo per i moduli, consentendo di organizzare il codice in file separati e di importare ed esportare funzioni, variabili e classi tra di essi.
5. **Template String:** I template string (delineati con backticks ``) consentono di creare stringhe multilinea e di interpolare variabili all'interno delle stringhe in modo più semplice e leggibile.
6. **Destructuring:** La destructuring assegna valori a variabili da oggetti e array in modo conciso. È utile per estrarre dati da strutture complesse.

7. **Valori predefiniti per i parametri delle funzioni:** È ora possibile definire valori predefiniti per i parametri delle funzioni, il che semplifica il controllo dei valori mancanti.
8. **Rest Parameters e Spread Operator:** I rest parameters (`...`) consentono di passare un numero variabile di argomenti a una funzione come un array, mentre lo spread operator (`...`) consente di espandere un array o un oggetto in posizioni di argomenti o proprietà separate.
9. **Promises:** ES6 ha introdotto le Promises, che semplificano la gestione delle operazioni asincrone e la gestione degli errori.
10. **Nuovi metodi per stringhe, array e oggetti:** ES6 ha aggiunto una serie di nuovi metodi per stringhe (`startsWith` , `endsWith` , `includes`), array (`find` , `findIndex` , `map` , `filter` , `reduce` , ecc.) e oggetti (`Object.keys` , `Object.values` , `Object.entries`) che semplificano il lavoro con queste strutture dati.

11. **Symbol e Map:** Introduce il tipo di dato Symbol, che è utile per la creazione di proprietà di oggetti non enumerabili. Inoltre, ES6 ha aggiunto la struttura dati Map per gestire associazioni chiave-valore in modo più flessibile rispetto agli oggetti.
12. **Generators:** I generatori sono funzioni speciali che possono essere interrotte e riprese, consentendo di scrivere il codice asincrono in un modo più lineare.
13. **Nuovi metodi per array e oggetti immutabili:** ES6 ha introdotto metodi come `Object.freeze` e `Object.seal` per rendere oggetti immutabili o sigillati, e metodi come `Array.from` per creare copie degli array con cui è più sicuro lavorare.

Questi sono solo alcuni dei miglioramenti introdotti da ES6 e versioni successive di ECMAScript. Le nuove versioni del linguaggio continuano a portare miglioramenti e nuove funzionalità per rendere JavaScript più potente e flessibile. È importante tenersi aggiornati con queste modifiche per scrivere codice JavaScript moderno ed efficiente.

Nuove keyword per variabili e costanti

```
var velocita = 100;

var velocita = 100: -> let velocita = 100;

let velocita = 100: -> const velocita = 100;

velocita += 40; //errore: velocita è una costante
```

Block Scoping

- Local Scope e Global Scope

```
let velocita = 100;

{
    let velocita = 80;
    console.log('interno',velocita);
}

    console.log('esterno',velocita);
```

Lo scope è diverso anche nei blocchi funzione: usando let posso limitare la visibilità delle variabili

Template literals

Le template literals, note anche come template strings, sono una caratteristica di JavaScript introdotta con ECMAScript 6 (ES6) che permette di creare stringhe multilinea e interpolare valori di variabili o espressioni all'interno di stringhe in modo più leggibile e flessibile rispetto alle stringhe tradizionali racchiuse tra virgolette singole o doppie.

Le template literals sono racchiuse tra backtick (`) anziché virgolette singole o doppie. Ecco come funzionano:

1. **Stringhe multilinea**: Puoi definire stringhe su più righe senza dover utilizzare caratteri di escape o concatenazione di stringhe. Ad esempio:

```
const testoMultilinea = `
Questa è una stringa multilinea.
Puoi scrivere su più righe senza problemi.
`;
```

2. **Interpolazione di variabili ed espressioni**: Puoi inserire il valore di variabili o espressioni direttamente all'interno di una template literal utilizzando il `${}` operatore di interpolazione. Ad esempio:

```
const nome = "Alice";
const messaggio = `Benvenuto, ${nome}!`;
```

In questo caso, `${nome}` verrà sostituito con il valore della variabile `nome` all'interno della stringa.

3. **Tagged templates**: Le template literals possono essere utilizzate con funzioni "tagged" personalizzate che consentono di elaborare la stringa in modi personalizzati. Ad esempio:

```
function maiuscolo(strings, ...valori) {
  let risultato = "";
  for (let i = 0; i < strings.length; i++) {
    risultato += strings[i];
    if (i < valori.length) {
      risultato += valori[i].toUpperCase();
    }
  }
  return risultato;
}

const nome = "Alice";
const messaggio = maiuscolo`Ciao, ${nome}!`;
```

In questo esempio, la funzione `maiuscolo` converte il valore della variabile `nome` in maiuscolo all'interno della stringa.

Le template literals rendono il codice più leggibile e consentono di gestire facilmente stringhe complesse e formattate. Sono particolarmente utili quando si costruiscono URL, query SQL o qualsiasi tipo di output multilinea.

```
let nome = 'mauro';
`Ciao ${nome}` //interpolazione
```

```
let nome = 'mauro';
let saluto  `ciao ${nome}`;
console.log(saluto);
```

Spread Operator

```
let a = [2, 3, 4];
let b = [1. ...a, 5];
console.log(b);
```

Rest Parameters

```
function calcolaMedia(...voti) {
    console.log(voti);
}
calcolaMedia(20,22,24,25);
```

Destructuring

La destructuring è una caratteristica introdotta in JavaScript con ECMAScript 6 (ES6) che consente di estrarre i valori da array e oggetti in variabili distinte in modo più conciso. È utile quando si desidera ottenere valori specifici da strutture dati complesse come array o oggetti. La sintassi della destructuring è contrassegnata dall'uso delle parentesi graffe `{}` per gli oggetti e delle parentesi quadre `[]` per gli array. Ecco come funziona:

Destructuring di Array

```
const numeri = [1, 2, 3, 4, 5];

// Estrai i valori da 'numeri' in variabili separate
const [primo, secondo, ...restanti] = numeri;

console.log(primo); // 1
console.log(secondo); // 2
console.log(restanti); // [3, 4, 5]
```

raccogliere i restanti valori in un altro array chiamato `restanti`.

Destructuring di Oggetti

```
const persona = {
  nome: "Alice",
  età: 30,
  indirizzo: {
    città: "Roma",
    paese: "Italia"
  }
};

// Estrai i valori da 'persona' in variabili separate
const { nome, età, indirizzo: { città } } = persona;

console.log(nome); // "Alice"
console.log(età); // 30
console.log(città); // "Roma"
```

Nell'esempio sopra, abbiamo estratto i valori delle proprietà `nome`, `età` e `indirizzo.città` dall'oggetto `persona` nelle variabili corrispondenti. Abbiamo anche

Destructuring con Valori Predefiniti

È possibile assegnare valori predefiniti alle variabili di destinazione nel caso in cui le proprietà o gli indici non esistano nell'array o nell'oggetto sorgente:

```
const persona = {
  nome: "Bob",
  età: 25
};

// Estrai le proprietà con valori predefiniti
const { nome, età, lavoro = "Sconosciuto" } = persona;

console.log(lavoro); // "Sconosciuto" (valore predefinito)
```

Destructuring in Funzioni

La destructuring può anche essere utilizzata con parametri di funzione per estrarre valori dagli oggetti passati come argomenti:

```
function stampaPersona({ nome, età }) {
  console.log(`Nome: ${nome}, Età: ${età}`);
}

const persona = {
  nome: "Charlie",
  età: 35
};

stampaPersona(persona); // "Nome: Charlie, Età: 35"
```

In questo caso, la funzione `stampaPersona` accetta un oggetto come argomento e utilizza la destructuring per estrarre le proprietà `nome` e `età` direttamente all'interno della funzione.

La destructuring è una tecnica potente che semplifica la gestione dei dati complessi in JavaScript, rendendo il codice più pulito e leggibile. Può essere utilizzata con array e oggetti in vari contesti per estrarre dati in modo efficiente.

Destrutturare array

```
let numeri = [1,2,3];
let [uno, due] = numeri;
console.log(uno, due);
```

Destrutturare oggetti

```
let auto = {marca: 'fiat', modello: 'panda'};
let {marca,modello} = auto;
console.log(marca, modello);

//alternativa
let marca, modello;
( {marca, modello} = auto );//notare le parentesi! altrimenti non funziona
```

Arrow functions

```
let saluta = function () {
console.log('ciao);

}

saluta();

//diventa

let saluta = () => { console.log('ciao); }
```

```
let valori = [2, 3, 4];
let raddoppia = (n) => n*2; //dato n ritorna n*2
```

Helper methods

map

Crea un nuovo array, chiamando una funzione per ciascun elemento dell'array base

```
let valori = [2, 3, 4];
let raddoppia = (n) => n*2; //dato n ritorna n*2

//uso map con arrow function
let raddoppiati = valori.map(raddoppia);
console.log(raddoppiati);
```

```
let valori = [2, 3, 4];

//versione sintetica: uso map con arrow function diretta
let raddoppiati = valori.map((n) => n*2);
console.log(raddoppiati);
```

filter

Crea un nuovo array, filtrando con un test l'array originale

```
let voti = [22, 30, 24, 28, 29, 23];

//versione sintetica: uso filter con arrow function diretta
//dato ciascun voto (n), lo ritorno solo se: n>=28
let votiAlti = voti.filter((n) => n>=28);
console.log(votiAlti);
```

manipolazione di stringhe

```
let lunghissimaStringa = `mauro${'oo'.repeat(10)}`;
console.log(lunghissimaStringa);
```

ricerca nelle stringhe

```
console.log("buonasera".startsWith("buona")); //true

console.log("buonasera".endsWith("giorno")); //false
console.log("buonasera".endsWith("sera")); //true

console.log("buonasera".includes("nas")); //true
```

controllo dei tipi numerici

```
const verificaUtente = (user, userId) => Number.isFinite(userId);
//alternativa
//const verificaUtente = (user, userId) => Number.isSafeInteger(userId);
console.log(verificaUtente('mauro', 6)); //true
```