Capitolo 1

Introduzione a JavaScript

JavaScript è il linguaggio di programmazione del Web. La stragrande maggioranza dei siti lo utilizza e tutti i browser web moderni, per computer, tablet e smartphone, includono interpreti JavaScript. Ciò lo rende il linguaggio più diffuso di sempre. Negli ultimi dieci anni, Node.js ha portato la programmazione JavaScript anche al di fuori dell'ambito dei browser web, e il notevole successo di Node ha fatto sì che JavaScript diventasse anche il linguaggio più utilizzato tra gli sviluppatori di software. Che stiate iniziando da zero o stiate già utilizzando JavaScript per attività professionali, questo libro vi aiuterà a impadronirvi appieno del linguaggio.

Se avete già una certa familiarità con altri linguaggi di programmazione, può esservi utile sapere che JavaScript è un linguaggio di alto livello, dinamico e interpretato, che ben si adatta agli stili di programmazione a oggetti e a funzioni. In JavaScript le variabili non sono tipizzate. La sua sintassi si basa vagamente su Java, ma per il resto i due linguaggi non sono correlati. JavaScript trae le sue funzioni di prima classe da Scheme e la sua ereditarietà basata su prototipi deriva da un linguaggio poco conosciuto: Self. Ma non è necessario conoscere nessuno di questi o avere familiarità con questi termini per poter utilizzare questo libro e per imparare a programmare.

Il nome "JavaScript" è piuttosto fuorviante. A parte una leggera somiglianza sintattica, JavaScript è completamente diverso da Java. E JavaScript ha da tempo superato le sue radici nel linguaggio per script, per diventare un linguaggio *general-purpose* solido ed efficiente, adatto all'ingegneria del software e a progetti costituiti da enormi basi di codice.

In questo capitolo

JavaScript: nomi, versioni e modalità

JavaScript è stato creato in Netscape agli esordi del Web e, tecnicamente, "JavaScript" è un *trademark* concesso in licenza da Sun Microsystems (ora Oracle) usato per descrivere l'implementazione del linguaggio da parte di Netscape (ora Mozilla). Netscape ha sottoposto il linguaggio per la standardizzazione all'ECMA (*European Computer Manufacturer's Association*) e, a causa di problemi col trademark, la versione standardizzata del linguaggio ha ottenuto un nome non troppo gradevole: "ECMAScript". All'atto pratico, tutti chiamano il linguaggio, semplicemente, *JavaScript*. Questo libro utilizza il nome "ECMAScript" e l'abbreviazione "ES" solo per far riferimento allo standard del linguaggio e alle versioni di quello standard.

Per la maggior parte degli anni Dieci, tutti i browser web hanno supportato la versione 5 dello standard ECMAScript. Questo libro impiega lo standard ES5 come base di compatibilità, e non tratta più le versioni precedenti del linguaggio. Nel 2015 è stato rilasciato ES6, che ha aggiunto nuove e importanti funzionalità, inclusa la sintassi per le classi e i moduli, che hanno trasformato JavaScript da un linguaggio per script a un linguaggio più completo e *general-purpose*, adatto ad attività di ingegneria del software su larga scala. A partire da ES6, le specifiche ECMAScript hanno adottato una cadenza di rilascio annuale e oggi le versioni del linguaggio – ES2016, ES2017, ES2018, ES2019 ed ES2020 – sono identificate per anno di rilascio.

Con l'evoluzione di JavaScript, i progettisti del linguaggio hanno tentato di correggere i difetti presenti nelle prime versioni (pre-ES5). Ma per mantenere la compatibilità con le versioni precedenti, non è possibile rimuovere le vecchie funzionalità, per quanto siano imperfette. Ma in ES5 e versioni successive, i programmi possono optare per la più rigorosa *strict mode* di JavaScript, in cui sono stati corretti numerosi errori iniziali del linguaggio. Per attivarla si utilizza la direttiva `use strict`, descritta nel paragrafo *"use strict"* del Capitolo 5. Tale paragrafo riassume anche le differenze esistenti tra il "vecchio" JavaScript e il JavaScript *strict*. In ES6 e versioni successive, l'uso di nuove funzionalità del linguaggio spesso richiama implicitamente la *strict mode*. Per esempio, se utilizzate la parola riservata ES6 `class` o se create un modulo ES6, tutto il codice situato all'interno della classe o del modulo diverrà automaticamente strict e dunque le vecchie funzionalità non saranno disponibili in quei contesti. Questo libro tratterà anche le funzionalità precedenti di JavaScript, ma facendo attenzione a sottolineare che esse non sono disponibili in *strict mode*.

Per poter essere davvero utile, ogni linguaggio deve avere una piattaforma, o libreria standard, per eseguire le attività di input e output. Il nucleo centrale del linguaggio JavaScript definisce un'API minima, per lavorare con i numeri, il testo, gli array, gli insiemi (set), le mappe e così via, ma non include alcuna funzionalità di input o output. L'input e l'output (così come le funzionalità più sofisticate, come il networking, l'archiviazione e la grafica) rientrano nelle responsabilità dell'"ambiente host" in cui è incorporato JavaScript. L'ambiente host originale di JavaScript era il browser web e questo è ancora l'ambiente di esecuzione più comune per il codice JavaScript. L'ambiente costituito dal browser web consente al codice JavaScript di ottenere l'input dal mouse e dalla tastiera tramite richieste HTTP. Inoltre consente al codice JavaScript di mostrare l'output con HTML e CSS.

Dal 2010 è disponibile un altro ambiente host per il codice JavaScript. Invece di obbligare JavaScript a funzionare con le API fornite dal browser web, Node fornisce a JavaScript l'accesso all'intero sistema operativo, consentendo quindi ai programmi JavaScript di leggere e scrivere file, inviare e ricevere dati sulla rete ed emettere o servire richieste

HTTP. Node è una scelta molto popolare per l'implementazione di server web ed è anche un comodo strumento per scrivere semplici script di servizio, in alternativa agli script della shell.
La maggior parte di questo libro è incentrata solo sul linguaggio JavaScript, ma il Capitolo 11 tratta la libreria standard JavaScript, il Capitolo 15 introduce l'ambiente host del browser web e il Capitolo 16 introduce l'ambiente host Node.
Questo libro esamina prima i fondamentali, di basso livello, quindi sviluppa tali concetti per affrontare le astrazioni più avanzate e di livello superiore. I suoi capitoli, pertanto, devono essere letti più o meno secondo l'ordine proposto. Tuttavia l'apprendimento di un nuovo linguaggio di programmazione non è mai un processo lineare, e nemmeno la descrizione di un linguaggio è lineare: ogni funzionalità del linguaggio è correlata ad altre, e questo libro è ricco di riferimenti incrociati, a volte indietro e talvolta in avanti, ad altro materiale correlato. Questo capitolo introduttivo non fa altro che presentare il linguaggio, introducendo le funzionalità chiave che aiuteranno a comprendere gli approfondimenti presentati nei capitoli successivi. Se siete già programmatori JavaScript, probabilmente potete anche pensare di saltare questo capitolo (ma forse potreste divertirvi a leggere l'Esempio 1.1 alla fine del capitolo, prima di procedere).

Esplorare JavaScript

Quando si impara un nuovo linguaggio di programmazione, è importante provare gli esempi presentati nel libro, poi modificarli e infine riprovarli per verificare la propria comprensione del linguaggio. Per poterlo fare, dovete dotarvi di un interprete JavaScript. Il modo più semplice per provare alcune righe di JavaScript consiste nell'aprire gli strumenti per sviluppatori web nel vostro browser web (con F12, Ctrl-Maiusc-I o Comando-Opzione-I) e selezionare la scheda *Console*. Qui potete digitare il codice al prompt e visualizzare i risultati, durante la digitazione. Gli strumenti di sviluppo, spesso vengono visualizzati come riquadri nella parte inferiore o destra della finestra del browser stesso, ma di solito è possibile "sganciarli" e usarli come finestre separate (come illustrato nella Figura 1.1), il che, spesso, è abbastanza comodo.
Un altro modo per provare il codice JavaScript è scaricare e installare Node da `https://nodejs.org`. Una volta installato Node sul sistema, potete semplicemente aprire una finestra del Terminale e digitare `node` e avviare così una sessione JavaScript interattiva, come la seguente:

```
$ node
Welcome to Node.js v12.13.0.
Type ".help" for more information.
> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the repl
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file
```

```
Press ^C to abort current expression, ^D to exit the repl
> let x = 2, y = 3;
undefined
> x + y
5
> (x === 2) && (y === 3)
true
> (x > 3) || (y < 3)
false
```

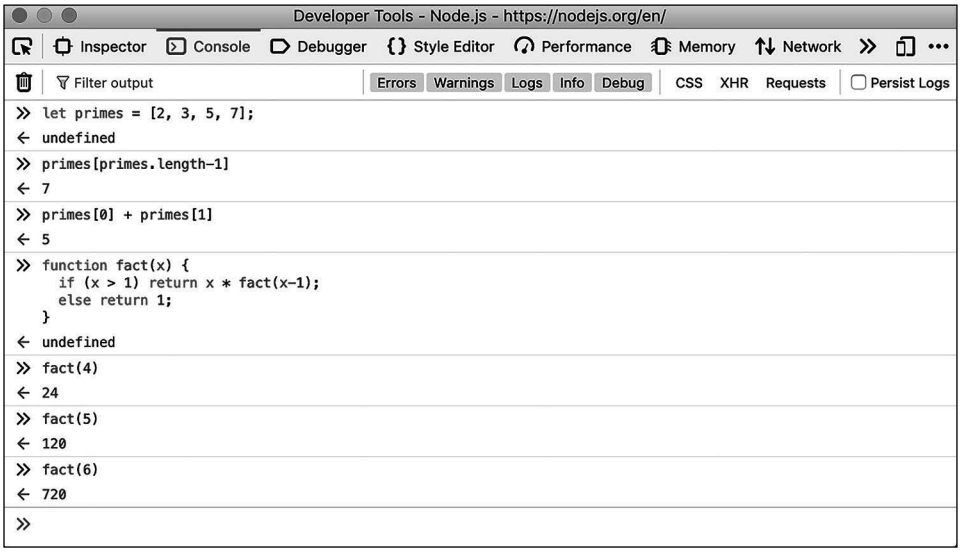

Figura 1.1 La console JavaScript negli strumenti per sviluppatori di Firefox.

Hello World

Quando sarete pronti per iniziare a sperimentare con blocchi di codice più lunghi, questi ambienti interattivi "riga per riga" potrebbero non essere più adatti e probabilmente preferirete scrivere il codice in un editor di testo. Da lì, potete copiare e incollare il codice nella console JavaScript o in una sessione Node. Oppure potete salvare il vostro codice in un file (l'estensione tradizionale per i file di codice JavaScript è `.js`) e poi eseguire quel file di codice JavaScript con Node:

```
$ node snippet.js
```

Se impiegate Node in questo modo non interattivo, non otterrete la visualizzazione automatica dei valori prodotti dal codice in esecuzione, per cui dovrete farlo esplicitamente. Per visualizzare del testo o altri valori JavaScript nella finestra del terminale o nella console degli strumenti di sviluppo del browser potete impiegare la funzione `console.log()`. Quindi, per esempio, se create un file `hello.js` contenente questa riga di codice:

```
console.log("Hello World!");
```

ed eseguite il file con `node hello.js`, otterrete sullo schermo il messaggio "`Hello World!`". Se volete vedere quello stesso messaggio nella console JavaScript di un browser web, create un nuovo file di nome `hello.html` e inserite questo testo:

```
<script src="hello.js"></script>
```

Ora caricate `hello.html` nel browser web impiegando un URL `file://` come il seguente:

```
file:///Users/username/javascript/hello.html
```

Aprite la finestra degli strumenti per sviluppatori e nella console comparirà il messaggio di benvenuto.

Breve tour di JavaScript

Questo paragrafo rappresenta una rapida introduzione, tramite esempi di codice, al linguaggio JavaScript. Dopo questo capitolo introduttivo, ci immergeremo in JavaScript a un livello più profondo: il Capitolo 2 parla di commenti, punti e virgola e set di caratteri Unicode. Il Capitolo 3 inizia a diventare più interessante: spiega le variabili JavaScript e i valori che potete assegnare loro.
Ecco alcuni esempi di codice per illustrare i punti salienti di questi due capitoli.

```
// Tutto ciò che segue le doppie barre è un commento.
// Leggete attentamente i commenti: spiegano il codice JavaScript.

// Una variabile è un nome simbolico per un valore.
// Le variabili vengono dichiarate con la parola riservata let:
let x;   // Dichiara una variabile denominata x.

// Per assegnare un valore a una variabile si usa il segno =
x = 0;   // Ora la variabile x ha il valore 0
x        // => 0: una variabile restituisce il proprio valore.

// JavaScript supporta diversi tipi di valori
x = 1;               // Numeri.
x = 0.01;            // I numeri possono essere numeri interi o reali.
x = "hello world";   // Stringhe di testo, tra doppi apici.
x = 'JavaScript';    // Le stringhe possono essere poste anche fra apici semplici.
x = true;            // Un valore booleano.
x = false;           // L'altro valore booleano.
x = null;            // Null è un valore speciale, che significa "nessun valore".
x = undefined;       // Undefined è un altro valore speciale, come null.
```

Altri due *tipi* molto importanti manipolabili dai programmi JavaScript sono gli oggetti e gli array. Questi sono gli argomenti dei Capitoli 6 e 7, ma sono così importanti che li vedrete comparire molte volte, anche prima di raggiungere tali capitoli:

```
// Il tipo di dati più importante di JavaScript è l'oggetto.
// Un oggetto è una raccolta di coppie nome/valore o una stringa per la mappa di valori.
let book = {               // Gli oggetti sono racchiusi tra parentesi graffe.
    topic: "JavaScript",   // La proprietà "topic" ha il valore "JavaScript".
    edition: 7             // La proprietà "edition" ha il valore 7
};                         // La parentesi graffa chiusa segna la fine dell'oggetto.

// Accesso alle proprietà di un oggetto con . o []:
book.topic                  // => "JavaScript"
book["edition"]             // => 7: un altro modo per accedere ai valori delle proprietà.
book.author = "Flanagan";   // Crea nuove proprietà tramite assegnamento.
book.contents = {};         // {} è un oggetto vuoto, senza proprietà.

// Accesso condizionale alle proprietà con ?. (ES2020):
book.contents?.ch01?.sect1   // => undefined: book.contents non ha la proprietà ch01.

// JavaScript supporta anche gli array, elenchi di valori indicizzati numericamente:
let primes = [2, 3, 5, 7];   // Un array di quattro valori, delimitato da [ e ].
primes[0]                    // => 2: il primo elemento (indice 0) dell'array.
primes.length                // => 4: numero di elementi contenuti nell'array.
primes[primes.length- 1]     // => 7: l'ultimo elemento dell'array.
primes[4] = 9;               // Aggiunta di un nuovo elemento tramite assegnamento.
primes[4] = 11;              // Modifica di un elemento, sempre tramite assegnamento.
let empty = [];              // [] è un array vuoto, senza elementi.
empty.length                 // => 0

// Gli array e gli oggetti possono contenere altri array e oggetti:
let points = [                // Un array di due elementi.
    { x: 0, y: 0 },           // Ogni elemento è un oggetto.
    { x: 1, y: 1}
];
let data = {                  // Un oggetto con due proprietà
    trial1: [[1,2], [3,4]],   // Il valore di ogni proprietà è un array.
    trial2: [[2,3], [4,5]]    // Gli elementi degli array sono array.
};
```

Sintassi dei commenti negli esempi di codice

Nel codice precedente avrete notato che alcuni commenti iniziano con una freccia (=>). Questi commenti mostrano il valore prodotto dal codice che precede il commento e, in pratica, sono il mio tentativo di emulare un ambiente JavaScript interattivo, come quello della console di un browser web, sulle pagine di un libro stampato.

Quei commenti `// =>` fungono anche da *asserzioni*, e ho scritto uno strumento che sottopone a test il codice e verifica che produca il valore specificato nel commento. Questo dovrebbe aiutare a contenere gli errori presenti nei listati, per i quali si rimanda comunque alla pagina `http://bit.ly/apo-js`.

Esistono due stili correlati di commenti/asserzioni. Se vedete un commento nella forma `// a == 42`, significa che dopo l'esecuzione del codice che precede il commento, la variabile `a` avrà il valore `42`. Se vedete un commento nella forma `// !`, significa che il codice che precede il commento lancia un'eccezione (e il resto del commento, dopo il punto esclamativo, di solito spiega quale tipo di eccezione viene lanciata). Vedrete questi commenti un po' in tutto il libro.

La sintassi appena illustrata per elencare gli elementi dell'array all'interno di parentesi quadre o per mappare i nomi delle proprietà degli oggetti ai rispettivi valori all'interno delle parentesi graffe è nota come *espressione di inizializzazione* ed è solo uno degli argomenti trattati nel Capitolo 4. Un'*espressione* è un frammento di codice JavaScript che può essere valutato per produrre un valore. Per esempio, l'uso dei simboli `.` e `[]` per far riferimento al valore di una proprietà di un oggetto o di un elemento di un array è un'espressione. Uno dei modi più comuni per costruire espressioni JavaScript prevede l'impiego degli *operatori*:

```
// Gli operatori agiscono sui loro valori (gli operandi) per produrre un nuovo valore.
// Gli operatori aritmetici sono alcuni degli operatori più semplici:
3 + 2                       // => 5: addizione
3 - 2                       // => 1: sottrazione
3 * 2                       // => 6: moltiplicazione
3 / 2                       // => 1.5: divisione
points[1].x - points[0].x   // => 1: potete impiegare anche operandi più complicati
"3" + "2"                   // => "32": + somma i numeri, ma concatena le stringhe

// JavaScript definisce alcuni operatori aritmetici abbreviati
let count = 0;   // Definisce una variabile
count++;         // Incrementa la variabile
count--;         // Decrementa la variabile
count += 2;      // Somma 2: equivale a count = count + 2;
count *= 3;      // Moltiplica per 3: equivale a count = count * 3;
count            // => 6: anche i nomi delle variabili sono espressioni.

// Gli operatori di uguaglianza e relazionali verificano se due valori sono uguali,
// differenti, minori di, maggiori di e così via. Il risultato è true (vero) o false.
let x = 2, y = 3;   // Questi segni = sono assegnamenti, non test di uguaglianza
x === y             // => false: uguaglianza
x !== y             // => true: disuguaglianza
x < y               // => true: minore di
x <= y              // => true: minore o uguale
x > y               // => false: maggiore di
x >= y              // => false: maggiore o uguale
"two" === "three"   // => false: le due stringhe sono diverse
"two" > "three"     // => true: "tw" è alfabeticamente maggiore di "th"
false === (x > y)   // => true: false è uguale a false

// Gli operatori logici combinano o invertono i valori booleani
(x === 2) && (y === 3)   // => true: entrambi i confronti sono true. && è AND
(x > 3) || (y < 3)       // => false: nessuno dei confronti è true. || è OR
!(x === y)               // => true: ! inverte un valore booleano
```

Se le espressioni JavaScript sono come frammenti di frasi, allora le *istruzioni* JavaScript sono frasi di senso compiuto. Le istruzioni sono l'argomento del Capitolo 5. In generale, un'espressione è qualcosa che calcola un valore ma non *fa* nulla: non altera in alcun modo lo stato del programma. Le istruzioni, al contrario, non hanno un valore, ma alterano lo stato. Qui avete visto istruzioni che eseguivano la dichiarazione e l'assegnamento di variabili. L'altra grande categoria di istruzioni è costituita dalle *strutture di controllo*, come le istruzioni condizionali e per cicli. Vedrete degli esempi di seguito, dopo aver trattato le funzioni.

Una *funzione* è un blocco di codice JavaScript dotato di un nome e di parametri, che viene definito una sola volta e può quindi essere richiamato più e più volte. Le funzioni saranno trattate formalmente nel Capitolo 8, ma, come nel caso degli oggetti e degli array, le vedrete molte volte, prima di arrivare a quel capitolo. Ecco alcuni semplici esempi:

```
// Le funzioni sono blocchi parametrizzati di codice JavaScript, che possiamo richiamare.
function plus1(x) {   // Definisce una funzione di nome "plus1" e con parametro "x"
    return x + 1;     // Restituisce un valore maggiore di 1 unità rispetto al valore passato
}                     // Le funzioni sono racchiusi fra parentesi graffe

plus1(y)                    // => 4: y è 3, quindi questa chiamata restituisce 3 + 1

let square = function(x) {  // Le funzioni sono valori e possono essere assegnate alle variabili
    return x * x;           // Calcola il valore della funzione
};                          // Il punto e virgola segna la fine dell'assegnamento.

square(plus1(y))           // => 16: richiama due funzioni in un'unica espressione
```

In ES6 e versioni successive, esiste una sintassi compatta per definire le funzioni. Questa sintassi compatta usa => per separare l'elenco degli argomenti dal corpo della funzione, motivo per cui le funzioni definite in questo modo sono chiamate *funzioni freccia*. Le funzioni freccia vengono comunemente utilizzate per passare una funzione senza nome come argomento a un'altra funzione. Il codice precedente assume il seguente aspetto, se riscritto in modo da utilizzare funzioni freccia:

```
const plus1 = x => x + 1;    // L'input x viene associato all'output x + 1
const square = x => x * x;   // L'input x viene associato all'output x * x
plus1(y)                     // => 4: la chiamata a funzione è la stessa
square(plus1(y))             // => 16
```

Quando usiamo le funzioni con gli oggetti, otteniamo i *metodi*:

```
// Quando le funzioni vengono assegnate alle proprietà di un oggetto, le chiamiamo
// "metodi". Tutti gli oggetti JavaScript (inclusi gli array) hanno dei metodi:
let a = [];      // Crea un array vuoto
a.push(1,2,3);   // Il metodo push() aggiunge elementi a un array
a.reverse();     // Un altro metodo: inverte l'ordine degli elementi

// Possiamo anche definire nuovi metodi. La parola riservata "this" fa riferimento all'oggetto
// in cui è definito il metodo: in questo caso, l'array points precedentemente definito.
Points.dist = function() {       // Definisce un metodo per calcolare la distanza tra i punti
    let p1 = this[0];            // Primo elemento dell'array su cui richiamiamo la funzione
    let p2 = this[1];            // Secondo elemento dell'oggetto "this"
    let a = p2.x - p1.x;         // Differenza sull'asse x
    let b = p2.y - p1.y;         // Differenza sull'asse y
    return Math.sqrt(a * a +     // Teorema di Pitagora
                     b * b);     // Math.sqrt() calcola la radice quadrata
};
points.dist()                    // => Math.sqrt(2): distanza tra i nostri due punti
```

Ora, come promesso, ecco alcune funzioni il cui contenuto mostra l'uso delle istruzioni per strutture di controllo:

```
// JavaScript include istruzioni condizionali e per cicli,
// con una sintassi analoga a quella dei linguaggi C, C++, Java e altri.
function abs(x) {      // Funzione per calcolare il valore assoluto.
    if (x >= 0) {      // L'istruzione if...
        return x;      // esegue questo codice se il confronto è true.
    }                  // Questa è la fine della clausola if.
    else {             // La clausola else opzionale esegue il suo codice
        return -x;     // solo se il confronto è false. Le parentesi graffe
    }                  // sono opzionali quando la clausola contiene una sola dichiarazione.
}                      // Notate le istruzioni return annidate all'interno dell'if/else.
abs(-10) === abs(10)   // => true

function sum(array) {        // Calcola la somma degli elementi di un array
    let sum = 0;             // Parte con sum uguale a 0.
    for(let x of array) {    // Ciclo sull'array, assegna ogni elemento a x.
        sum += x;            // Somma a sum il valore dell'elemento.
    }                        // Questa è la fine del ciclo.
    return sum;              // Restituisce sum.
}
sum(primes)                  // => 28: somma dei primi 5 numeri primi 2 + 3 + 5 + 7 + 11

function factorial(n) {   // Funzione per calcolare i fattoriali
    let product = 1;      // Inizia con product uguale a 1
    while(n > 1) {        // Ripete le istruzioni fra {} mentre è vera l'espressione fra ()
        product *= n;     // Equivale a product = product * n;
        n--;              // Equivale a n = n - 1
    }                     // Fine del ciclo
    return product;       // Restituisce product
}
factorial(4)              // => 24: 4 * 3 * 2 * 1

function factorial2(n) {       // Un'altra versione, che utilizza un ciclo differente
    let i, product = 1;        // Tutto parte da 1
    for(i = 2; i <= n; i++)    // Incrementa automaticamente i da 2 fino a n
        product *= i;          // Ripete ogni volta. {} non necessarie per cicli di 1 riga.
    return product;            // Restituisce il fattoriale
}
factorial2(5)                  // => 120: 1 * 2 * 3 * 4 * 5
```

JavaScript supporta anche lo stile di programmazione a oggetti, ma in modo significativamente diverso dai linguaggi di programmazione a oggetti "classici". Il Capitolo 9 tratta in dettaglio la programmazione a oggetti in JavaScript, offrendo molti esempi. Ecco però un esempio molto semplice, che mostra come definire una classe JavaScript per rappresentare punti geometrici bidimensionali. Gli oggetti che sono istanze di questa classe hanno un unico metodo, `distance()`, che calcola la distanza del punto dall'origine:

```
class Point {                   // Per convenzione, i nomi delle classi sono in maiuscolo.
    constructor(x, y) {         // Funzione costruttore per inizializzare le nuove istanze.
        this.x = x;             // this è il nuovo oggetto che viene inizializzato. Memorizza
        this.y = y;             // gli argomenti della funzione come proprietà dell'oggetto.
    }                           // Le funzioni constructor non richiedono alcun return.

    distance() {                // Metodo per calcolare la distanza del punto dall'origine.
        return Math.sqrt(       // Restituisce la radice quadrata di x² + y².
            this.x * this.x +   // Fa riferimento all'oggetto Point su cui
            this.y * this.y     // viene richiamato il metodo distance.
        );
    }
}

// Usa la funzione construct di Point() con "new" per creare un nuovo oggetto Point
let p = new Point(1, 1);        // Il punto geometrico (1,1).

// Ora usa un metodo dell'oggetto Point p
p.distance()                    // => Math.SQRT2
```

Questo tour introduttivo sulla sintassi e sulle funzionalità fondamentali di JavaScript termina qui, ma i singoli capitoli del libro tratteranno dettagliatamente tutte funzionalità del linguaggio.

- *Capitolo 10, I moduli* - Mostra in quale modo il codice JavaScript di un file o script può utilizzare le funzioni e le classi JavaScript definite in altri file o script.
- *Capitolo 11, La libreria standard JavaScript* - Tratta le funzioni e le classi integrate disponibili per tutti i programmi JavaScript. Ciò include alcune strutture di dati importanti, come le mappe e i set, una classe per espressioni regolari, per individuare pattern testuali, funzioni per serializzare le strutture di dati JavaScript e molto altro ancora.
- *Capitolo 12, Gli iteratori e i generatori* - Spiega il funzionamento dei cicli `for/of` e come creare classi iterabili con `for/of`. Inoltre tratta le funzioni generative e l'istruzione `yield`.
- *Capitolo 13, JavaScript asincrono* - Questo capitolo esplora in modo approfondito la programmazione asincrona in JavaScript, parlando di callback ed eventi, di API Promise e delle parole riservate `async` e `await`. Sebbene il nucleo centrale del linguaggio JavaScript non sia asincrono, le API asincrone sono normalmente utilizzate sia nei browser web sia in Node, e questo capitolo spiega le tecniche impiegabili per lavorare con tali API.
- *Capitolo 14, Metaprogrammazione* - Introduce una serie di funzionalità avanzate di JavaScript che potrebbero essere utili per chi scrive librerie di codice destinate all'utilizzo da parte di altri programmatori JavaScript.
- *Capitolo 15, JavaScript nei browser web* - Presenta l'ambiente host del browser web, spiega in quale modo i browser web eseguono il codice JavaScript e tratta la più importante delle API definite dai browser web. Questo è di gran lunga il capitolo più lungo del libro.
- *Capitolo 16, JavaScript server-side con Node* - Introduce l'ambiente host Node, trattando il suo modello di programmazione e le strutture di dati e API più importanti.

- *Capitolo 17, Tool ed estensioni JavaScript* - Tratta gli strumenti e le estensioni del linguaggio che vale la pena di conoscere perché sono ampiamente utilizzati e possono migliorare la vostra produttività.

Esempio: istogrammi della frequenza dei caratteri

Questo capitolo si conclude con un breve (ma non banale) programma JavaScript. L'Esempio 1.1 è un programma Node che legge il testo dallo standard input, calcola un istogramma della frequenza dei caratteri in quel testo e quindi mostra l'istogramma. Potete richiamare il programma in questo modo per analizzare la frequenza dei caratteri presenti nel vostro codice sorgente:

```
$ node charfreq.js < charfreq.js
T: ########## 11.22%
E: ########## 10.15%
R: ####### 6.68%
S: ###### 6.44%
A: ###### 6.16%
N: ###### 5.81%
O: ##### 5.45%
I: ##### 4.54%
H: #### 4.07%
C: ### 3.36%
L: ### 3.20%
U: ### 3.08%
/: ### 2.88%
```

Questo esempio utilizza una serie di funzionalità JavaScript avanzate, e ha il solo scopo di mostrarvi l'aspetto dei "veri" programmi JavaScript. Non aspettatevi di comprendere appieno il funzionamento del codice, ma siate certi che dopo aver letto i capitoli del libro, tutto risulterà molto più chiaro.

Esempio 1.1 Calcolo degli istogrammi della frequenza dei caratteri con JavaScript.

```
/**
 * Questo programma Node legge il testo dallo standard input, calcola la frequenza
 * di ogni lettera in quel testo e visualizza un istogramma dei caratteri
 * più utilizzati. Richiede Node 12 o superiore per funzionare.
 *
 * In un ambiente di tipo Unix potete richiamare il programma nel seguente modo:
 *     node charfreq.js < corpus.txt
 */

// Questa classe estende Map in modo che il metodo get() restituisca il valore
// specificato invece di null quando la chiave non è nella mappa
class DefaultMap extends Map {
    constructor(defaultValue) {
        super();                            // Richiama il costruttore della superclasse
```

```
        this.defaultValue = defaultValue;   // memorizza il valore di default
    }

    get(key) {
        if (this.has(key)) {                // Se la chiave è già nella mappa
            return super.get(key);          // restituisce il suo valore dalla superclasse.
        }
        else {
            return this.defaultValue;       // Altrimenti restituisce il valore di default
        }
    }
}

// Questa classe calcola e visualizza gli istogrammi della frequenza delle lettere
class Histogram {
    constructor() {
        this.letterCounts = new DefaultMap(0);   // Mappa dalle lettere ai conteggi
        this.totalLetters = 0;                   // Quante lettere, in tutto
    }

    // Questa funzione aggiorna l'istogramma con le lettere del testo.
    add(text) {
        // Rimuove gli spazi dal testo e converte tutto in maiuscole
        text = text.replace(/\s/g, "").toUpperCase();
        // Ora esamina in un ciclo i caratteri del testo
        for(let character of text) {
            let count = this.letterCounts.get(character);   // Prende il vecchio count
            this.letterCounts.set(character, count+1);      // Lo incrementa
            this.totalLetters++;
        }
    }

    // Converte l'istogramma in una stringa che genera un grafico ASCII
    toString() {
        // Converte la mappa in un array di array [chiave, valore]
        let entry = [...this.letterCounts];

        // Ordina l'array prima per conteggio, poi in ordine alfabetico
        entries.sort((a, b) => {              // Una funzione per definire l'ordinamento.
            if (a[1] === b[1]) {              // Se i conteggi sono gli stessi
                return a[0] < b[0] ? -1: 1;   // le ordina alfabeticamente.
            } else {                          // Se i conteggi differiscono
                return b[1] - a[1];           // ordina per conteggio maggiore.
            }
        });

        // Converte i conteggi in percentuali
        for(let entry of entries) {
            entry[1] = entry[1] / this.totalLetters * 100;
        }
```

```
        // Elimina le voci inferiori all'1%
        entries = entries.filter(entry => entry[1] >= 1);

        // Ora converte ogni voce in una riga di testo
        let lines = entries.map(
            ([l,n]) => `${l}: ${"#".repeat(Math.round(n))} ${n.toFixed(2)}%`
        );

        // Restituisce le righe concatenate, separate da un codice di fine riga.
        return lines.join ("\n");
    }
}

// Questa funzione asincrona (Promise-returning) crea un oggetto Histogram,
// legge in modo asincrono delle porzioni di testo dall'input standard e le aggiunge
// all'istogramma. Quando raggiunge la fine dello stream, restituisce l'istogramma
async function histogramFromStdin() {
    process.stdin.setEncoding("utf-8");   // Legge le stringhe Unicode, non i byte
    let histogram = new Histogram();
    for await (let chunk of process.stdin) {
        histogram.add(chunk);
    }
    Return histogram;
}

// Quest'ultima riga di codice è il corpo principale del programma.
// Crea un oggetto Histogram dallo standard input, quindi visualizza l'istogramma.
histogramFromStdin().then(histogram => {console.log(histogram.toString()); });
```

Riepilogo

Questo libro descrive il linguaggio JavaScript partendo dalle basi. Ciò significa che iniziamo con alcuni dettagli di basso livello come i commenti, gli identificatori, le variabili e i tipi; poi creeremo espressioni, istruzioni, oggetti e funzioni; quindi tratteremo le astrazioni di alto livello del linguaggio, come le classi e i moduli. Prendo sul serio la parola "definitiva" che è nel titolo del libro, e i prossimi capitoli spiegano il linguaggio a un livello di dettaglio che all'inizio può anche sembrare scoraggiante. Ma per impadronirsi davvero di JavaScript occorre comprendere anche i dettagli, e spero che troverete il tempo per leggere questo libro da cima a fondo. Ma non siete certo obbligati a farlo fin dalla prima lettura. Se un certo paragrafo proprio "non vi entra in testa", saltate pure a quello successivo. Potrete sempre ritornarvi una volta che avete una maggiore conoscenza pratica del linguaggio nel suo complesso.