

JACK FELLERS

HTML&CSS



LA GUIDA COMPLETA AL WEB DESIGN
PER PROGETTARE E SVILUPPARE SITI WEB
IN 7 GIORNI



WEBHAWK

HTML CSS

JACK FELLERS

Caro lettore, per ringraziarti per la fiducia dimostratami acquistando il mio libro, ecco per te in **regalo**, una guida per fortificare ancora di più la tua conoscenza nella programmazione web!

Scansiona il codice o clicca sul link per riscattarlo in meno di un minuto:



Link alternativo al Qr code:

<https://webhawk.tech/optin-it/>

Buona lettura!

INDICE

1. [Introduzione a html](#)
2. [HTML5, CSS3 e Responsive Web Design](#)
3. [Media Query e supporto a diversi viewport](#)
4. [Usare i layout fluidi](#)
5. [HTML5 per Responsive Designs](#)
6. [Introduzione a css](#)
7. [Le basi di CSS](#)
8. [Unità relative](#)
9. [Padroneggiare il box model](#)
10. [Floats](#)

INTRODUZIONE A HTML

Questo libro ha due scopi, insegnarti il linguaggio HTML ma con un occhio alle versioni mobile, che spesso i programmatori tendono a trascurare. Se pensi di dover creare una versione "mobile" del tuo sito web, ripensaci! È possibile creare un sito web reattivo, con un design che si presenta benissimo su smartphone, desktop e tutti gli altri dispositivi. Si adatterà senza alcuno sforzo alle dimensioni dello schermo dell'utente, fornendo la migliore user experience sia per i dispositivi di oggi che per quelli di domani.

Questo libro fornisce il "know how" necessario e per farlo useremo un progetto a larghezza fissa esistente e lo renderemo reattivo. Inoltre, applicheremo le ultime e più utili tecniche fornite da HTML5 e CSS3, rendendo il design più snello e manutenibile. Spiegheremo anche quali sono le best practice comuni per scrivere e distribuire il codice, le immagini e i file. Alla fine del libro sarai in grado di capire HTML e CSS e potrai creare il tuo web design reattivo.

Di cosa tratta questo libro

Il Capitolo 1, Introduzione a HTML5, CSS3 e Responsive Web Design, definisce cos'è il design web reattivo, fornisce esempi di design reattivo e mette in evidenza i vantaggi dell'utilizzo di HTML5 e CSS3.

Il Capitolo 2, Media query: supporto a diverse viste, spiega quali sono le media query, come scriverle e come possono essere applicate a qualsiasi progetto per adattare il CSS alle capacità di un dispositivo.

Il Capitolo 3, Layout "fluidi", spiega i vantaggi di un layout fluido e mostra come convertire facilmente un progetto a larghezza fissa in un layout fluido o utilizzare un framework CSS per prototipare rapidamente un design reattivo.

Il Capitolo 4, HTML5 per il Responsive Designs, esplora i numerosi vantaggi della codifica con HTML5 (codice più snello, elementi semantici, memorizzazione nella cache offline e WAI-ARIA per tecnologie assistive).

Cosa ti serve per questo libro

Avrai bisogno di un po' di dimestichezza con HTML e CSS ma se non ne hai mai sentito parlare, ti basterà solo un po' di curiosità. Può esserti utile anche una conoscenza di base di JavaScript ma non è necessaria.

A chi è rivolto questo libro

Stai scrivendo due siti Web, uno per dispositivi mobile e uno per display più grandi? O forse hai sentito parlare di "design reattivo" ma non sei sicuro di come unire HTML5 e CSS3 in un design reattivo. Se questa è la tua condizione, questo libro fornisce tutto ciò di cui hai bisogno per portare le tue pagine web ad un livello superiore, evitando di restare indietro! Questo libro è rivolto a web designer e sviluppatori web che attualmente creano siti web a larghezza fissa con HTML e CSS. Questo libro spiega, inoltre, come creare siti web responsive con HTML5 e CSS3 che si adattano a qualsiasi dimensione dello schermo.

Convenzioni

In questo libro troverai una serie di stili di testo che distinguono tra diversi tipi di informazioni. Ecco alcuni esempi di questi stili e una spiegazione del loro significato. Le parole che fanno riferimento al codice sono mostrate come segue: "HTML5 accetta anche una sintassi molto slacker per essere considerata "valida".

Ad esempio, `<script src=js/jquery-1.6.2.js></script>` è valido quanto l'esempio precedente.

Un blocco di codice è impostato come segue:

```
<div class="header">
<div class="navigation">
<ul class="nav-list">
<li><a href="#" title="Home">Home</a></li>
<li><a href="#" title="About">Chi siamo</a></li>
</ul>
</div> <!--fine di navigation -->
</div> <!-- fine di header -->
```

Quando desideriamo attirare la tua attenzione su una parte particolare di un blocco di codice, le righe o gli elementi pertinenti sono impostati in grassetto:

```
#wrapper {
margin-right: auto;
margin-left: auto;
width: 96%; /* Holding outermost DIV */
}
#header {
margin-right: 10px;
margin-left: 10px;
width: 97.9166667%; /* 940 ÷ 960 */
}
```

I NUOVI TERMINI e le parole importanti sono mostrati in grassetto. Le parole che vedi sullo schermo, nei menu o nelle finestre di dialogo, ad

esempio, appaiono nel testo in questo modo: "Ad esempio, il menu di navigazione non alterna i colori rosso e nero, il pulsante principale **HAI VINTO** nell'area dei contenuti e il pulsante **informazioni complete** dalla barra laterale, così come i caratteri, sono tutti molto lontani da quelli mostrati nel file grafico”.

HTML5, CSS3 E RESPONSIVE WEB DESIGN

Fino a poco fa, i siti Web potevano essere creati con una larghezza fissa, ad esempio 960 pixel, con l'aspettativa che tutti gli utenti finali ricevessero un'esperienza abbastanza coerente. Questa larghezza fissa non era troppo ampia per gli schermi dei laptop e gli utenti con monitor ad alta risoluzione avevano semplicemente un'abbondanza di margine su entrambi i lati. Ma ora ci sono gli smartphone. L'iPhone di Apple ha inaugurato la prima esperienza di navigazione del telefono veramente utilizzabile e molti altri hanno ora seguito quell'esempio. A differenza delle implementazioni di navigazione web su piccolo schermo precedenti, che richiedevano la destrezza del pollice di un campione del mondo per essere utilizzate, le persone ora usano comodamente i loro telefoni per navigare sul Web. Inoltre, c'è una crescente tendenza dei consumatori a utilizzare dispositivi a schermo piccolo (tablet e netbook, ad esempio) rispetto ai loro fratelli a schermo intero per il consumo di contenuti multimediali. Il fatto indiscutibile è che il numero di persone che utilizzano questi dispositivi con schermo più piccolo per visualizzare Internet sta crescendo a un ritmo sempre crescente, mentre all'altro capo della scala, ora anche i display da 27 e 30 pollici sono all'ordine del giorno. Oggi c'è una differenza maggiore tra gli schermi più piccoli che navigano sul Web e quelli più grandi.

Per fortuna, esiste una soluzione a questo panorama di browser e dispositivi in continua espansione. Un web design reattivo, realizzato con HTML5 e CSS3, consente a un sito Web di "funzionare" su più dispositivi e schermi. E la parte migliore è che le tecniche sono tutte implementate senza la necessità di soluzioni basate su server/backend.

In questo capitolo dovremo:

- Scoprire l'importanza di supportare dispositivi con schermo piccolo
- Definire il design di un "sito web mobile"
- Definire il design di un "sito web reattivo"
- Osservare ottimi esempi di web design reattivo
- Scoprire la differenza tra viewport e dimensioni dello schermo
- Installare e usare le estensioni del browser per modificare il viewport

- Usare HTML5 per creare markup più puliti e snelli
- Utilizzare CSS3 per risolvere problemi di progettazione comuni

Perché gli smartphone sono importanti (e non il vecchio IE)

Sebbene le statistiche debbano essere utilizzate solo come guida approssimativa, è interessante notare che secondo gs.statcounter.com, nei 12 mesi da luglio 2010 a luglio 2011, l'utilizzo globale del browser mobile è aumentato dal 2,86 al 7,02%, immaginiamo come possa essere la situazione oggi. Molte più persone stanno ora navigando da un telefono cellulare rispetto a un desktop o laptop. C'è un numero crescente di persone che utilizzano dispositivi con schermo piccolo per navigare in Internet e i browser Internet di questi dispositivi sono stati generalmente progettati per gestire i siti Web esistenti senza problemi. Lo fanno rimpicciolendo un sito Web standard per adattarlo all'area visibile (o **viewport** per dargli il termine tecnico corretto) del dispositivo. L'utente, quindi, ingrandisce l'area del contenuto a cui è interessato. Eccellente, quindi perché noi, come designer e sviluppatori frontend, dobbiamo intraprendere ulteriori azioni? Bene, più navighi su siti Web, su iPhone e telefoni Android, più diventano evidenti i motivi. È noioso e frustrante ingrandire e rimpicciolire costantemente le aree della pagina per vederle a una dimensione leggibile e quindi spostare la pagina a sinistra e a destra per leggere le frasi che sono fuori dallo schermo. Tutto ciò è abbastanza fastidioso perché devi anche evitare di toccare inavvertitamente un link che non vuoi aprire. Sicuramente possiamo fare di meglio!

Ci sono momenti in cui un design reattivo non è la scelta giusta

Laddove i budget lo consentano e la situazione lo richieda, la versione mobile di un sito Web è sicuramente l'opzione preferita. Si tratta di fornire contenuti, design e interazioni adeguati al dispositivo, alla posizione, alla velocità di connessione e a tante altre variabili, comprese le capacità tecniche del dispositivo. Come esempio pratico, immagina una catena di negozi di abbigliamento, potrebbe avere un sito Web "standard" e una versione "mobile" che aggiunga una funzionalità di realtà aumentata che, sfruttando la posizione GPS corrente, aiuti a trovare il negozio più vicino. Questo tipo di soluzione ha bisogno di molto più di un design reattivo. Tuttavia, sebbene non tutti i progetti richiedano quelle funzionalità, in quasi tutti gli altri casi sarebbe comunque preferibile fornire agli utenti una visione personalizzata dei contenuti in base alle dimensioni del loro viewport. Ad esempio, sulla maggior parte dei siti, sebbene vengano offerti gli stessi contenuti, sarebbe meglio variare il modo in cui vengono visualizzati. Su schermi piccoli, gli elementi di minore importanza verranno posti sotto il contenuto principale, o come scenario peggiore, nascosti del tutto. Sarebbe utile anche alterare i pulsanti di navigazione per adattarsi alla pressione delle dita, piuttosto che offrire un'esperienza utilizzabile solo a coloro in grado di offrire un clic preciso del mouse! Anche i caratteri dovrebbero essere ridimensionati per motivi di leggibilità, consentendo la lettura del testo senza richiedere continui scorrimenti da un lato all'altro. Allo stesso modo, mentre ci occupiamo dei viewport più piccoli, non dobbiamo compromettere il design per coloro che utilizzano schermi grandi per laptop, desktop o addirittura TV.

Definizione di responsive design

Il termine **responsive design** è stato coniato da Ethan Marcotte. Nel suo articolo fondamentale su List Apart ha consolidato tre tecniche esistenti (layout flessibile con griglia, immagini flessibili, media e media query) in un approccio unificato e lo ha chiamato responsive web design. Il termine è spesso usato per dedurre lo stesso significato di una serie di altre descrizioni come design fluido, layout elastico, design liquido, layout adattivo, design cross-device e design flessibile. Solo per citarne alcuni! Tuttavia, come hanno eloquentemente affermato Mr. Marcotte e altri, una metodologia veramente reattiva è in realtà molto più che modificare il layout di un sito in base alle dimensioni del viewport, infatti, si tratta di invertire il nostro intero approccio attuale al web design. Al posto di iniziare con un design del sito desktop a larghezza fissa e ridimensionarlo per ridistribuire il contenuto per viewport più piccoli, dovremmo prima progettare per il viewport più piccolo e poi migliorare progressivamente il design e il contenuto per viewport più grandi. Per tentare di riassumere la filosofia del responsive web design, direi che è la presentazione dei contenuti nel modo più accessibile per qualsiasi viewport. Al contrario, un vero "sito web mobile" è necessario quando richiede contenuti e funzionalità specifici in base al dispositivo che vi accede. In questi casi, un sito Web mobile presenta un'esperienza utente del tutto diversa dal suo equivalente desktop.

Perché fermarsi al design reattivo?

Un web design reattivo gestirà il flusso del contenuto della nostra pagina man mano che i viewport cambiano, ma andiamo oltre. HTML5 ci offre molto di più rispetto ad HTML 4 e i suoi elementi semantici più significativi formeranno la base del nostro markup. Le media query CSS3 sono un ingrediente essenziale per un design reattivo, infatti, i moduli aggiuntivi CSS3 ci conferiscono livelli di flessibilità mai visti prima. Elimineremo porzioni di sfondo e complicato codice JavaScript, sostituendoli con gradienti, ombre, tipografia, animazioni e trasformazioni in CSS3 semplici e snelli. Prima di procedere con la creazione di un web design reattivo basato su HTML5 e CSS3, diamo prima un'occhiata ad alcuni esempi come stato dell'arte. C'è già chi ha fatto un buon lavoro con HTML5 reattivo e CSS3 quindi, cosa possiamo imparare dai loro sforzi pionieristici?

Esempi di design web reattivo

Per testare completamente il design del tuo sito Web reattivo e quello degli altri sarebbe necessaria una configurazione dedicata per ogni dispositivo e dimensione dello schermo. Sebbene nulla migliori questa pratica, la maggior parte dei test può essere ottenuta semplicemente ridimensionando la finestra del browser. Per aiutare ulteriormente questo metodo, ci sono vari plug-in di terze parti ed estensioni del browser che mostrano la finestra del browser corrente o le dimensioni del viewport in pixel. Oppure, in alcuni casi, essi cambiano automaticamente la finestra o la adattano ad una dimensione dello schermo predefinita (1024 x 768 pixel, ad esempio). Ciò ti consente di testare più facilmente cosa accade quando le dimensioni dello schermo cambiano. Ricorda, non attaccarti molto ai pixel come unità di misura perché in molti casi li abbandoneremo e ci sposteremo su unità di misura relative (in genere, "em" o "ems" e percentuali), non appena ci addentriamo nel responsive design.

HTML5: perché?

HTML5 pone l'accento sullo snellimento del markup necessario per creare una pagina che sia conforme agli standard del W3C e che colleghi tutti i nostri file tra cui CSS, JavaScript e immagini. Per gli utenti di smartphone, che possono visualizzare le nostre pagine con una larghezza di banda limitata, vogliamo che il nostro sito Web non solo risponda alla loro visualizzazione più limitata, ma soprattutto che venga caricato nel più breve tempo possibile. Nonostante la rimozione di elementi di markup superflui rappresenta solo un piccolo risparmio di dati, HTML5 offre ulteriori vantaggi e funzionalità aggiuntive rispetto alla precedente versione (HTML 4.01). È probabile che gli sviluppatori web frontend siano principalmente interessati ai nuovi elementi semantici di HTML5 che forniscono codice più significativo ai motori di ricerca. HTML5, inoltre, consente anche un feedback all'utente sull'interattività di base del sito come l'invio di form e così via, evitando l'elaborazione di moduli JavaScript, solitamente più pesanti. Ancora una volta, questa è una buona notizia per il nostro design reattivo, che ci consente di creare una codebase più snella e con tempi di caricamento più rapidi.

La prima riga di qualsiasi documento HTML inizia con Doctype (Dichiarazione del tipo di documento). Questa è la parte che, ad essere onesti, viene aggiunta automaticamente dal nostro editor di codice preferito o possiamo incollarla da un template esistente (nessuno davvero ricorda a memoria il Doctype HTML 4.01 completo). Prima di HTML5, il Doctype per una pagina HTML 4.01 standard avrebbe avuto il seguente aspetto:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

Adesso con HTML5, si riduce a:

```
<!DOCTYPE html>
```

Ora, come ho già ammesso, non digito fisicamente il Doctype ogni volta che scrivo una pagina, e sospetto che nemmeno tu lo faccia. Bene, che ne dici di aggiungere link a JavaScript o CSS nelle tue pagine? Con HTML 4.01, il modo corretto di collegare un file di script sarebbe il seguente:

```
<script src="js/jquery-1.6.2.js" type="text/javascript"></script>
```

In HTML5 è molto più semplice:

```
<script src="js/jquery-1.6.2.js"></script>
```

Come si può notare, la necessità di specificare l'attributo type non è più considerata necessaria. In modo analogo avviene il collegamento a file CSS; HTML5 accetta anche una sintassi molto più blanda per essere considerata "valida". Ad esempio, `<sCRipt SrC=js/jquery1.6.2.js></script>` è valido proprio come l'esempio precedente. Abbiamo ommesso le virgolette intorno all'origine dello script e abbiamo utilizzato una combinazione di caratteri maiuscoli e minuscoli nei nomi dei tag e degli attributi. Ma ad HTML5 non importa: verrà comunque convalidato dal validatore HTML5 del W3C (<https://validator.w3.org/>), questa è una buona notizia se sei un po' incurante o distratto nella scrittura del codice ma anche, in modo più utile, se vuoi eliminare ogni possibile carattere in eccesso dal tuo markup. In realtà, ci sono altre specifiche che semplificano la vita ma immagino che tu non sia convinto che sia tutto così eccitante. Quindi, diamo una rapida occhiata ai nuovi elementi semantici di HTML5.

Nuovi tag HTML5

Quando stai strutturando una pagina HTML, è normale indicare un'intestazione e una sezione dedicata alla navigazione in modo simile a questo:

```
<div class="header">
<div class="navigation">
<ul class="nav-list">
<li><a href="#" title="Home">Home</a></li>
<li><a href="#" title="Chi siamo">Chi siamo</a></li>
</ul>
</div> <!--fine di navigation -->
</div> <!--fine di header -->
```

Tuttavia, dai un'occhiata a come sarebbe con HTML5:

```
<header>
<nav>
<ul id="nav-list">
<li><a href="#" title="Home">Home</a></li>
<li><a href="#" title="Chi siamo">Chi siamo</a></li>
</ul>
</nav>
</header>
```

Hai notato la differenza? Invece di tag `<div>` per ogni elemento strutturale (sebbene con l'aggiunta di nomi di classe per scopi di stile), HTML5 ci fornisce invece alcuni elementi semanticamente più significativi da usare. Le sezioni strutturali comuni all'interno di pagine come l'intestazione e la navigazione (e molte altre come vedremo presto) ottengono i propri tag di elemento. Il nostro codice è appena diventato molto più "semantico" con il tag `<nav>` che dice al browser: "Ehi, questa sezione qui è dedicata alla navigazione". Questa è una buona indicazione per noi, ma forse ancora più importante, per i motori di ricerca infatti ora saranno in grado di comprendere meglio le nostre pagine e di classificare i nostri contenuti di conseguenza.

Quando scrivo pagine HTML, lo faccio spesso sapendo che a loro volta verranno passate alla squadra di backend (quei ragazzi che si occupano di PHP, Ruby, .NET e così via) prima che le pagine raggiungano il www. Per

non intralciare il lavoro dei colleghi del backend, commento spesso i tag di chiusura `</div>` all'interno del codice per consentire ad altri (e spesso anche a me stesso) di stabilire facilmente dove finiscono gli elementi `<div>`. HTML5 non ha bisogno di gran parte di questo compito, infatti, quando guardi il codice HTML5, un tag di chiusura di un elemento, `</header>` ad esempio, ti dice istantaneamente quale elemento si sta chiudendo, senza la necessità di aggiungere un commento. Siamo solo scoprendo alcune semantiche di HTML5 ma, prima di lasciarci trasportare, abbiamo un altro amico con cui fare conoscenza. Se c'è una cosa essenziale per questa nuova era del web design e in particolare del responsive design, è CSS3.

CSS3 consente design reattivi

Se hai vissuto l'epoca del web design dalla metà degli anni '90, ricorderai che tutti i design erano basati su tabelle e lo stile era annidato e legato al contenuto. I **Cascading Style Sheets (CSS)** sono stati introdotti come un modo per separare lo stile dal contenuto. Ci è voluto del tempo prima che i web designer entrassero nel nuovo e audace mondo del design basato su CSS, ma alcuni siti hanno aperto la strada, mostrando esattamente ciò che si poteva ottenere, visivamente, con un sistema basato su CSS. Da allora, CSS è diventato il modo standard per definire il livello di presentazione di una pagina Web. Attualmente viene usato CSS3, che si basa su CSS Livello 2 modulo per modulo, usando la specifica CSS2.1 come base. Ogni modulo aggiunge funzionalità e/o sostituisce parte della specifica CSS2.1. In termini molto semplici, ciò che conta per noi è sapere che CSS3 è costruito come un insieme di moduli "imbullonati" piuttosto che come un unico insieme consolidato. In conclusione, CSS3 non crea alcun problema, nemmeno con i browser più datati! Infatti, non c'è alcun problema per i browser più vecchi nell'includere proprietà che non capiscono. I browser meno recenti (ad esempio Internet Explorer) semplicemente salteranno le proprietà CSS3 che non possono elaborare e questo ci dà la possibilità di migliorare progressivamente i layout per i browser recenti, garantendo al contempo un ragionevole ripiego per quelli meno recenti.

Consideriamo un ostacolo di progettazione comune che tutti affrontiamo nella maggior parte dei progetti: creare un angolo arrotondato su un elemento dello schermo, ad esempio per un'interfaccia a tab o schede oppure l'angolo di un elemento come un'intestazione. Usando CSS 2.1 questo risultato potrebbe essere ottenuto usando la tecnica "a porte scorrevoli", per cui un'immagine si trova dietro l'altra. L'HTML potrebbe apparire così semplice:

```
<a href="#"><span>Box Title</span></a>
```

Aggiungiamo uno sfondo arrotondato all'elemento `<a>` creando due immagini. Il primo, chiamato `headerLeft.png`, sarebbe largo 15 pixel e alto 40 pixel e il secondo, chiamato `headerRight.png` in questo esempio, sarebbe più largo di quanto ci si aspetterebbe (280 pixel). Ciascuno sarebbe una metà della "porta scorrevole" quindi man mano che un elemento cresce (il testo all'interno dei nostri tag ``), lo sfondo riempie lo spazio creando

una soluzione con angoli arrotondati in qualche modo “a prova di futuro”. Ecco come appare il CSS in questo esempio:

```
a {  
  display: block;  
  height: 40px;  
  float: left;  
  font-size: 1.2em;  
  padding-right: 0.8em;  
  background: url(images/headerRight.png) no-repeat scroll top right;  
}  
a span {  
  background: url(images/headerLeft.png) no-repeat;  
  display: block;  
  line-height: 40px;  
  padding-left: 0.8em;  
}
```

Lo screenshot seguente mostra come appare in Google Chrome:



Questo risolve il problema di progettazione ma richiede del markup aggiuntivo (semanticamente l'elemento `` non ha valore) oltre ad aggiungere due richieste HTTP (per le immagini) verso il server per creare l'effetto sullo schermo. Potremmo combinare le due immagini in una per creare uno sprite e quindi utilizzare la proprietà CSS `background-position` per spostarla, ma in questo caso si tratta di una soluzione non flessibile. Cosa succede se il cliente vuole gli angoli abbiano un raggio più stretto? O con un colore diverso? In tal caso avremmo bisogno di rifare di nuovo le nostre immagini e, purtroppo, fino a CSS3 questa è stata la realtà della situazione in cui ci siamo trovati noi, designer e sviluppatori di frontend.

Signore e signori, siamo nel futuro e questo è cambiato con CSS3! Revisioniamo l'HTML in modo che sia solo:

```
<a href="#">Box Title</a>
```

E, per cominciare, il CSS può diventare il seguente:

```
a {  
  float: left;  
  height: 40px;  
  line-height: 40px;  
  padding-left: 0.8em;  
  padding-right: 0.8em;  
  border-top-left-radius: 8px;  
  border-top-right-radius: 8px;  
  background-image: url(images/headerTiny.png);  
  background-repeat: repeat-x;  
}
```

Lo screenshot seguente mostra come appare la versione CSS3 del pulsante nello stesso browser:



In questo esempio, le due immagini precedenti sono state sostituite con una singola immagine larga 1 pixel che viene ripetuta lungo l'asse x. Sebbene l'immagine sia larga solo 1 pixel, è alta 40 pixel, si spera più alta di qualsiasi contenuto che verrà inserito. Quando si utilizza un'immagine come sfondo, è sempre necessario "superare" l'altezza, in previsione dell'eccedenza del contenuto, il che purtroppo comporta immagini più grandi e un uso di larghezza di banda maggiore. Qui, tuttavia, a differenza della soluzione interamente basata su immagini, CSS3 si occupa degli angoli con il raggio e le relative proprietà. Il cliente vuole che gli angoli siano un po' più rotondi, diciamo 12 pixel? Nessun problema, basta modificare la proprietà border-radius a 12px e il tuo lavoro è fatto. La

proprietà CSS3 per gli angoli arrotondati è veloce, flessibile e supportata in Safari, Firefox, Opera, Chrome e Internet Explorer (dalla versione 9 in poi).

CSS3 può andare oltre, eliminando la necessità di un'immagine di sfondo sfumata e producendo l'effetto nel browser. Questa proprietà è ben supportata ma con qualcosa sulla falsariga del `linear-gradient(yellow, blue)`, lo sfondo di qualsiasi elemento può godere di un gradiente generato da CSS3. Il gradiente può essere specificato in colori solid, come valori HEX tradizionali (ad esempio, `#BFBFBF`) o utilizzando una delle modalità colore CSS3. In realtà possiamo fare qualcosa di meglio, se accettiamo che gli utenti dei browser più vecchi vedano uno sfondo a tinta unita invece di un gradiente, può essere utile uno stack CSS simile a questo, in grado di fornire un colore solid nel caso in cui il browser non sia in grado di gestire il gradiente:

```
background-color: #42c264;  
background-image: -webkit-linear-gradient(#4fec50, #42c264);  
background-image: -moz-linear-gradient(#4fec50, #42c264);  
background-image: -o-linear-gradient(#4fec50, #42c264);  
background-image: -ms-linear-gradient(#4fec50, #42c264);  
background-image: -chrome-linear-gradient(#4fec50, #42c264);  
background-image: linear-gradient(#4fec50, #42c264);
```

La proprietà `linear-gradient` indica al browser di iniziare con il primo valore di colore (`#4fec50`, in questo esempio) e passare al secondo valore di colore (`#42c264`). Noterai che nel codice CSS, la proprietà del gradiente lineare dell'immagine di sfondo è stata ripetuta con alcuni prefissi; ad esempio, `-webkit-`. Ciò consente a diversi fornitori di browser (ad esempio, `-moz-` per Mozilla Firefox, `-ms-` per Microsoft Internet Explorer e così via) di sperimentare la propria implementazione delle nuove proprietà CSS3 prima di introdurre la versione definitiva, a quel punto i prefissi non sono più necessari. Poiché i fogli di stile per loro natura si sovrappongono, posizioniamo la versione senza prefisso per ultima, in modo che sostituisca le precedenti dichiarazioni se disponibili.

Lo screenshot seguente mostra come appare il pulsante CSS3 completo nello stesso browser:



Penso che siamo d'accordo: qualsiasi differenza tra la versione dell'immagine e la versione interamente CSS è banale. La creazione di elementi visivi con CSS3 consente al nostro design reattivo di essere molto più snello rispetto a quello costruito con le immagini. Inoltre, i gradienti delle immagini sono ben supportati nei moderni browser mobile, l'unico compromesso è la mancanza di supporto per i gradienti per browser come IE 9 e versioni precedenti.

Cos'altro ha da offrire CSS3? Finora, abbiamo esaminato un esempio molto banale in cui CSS3 può aiutarti nelle attività di sviluppo quotidiane. Tuttavia, stuzzichiamo un po' il nostro appetito e vediamo quali vere prelibatezze ci consente CSS3. Sul Web troverai diversi siti che utilizzano le più recenti funzionalità, ad esempio, passando il mouse sopra alcuni elementi, questi iniziano a fluttuare. Bello, vero? In passato questo tipo di effetto sarebbe stato creato con Flash o JavaScript con diverse risorse necessarie. Qui, viene creato interamente attraverso le trasformazioni CSS3. L'uso di CSS3 anziché JavaScript o Flash rende l'animazione leggera, manutenibile e quindi perfetta per un design reattivo. I browser che supportano la funzione la usano, gli altri vedono semplicemente un'immagine statica al suo posto. Ovviamente, questi effetti non sono essenziali per nessun sito web ma sono un perfetto esempio di "miglioramento progressivo". Il supporto per le regole CSS3 come ombre di testo, gradienti, bordi arrotondati, colore RGBA e più immagini di sfondo sono tutti ampiamente supportati e forniscono modi flessibili per fornire soluzioni a problemi di progettazione comuni che hanno ci ha fatto lavorare in modo meno facile per anni.

HTML5 e CSS3 possono esserci utili oggi?

Qualsiasi strumento o tecnica dovrebbe essere utilizzata solo se l'applicazione lo richiede. In qualità di sviluppatori/progettisti frontend, i nostri progetti in genere hanno una quantità limitata di tempo e risorse disponibili per renderli finanziariamente sostenibili. Il fatto che alcuni vecchi browser non supportino i nuovi elementi semantici HTML5 o le proprietà CSS3, può essere “aggirato” grazie al numero crescente di strumenti (denominati **polyfills** poiché coprono le lacune dei browser più vecchi) per correggere i browser (principalmente IE). Alla luce di ciò, adottare un approccio per l'implementazione di un web design reattivo fin dall'inizio è sempre la politica migliore. In base alla mia esperienza, in genere chiedo quanto segue fin dall'inizio:

- Il cliente desidera supportare il maggior numero degli utenti di Internet? Se sì, è adatta una metodologia reattiva.
- Il cliente desidera la codebase più pulita, veloce e gestibile? Se sì, è adatta una metodologia reattiva.
- Il cliente comprende che l'esperienza può e deve essere leggermente diversa nei diversi browser? Se sì, è adatta una metodologia reattiva.
- Il cliente richiede che il design sia identico in tutti i browser, incluso IE in tutte le sue versioni? Se sì, il design reattivo non è più adatto.
- È probabile che il 70% o più dei visitatori attuali o previsti del sito utilizzi Internet Explorer 8 o versioni precedenti? Se sì, il design reattivo non è più adatto.

È anche importante ribadire che, laddove il budget lo consenta, a volte può capitare che una versione "mobile" completamente personalizzata di un sito Web sia un'opzione più pertinente rispetto a un design reattivo. Per motivi di chiarezza, definisco "siti web mobile" soluzioni interamente incentrate sui dispositivi mobili che forniscono contenuti o esperienze diversi ai loro utenti mobili. Non credo che qualcuno che sostenga le tecniche di progettazione web reattive sosterrrebbe che un web design

reattivo sia un sostituto adatto per un "sito web mobile" in ogni situazione. Vale la pena ribadire che un web design HTML5 e CSS3 reattivo non è una panacea per tutte le sfide di design e fruizione di contenuti. Come sempre con il web design, le specifiche di un progetto (vale a dire budget, target demografico e scopo) dovrebbero dettare l'attuazione. Tuttavia, secondo la mia esperienza, se il budget è limitato e/o la programmazione di un "sito web mobile" interamente su misura non è un'opzione praticabile, un web design reattivo offre quasi sempre un'esperienza utente migliore e più inclusiva rispetto a uno standard, a larghezza fissa. Bisogna educare i nostri clienti al fatto che i siti Web non dovrebbero apparire uguali in tutti i browser, l'ultimo ostacolo da superare prima di intraprendere un design reattivo è spesso quello della mentalità, e per certi versi, questo è forse il più difficile da superare. Ad esempio, mi viene chiesto frequentemente di convertire i progetti grafici esistenti in pagine Web basate su HTML/CSS e jQuery conformi agli standard. Nella mia esperienza, è raro (e quando dico raro, intendo che non è mai successo) che i grafici abbiano in mente qualcosa di diverso da una "versione desktop" a larghezza fissa di un sito quando producono i loro componenti di design. Il mio compito è quindi quello di creare una riproduzione perfetta in pixel di quel design in ogni browser conosciuto. La riuscita o il fallimento in questo compito definisce il successo agli occhi del mio cliente, il grafico. Questa mentalità è particolarmente radicata nei clienti con un passato nel design dei media stampati, è facile capire il loro modo di pensare: un design del progetto può essere firmato dai propri clienti, lo consegnano al progettista o sviluppatore frontend (tu o io) e quindi passiamo il nostro tempo assicurandoci che il codice finito appaia il più umanamente possibile in tutti i principali browser. Ciò che il cliente vede è ciò che il cliente ottiene. Tuttavia, se hai mai provato a ottenere un web design moderno con lo stesso aspetto in Internet Explorer di un browser conforme agli standard moderni come Safari, Firefox o Chrome, capisci le difficoltà intrinseche.

Spesso mi ci è voluto fino al 30 percento del tempo/budget assegnato a un progetto per correggere i difetti e gli errori intrinseci in questi vecchi browser. Quel tempo avrebbe potuto essere speso per migliorare e risparmiare codice per il numero crescente di utenti che visualizzano i siti nei browser moderni, piuttosto che applicare patch e modificare il codice per fornire angoli arrotondati, immagini trasparenti, elementi del modulo correttamente allineati e così via per un numero sempre più ridotto di utenti

di Internet Explorer. Sfortunatamente, l'unico antidoto a questo scenario è l'istruzione. Il cliente ha bisogno di una spiegazione del motivo per cui un design reattivo è utile, cosa comporta e perché il design finito non sarà e non dovrebbe avere lo stesso aspetto in tutti i viewport e browser. Alcuni clienti arrivano a capirlo, altri no e sfortunatamente, alcuni vogliono ancora che tutti gli angoli arrotondati e le ombre esterne appaiano identici anche in Internet Explorer 11! Quando mi avvicino a un nuovo progetto, indipendentemente dal fatto che un design responsive sia applicabile o meno, cerco di spiegare i seguenti punti al mio cliente:

- Consentire ai browser più vecchi di visualizzare le pagine in modo leggermente diverso, significa che il codice è più gestibile ed è più facile da aggiornare in futuro.
- Rendere tutti gli elementi uguali, anche su browser meno recenti (ad esempio Internet Explorer 11) aggiunge una quantità significativa di immagini a un sito Web. Questo lo rende più lento, più costoso da produrre e più difficile da mantenere.
- Un codice più snello che i browser moderni comprendono equivale a un sito web più veloce. Un sito web più veloce è mostrato più in alto nei motori di ricerca rispetto ad uno lento.
- Il numero di utenti con browser meno recenti sta diminuendo, il numero di utenti con browser moderni sta crescendo: supportiamoli!
- Soprattutto, supportando i browser moderni, puoi goderti un design web reattivo che risponde alle diverse visualizzazioni dei browser su dispositivi diversi.

Ora che abbiamo stabilito cosa intendiamo per design "reattivo" ed abbiamo esaminato ottimi esempi di design reattivo che fanno uso degli strumenti e delle tecniche che stiamo per trattare, abbiamo anche riconosciuto che dobbiamo passare da una mentalità di progettazione incentrata sul desktop a una posizione più indipendente dal dispositivo, dobbiamo pianificare prima i nostri contenuti attorno all'area di visualizzazione più piccola possibile e migliorare progressivamente l'esperienza utente. Abbiamo dato un'occhiata alla nuova specifica HTML5, abbiamo stabilito che ci sono grandi porzioni di essa che possiamo usare a nostro vantaggio, sappiamo che il nuovo markup semantico ci permetterà di

creare pagine con meno codice e più significato di quanto sarebbe stato possibile in precedenza. Il fulcro nella realizzazione di un web design completamente reattivo è CSS3. Prima di usare CSS3 per aggiungere un tocco visivo come i gradienti, gli angoli arrotondati, le ombre del testo, le animazioni e le trasformazioni al nostro design, lo useremo prima per svolgere un ruolo più fondamentale. Utilizzando le media query CSS3, saremo in grado di indirizzare regole CSS specifiche a viste specifiche. Il prossimo capitolo è il punto in cui inizieremo sul serio la nostra ricerca di "design reattivo".

MEDIA QUERY E SUPPORTO A DIVERSI VIEWPORT

Come abbiamo notato nell'ultimo capitolo, CSS3 è costituito da una serie di moduli “imbullonati” tra loro e le media query sono solo uno di questi moduli CSS3. Le media query ci consentono di indirizzare stili CSS specifici a seconda delle capacità di visualizzazione di un dispositivo. Ad esempio, con poche righe di CSS possiamo cambiare il modo in cui il contenuto viene visualizzato in base alla larghezza del viewport, le proporzioni dello schermo, l'orientamento (orizzontale o verticale) e così via.

In questo capitolo:

- Scopriremo perché le media query sono necessarie per un web design reattivo
- Scopriremo come viene costruita una media query CSS3
- Capiremo quali caratteristiche del dispositivo possiamo sfruttare
- Scriveremo la nostra prima media query CSS3
- Indirizzeremo le regole di stile CSS a viste specifiche
- Scopriremo come far funzionare le media query su dispositivi iOS e Android.

Oggi puoi già utilizzare le media query e godere di un ampio livello di supporto dei browser (Firefox, Safari, Chrome, Opera, iOS Safari, Opera Mobile, Android e Internet Explorer 9+). Inoltre, ci sono facili correzioni da implementare (sebbene basate su JavaScript) per i browser obsoleti come Internet Explorer.

Perché i design reattivi richiedono media query? Senza il modulo di media query CSS3, non saremmo in grado di indirizzare particolari stili CSS a particolari capacità del dispositivo, come la larghezza del viewport. Se leggi le specifiche W3C del modulo di query multimediali CSS3, vedrai che questa è la loro introduzione ufficiale a cosa sono le media query:

HTML 4 e CSS2 attualmente supportano fogli di stile dipendenti dai media adattati per diversi tipi di media. Ad esempio, un documento può utilizzare font sans-serif quando viene visualizzato su uno schermo e font serif quando viene stampato. 'screen' e 'print' sono due tipi di supporto che sono stati definiti ma le media query estendono la funzionalità consentendo un'etichettatura più precisa dei fogli di stile. Una media query è costituita

da un tipo di supporto e da zero o più espressioni che controllano le condizioni di funzionalità multimediali. Tra le funzionalità multimediali che possono essere utilizzate nelle media query ci sono "width", "height" e "color". Utilizzando le media query, le presentazioni possono essere adattate a una gamma specifica di dispositivi di output senza modificare il contenuto stesso.

Quindi, che aspetto ha una media query CSS e, soprattutto, come funziona? Scrivi il seguente codice in fondo a qualsiasi file CSS e visualizzare in anteprima la relativa pagina Web:

```
body {  
  background-color: grey;  
}  
@media screen and (max-width: 960px) {  
  body {  
    background-color: red;  
  }  
}  
@media screen and (max-width: 768px) {  
  body {  
    background-color: orange;  
  }  
}  
@media screen and (max-width: 550px) {  
  body {  
    background-color: yellow;  
  }  
}  
@media screen and (max-width: 320px) {  
  body {  
    background-color: green;  
  }  
}
```

Ora, visualizza in anteprima il file in un browser moderno (almeno IE 9 se usi IE) e ridimensiona la finestra del browser. Il colore dello sfondo della pagina varia in base alle dimensioni del viewport corrente. Ho usato il nome dei colori per chiarezza, ma normalmente potresti usare un codice HEX; ad esempio, #ffffff. Ora, andiamo avanti e analizziamo queste domande sulle

media query per capire come possiamo sfruttarle al meglio. Se sei abituato a lavorare con i fogli di stile CSS2 saprai che è possibile specificare il tipo di dispositivo (ad esempio, screen o print) applicabile a un foglio di stile con l'attributo media del tag <link>. Puoi farlo inserendo un link come fatto nel seguente snippet di codice all'interno dei tag <head> del tuo HTML:

```
<link rel="stylesheet" type="text/css" media="screen" href="screenstyles.css">
```

Ciò che le media query forniscono principalmente è la capacità di indirizzare gli stili in base alla capacità o alle caratteristiche di un dispositivo, piuttosto che semplicemente al tipo di dispositivo. Pensala come una domanda per il browser. Se la risposta del browser è "true", vengono applicati gli stili inclusi, se invece la risposta è "false", non vengono applicati. Invece di chiedere semplicemente al browser "Sei uno schermo?", per quanto potremmo effettivamente chiedere con solo CSS2, le media query chiedono delle informazioni in più. Una media query potrebbe chiedere: "Sei uno schermo e sei in orientamento verticale?" Diamo un'occhiata a questo come esempio:

```
<link rel="stylesheet" media="screen and (orientation: portrait)" href="portrait-screen.css" />
```

Innanzitutto, l'espressione della media query chiede il tipo (sei uno schermo?), quindi la funzione (lo schermo è con orientamento verticale?). Il foglio di stile portrait-screen.css verrà caricato per qualsiasi dispositivo con schermo con orientamento verticale e verrà ignorato per tutti gli altri. È possibile invertire la logica di qualsiasi espressione di media query aggiungendo la parola chiave *not* all'inizio della media query. Ad esempio, il codice seguente annullerebbe il risultato nel nostro esempio precedente, caricando il file per qualsiasi vista che non sia uno schermo con orientamento verticale:

```
<link rel="stylesheet" media="not screen and (orientation: portrait)" href="portrait-screen.css" />
```

È anche possibile mettere insieme più espressioni. Estendiamo il nostro primo esempio e limitiamo anche il file ai dispositivi con una finestra di visualizzazione maggiore di 800 pixel.

```
<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px)" href="800wide-portrait-screen.css" />
```

Inoltre, potremmo avere un elenco di media query. Se una delle query elencate è vera, il file verrà caricato, se invece nessuna è vera, non verrà

caricato:

```
<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px), projection" href="800wide-portrait-screen.css" />
```

Ci sono due punti da notare qui. In primo luogo, una virgola separa ogni media query. In secondo luogo, noterai che dopo la *projection* non c'è alcuna combinazione finale e/o caratteristica/valore tra parentesi. Questo perché in assenza di questi valori, la media query viene applicata a tutti i tipi di media. Nel nostro esempio, gli stili verranno applicati a tutti i proiettori. Proprio come le regole CSS esistenti, le media query possono caricare condizionalmente gli stili in vari modi. Finora li abbiamo inclusi come collegamenti a file CSS che inseriremmo nella sezione `<head>` del nostro HTML. Tuttavia, possiamo anche utilizzare le media query all'interno degli stessi fogli di stile CSS. Ad esempio, se aggiungiamo il seguente codice in un foglio di stile, tutti gli elementi `h1` saranno verdi, a condizione che il dispositivo abbia una larghezza dello schermo di 400 pixel o meno:

```
@media screen and (max-device-width: 400px) {  
  h1 { color: green }  
}
```

Possiamo anche utilizzare la funzione `@import` di CSS per caricare condizionalmente i fogli di stile nel nostro foglio di stile esistente. Ad esempio, il codice seguente importerebbe il foglio di stile chiamato `phone.css`, a condizione che il dispositivo sia basato su schermo e abbia un viewport massimo di 360 pixel:

```
@import url("phone.css") screen and (max-width:360px);
```

Ricorda che l'utilizzo della funzione `@import` di CSS, aggiunge delle richieste HTTP (che influiscono sulla velocità di caricamento); quindi usa questo metodo con parsimonia.

Per cosa possono essere usate le media query?

Quando si creano progetti reattivi, le media query che vengono utilizzate più spesso si riferiscono alla larghezza del viewport di un dispositivo (*width*) e alla larghezza dello schermo del dispositivo (*device-width*). Nella mia esperienza, ho trovato poca richiesta per le altre capacità che possiamo testare. Tuttavia, nel caso se ne presentasse la necessità, ecco un elenco di tutte le funzionalità per le quali le media query possono essere testate.

Si spera che alcune suscitino il tuo interesse:

- *width*: la larghezza del viewport.
- *height*: l'altezza del viewport.
- *device-width*: la larghezza della superficie di rendering (per i nostri scopi, questa è in genere la larghezza dello schermo di un dispositivo).
- *device-height*: l'altezza della superficie di rendering (per i nostri scopi, questa è in genere l'altezza dello schermo di un dispositivo).
- *orientation*: questa funzionalità controlla se un dispositivo è con orientamento verticale o orizzontale.
- *aspect-ratio*: il rapporto tra larghezza e altezza in base alla larghezza e all'altezza del viewport. Un display widescreen 16:9 può essere scritto come *aspect-ratio: 16/9*;
- *device-aspect-ratio*: questa capacità è simile alla precedente ma si basa sulla larghezza e l'altezza della superficie di rendering del dispositivo, piuttosto che sul viewport.
- *color*: il numero di bit per la componente colore. Ad esempio, *min-color: 16* verificherà che il dispositivo abbia un colore a 16 bit.
- *color-index*: il numero di voci nella tabella di ricerca dei colori del dispositivo. I valori devono essere numeri e non possono essere negativi.
- *monochrome*: questa funzionalità verifica quanti bit per pixel si trovano in un frame buffer monocromatico. Il valore è un

numero (intero), ad esempio *monochrome: 2*, e non può essere negativo.

- *resolution*: questa funzionalità può essere utilizzata per testare la risoluzione dello schermo o della stampa; ad esempio, *min-resolution: 300 dpi*. Può accettare anche misure in punti per centimetro; ad esempio, *min-resolution: 118 dpcm*.
- *scan*: può trattarsi di funzioni progressive o interlacciate in gran parte specifiche dei televisori. Ad esempio, un televisore HD 720p (la p di 720p indica "progressivo") potrebbe essere indicato con *scan: progressive* mentre un televisore HD 1080i (la i di 1080i indica "interlacciato") potrebbe essere indicato con *scan: interlace*.
- *grid*: questa funzionalità indica se il dispositivo è basato su griglia o bitmap.

Tutte le funzioni di cui sopra, ad eccezione di *scan* e *grid*, possono essere precedute da *min* o *max* per creare intervalli. Ad esempio, considera il seguente frammento di codice:

```
@import url("phone.css") screen and (min-width:200px) and (maxwidth:360px);
```

In questo caso, un minimo (min) e un massimo (max) sono stati applicati alla larghezza per impostare un intervallo. Il file *phone.css* verrà importato solo per i dispositivi con schermo con una larghezza di viewport minima di 200 pixel e una larghezza di viewport massima di 360 pixel.

Per la serie "*repetita iuvant*", CSS sta per Cascading Style Sheet e, per loro stessa natura, gli stili posizionati più in basso in un foglio di stile a cascata sovrascrivono gli stili equivalenti e posti più in alto (a meno che gli stili più in alto non siano più specifici). Possiamo quindi impostare gli stili di base all'inizio di un foglio di stile, applicabili a tutte le versioni del nostro progetto e quindi sovrascrivere le sezioni pertinenti con le media query in seguito nel documento.

Ad esempio, impostare i link di navigazione come semplici collegamenti di testo per la versione desktop di un progetto (dove è più probabile che gli utenti utilizzino un mouse) e sovrascrivere quegli stili con una media query per offrire un'area più ampia (adatta ai dispositivi touchscreen) per viewport più limitati. Sebbene i browser moderni siano abbastanza intelligenti da ignorare i file di media query non destinati a loro,

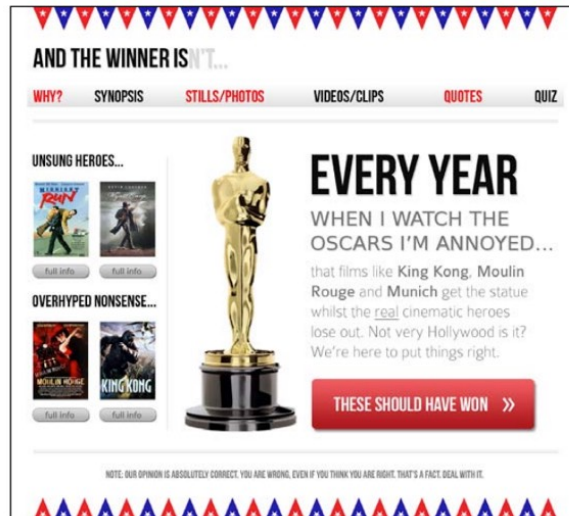
non sempre questo impedisce loro di scaricare effettivamente i file. C'è quindi poco vantaggio (a parte preferenze personali e/o la modulazione del codice) nel separare stili di media query diversi in file separati. L'uso di file separati aumenta il numero di richieste HTTP necessarie per eseguire il rendering di una pagina, il che a sua volta rende la pagina più lenta da caricare. Consiglierei quindi di aggiungere stili di media query all'interno di un foglio di stile esistente. Ad esempio, nel foglio di stile esistente, aggiungi semplicemente la media query utilizzando la seguente sintassi:

```
@media screen and (max-width: 768px) { le tue regole di stile }
```

Il nostro primo design reattivo

Non so voi, ma io non vedo l'ora di iniziare con un design Web reattivo! Ora che comprendiamo i principi delle media query, proviamoli e vediamo come funzionano in pratica. E ho anche il progetto su cui possiamo testarli, concedimi una breve digressione... Mi piacciono i film. Tuttavia, mi ritrovo comunemente in disaccordo con gli amici, in particolare su quali sono e quali non sono bei film. Quando vengono annunciati i candidati all'Oscar, ho spesso la sensazione che altri film avrebbero dovuto ricevere dei riconoscimenti. Vorrei lanciare un piccolo sito in inglese chiamato "E il vincitore non è...", proprio per questo motivo. Mostrerà i film che avrebbero dovuto vincere, criticherà quelli che hanno vinto (e non avrebbero dovuto) e includerà videoclip, citazioni, immagini e quiz per illustrare che ho ragione.

Proprio come i grafici che ho precedentemente rimproverato per non aver preso in considerazione viewport diversi, ho iniziato un mockup grafico basato su una griglia fissa di 960 pixel di larghezza. In realtà, anche se in teoria sarebbe sempre meglio iniziare un progetto pensando all'esperienza mobile/schermo piccolo e costruendo da lì, ci vorranno alcuni anni prima che tutti capiscano i vantaggi di quel modo di pensare. Fino ad allora, è probabile che dovrai prendere i progetti desktop esistenti e "adattarli" per farli funzionare in modo reattivo. Poiché questo è lo scenario in cui probabilmente ci troveremo nel prossimo futuro, inizieremo il nostro processo con un nostro progetto a larghezza fissa. Lo screenshot seguente mostra l'aspetto del mockup a larghezza fissa incompiuto, ha una struttura molto semplice e comune: intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina.



Si spera che questo sia tipico del tipo di struttura che ti viene chiesto di costruire settimana dopo settimana. Nel Capitolo 4, ti spiegherò perché dovresti usare HTML5 per il tuo markup. Tuttavia, per ora tralascerò questa parte, poiché siamo così ansiosi di testare le nostre capacità per le media query. Quindi, il nostro primo tentativo per l'utilizzo delle media query utilizza il buon vecchio markup HTML 4. Senza il contenuto effettivo, la struttura di base nel markup HTML 4 è simile al codice seguente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
<title>And the winner isn't</title>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
<!-- the header and navigation -->
<div id="header">
<div id="navigation">
<ul>
<li><a href="#">navigation1</a></li>
```

```
<li><a href="#">navigation2</a></li>
</ul>
</div>
</div>
<!-- the sidebar -->
<div id="sidebar">
<p>here is the sidebar</p>
</div>
<!-- the content -->
<div id="content">
<p>here is the content</p>
</div>
<!-- the footer -->
<div id="footer">
<p>Here is the footer</p>
</div>
</div>
</body>
</html>
```

Osservando il file di progettazione in Photoshop, possiamo vedere che l'intestazione e il piè di pagina sono larghi 940 pixel (con un margine di 10 pixel su entrambi i lati) e la barra laterale e il contenuto occupano rispettivamente 220 e 700 pixel, con un margine di 10 pixel su entrambi i lati di ognuno.



Prima di tutto, impostiamo i nostri blocchi strutturali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina) nel CSS. Dopo aver inserito gli stili di "reset", il nostro CSS per la pagina si presenta come segue:

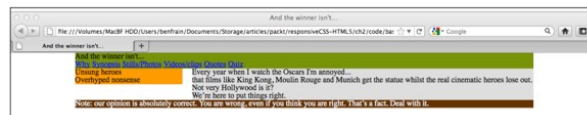
```
#wrapper {
margin-right: auto;
margin-left: auto;
width: 960px;
}
#header {
margin-right: 10px;
margin-left: 10px;
width: 940px;
background-color: #779307;
}
#navigation ul li {
display: inline-block;
}
#sidebar {
margin-right: 10px;
margin-left: 10px;
float: left;
background-color: #fe9c00;
```

```

width: 220px;
}
#content {
margin-right: 10px;
float: right;
margin-left: 10px;
width: 700px;
background-color: #dedede;
}
#footer {
margin-right: 10px;
margin-left: 10px;
clear: both;
background-color: #663300;
width: 940px;
}

```

Per illustrare come funziona la struttura, oltre ad aggiungere il contenuto aggiuntivo (senza immagini) ho anche aggiunto un colore di sfondo a ciascuna sezione strutturale. In un browser con una finestra più grande di 960 pixel, lo screenshot seguente mostra come appare la struttura di base:

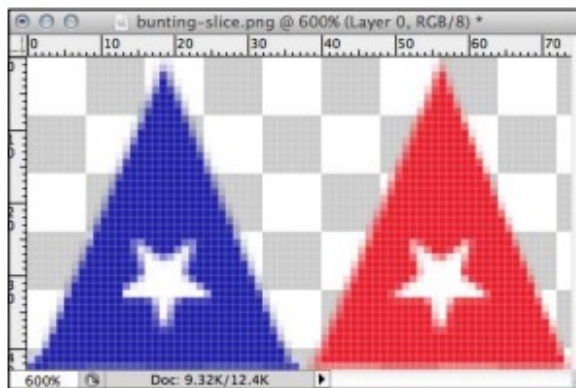


Ci sono molti altri modi in cui ottenere con CSS lo stesso tipo di struttura di contenuto sinistra/destra; senza dubbio avrai le tue preferenze. Ciò che è universalmente vero per tutti, è che quando il viewport diminuisce a meno di 960 pixel, le aree del contenuto a destra iniziano a essere “tagliate”.

Nel caso te lo fossi perso, gli stili di "reset" sono un mucchio di dichiarazioni CSS generiche che ripristinano i vari stili predefiniti con cui browser diversi renderizzano gli elementi HTML. Vengono aggiunti all'inizio del foglio di stile principale nel tentativo di reimpostare gli stili di ciascun browser su condizioni di parità in modo che gli stili aggiunti

successivamente nel foglio di stile abbiano lo stesso effetto su browser diversi. Non esiste un set "perfetto" di stili di ripristino e la maggior parte degli sviluppatori ha la propria preferenza a riguardo, ti invito a fare qualche ricerca per approfondire questo tema.

Per illustrare i problemi con la struttura del codice così com'è, sono andato avanti e ho aggiunto alcuni degli stili dal nostro file grafico nel CSS. Poiché alla fine si tratterà di un design reattivo, ho tagliato le immagini di sfondo nel modo migliore. Ad esempio, nella parte superiore e inferiore del disegno, invece di creare una lunga striscia come file grafico, ho tagliato due bandiere. Questa parte verrà quindi ripetuta orizzontalmente come immagine di sfondo attraverso il viewport per dare l'illusione di una lunga striscia (non importa quanto siano larghe). In termini reali, questo fa una differenza di 16 KB (l'intera striscia larga 960 pixel era un file .png da 20 KB mentre la sezione pesa solo 4 KB) su ciascuna striscia. Un utente mobile che visualizza il sito tramite apprezzerà questo risparmio di dati e il sito verrà caricato più velocemente! Lo screenshot seguente mostra l'aspetto della sezione (ingrandita al 600 per cento) prima dell'esportazione:

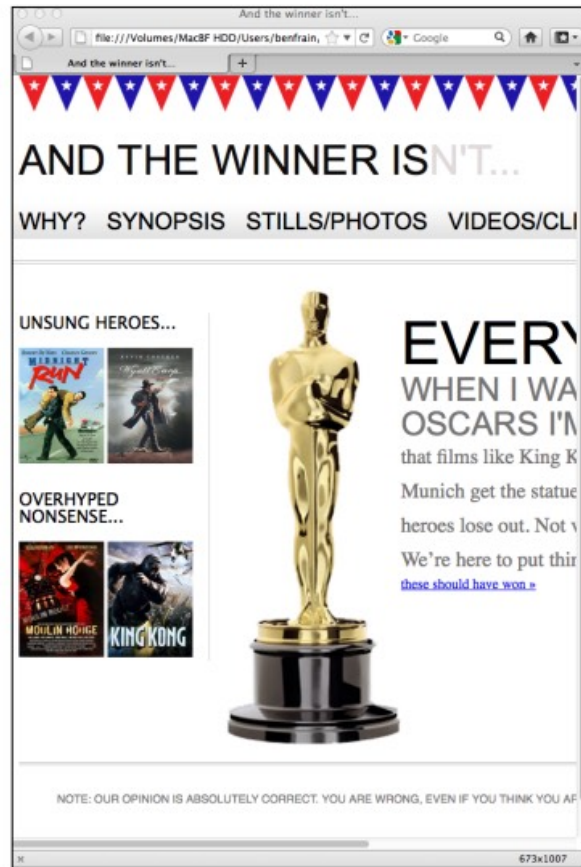


Ecco come appare il sito “E il vincitore non è...” in una finestra del browser:

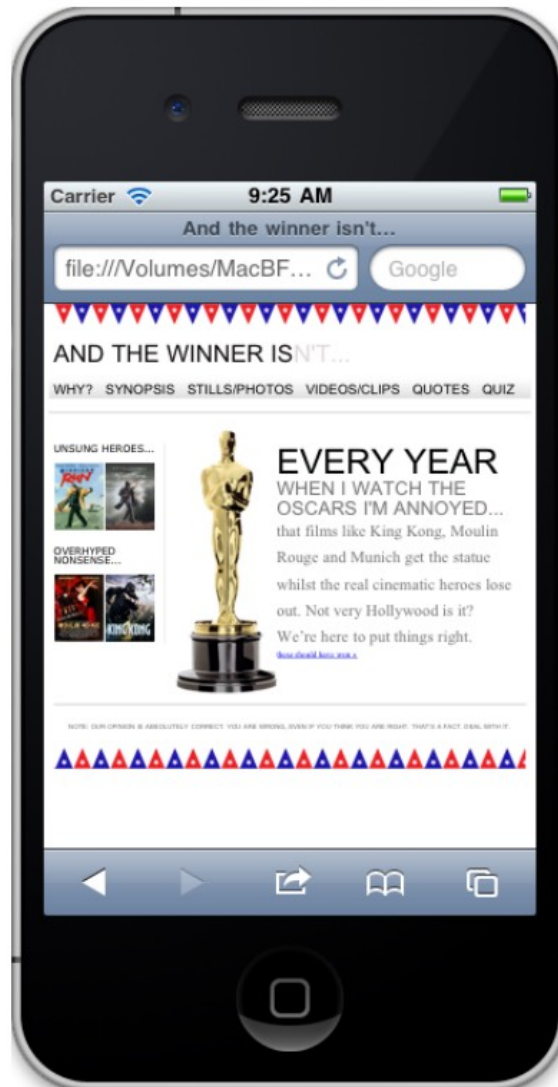


Per quanto riguarda lo stile, c'è ancora molto lavoro da fare. Ad esempio, il menu di navigazione non alterna rosso e nero, il pulsante principale AVREBBE DOVUTO VINCERE nell'area dei contenuti e mancano i pulsanti delle informazioni complete dalla barra laterale, oltretutto, i caratteri sono tutti molto lontani da quelli mostrati nel file grafico. Tuttavia, tutti questi aspetti sono risolvibili con HTML5 e CSS3. L'uso di HTML5 e CSS3 per risolvere questi problemi, piuttosto che inserire semplicemente file di immagine (come potremmo aver fatto in precedenza), renderà il sito Web reattivo, in sintonia con il nostro obiettivo. Ricorda che vogliamo che il nostro codice e il sovraccarico dei dati siano al minimo, per avere codice il più snello possibile per offrire anche agli utenti con velocità di larghezza di banda limitate un'esperienza piacevole.

Per ora, mettiamo da parte i problemi estetici e restiamo concentrati sul fatto che quando il viewport è ridotto al di sotto di 960 pixel, la nostra home page viene tagliata dal bordo del dispositivo:



L'abbiamo ridotta a 673 pixel di larghezza; immagina quanto sembrerà brutto su qualcosa come un iPhone con lo schermo piccolo? Basta dare un'occhiata al seguente screenshot:



Naturalmente, il browser Safari disegna automaticamente le pagine su una “tela” larga 980 pixel e quindi stringe quella tela per adattarla all'area della vista. Dobbiamo ancora ingrandire per vedere le aree ma non ci sono contenuti ritagliati. Come possiamo impedire a Safari e ad altri browser mobili di farlo?

Impedire ai moderni browser di ridimensionare la pagina

Sia i browser iOS che Android sono basati su WebKit (<https://www.webkit.org/>). Questi browser, e un numero crescente di tanti altri (Opera Mobile, ad esempio), consentono l'uso di un elemento meta viewport specifico per risolvere il problema. Il tag <meta> viene semplicemente aggiunto all'interno dei tag <head> dell'HTML. Può essere impostato su una larghezza specifica (che potremmo specificare in pixel, ad esempio) o in scala, ad esempio 2.0 (il doppio della dimensione effettiva). Ecco un esempio del meta tag viewport impostato per mostrare il browser al doppio (200%) delle dimensioni effettive:

```
<meta name="viewport" content="initial-scale=2.0,width=device-width" />
```

Usiamo questo tag nel nostro HTML come fatto nel seguente snippet di codice:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"  
  />  
  <meta name="viewport" content="initial-scale=2.0,width=device-  
width"/>  
  <title>And the winner isn't...</title>
```

Ora ricarichiamo quella pagina su Android e guarda come appare:



Come puoi vedere, questo non è esattamente ciò che stiamo cercando, ma illustra ciò che volevamo dimostrare, in grande stile! Sebbene non vi sia alcun sostituto per testare i siti su dispositivi reali, ci sono emulatori per Android e iOS. L'emulatore Android per Windows, Linux e Mac è disponibile gratuitamente scaricando e installando l'Android Software Development Kit (SDK) all'indirizzo <https://developer.android.com/sdk/>. È una configurazione da riga di comando; non adatta ai deboli di cuore. Il simulatore iOS è disponibile solo per gli utenti di Mac OS e fa parte del pacchetto Xcode (gratuito dal Mac App Store). Una volta installato Xcode, puoi accedere da `~/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications` iOS Simulator.app. Analizziamo il tag `<meta>` sopra e capiamo cosa sta succedendo. L'attributo `name="viewport"` è abbastanza ovvio. La sezione `content="initial-scale=2.0"` indica di ridimensionare il contenuto al doppio della dimensione (dove 0.5 sarebbe la metà della dimensione, 3.0 sarebbe tre volte la dimensione e così via) mentre la parte `width=device-width` indica al browser che la larghezza della pagina deve essere uguale alla larghezza del dispositivo. Il tag `<meta>` può essere utilizzato anche per

controllare la quantità di zoom per un utente ovvero quanto può ingrandire e rimpicciolire la pagina. Questo esempio consente agli utenti di effettuare uno zoom fino a tre volte la larghezza del dispositivo e fino alla metà della larghezza del dispositivo:

```
<meta name="viewport" content="width=device-width, maximum-scale=3,minimum-scale=0.5" />
```

Puoi anche disabilitare del tutto gli utenti dallo zoom ma, poiché lo zoom è un importante strumento di accessibilità, è sconsigliato farlo:

```
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
```

User-scalable=no è la parte rilevante. Bene, cambiamo la scala in 1.0, il che significa che il browser mobile visualizzerà la pagina al 100% del suo viewport. Impostare lo zoom alla larghezza del dispositivo significa che la nostra pagina dovrebbe essere visualizzata al 100% della larghezza di tutti i browser mobile supportati. Ecco il tag <meta> che useremo:

```
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>
```

Guardando la nostra pagina su un iPad in modalità verticale ora mostrerà il contenuto ritagliato ma in modo migliore rispetto a prima! È così che lo vogliamo a questo punto. Questo è già un progresso, fidati!

Il W3C sta tentando di portare maggiori funzionalità nei CSS, infatti, se visiti il sito del W3C noterai che al posto di scrivere un tag <meta> nella sezione <head> del tuo markup, potresti scrivere @viewport { width: 320px; } nel CSS. Ciò imposterebbe la larghezza del browser su 320 pixel. Alcuni browser supportano già questa sintassi (Opera Mobile, ad esempio), anche se utilizzano il proprio prefisso fornitore; ad esempio, @-o-viewport { width: 320px; }.

Correzione del design per diverse larghezze della finestra

Con il problema del viewport risolto, nessun browser ora ingrandisce la pagina; quindi, possiamo iniziare a correggere il design per diversi viewport. Nel CSS, aggiungeremo una media query per dispositivi come tablet (ad esempio, iPad) che hanno una larghezza del viewport di 768 pixel nella visualizzazione verticale (poiché la larghezza del viewport orizzontale è di 1024 pixel, rende la pagina adatta alla visualizzazione in orizzontale).

```
@media screen and (max-width: 768px) {  
  #wrapper {  
    width: 768px;  
  }  
  #header,#footer,#navigation {  
    width: 748px;  
  }  
}
```

La nostra media query ridimensiona la larghezza del wrapper, dell'intestazione, del piè di pagina e degli elementi di navigazione se la dimensione del viewport non supera i 768 pixel. Lo screenshot seguente mostra come appare sul nostro iPad:



In realtà sono abbastanza incoraggiato da questo risultato. Il contenuto ora si adatta al display dell'iPad (o qualsiasi altra finestra non più grande di 768 pixel) senza alcuna sezione ritagliata. Tuttavia, è necessario correggere l'area di navigazione perché i link si estendono dall'immagine di sfondo e l'area del contenuto principale fluttua sotto la barra laterale (è troppo ampia per adattarsi allo spazio disponibile). Modifichiamo la nostra media query nel CSS, come dimostrato nel seguente frammento di codice:

```
@media screen and (max-width: 768px) {
```

```
#wrapper {
```

```
width: 768px;
```

```
}
```

```
#header,#footer,#navigation {
```

```
width: 748px;
```

```
}
```

```
#content,#sidebar {
```

```
padding-right: 10px;
```

```
padding-left: 10px;
```

```
width: 728px;
```

```
}
```


}

Ora la barra laterale e l'area del contenuto stanno riempiendo l'intera pagina e sono ben distanziate con un piccolo riempimento su entrambi i lati. Tuttavia, questo non è molto convincente. Voglio prima il contenuto e poi la barra laterale (per sua natura è un'area di interesse secondaria). Ho commesso un altro errore da principiante, se sto tentando di avvicinarmi a questo progetto con una metodologia di progettazione veramente reattiva.

Con i design reattivi, i contenuti dovrebbero sempre essere al primo posto

Vogliamo mantenere tutte le caratteristiche del nostro design su più piattaforme e finestre (piuttosto che nascondere alcune parti con display: none o simili), ma è anche importante considerare l'ordine in cui appaiono le cose. Al momento, a causa dell'ordine della barra laterale e delle sezioni dei contenuti principali del nostro markup, la barra laterale dovrà sempre essere visualizzata prima del contenuto principale. È ovvio che un utente con una vista più limitata dovrebbe ottenere il contenuto principale prima della barra laterale, altrimenti vedrà il contenuto correlato prima del contenuto principale stesso. Potremmo (e forse dovremmo) spostare i nostri contenuti anche sopra la nostra barra di navigazione. In modo che coloro con i dispositivi più piccoli ottengano il contenuto prima di ogni altra cosa. Questa sarebbe certamente la logica continuazione dell'adesione alla massima: "Prima il contenuto". Tuttavia, nella maggior parte dei casi, vorremmo un po' di navigazione in cima a ogni pagina; quindi, sono più felice nello scambiare semplicemente l'ordine della barra laterale e dell'area del contenuto nel mio HTML: farò in modo che la sezione del contenuto venga prima della barra laterale. Si consideri ad esempio il seguente codice:

```
<div id="sidebar">  
<p>here is the sidebar</p>  
</div>  
<div id="content">  
<p>here is the content</p>  
</div>
```

Invece del codice precedente, abbiamo il codice come segue:

```
<div id="content">  
<p>here is the content</p>  
</div>  
<div id="sidebar">  
<p>here is the sidebar</p>  
</div>
```

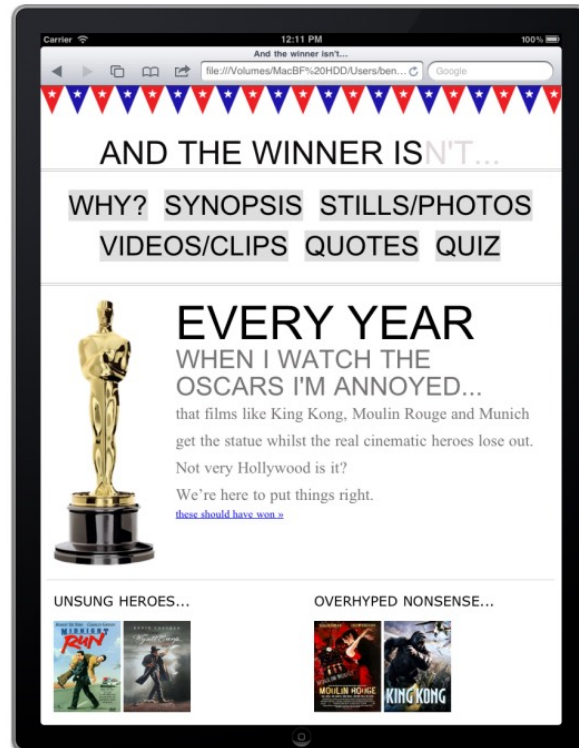
Sebbene abbiamo modificato il markup, la pagina ha ancora esattamente lo stesso aspetto nelle finestre più grandi a causa delle proprietà float:left e float:right sulla barra laterale e nelle aree di contenuto. Tuttavia, nell'iPad, i

nostri contenuti ora appaiono per primi, con i nostri contenuti secondari (la barra laterale) in seguito. Tuttavia, con il nostro markup strutturato nell'ordine corretto, ho anche iniziato ad aggiungere e modificare più stili, specifici per il viewport largo 768 pixel. Ecco come appare ora la media query:

```
@media screen and (max-width: 768px) {  
  #wrapper,#header,#footer,#navigation {  
    width: 768px;  
    margin: 0px;  
  }  
  #logo {  
    text-align:center;  
  }  
  #navigation {  
    text-align: center;  
    background-image: none;  
    border-top-color: #bfbfbf;  
    border-top-style: double;  
    border-top-width: 4px;  
    padding-top: 20px;  
  }  
  #navigation ul li a {  
    background-color: #dedede;  
    line-height: 60px;  
    font-size: 40px;  
  }  
  #content, #sidebar {  
    margin-top: 20px;  
    padding-right: 10px;  
    padding-left: 10px;  
    width: 728px;  
  }  
  .oscarMain {  
    margin-right: 30px;  
    margin-top: 0px;  
    width: 150px;  
    height: 394px;
```

```
}  
#sidebar {  
border-right: none;  
border-top: 2px solid #e8e8e8;  
padding-top: 20px;  
margin-bottom: 20px;  
}  
.sideBlock {  
width: 46%;  
float: left;  
}  
.overHyped {  
margin-top: 0px;  
margin-left: 50px;  
}  
}
```

Ricorda, gli stili aggiunti qui influenzeranno solo i dispositivi dello schermo con un riquadro di visualizzazione pari a 768 pixel o meno. I viewport più grandi li ignoreranno. Inoltre, poiché questi stili sono posti dopo qualsiasi stile esistente, li sovrascriveranno in modo pertinente. Il risultato è che le finestre più grandi otterranno il design che avevano prima. I dispositivi con un riquadro di visualizzazione largo 768 pixel, vedranno la schermata seguente:



Inutile dire che non vinceremo alcun premio di design qui, ma con poche righe di codice CSS all'interno di una media query, abbiamo creato un layout completamente diverso per un viewport diverso. Cosa abbiamo fatto? Innanzitutto, reimpostiamo tutte le aree di contenuto sull'intera larghezza della media query, come illustrato nel frammento di codice seguente:

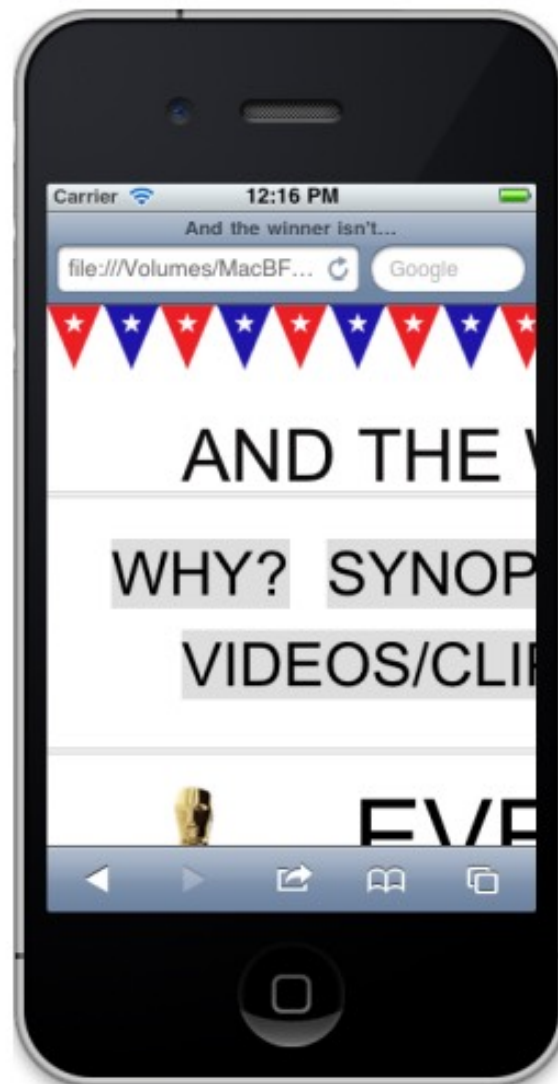
```
#wrapper,#header,#footer,#navigation {
width: 768px;
margin: 0px;
}
```

Si trattava semplicemente di aggiungere stili per alterare la disposizione estetica degli elementi. Ad esempio, il frammento di codice seguente modifica le dimensioni, il layout e lo sfondo della barra di navigazione, in modo che sia più facile per gli utenti con tablet (o qualsiasi utente con una finestra di 768 pixel o meno) selezionare un elemento di navigazione:

```
#navigation {
text-align: center;
background-image: none;
border-top-color: #bfbfbf;
```

```
border-top-style: double;  
border-top-width: 4px;  
padding-top: 20px;  
}  
#navigation ul li a {  
background-color: #dedede;  
line-height: 60px;  
font-size: 40px;  
}
```

Ora abbiamo esattamente lo stesso contenuto visualizzato con un layout diverso a seconda delle dimensioni del riquadro di visualizzazione. Le media query sono interessanti, no? Diamo un'occhiata al mio iPhone per vedere come appare... Puoi dargli un'occhiata nel seguente screenshot:



Media queries - parte della soluzione

Chiaramente il nostro lavoro è tutt'altro che finito; sembra orribile sul nostro iPhone. La nostra media query sta facendo esattamente quello che dovrebbe, applicando stili che dipendono dalle caratteristiche del nostro dispositivo. Il problema è tuttavia che la media query copre uno spettro molto ristretto di viewport. Qualsiasi cosa con una vista inferiore a 768 pixel verrà ritagliata così come tra 768 e 960 pixel poiché otterrà la versione non media query degli stili CSS che, come già sappiamo, non si adatta quando abbiamo una larghezza inferiore a 960 pixel. L'utilizzo delle sole media query per modificare un design va bene se disponiamo di un dispositivo di destinazione noto specifico; abbiamo già visto quanto sia facile adattare un dispositivo all'iPad. Ma questa strategia ha gravi carenze; vale a dire, non è davvero a prova di futuro. Al momento, quando ridimensioniamo il nostro viewport, il design scatta nei punti in cui intervengono le media query e la forma del nostro layout cambia. Tuttavia, rimane statico fino a quando non viene raggiunto il "punto di interruzione" della finestra successiva. Abbiamo bisogno di qualcosa di meglio di questo. Scrivere stili CSS specifici per ogni permutazione del viewport non tiene conto dei dispositivi futuri e un design è davvero eccezionale se è a prova di futuro. A questo punto la nostra soluzione è incompleta. Questo è più un design adattivo rispetto a quello veramente reattivo che vogliamo. Abbiamo bisogno che il nostro design si adatti prima di "rompersi". Per fare ciò, dobbiamo passare da un layout rigido e fisso a un layout fluido. In questo capitolo abbiamo imparato cosa sono le media query CSS3, come includerle nei nostri file CSS e come possono aiutare la nostra ricerca a creare un web design reattivo. Abbiamo anche imparato come fare in modo che i browser mobile moderni visualizzino le nostre pagine allo stesso modo delle loro controparti desktop e abbiamo toccato la necessità di considerare una politica "prima il contenuto" durante la strutturazione del nostro markup. Abbiamo anche appreso l'importanza di risparmiare dati quando utilizziamo le immagini nel nostro design nel modo più efficiente. Tuttavia, abbiamo anche appreso che le media query possono fornire solo un web design adattabile, non veramente reattivo. Le media query sono una componente essenziale in un design reattivo, ma è essenziale anche un layout fluido che consenta al nostro design di flettersi tra i punti di

interruzione gestiti dalle media query. La creazione di una base fluida per il nostro layout per facilitare la transizione tra i punti di interruzione delle nostre query multimediali è ciò che tratteremo nel prossimo capitolo.

USARE I LAYOUT FLUIDI

Quando ho iniziato a creare siti Web alla fine degli anni '90, le strutture di layout erano basate su tabelle. Il più delle volte, tutta la sezionatura sullo schermo era eseguita con percentuali. Ad esempio, alla colonna di navigazione a sinistra era relegato il 20% mentre all'area del contenuto principale il restante 80%. Non c'erano le grandi differenze nelle finestre del browser che vediamo oggi; quindi, questi layout funzionavano e si adattavano bene all'intervallo limitato di finestre. A nessuno importava molto che le frasi apparissero un po' diverse su uno schermo rispetto all'altro. Tuttavia, quando i progetti basati su CSS hanno preso il sopravvento, ha consentito ai progetti basati sul Web di imitare più da vicino la stampa. Con quella transizione, per molti (me compreso), i layout basati sulla proporzione sono diminuiti, a favore delle loro controparti rigide basate su pixel. Ora è tempo che i layout proporzionali riappaiano e in questo capitolo:

- Impareremo perché i layout proporzionali sono necessari per la progettazione reattiva
- Convertire le larghezze degli elementi basati su pixel in percentuali
- Convertire le dimensioni tipografiche basate sui pixel nel loro equivalente basato su em
- Comprendere come trovare il contesto per qualsiasi elemento
- Scoprire come ridimensionare le immagini in modo fluido
- Scoprire come fruire di immagini diverse su schermi di dimensioni diverse
- Scoprire come le media query possono funzionare con immagini e layout fluidi
- Creare un layout reattivo da zero utilizzando un sistema a griglia CSS

Come ho già detto, in genere, mi è sempre stato chiesto di codificare HTML e CSS che si adattano meglio a un composito di progettazione che misura quasi sempre una larghezza di 950-1000 pixel. Se il layout fosse

stato costruito con una larghezza proporzionale (diciamo, 90 per cento), le lamentele sarebbero arrivate rapidamente dai miei clienti: "Sembra diverso sul mio monitor!". Le pagine Web con dimensioni fisse basate su pixel erano il modo più semplice per abbinare le dimensioni fisse basate su pixel del composito. Anche in tempi più recenti, quando si utilizzano media query per produrre una versione ottimizzata di un layout, specifica per un determinato dispositivo popolare come un iPad o iPhone (come abbiamo fatto nel Capitolo 2), le dimensioni potrebbero essere ancora basate sui pixel dato che era noto il viewport. Tuttavia, mentre molti potrebbero monetizzare l'esigenza del cliente ogni volta che hanno bisogno di un sito ottimizzato, non è esattamente un modo a prova di futuro per costruire pagine web. Poiché vengono introdotti sempre più viewport, abbiamo bisogno di un modo a prova di futuro per qualcosa che non conosciamo ancora.

Perché i layout proporzionali sono essenziali per i design reattivi

Sappiamo che le media query sono incredibilmente potenti, ma siamo consapevoli di alcune limitazioni. Qualsiasi progetto a larghezza fissa, che utilizza solo media query per adattarsi a viste diverse, semplicemente "scatterà" da una serie di regole di media query CSS a quella successiva senza alcuna progressione lineare tra le due. Dalla nostra esperienza nel Capitolo 2, dove un viewport rientrava tra gli intervalli di larghezza fissa delle nostre media query (come potrebbe essere il caso per i futuri dispositivi sconosciuti e i loro viewport) il design richiedeva lo scorrimento orizzontale nel browser. Noi, invece, vogliamo creare un design che si fletta e abbia un bell'aspetto su tutte le finestre, non solo su quelle specificate in una media query. Andiamo al sodo. Dobbiamo cambiare il nostro layout fisso, basato su pixel, in uno fluido proporzionale. Ciò consentirà agli elementi di ridimensionarsi rispetto al viewport finché una media query non ne modifica lo stile. Ho già citato l'articolo di Ethan Marcotte su Responsive Web Design su A List Apart, Sebbene gli strumenti da lui utilizzati (impaginazione fluida, immagini e media query) non fossero nuovi, l'applicazione e l'incarnazione delle idee in un'unica metodologia coerente lo erano. Per molti che lavorano nel web design, il suo articolo è stato la genesi di nuove possibilità. In effetti, ha definito nuovi modi per creare pagine web che offrissero il meglio di entrambi i mondi; un modo per avere un design fluido e flessibile basato su un layout proporzionale. Metterli insieme costituisce il fulcro di un design responsive, creando qualcosa di veramente più grande della somma delle sue parti. In genere, nel prossimo futuro, qualsiasi design che ricevi o crei avrà dimensioni fisse. Attualmente misuriamo (in pixel) le dimensioni degli elementi, i margini e così via all'interno dei file grafici di Photoshop e altri strumenti grafici. Quindi inseriamo queste dimensioni direttamente nel nostro CSS e lo stesso vale per le dimensioni del testo. Facciamo clic su un elemento di testo nel nostro editor di immagini preferito, prendiamo nota della dimensione del carattere e quindi la inseriamo (di nuovo, spesso misurata in pixel) nella relativa regola CSS. Quindi come convertiamo le nostre dimensioni fisse in proporzionali?

Una formula da ricordare

È possibile che io mi stia spingendo oltre, essendo un fan di Ethan Marcotte, ma a questo punto è essenziale fare una precisazione. Nell'eccellente libro di Dan Cederholm, *Handcrafted CSS*, Marcotte ha contribuito con un capitolo sulle griglie fluide. In esso, ha fornito una formula semplice e coerente per convertire pixel a larghezza fissa in percentuali proporzionali: $\text{target} \div \text{contesto} = \text{risultato}$.

Ti sembra un po' un'equazione? Non temere, quando creerai un design reattivo, questa formula diventerà presto la tua nuova migliore amica. Piuttosto che parlare di altre teorie, mettiamo in pratica la formula convertendo l'attuale dimensione fissa per il sito “E il vincitore non è...” in un layout fluido basato sulla percentuale. Se ricordi, nel Capitolo 2, abbiamo stabilito che la struttura di markup di base del nostro sito era simile a questa:

```
<div id="wrapper">
  <!-- the header and navigation -->
  <div id="header">
    <div id="navigation">
      <ul>
        <li><a href="#">navigation1</a></li>
        <li><a href="#">navigation2</a></li>
      </ul>
    </div>
  </div>
  <!-- the sidebar -->
  <div id="sidebar">
    <p>here is the sidebar</p>
  </div>
  <!-- the content -->
  <div id="content">
    <p>here is the content</p>
  </div>
  <!-- the footer -->
  <div id="footer">
    <p>Here is the footer</p>
```

</div>

</div>

Il contenuto è stato aggiunto in seguito, ma ciò che è importante notare qui è il CSS che stiamo attualmente utilizzando per impostare le larghezze degli elementi strutturali principali (intestazione, barra di navigazione, barra laterale, contenuto e piè di pagina). Nota, ho omesso molte delle regole di stile in modo da poterci concentrare sulla struttura:

```
#wrapper {  
  margin-right: auto;  
  margin-left: auto;  
  width: 960px;  
}  
#header {  
  margin-right: 10px;  
  margin-left: 10px;  
  width: 940px;  
}  
#navigation {  
  padding-bottom: 25px;  
  margin-top: 26px;  
  margin-left: -10px;  
  padding-right: 10px;  
  padding-left: 10px;  
  width: 940px;  
}  
#navigation ul li {  
  display: inline-block;  
}  
#content {  
  margin-top: 58px;  
  margin-right: 10px;  
  float: right;  
  width: 698px;  
}  
#sidebar {  
  border-right-color: #e8e8e8;  
  border-right-style: solid;
```

```

border-right-width: 2px;
margin-top: 58px;
padding-right: 10px;
margin-right: 10px;
margin-left: 10px;
float: left;
width: 220px;
}
#footer {
float: left;
margin-top: 20px;
margin-right: 10px;
margin-left: 10px;
clear: both;
width: 940px;
}

```

Tutti i valori sono attualmente impostati utilizzando i pixel. Lavoriamo a partire dall'elemento più esterno e cambiamoli in percentuali proporzionali usando la formula $\text{target} \div \text{contesto} = \text{risultato}$. Tutti i nostri contenuti si trovano attualmente all'interno di un div con un ID #wrapper. Puoi vedere dal CSS sopra che è impostato con margine automatico e una larghezza di 960 px. Essendo il div più esterno, come definiamo la sua percentuale di larghezza del viewport?

Abbiamo bisogno di qualcosa da "contenere" e che diventi il contesto per tutti gli elementi proporzionali (contenuto, barra laterale, piè di pagina e così via) che intendiamo inglobare all'interno del nostro design. Dobbiamo quindi impostare un valore proporzionale per la larghezza che il #wrapper dovrebbe avere in relazione alla dimensione del viewport. Per ora, impostiamo 96 percento e vediamo cosa succede. Ecco la regola modificata per #wrapper:

```

#wrapper {
margin-right: auto;
margin-left: auto;
width: 96%; /* Holding outermost DIV */
}

```

Ed ecco come appare nella finestra del browser:



Fin qui tutto bene! Il 96 percento in realtà funziona abbastanza bene in questo caso, anche se avremmo potuto optare per il 100 o il 90 percento. Ora il passaggio da fisso a proporzionale diventa un po' più complicato man mano che ci spostiamo verso l'interno. Diamo prima un'occhiata alla sezione dell'intestazione. Considera di nuovo la formula, $\text{target} \div \text{contesto} = \text{risultato}$. Il nostro div #header (il target) si trova all'interno del div #wrapper (il contesto). Pertanto, prendiamo la larghezza del nostro #header (il target) di 940 pixel, lo dividiamo per la larghezza del contesto (il #wrapper), che era 960 px e il nostro risultato è 0,979166667. Possiamo trasformarlo in una percentuale spostando la posizione decimale di due cifre a destra e ora abbiamo una larghezza percentuale per l'intestazione di 97,9166667. Aggiungiamolo al nostro CSS:

```
#header {
margin-right: 10px;
margin-left: 10px;
width: 97.9166667%; /* 940 ÷ 960 */
}
```

E poiché anche i div #navigation e #footer hanno la stessa larghezza dichiarata, possiamo cambiare entrambi i valori dei pixel con la stessa regola basata sulla percentuale. Infine, prima di dare un'occhiata al browser, passiamo ai div #content e #sidebar. Poiché il contesto è sempre lo stesso (960 px), dobbiamo solo dividere la nostra dimensione target per quella

cifra. Il nostro #contenuto è attualmente di 698 px, quindi dividi quel valore per 960 e la nostra risposta è 0,727083333. Spostando la cifra decimale, avremo un risultato di 72,70833333 per cento, ovvero la larghezza del div #content in termini percentuali. La nostra barra laterale è attualmente di 220 px, ma c'è anche un bordo di 2 px da considerare. Non voglio che lo spessore del bordo destro si espanda o si contragga proporzionalmente in modo che rimanga a 2 px. Per questo motivo ho bisogno di sottrarre alcune dimensioni dalla larghezza della barra laterale. Quindi, nel caso di questa barra laterale, ho sottratto 2 px dalla larghezza della barra laterale e quindi ho eseguito lo stesso calcolo. Ho diviso il target (ora, 218 px) per il contesto (960 px) e la risposta è 0,227083333. Spostando il decimale, avremo un risultato di 22,70833333 per cento per la barra laterale. Dopo aver modificato tutte le larghezze dei pixel in percentuali, il seguente è l'aspetto del CSS pertinente:

```
#wrapper {  
  margin-right: auto;  
  margin-left: auto;  
  width: 96%; /* Holding outermost DIV */  
}  
#header {  
  margin-right: 10px;  
  margin-left: 10px;  
  width: 97.9166667%; /* 940 ÷ 960 */  
}  
#navigation {  
  padding-bottom: 25px;  
  margin-top: 26px;  
  margin-left: -10px;  
  padding-right: 10px;  
  padding-left: 10px;  
  width: 72.7083333%; /* 698 ÷ 960 */  
}  
#navigation ul li {  
  display: inline-block;  
}  
#content {  
  margin-top: 58px;
```

```
margin-right: 10px;
float: right;
width: 72.7083333%; /* 698 ÷ 960 */
}
#sidebar {
border-right-color: #e8e8e8;
border-right-style: solid;
border-right-width: 2px;
margin-top: 58px;
margin-right: 10px;
margin-left: 10px;
float: left;
width: 22.7083333%; /* 218 ÷ 960 */
}
#footer {
float: left;
margin-top: 20px;
margin-right: 10px;
margin-left: 10px;
clear: both;
width: 97.9166667%; /* 940 ÷ 960 */
}
```

Lo screenshot seguente mostra come appare in Firefox con il viewport di circa 1000px di larghezza:



Tutto bene finora. Ora, andiamo avanti e sostituiamo tutte le istanze di 10 px utilizzate per il riempimento e il margine in tutto con il loro equivalente proporzionale utilizzando la stessa formula $\text{target} \div \text{contesto} = \text{risultato}$. Poiché tutte le larghezze di 10 px hanno lo stesso contesto di 960 px, la larghezza in termini percentuali è 1,0416667 percento ($10 \div 960$).

Tutto sembra a posto con le stesse dimensioni del viewport. Tuttavia, l'area di navigazione non funziona. Se inserisco le dimensioni del viewport, i collegamenti iniziano a estendersi su due righe:



Inoltre, se espando il mio viewport, il margine tra i link non aumenta proporzionalmente. Diamo un'occhiata ai CSS associati alla navigazione e cerchiamo di capire perché:

```
#navigation {
padding-bottom: 25px;
margin-top: 26px;
margin-left: -1.0416667%; /* 10 ÷ 960 */
padding-right: 1.0416667%; /* 10 ÷ 960 */
padding-left: 1.0416667%; /* 10 ÷ 960 */
width: 97.9166667%; /* 940 ÷ 960 */
background-repeat: repeat-x;
background-image: url(../img/atwiNavBg.png);
border-bottom-color: #bfbfbf;
border-bottom-style: double; border-bottom-width: 4px;
}
#navigation ul li {
display: inline-block;
}
#navigation ul li a {
height: 42px;
line-height: 42px;
margin-right: 25px;
```

```
text-decoration: none;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
font-size: 27px;
color: black;
}
```

Bene, a prima vista, sembra che la nostra terza regola, la `#navigation ul li a`, abbia ancora un margine basato sui pixel di 25 px. Andiamo avanti e risolviamo il problema con la nostra affidabile formula. Poiché il div di `#navigazione` è basato su 940 px, il nostro risultato dovrebbe essere 2,6595745 percento. Quindi cambieremo quella regola in modo che sia la seguente:

```
#navigation ul li a {
height: 42px;
line-height: 42px;
margin-right: 2.6595745%; /* 25 ÷ 940 */
text-decoration: none;
text-transform: uppercase;
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
font-size: 27px;
color: black;
}
```

È stato abbastanza facile! Verifichiamo che tutto sia a posto nel browser...



Attenzione, non è esattamente quello che stavamo cercando. I collegamenti non si estendono su due righe ma non abbiamo il valore del margine proporzionale corretto.

Considerando di nuovo la nostra formula ($\text{target} \div \text{contesto} = \text{risultato}$), è possibile capire perché si verifica questo problema. Il nostro problema qui è il contesto, ecco il markup pertinente:

```
<div id="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>
```

Come puoi vedere, i nostri link `` si trovano all'interno dei tag ``. Sono il contesto per il nostro margine proporzionale. Osservando il CSS per i tag ``, possiamo vedere che non ci sono valori di larghezza impostati:

```
#navigation ul li { display: inline-block; }
```

Come spesso accade, si scopre che ci sono vari modi per risolvere questo problema. Potremmo aggiungere una larghezza esplicita ai tag ``

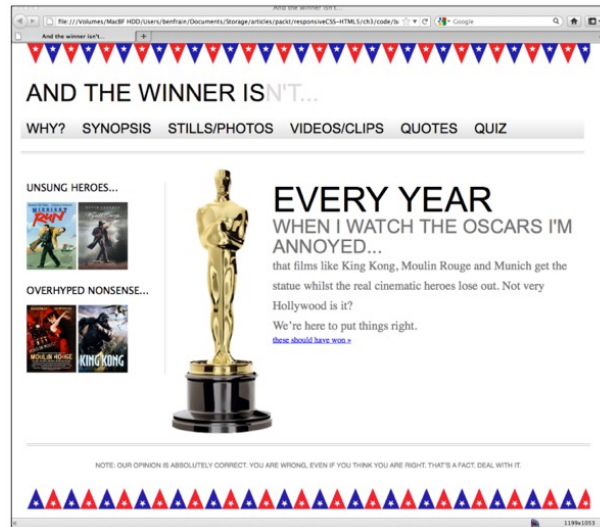
ma dovrebbe essere espressa in pixel a larghezza fissa o con una percentuale dell'elemento contenitore (il div di navigazione), nessuno dei due consente flessibilità per il testo che alla fine si trova al loro interno. Potremmo invece modificare il CSS per i tag , cambiando inline-block in modo che sia semplicemente inline:

```
#navigation ul li {  
  display: inline;  
}
```

Optando per la display:inline; (che impedisce agli elementi di comportarsi come elementi a livello di blocco), non avremo problemi nelle vecchie versioni di Internet Explorer. Tuttavia, sono un fan di inline-block in quanto offre un maggiore controllo sui margini e sul riempimento per i browser moderni, quindi lascerò invece i tag come inline-block (e forse aggiungerò uno stile di override per IE) e invece sposterò la mia regola del margine basata sulla percentuale dal tag <a> (che non ha un contesto esplicito) al blocco che lo contiene. Ecco come appaiono ora le regole modificate:

```
#navigation ul li {  
  display: inline-block;  
  margin-right: 2.6595745%; /* 25 ÷ 940 */  
}  
#navigation ul li a {  
  height: 42px;  
  line-height: 42px;  
  text-decoration: none;  
  text-transform: uppercase;  
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;  
  font-size: 27px;  
  color: black;  
}
```

E lo screenshot seguente mostra come appare nel browser con una finestra ampia 1200 px:



Ho ancora il problema dei collegamenti di navigazione che si estendono su due righe man mano che il viewport diventa più piccolo, fino a quando non arrivo al di sotto di 768 px di larghezza quando la media query che abbiamo scritto nel Capitolo 2, sovrascrive gli stili di navigazione correnti. Prima di iniziare a correggere la barra di navigazione, passerò tutte le dimensioni dei miei caratteri da pixel di dimensione fissa all'unità proporzionale, "ems". Una volta fatto ciò, guarderemo l'altro elefante nella stanza, facendo in modo che le nostre immagini si adattino al design.

Utilizzare ems anziché pixel

Negli anni passati, i web designer utilizzavano principalmente ems per ridimensionare i caratteri, piuttosto che i pixel, perché le versioni precedenti di Internet Explorer non erano in grado di ingrandire il testo impostato in pixel. Da tempo i browser moderni sono in grado di ingrandire il testo sullo schermo, anche se i valori di dimensione del testo sono stati dichiarati in pixel. Quindi, perché è necessario o preferibile utilizzare ems al posto dei pixel? Ecco due ovvi motivi: in primo luogo chiunque utilizzi ancora Internet Explorer ottiene automaticamente la possibilità di ingrandire il testo e in secondo luogo rende la vita per te, designer/sviluppatore, molto più semplice. La dimensione di un em è in relazione alla dimensione del suo contesto. Se impostiamo una dimensione del carattere del 100 percento sul nostro tag <body> e impostiamo con stile tutti gli ulteriori caratteri tipografici usando ems, saranno tutti influenzati da quella dichiarazione iniziale. Il risultato è che se, dopo aver completato tutta la configurazione necessaria, un cliente richiede che tutti i nostri caratteri siano un po' più grandi, possiamo semplicemente modificare la dimensione del carattere del body e di tutte le altre aree in proporzione. Usando la nostra stessa formula $\text{target} \div \text{contesto} = \text{risultato}$, convertirò ogni dimensione del carattere basata su pixel in ems. Vale la pena sapere che tutti i moderni browser desktop utilizzano 16 px come dimensione del carattere predefinita (se non diversamente specificato). Pertanto, fin dall'inizio, l'applicazione di una delle seguenti regole al tag body fornirà lo stesso risultato:

font-size: 100%;

font-size: 16px;

font-size: 1em;

Ad esempio, la prima dimensione del carattere basata sui pixel nel nostro foglio di stile controlla il titolo del sito **“E IL VINCITORE NON È...”** in alto a sinistra:

```
#logo {
```

```
display: block;
```

```
padding-top: 75px;
```

```
color: #0d0c0c;
```

```
text-transform: uppercase;
```

```
font-family: Arial, "Lucida Grande", Verdana, sans-serif;
```

font-size: 48px;

}

#logo span { color: #dfdada; }

Pertanto, $48 \div 16 = 3$. Quindi il nostro stile cambia nel seguente:

#logo {

display: block;

padding-top: 75px;

color: #0d0c0c;

text-transform: uppercase;

font-family: Arial, "Lucida Grande", Verdana, sans-serif;

font-size: 3em; /* $48 \div 16 = 3$ */

}

Puoi applicare questa stessa logica in tutto. Se in qualsiasi momento le cose vanno in tilt, è probabilmente il contesto per il tuo obiettivo che è cambiato. Ad esempio, considera `<h1>` all'interno del markup della nostra pagina:

`<h1>Every year when I watch the Oscars I'm annoyed...
</h1>`

Il nostro nuovo CSS basato su em si presenta così:

#content h1 {

font-family: Arial, Helvetica, Verdana, sans-serif;

text-transform: uppercase;

font-size: 4.3125em; } /* $69 \div 16$ */

#content h1 span {

display: block;

line-height: 1.052631579em; /* $40 \div 38$ */

color: #757474;

font-size: .550724638em; /* $38 \div 69$ */

}

Puoi vedere qui che la dimensione del carattere (che era 38 px) dell'elemento `` è in relazione all'elemento genitore (che era 69 px). Inoltre, line-height (che era 40 px) è impostata in relazione al carattere stesso (che era 38 px). Quindi la nostra struttura ora si sta ridimensionando e abbiamo cambiato il nostro tipo basato sui pixel in ems. Tuttavia, dobbiamo ancora capire come ridimensionare le immagini quando si ridimensiona il viewport, quindi diamo un'occhiata a questo problema ora, ma prima... Cosa diavolo è un em? Il termine em è semplicemente un modo

per esprimere la lettera "M" in forma scritta ed è pronunciato come tale. Storicamente, la lettera "M" veniva utilizzata per stabilire la dimensione di un determinato carattere poiché la lettera "M" era la più grande (la più ampia) delle lettere. Al giorno d'oggi, em come misura definisce la proporzione della larghezza e dell'altezza di una determinata lettera rispetto alla dimensione in punti di un determinato carattere.

Immagini fluide

La scalabilità delle immagini con un layout fluido può essere ottenuta in modo semplice nei browser moderni. È così semplice che basta dichiarare quanto segue nel CSS:

```
img { max-width: 100%; }
```

Questa dichiarazione fa sì che qualsiasi immagine venga ridimensionata automaticamente fino al 100 percento dell'elemento che la contiene. Inoltre, lo stesso attributo e proprietà possono essere applicati ad altri media. Per esempio:

```
img,object,video,embed {  
  max-width: 100%;  
}
```

E aumenteranno anche le dimensioni, a parte alcune eccezioni degne di nota come i video `<iframe>` di YouTube, ma li esamineremo nel Capitolo 4. Per ora, però, ci concentreremo sulle immagini poiché i principi sono gli stessi, indipendentemente dai media.

Ci sono alcune considerazioni importanti nell'utilizzo di questo approccio. In primo luogo, richiede una pianificazione anticipata: le immagini inserite devono essere sufficientemente grandi da poter essere ridimensionate a dimensioni maggiori del viewport. Questo porta a un'ulteriore considerazione, forse più importante. Indipendentemente dalle dimensioni del viewport o dal dispositivo che visualizza il sito, dovranno comunque scaricare le immagini di grandi dimensioni, anche se su alcuni dispositivi il viewport potrebbe dover visualizzare un'immagine solo il 25% delle sue dimensioni effettive. Questa è una considerazione importante sulla larghezza di banda in alcuni casi; quindi, rivisiteremo questo secondo problema a breve. Per ora, pensiamo solo al ridimensionamento delle nostre immagini.

Considera la nostra barra laterale con le locandine di alcuni film. Il markup è attualmente il seguente:

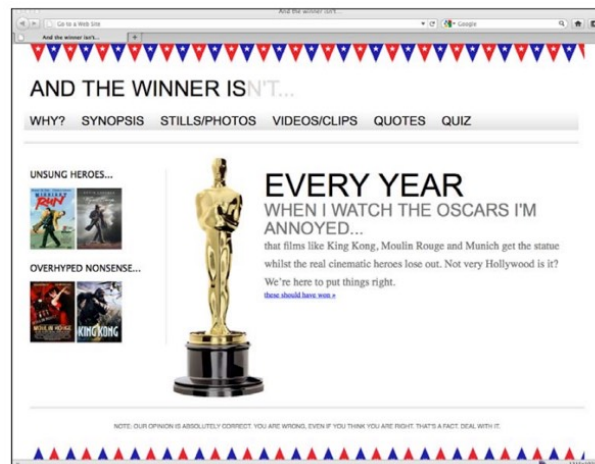
```
<!-- the sidebar -->  
<div id="sidebar">  
  <div class="sideBlock unsung">  
    <h4>Unsung heroes...</h4>
```

```

<a href="#"></a>
<a href="#"></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
</div>

```

Anche se ho aggiunto la dichiarazione max-width: 100% all'elemento img nel mio CSS, nulla è cambiato e le immagini non vengono ridimensionate quando espando il viewport:



Il motivo qui è che ho dichiarato esplicitamente sia la larghezza che l'altezza delle mie immagini nel markup:

```



```

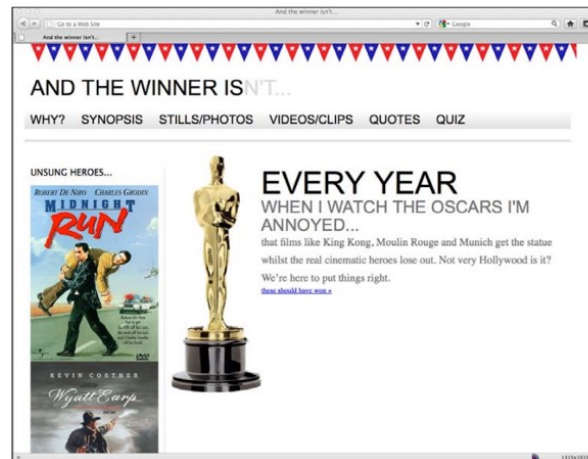
Un altro errore da principiante! Quindi modifico il markup associato alle immagini, rimuovendo gli attributi di altezza e larghezza:

```



```

Vediamo cosa accade, aggiornando la finestra del browser:



Beh, sicuramente funziona! Ma questo ha introdotto un ulteriore problema. Poiché le immagini vengono ridimensionate per riempire fino al 100 per cento la larghezza dell'elemento contenitore, ciascuna riempie la barra laterale. Come sempre, ci sono vari modi per risolvere questo problema...

Potrei aggiungere una classe aggiuntiva a ciascuna immagine come fatto nel seguente frammento di codice:

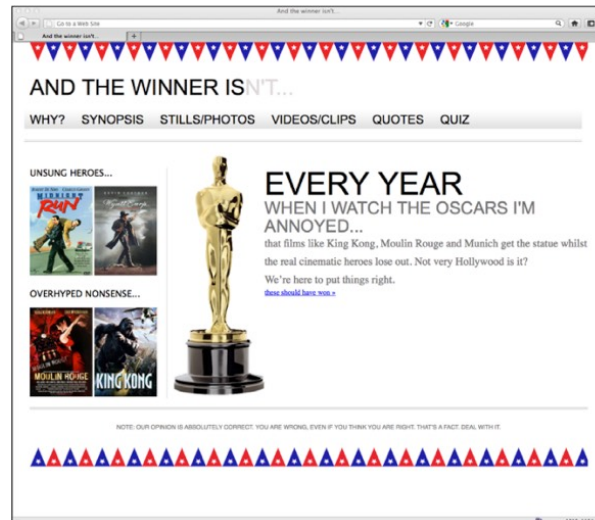
```

```

E quindi impostare una regola specifica per la larghezza. Tuttavia, lascerò il markup così com'è e userò la specificità CSS per annullare la regola della larghezza massima esistente con un'ulteriore regola più specifica per le mie immagini della barra laterale:

```
img {
  max-width: 100%;
}
.sideBlock img {
  max-width: 45%;
}
```

Lo screenshot seguente mostra come appaiono adesso gli elementi nel browser:



L'utilizzo della specificità CSS in questo modo ci consente di aggiungere ulteriore controllo alla larghezza di qualsiasi altra immagine o supporto. Inoltre, i nuovi potenti selettori di CSS3 ci consentono di indirizzare quasi tutti gli elementi senza la necessità di markup extra o l'introduzione di framework JavaScript come jQuery per fare il nostro lavoro sporco. Per le immagini della barra laterale ho deciso una larghezza del 45 percento semplicemente perché so che ho bisogno di aggiungere un piccolo margine tra le immagini in un secondo momento; quindi, avere due immagini per un totale del 90 percento della larghezza mi dà un po' di spazio (10 percento) extra. Ora che le immagini della barra laterale sono ben disposte, rimuovo anche gli attributi di larghezza e altezza sull'immagine della statua degli Oscar nel markup. Tuttavia, a meno che non imposti un valore di larghezza proporzionale per esso, non verrà ridimensionato; quindi, ho ottimizzato il CSS associato per impostare una larghezza proporzionale usando l'ormai collaudata formula $\text{target} \div \text{contesto} = \text{risultato}$.

```
.oscarMain {
float: left;
margin-top: -28px;
width: 28.9398281%; /* 698 ÷ 202 */
}
```

Quindi ora le immagini si ridimensionano bene man mano che il viewport si espande e si contrae. Tuttavia, se espandendo il viewport l'immagine viene ridimensionata oltre la sua dimensione nativa, le cose

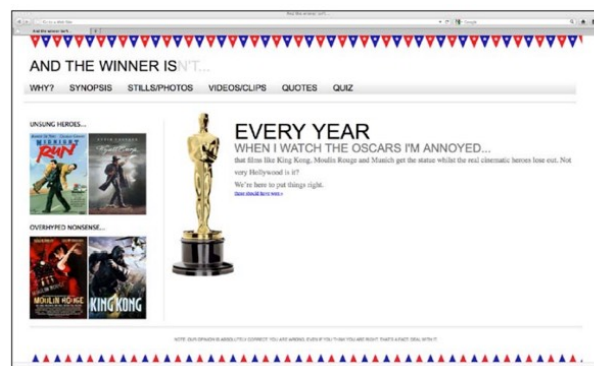
diventano sgradevoli e poco estetiche. Dai un'occhiata al seguente screenshot, con il viewport fino a 1900 px:



L'immagine oscar.png è in realtà larga 202 px. Tuttavia, con la finestra di oltre 1900 px di larghezza e il ridimensionamento dell'immagine, viene visualizzata con oltre 300 px di larghezza. Possiamo facilmente "mettere i freni" su questa immagine impostando un'altra regola più specifica:

```
.oscarMain {  
  float: left;  
  margin-top: -28px;  
  width: 28.9398281%; /* 698 ÷ 202 */  
  max-width: 202px;  
}
```

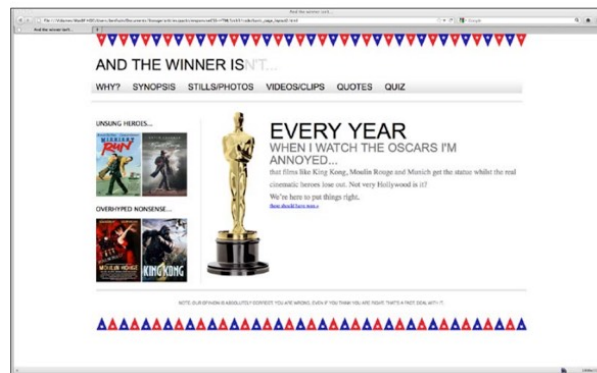
Ciò consentirebbe all'immagine di oscar.png di ridimensionarsi a causa della regola dell'immagine più generale, ma non andrebbe mai oltre la proprietà max-width più specifica impostata sopra. Ecco come appare la pagina con questo set di regole:



Un altro trucco per limitare gli oggetti che si espandono illimitatamente sarebbe impostare una proprietà di larghezza massima sull'intero div #wrapper in questo modo:

```
#wrapper {  
margin-right: auto;  
margin-left: auto;  
width: 96%; /* Holding outermost DIV */  
max-width: 1414px;  
}
```

Ciò significa che il design si ridimensionerà al 96 percento del viewport ma non si espanderà mai oltre i 1414 px di larghezza (ho optato per 1414 px poiché nella maggior parte dei browser moderni taglia le bandiere della bandierina alla fine di una bandiera anziché a metà di una). Lo screenshot seguente mostra come appare con un viewport di circa 1900 px:

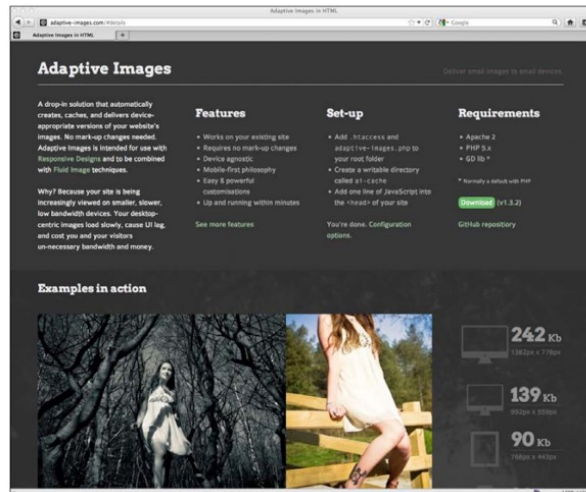


Ovviamente queste sono solo opzioni. Tuttavia, dimostra la versatilità di una griglia fluida e come possiamo controllare il flusso con poche dichiarazioni ma specifiche.

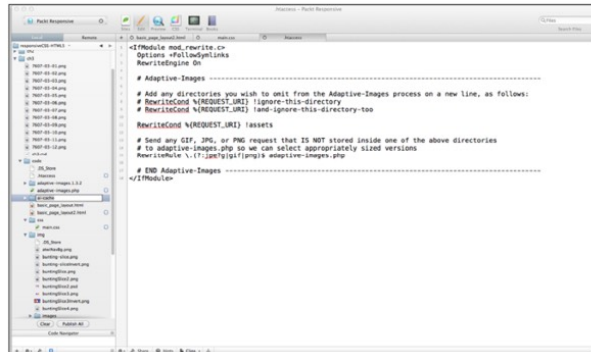
Immagini diverse per dimensioni dello schermo diverse

Adesso abbiamo le nostre immagini ben ridimensionate e ora capiamo come possiamo limitare la dimensione di visualizzazione di immagini specifiche. Tuttavia, in precedenza nel capitolo abbiamo notato il problema intrinseco con il ridimensionamento delle immagini. Devono essere fisicamente più grandi di quanto non siano visualizzate per essere visualizzate correttamente. Se non lo sono, iniziano a scombinare il design. Per questo motivo, le immagini, in termini di dimensioni del file, sono quasi sempre più grandi di quelle necessarie per la probabile dimensione di visualizzazione. Diverse persone hanno affrontato il problema, tentando di fornire immagini più piccole su schermi più piccoli. Il primo esempio degno di nota è stato "Responsive Images" del Filament Group. Tuttavia, di recente, sono passato a "Adaptive Images" di Matt Wilcox. La soluzione di Filament Group richiedeva la modifica del markup relativo all'immagine. La soluzione di Matt non ha questa necessità e crea automaticamente le immagini ridimensionate (più piccole) in base all'immagine a dimensione intera già specificata nel markup. Questa soluzione consente quindi di ridimensionare le immagini e di servirle all'utente in base alle esigenze e in base a un numero di punti di interruzione delle dimensioni dello schermo. Usiamo le immagini adattive!

La soluzione Adaptive Images richiede Apache 2, PHP 5.x e GD Lib. Quindi dovrai sviluppare su un server appropriato per vederne i vantaggi. Scarica il file .zip e iniziamo:



Estrai il contenuto del file ZIP e copia i file `adaptive-images.php` e `.htaccess` nella directory principale del tuo sito. Se stai già utilizzando un file `.htaccess` nella directory principale del tuo sito, non sovrascriverlo. Leggi le informazioni aggiuntive nel file `istruzioni.htm` incluso nel download. Ora crea una cartella nella root del tuo sito chiamata **ai-cache**.



Usa il tuo client FTP preferito per impostare i permessi di scrittura pari a 777 e copia il seguente JavaScript nel tag <head> di ogni pagina che necessita di immagini adattive:

```
<script>document.cookie='resolution='+Math.max(screen.width,screen.
height)+'; path=/';</script>
```

Nota che se non stai usando HTML5 (passeremo ad HTML5 nel prossimo capitolo), se vuoi che la pagina venga convalidata, dovrai aggiungere l'attributo `type`. Quindi lo script dovrebbe essere il seguente:

```
<script
type="text/javascript">document.cookie='resolution='+Math.max(screen.wi
dth,screen.height)+'; path=/'></script>
```

È importante che JavaScript sia in testa (e preferibilmente il primo pezzo di script) perché deve funzionare prima che la pagina abbia terminato il caricamento e prima che siano state richieste immagini. Qui viene aggiunto alla sezione <head> del nostro sito:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
```

```
<meta name="viewport" content="width=device-width,initial-
scale=1.0"/>
```

```
<title>And the winner isn't...</title>
```

```
<script
```

```
type="text/javascript">document.cookie='resolution='+Math.max(scre
en.width,screen.height)+'; path=/'></script>
```

```
<link href="css/main.css" rel="stylesheet" type="text/css" />
```

```
</head>
```

In passato, in genere ho posizionato tutte le mie immagini (sia quelle utilizzate per gli elementi CSS di sfondo che le immagini inline inserite nel markup) in un'unica cartella come immagini o img. Tuttavia, se si utilizzano le immagini adattive, è consigliabile che le immagini da utilizzare con CSS come immagini di sfondo (o qualsiasi altra immagine che non si desidera ridimensionare) siano collocate in una directory diversa. Adaptive Images per impostazione predefinita definisce una cartella denominata asset in cui conservare le immagini che non desideri ridimensionare. Pertanto, se vuoi che le immagini non vengano ridimensionate, tienile in questa cartella. Se desideri utilizzare una cartella diversa (o più di una) puoi modificare il file .htaccess come segue:

```
<IfModule mod_rewrite.c>
```

```
Options +FollowSymlinks
```

```
RewriteEngine On
```

```
# Adaptive-Images -----
```

```
REWRITECOND %{REQUEST_URI} !assets  
RewriteCond %{REQUEST_URI} !bkg
```

```
# SEND ANY GIF, JPG, or PNG request that IS NOT stored inside one of  
the above directories  
# to adaptive-images.php so we can select appropriately sized  
versions  
RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php  
# END Adaptive-Images -----  
</IfModule>
```

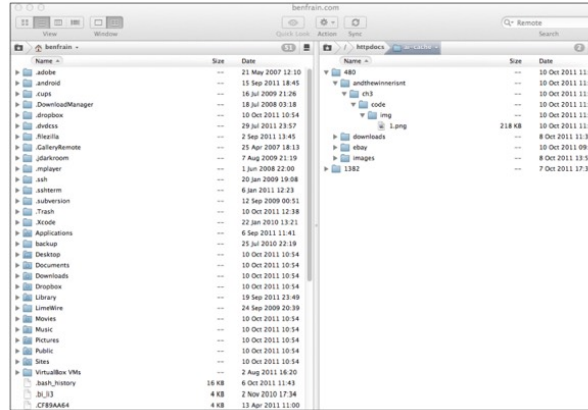
In questo esempio, abbiamo specificato che non vogliamo che le immagini all'interno di asset o bkg si adattino. Al contrario, se desideri affermare esplicitamente che desideri adattare solo le immagini all'interno di determinate cartelle, puoi omettere il punto esclamativo dalla regola. Ad esempio, se volessi solo immagini in una sottocartella del mio sito, chiamata andthewinnerisnt, modificherei il file .htaccess come segue:

```
<IfModule mod_rewrite.c>  
Options +FollowSymlinks  
RewriteEngine On  
# Adaptive-Images -----
```

```
REWRITECOND %{REQUEST_URI} andthewinnerisnt
```

```
# SEND ANY GIF, JPG, or PNG request that IS NOT stored inside one of  
the above directories  
# to adaptive-images.php so we can select appropriately sized  
versions  
RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php  
# END Adaptive-Images -----  
</IfModule>
```

Questo è tutto ciò che c'è da fare. Il modo più semplice per verificare che sia attivo e funzionante è inserire un'immagine di grandi dimensioni in una pagina, quindi visitare la pagina con uno smartphone. Se controlli il contenuto della tua cartella ai-cache con un programma FTP, dovresti vedere file e cartelle all'interno di cartelle di punti di interruzione con nome, ad esempio 480 (vedi lo screenshot seguente):



Le immagini adattive non sono limitate ai siti statici. Può anche essere utilizzato insieme ai sistemi di gestione dei contenuti (CMS) e ci sono anche soluzioni alternative per quando JavaScript non è disponibile. Con le immagini adattive, c'è un modo per offrire immagini completamente diverse in base alle dimensioni dello schermo, risparmiando larghezza di banda per i dispositivi che non vedrebbero il vantaggio delle immagini a dimensione intera. Se ricordi, all'inizio del capitolo, i nostri link di navigazione si estendevano ancora su più righe a determinate larghezze della finestra. Possiamo risolvere questo problema con le media query. Se i nostri collegamenti si “rompono” a 1060 px e “riprendono a funzionare” a 768 px (dove la nostra precedente media query prende il sopravvento), impostiamo alcuni stili di carattere aggiuntivi per gli intervalli intermedi:

```
@media screen and (min-width: 1001px) and (max-width: 1080px) {
#navigation ul li a { font-size: 1.4em; }
}
@media screen and (min-width: 805px) and (max-width: 1000px) {
#navigation ul li a { font-size: 1.25em; }
}
@media screen and (min-width: 769px) and (max-width: 804px) {
```

```
#navigation ul li a { font-size: 1.1em; }  
}
```

Come puoi vedere, stiamo cambiando la dimensione del carattere in base alla larghezza della finestra e il risultato è un insieme di link di navigazione che si trovano sempre su una riga, nell'intervallo da 769 px fino all'infinito. Questa è ancora una prova della simbiosi tra query multimediali e layout fluidi: le media query limitano le carenze di un layout fluido, un layout fluido facilita il passaggio da un insieme di stili definiti all'interno di una media query all'altro.

Sistemi a griglia CSS

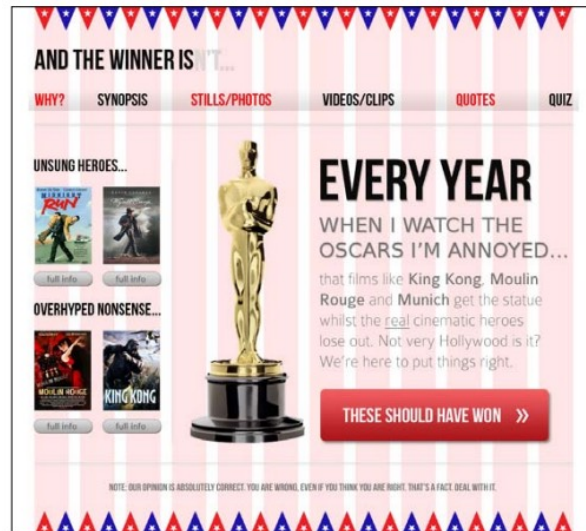
I CSS Grid sono un argomento potenzialmente divisivo. Alcuni designer li apprezzano in particolar modo, altri non li amano affatto. Nel tentativo di ridurre al minimo le mail di odio, dirò che mi pongo nel mezzo, poiché posso capire che alcuni sviluppatori pensano che siano superflui e in alcuni casi creano codice estraneo ma posso anche apprezzare il loro valore per la prototipazione rapida dei layout. Ecco alcuni framework CSS che offrono vari gradi di supporto "reattivo":

- Semantic (<http://semantic.gs>)
- Skeleton (<http://getskeleton.com>)
- Less Framework (<http://lessframework.com>)
- 1140 CSS Grid (<http://cssgrid.net>)
- Columnal (<http://www.columnal.com>)

Di questi, personalmente preferisco il sistema di griglia a colonne in quanto ha una griglia fluida incorporata accanto alle media query e utilizza anche classi CSS simili a 960.gs, il popolare sistema di griglia a larghezza fissa con cui la maggior parte degli sviluppatori e designer ha familiarità.

Molti sistemi di griglia CSS utilizzano classi CSS specifiche per eseguire le attività di layout quotidiane. Le classi row e container sono autoesplicative ma spesso ce ne sono molte di più. Pertanto, controlla sempre la documentazione di qualsiasi sistema di griglia per eventuali altre classi, semplificheranno di certo la vita professionale. Ad esempio, altre classi de facto tipiche utilizzate nei sistemi CSS Grid sono alfa e omega, rispettivamente per il primo e l'ultimo elemento di una riga (le classi alfa e omega rimuovono il riempimento o il margine) e .col_x dove x è il numero per l'importo di colonne su cui deve essere compreso l'elemento (ad esempio, col_6 per sei colonne). Supponiamo di non aver già costruito la nostra griglia fluida, né di aver scritto alcuna media query. Ci viene consegnata l'homepage originale di "E il vincitore non è..." sottoforma di PSD e ci viene detto di rendere operativa la struttura del layout di base in HTML e CSS il più rapidamente possibile. Vediamo se il sistema della griglia a colonne ci aiuta a raggiungere questo obiettivo.

Nel nostro PSD originale, era facile vedere che il layout era basato su 16 colonne. Il sistema di griglia a colonne, tuttavia, supporta un numero massimo di 12 colonne, quindi sovrapponiamo 12 colonne sul PSD anziché le 16 originali:



Dopo aver scaricato Columnal ed estratto il contenuto del file ZIP, duplicheremo la pagina esistente e quindi collegheremo columnal.css anziché main.css nella <head>. Per creare una struttura visiva usando Columnal, la chiave sta nell'aggiungere le classi div corrette nel markup. Ecco il markup completo della pagina fino a questo punto:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
<meta name="viewport" content="width=device-width,initial-
scale=1.0"/>
<title>And the winner isn't...</title>
<script
type="text/javascript">document.cookie='resolution='+Math.max(screen.wi
dth,screen.height)+'; path=/;</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
```

```

</head>
<body>
<div id="wrapper">
<!-- the header and navigation -->
<div id="header">
<div id="logo">And the winner is<span>n't...</span></div>
<div id="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>
</div>
<!-- the content -->
<div id="content">

<h1>Every year <span>when I watch the Oscars I'm annoyed...</span>
</h1>
<p>that films like King Kong, Moulin Rouge and Munich get the statue
whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
<p>We're here to put things right. </p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unsung">
<h4>Unsung heroes...</h4>
<a href="#">
</a>
<a href="#"></a>
</div>
<div class="sideBlock overHyped">

```

```

<h4>Overhyped nonsense...</h4>
<a href="#">
</a>
<a href="#"></a>
</div>
</div>
<!-- the footer -->
<div id="footer">
<p>Note: our opinion is absolutely correct. You are wrong, even if you
think you are right. That's a fact. Deal with it.</p>
</div>
</div>
</body>
</html>

```

Prima di tutto, dobbiamo specificare che il nostro div #wrapper è il contenitore per tutti gli elementi, quindi aggiungeremo la classe .container:

```
<div id="wrapper" class="container">
```

Scorrendo la pagina possiamo vedere che il nostro testo “E IL VINCITORE NON È” è la prima riga. Pertanto, aggiungeremo la classe .row a quell'elemento:

```
<div id="header" class="row">
```

Il nostro logo, anche se è solo testo, si trova all'interno di questa riga e si estende su tutte le 12 colonne. Pertanto aggiungeremo .col_12 ad esso:

```
<div id="logo" class="col_12">And the winner is<span>n't...</span>
</div>
```

Quindi la barra di navigazione è la riga successiva, aggiungeremo una classe .row a quel div:

```
<div id="navigation" class="row">
```

E il processo continua, aggiungendo le classi .row e .col_x se necessario. A questo punto faremo un salto in avanti, poiché temo che la ripetizione di questo processo possa farti addormentare. Pertanto, ecco l'intero markup modificato. Nota, era anche necessario spostare l'immagine dell'Oscar e darle la propria colonna. Inoltre ho aggiunto un div .row attorno al nostro #content e alla #sidebar.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

```

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
<meta name="viewport" content="width=device-width,initial-
scale=1.0"
/>
<title>And the winner isn't...</title>
<script type="text/javascript">document.cookie='resolution='+Math.
max(screen.width,screen.height)+'; path='/;</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
<link href="css/custom.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper" class="container">
<!-- the header and navigation -->
<div id="header" class="row">
<div id="logo" class="col_12">And the winner is<span>n't...</
span></div>
<div id="navigation" class="row">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>
</div>
<div class="row">
<!-- the content -->
<div id="content" class="col_9 alpha omega">

<div class="col_6 omega">
<h1>Every year <span>when I watch the Oscars I'm annoyed...</

```

```

span></h1>
<p>that films like King Kong, Moulin Rouge and Munich get the
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
<p>We're here to put things right. </p>
<a href="#">these should have won &raquo;</a>
</div>
</div>
<!-- the sidebar -->
<div id="sidebar" class="col_3">
<div class="sideBlock unsung">
<h4>Unsung heroes...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
</div>
</div>
<!-- the footer -->
<div id="footer" class="row">
<p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
</div>
</div>
</body>
</html>

```

Era anche necessario aggiungere alcuni stili CSS in un file chiamato custom.css. Il contenuto di questo file è il seguente:

```

#navigation ul li {
display: inline-block;

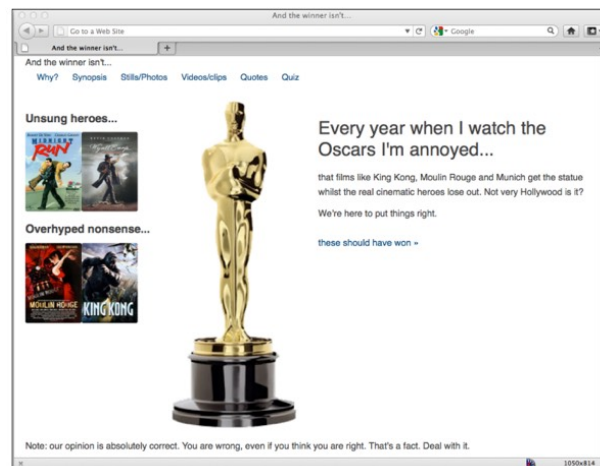
```

```

}
#content {
float: right;
}
#sidebar {
float: left;
}
.sideBlock {
width: 100%;
}
.sideBlock img {
max-width: 45%;
float:left;
}
.footer {
float: left;
}

```

Dopo aver apportato queste modifiche di base, con una rapida occhiata nella finestra del browser ci accorgiamo che la nostra struttura di base è a posto e si adatta alla finestra del browser:



Ovviamente c'è ancora molto lavoro da fare (lo so, è più di un leggero eufemismo), ma se hai bisogno di un modo rapido per creare una struttura reattiva di base, i sistemi CSS Grid come Columnal sono degni di considerazione. In questo capitolo abbiamo imparato come cambiare una

struttura rigida basata sui pixel in una flessibile basata sulla percentuale. Abbiamo anche imparato a usare ems, piuttosto che i pixel per una composizione più flessibile. Ora comprendiamo anche come possiamo fare in modo che le immagini siano responsive e si ridimensionino in modo fluido, oltre a implementare una soluzione basata su server per servire immagini completamente diverse in base alle dimensioni dello schermo del dispositivo.

Infine, abbiamo sperimentato un sistema CSS Grid reattivo che ci consente di prototipare rapidamente strutture reattive con il minimo sforzo. Tuttavia, fino a questo punto abbiamo perseguito la nostra ricerca reattiva utilizzando HTML 4.01 per il nostro markup. Nel Capitolo 1, abbiamo toccato alcune delle caratteristiche offerte da HTML5. Queste sono particolarmente importanti e rilevanti per i progetti reattivi in cui una mentalità "mobile first", che si presta ad un codice più snello, veloce e semantico. Nel prossimo capitolo, faremo i conti con HTML5 e modificheremo il nostro markup per sfruttare l'ultima e più ampia iterazione della specifica HTML.

HTML5 PER RESPONSIVE DESIGNS

HTML5 si è evoluto dal progetto Web Applications 1.0, avviato dal Web Hypertext Application Technology Working Group (WHATWG) prima di essere successivamente abbracciato dal W3C. Successivamente, gran parte delle specifiche sono state ponderate per gestire le applicazioni web. Se non stai creando applicazioni web, ciò non significa che non ci siano molte cose in HTML5 che potresti (e in effetti dovresti) usare per un design reattivo. Quindi, mentre alcune funzionalità di HTML5 sono direttamente rilevanti per la creazione di pagine Web più reattive (ad esempio, codice più snello), altre sono al di fuori del nostro ambito reattivo. HTML5 fornisce anche strumenti specifici per la gestione dei form e dell'input dell'utente. Tutto questo insieme di funzionalità elimina gran parte del carico di tecnologie più pesanti come JavaScript per fasi come la convalida dei form. In questo capitolo tratteremo quanto segue:

- Come scrivere pagine HTML5
- L'uso di HTML5
- Funzionalità HTML obsolete
- Nuovi elementi semantici HTML5
- Utilizzo di Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) per aumentare la semantica e aiutare le tecnologie assistive
- Incorporamento dei media
- Video HTML5 e iFrame reattivi
- Rendere un sito web disponibile offline

Quali parti di HTML5 possiamo utilizzare oggi?

Sebbene la specifica completa di HTML5 debba ancora essere rivista, la maggior parte delle nuove funzionalità di HTML5 sono già supportate, a vari livelli, dai moderni browser Web tra cui Safari di Apple, Google Chrome, Opera e Mozilla Firefox e persino Internet Explorer 9! Ci sono molte nuove funzionalità che possono essere implementate in questo momento e la maggior parte dei siti può essere scritta in HTML5. Attualmente, se ho il compito di creare un sito Web, il mio markup predefinito sarebbe HTML5 anziché HTML 4.01. Laddove solo pochi anni fa avveniva il contrario, al momento, deve esserci una ragione convincente per non eseguire il markup di un sito in HTML5. Tutti i browser moderni comprendono le funzionalità HTML5 comuni senza problemi (i nuovi elementi strutturali, i tag video e audio) e le versioni precedenti di IE possono sfruttare i **polyfill** per affrontare tutte le carenze che ho riscontrato. Cosa sono i polyfill? Il termine polyfill è stato originato da Remy Sharp come un'allusione al riempimento delle crepe nei vecchi browser con Polyfilla (noto come Spackling Paste negli Stati Uniti). Pertanto, un polyfill è uno shim JavaScript che replica efficacemente le funzionalità più recenti nei browser meno recenti. Tuttavia, è importante capire che i polyfill aggiungono codice extra al tuo codice. Pertanto, solo perché puoi aggiungere tre script polyfill per fare in modo che Internet Explorer renda il tuo sito uguale a qualsiasi altro browser non significa che dovresti necessariamente farlo! Normalmente, le versioni precedenti di Internet Explorer (precedente alla v9) non comprendono nessuno dei nuovi elementi semantici di HTML5. Tuttavia, qualche tempo fa, Sjoerd Visscher ha scoperto che se gli elementi vengono creati prima con JavaScript, Internet Explorer è in grado di riconoscerli e di adattarli di conseguenza. Forte di questa conoscenza, il mago JavaScript Remy Sharp ha creato uno script che, se incluso in una pagina HTML5, attivava magicamente questi elementi per le versioni precedenti di Internet Explorer. Per molto tempo, i pionieri di HTML5 hanno inserito questo script nel loro markup per consentire agli utenti che visualizzano in Internet Explorer 6, 7 e 8 di godere di un'esperienza comparabile. Tuttavia, le cose ora sono progredite in modo significativo. Ora c'è un nuovo strumento che fa tutto questo e molto altro ancora. Il suo nome è Modernizr (<https://www.modernizr.com>) e

se stai scrivendo pagine in HTML5, vale la pena prestare attenzione. Oltre ad abilitare elementi strutturali HTML5 per IE, offre anche la possibilità di caricare condizionalmente ulteriori polyfill, file CSS e file JavaScript aggiuntivi in base a una serie di test di funzionalità. Quindi, poiché ci sono alcune buone ragioni per non utilizzare HTML5, andiamo avanti e iniziamo a scrivere un po' di markup, in stile HTML5.

Vuoi una scorciatoia per un ottimo codice HTML5? Se il tempo è poco e hai bisogno di un buon punto di partenza per il tuo progetto, considera l'utilizzo di HTML5 Boilerplate (<https://html5boilerplate.com/>). È un file HTML5 di "best practice" predefinito, che include stili essenziali (come il suddetto normalize.css), polyfill e strumenti come Modernizr. Include anche uno strumento di compilazione che concatena automaticamente i file CSS e JS e rimuove i commenti per creare codice pronto per la produzione. Davvero ben fatto e fortemente raccomandato!

Come scrivere pagine HTML5

Apri una pagina Web esistente. C'è la possibilità che le prime righe assomiglino a queste:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"  
>
```

Elimina il frammento di codice precedente e sostituiscilo con il frammento di codice seguente:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset=utf-8>
```

Salva il documento e ora dovresti avere la tua prima pagina HTML5 per quanto riguarda il validatore W3C (<https://validator.w3.org/>). Non preoccuparti, non è finito il capitolo! Quell'esercizio grezzo ha semplicemente lo scopo di dimostrare la flessibilità di HTML5. È un'evoluzione del markup che scrivi già, non una rivoluzione. Possiamo usarlo per potenziare il markup che sappiamo già scrivere. Allora, cosa abbiamo effettivamente fatto? Prima di tutto, abbiamo usato la nuova dichiarazione HTML5 Doctype:

```
<!DOCTYPE html>
```

Se sei un fan del minuscolo, allora `<!doctype html>` è altrettanto valido. Non fa differenza. HTML5 Doctype: perché è così breve? Il Doctype `<!DOCTYPE html>` HTML5 è così breve perché è stato determinato come il metodo più breve per dire a un browser di visualizzare la pagina in "modalità standard". Questa mentalità sintattica più efficiente è prevalente in gran parte di HTML5. Dopo la dichiarazione Doctype, abbiamo aperto il tag HTML, specificato la lingua e quindi aperto la sezione `<head>`:

```
<html lang="en">  
<head>
```

Infine, abbiamo specificato la codifica dei caratteri. Poiché è un elemento void non richiede un tag di chiusura:

```
<meta charset=utf-8>
```

A meno che tu non abbia una buona ragione per specificare un encoding diverso, è quasi sempre UTF-8. Ricordo che, a scuola, ogni tanto il nostro insegnante di matematica super cattivo (ma in realtà molto bravo) doveva assentarsi. La classe tirava un sospiro di sollievo collettivo poiché, rispetto al signor "Rossi", il sostituto era solitamente un uomo accomodante e amabile che sedeva in silenzio, senza mai rimproverarci. Non ha insistito sul silenzio mentre lavoravamo, non gli importava molto di quanto fossero eleganti i nostri lavori sulla pagina – tutto ciò che contava erano le risposte. Se HTML5 fosse un insegnante di matematica, sarebbe quel supplente accomodante. Se presti attenzione a come scrivi il codice, in genere utilizzerai minuscolo per la maggior parte, racchiuderai i valori degli attributi tra virgolette e dichiarerai un "type" per script e fogli di stile. Ad esempio, potresti collegarti a un foglio di stile come questo:

```
<link href="CSS/main.css" rel="stylesheet" type="text/css" />
```

HTML5 non richiede tali dettagli, è altrettanto felice di vedere questo:

```
<link href=CSS/main.css rel=stylesheet >
```

Lo so, lo so, fa strano anche a me. Non ci sono tag di chiusura, non ci sono virgolette intorno ai valori degli attributi e non c'è una dichiarazione di tipo. Il secondo esempio vale quanto il primo. Questa sintassi più lassista si applica all'intero documento, non solo agli elementi CSS e JavaScript collegati alla pagina. Ad esempio, specifica un div come questo se ti piace:

```
<div id=wrapper>
```

Questo è HTML5 perfettamente valido. Lo stesso vale per l'inserimento di un'immagine:

```
<img SRC=frontCarousel.png aLt=frontCarousel>
```

Anche questo è HTML5 valido. Nessun tag di chiusura, nessuna virgoletta e un mix di lettere maiuscole e minuscole. Puoi anche omettere elementi come il tag di apertura <head> e la pagina viene comunque convalidata. Cosa direbbe XHTML 1.0 a riguardo! Sebbene miriamo ad abbracciare una mentalità mobile first per le nostre pagine Web e design reattivi, ammetto che non posso rinunciare completamente a scrivere quello che considero il markup delle best practice (nota, nel mio caso aderisce all'XHTML standard di markup 1.0 che richiedevano la sintassi XML). È vero che possiamo perdere alcune piccole quantità di dati dalle nostre

pagine abbracciando questo tipo di codifica, ma in tutta onestà, se necessario, risparmierei dati il più possibile sulle immagini! Per me, i caratteri extra (tag di chiusura e virgolette attorno ai valori degli attributi) sono fondamentali per una maggiore leggibilità del codice. Quando scrivo documenti HTML5, quindi, tendo a cadere da qualche parte tra il vecchio stile di scrittura del markup (che è ancora codice valido per quanto riguarda HTML5, sebbene possa generare avvisi nei validatori/controllori di conformità). Per esemplificare, per il collegamento CSS sopra, userei quanto segue:

```
<link href="CSS/main.css" rel="stylesheet"/>
```

Ho mantenuto il tag di chiusura e le virgolette ma ho ommesso l'attributo type. Il punto da sottolineare qui è che puoi trovare un livello adatto per te. HTML5 non ti sgriderà, segnalando il tuo markup e mettendoti in un angolo per non averlo convalidato. Un'altra caratteristica davvero utile in HTML5 è che ora possiamo racchiudere più elementi in un tag `<a>`. (Era ora, giusto?) In precedenza, se volevi che il tuo markup venisse convalidato, era necessario racchiudere ogni elemento nel proprio tag `<a>`. Ad esempio, vedi il frammento di codice seguente:

```
<h2><a href="index.html">The home page</a></h2>
```

```
<p><a href="index.html">This paragraph also links to the home page</a></p>
```

```
<a href="index.html"></a>
```

Tuttavia, possiamo abbandonare tutti i singoli tag `<a>` e avvolgere invece il gruppo come mostrato nel seguente frammento di codice:

```
<a href="index.html">
```

```
<h2>The home page</h2>
```

```
<p>This paragraph also links to the home page</p>
```

```

```

```
</a>
```

Le uniche limitazioni da tenere a mente sono che, comprensibilmente, non puoi racchiudere un tag `<a>` all'interno di un altro tag `<a>` e non puoi nemmeno racchiudere un form in un tag `<a>`.

Oltre a cose come gli attributi della lingua nei collegamenti agli script, ci sono altre parti dell'HTML a cui potresti essere abituato a utilizzare che ora sono considerate "obsolete" in HTML5. È importante essere consapevoli del fatto che ci sono due campi di funzionalità obsolete in

HTML5: conformi e non conformi. Le funzionalità di conformità continueranno a funzionare ma genereranno avvisi nei validatori. Realisticamente, evitale se possibile, ma potrai comunque usarle. Le funzionalità non conformi possono ancora essere visualizzate in alcuni browser, ma se le usi, sarai considerato una brutta persona! Un esempio di funzionalità obsoleta ma conforme sarebbe l'uso di un attributo border su un'immagine. Questo è stato storicamente utilizzato per impedire alle immagini di mostrare un bordo blu su di esse se erano nidificate all'interno di un collegamento. Ad esempio, vedi quanto segue:

```

```

Invece, si consiglia di utilizzare CSS per lo stesso effetto. Confesso che molte caratteristiche obsolete non le ho mai usate (alcune non le ho nemmeno mai viste!). È possibile che tu abbia una reazione simile. Tuttavia, se sei curioso, puoi trovare l'elenco completo delle funzionalità obsolete e non conformi online. Le caratteristiche obsolete e non conformi degne di nota sono strike, center, font, acronym, frame e frameset.

Nuovi elementi semantici in HTML5

Il mio dizionario definisce la semantica come "il ramo della linguistica e della logica che si occupa del significato". Per i nostri scopi, la semantica è il processo per dare significato al nostro markup. Perché questo è importante? Sono fiero che tu l'abbia chiesto. Considera la struttura del nostro attuale markup per il sito "E il vincitore non è..."

```
<body>
  <div id="wrapper">
    <div id="header">
      <div id="logo"></div>
      <div id="navigation">
        <ul>
          <li><a href="#">Why?</a></li>
        </ul>
      </div>
    </div>
    <!-- the content -->
    <div id="content">
```

```
</DIV>
  <!-- the sidebar -->
  <div id="sidebar">
```

```
</DIV>
  <!-- the footer -->
  <div id="footer">
```

```
</DIV>
</div>
</body>
```

La maggior parte degli autori di markup vedrà le convenzioni comuni per i nomi ID dei div utilizzati: intestazione, contenuto, barra laterale e così via. Tuttavia, per quanto riguarda il codice stesso, qualsiasi user-agent (browser web, screen reader, crawler dei motori di ricerca e così via) che lo guardi non potrebbe dire con certezza quale sia lo scopo di ciascuna sezione div. HTML5 mira a risolvere questo problema con nuovi elementi semantici. Dal punto di vista della struttura, questi sono spiegati nelle sezioni che seguono. L'elemento `<section>` viene utilizzato per definire una sezione generica di un documento o di un'applicazione. Ad esempio, puoi scegliere di creare sezioni attorno al tuo contenuto; una sezione per le informazioni di contatto, un'altra sezione per i feed di notizie e così via. È importante capire che non è inteso per scopi di stile, se hai bisogno di avvolgere un elemento semplicemente per modellarlo, dovresti continuare a usare un `<div>` come avresti fatto prima. L'elemento `<nav>` viene utilizzato per definire i principali blocchi di navigazione: collegamenti ad altre pagine o a parti all'interno della pagina. Poiché è indicato per l'uso nei principali blocchi di navigazione, non è strettamente inteso per l'uso nei piè di pagina (sebbene possa esserlo) e simili, dove i gruppi di collegamenti ad altre pagine sono comuni. L'elemento `<article>`, insieme a `<section>` può facilmente creare confusione. Ho sicuramente dovuto leggere e rileggere le specifiche di ciascuno prima di usarli. L'elemento `<article>` viene utilizzato per avvolgere un contenuto autonomo. Durante la strutturazione di una pagina, chiediti se il contenuto che intendi utilizzare all'interno di un tag `<article>` può essere copiato e incollato come un pezzo unico su un sito diverso e ha comunque un senso completo? Un altro modo è pensare che il contenuto racchiuso in `<article>` possa costituire effettivamente un articolo separato in un feed RSS? L'esempio ovvio di contenuto che dovrebbe essere racchiuso con un elemento `<article>` sarebbe un post di un blog. Tieni presente che se nidifichi elementi `<article>`, si presume che gli elementi nidificati `<article>` siano principalmente correlati all'articolo esterno. L'elemento `<aside>` viene utilizzato per il contenuto che è correlato al contenuto che lo circonda. In termini pratici, lo uso spesso per le barre laterali (quando contiene contenuti adatti). È anche considerato adatto per citazioni, pubblicità e gruppi di elementi di navigazione (come blog roll e così via).

Se hai una serie di intestazioni, tagline e sottotitoli in `<h1>`, `<h2>`, `<h3>` e i tag successivi, considera la possibilità di racchiuderli nel tag `<hgroup>`.

In questo modo si nasconderanno gli elementi secondari dall'algoritmo di struttura HTML5 poiché solo il primo elemento di intestazione all'interno di un `<hgroup>` contribuisce alla struttura dei documenti. HTML5 consente a ogni contenitore di avere il proprio schema autonomo. Ciò significa che non è più necessario pensare costantemente a quale livello di tag di intestazione ti trovi. Ad esempio, all'interno di un blog, posso impostare i titoli dei miei post in modo che utilizzino il tag `<h1>`, quando il titolo stesso del mio blog ha anche un tag `<h1>`. Si consideri ad esempio la seguente struttura:

```
<hgroup>
<h1>Ben's blog</h1>
<h2>All about what I do</h2>
</hgroup>
<article>
<header>
<hgroup>
<h1>A post about something</h1>
<h2>Trust me this is a great read</h2>
<h3>No, not really</h3>
<p>See. Told you.</p>
</hgroup>
</header>
</article>
```

Nonostante abbia più intestazioni `<h1>` e `<h2>`, lo schema appare ancora come segue:

- Ben's blog
- A post about something

Pertanto, non è necessario tenere traccia del tag di intestazione che è necessario utilizzare. Puoi semplicemente utilizzare qualsiasi livello di tag di intestazione che ti piace all'interno di ogni parte di contenuto sezionato e l'algoritmo di struttura HTML5 lo ordinerà di conseguenza. Puoi testare la struttura dei tuoi documenti utilizzando `outliner HTML5` in uno dei seguenti URL:

- <http://gsnedders.html5.org/outliner/>

- <http://hoyois.github.com/html5outliner/>

L'elemento `<header>` non partecipa all'algoritmo di struttura, quindi non può essere utilizzato per sezionare il contenuto. Invece dovrebbe essere usato come introduzione al contenuto. In pratica, l'`<header>` può essere utilizzato per l'area "masthead" dell'intestazione di un sito ma anche come introduzione ad altri contenuti come un'introduzione a un elemento `<article>`. Come l'`<header>`, l'elemento `<footer>` non prende parte all'algoritmo di struttura, quindi non seziona il contenuto. Invece dovrebbe essere usato per contenere informazioni sulla sezione in cui si trova. Potrebbe contenere collegamenti ad altri documenti o informazioni sul copyright, ad esempio e, come l'`<header>`, può essere utilizzato più volte all'interno di una pagina, se necessario. Ad esempio, potrebbe essere utilizzato per il footer di un blog ma anche per il footer all'interno di un post di blog `<article>`. Tuttavia, la specifica rileva che le informazioni di contatto per l'autore di un post del blog dovrebbero invece essere racchiuse da un elemento `<address>`. L'elemento `<address>` deve essere utilizzato esplicitamente per contrassegnare le informazioni di contatto per il suo predecessore `<article>` o `<body>` più vicino. Per confondere le cose, tieni presente che non deve essere utilizzato per indirizzi postali e simili a meno che non siano effettivamente gli indirizzi di contatto per il contenuto in questione. Invece gli indirizzi postali e altre informazioni di contatto arbitrarie dovrebbero essere racchiuse in vecchi tag `<p>`.

Utilizzo pratico degli elementi strutturali di HTML5

Diamo un'occhiata ad alcuni esempi pratici di questi nuovi elementi. Penso che gli elementi `<header>`, `<nav>` e `<footer>` siano abbastanza autoesplicativi, quindi per cominciare, prendiamo il markup corrente della homepage di “E il vincitore non è...” e modifichiamo le aree di intestazione, navigazione e piè di pagina (vedi le aree evidenziate nel seguente frammento di codice):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
<meta      name="viewport"      content="width=device-width,initial-
scale=1.0"
/>
<title>And the winner isn't...</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen.
height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" />
</head>
<body>
<div id="wrapper">
<!-- the header and navigation -->
<header>
<div id="logo">And the winner is<span>n't...</span></div>
<nav>
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
</header>
```

```

<!-- the content -->
<div id="content">

<h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
<p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>
<p>We're here to put things right. </p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<div id="sidebar">
<div class="sideBlock unsung">
<h4>Unsung heroes...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
</div>
<!-- the footer -->
<footer>
<p>Note: our opinion is absolutely correct. You are wrong, even if you think you are right. That's a fact. Deal with it.</p>
</footer>
</div>
</body>
</html>

```

Come abbiamo visto, tuttavia, laddove esistono articoli e sezioni all'interno di una pagina, questi elementi non sono limitati a un uso per

pagina. Ogni articolo o sezione può avere la propria intestazione, piè di pagina e navigazione. Ad esempio, se aggiungiamo un elemento <article> nel nostro markup, potrebbe apparire come segue:

```
<body>
<div id="wrapper">
  <!-- the header and navigation -->
  <header>
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav>
      <ul>
        <li><a href="#">Why?</a></li>
      </ul>
    </nav>
  </header>
  <!-- the content -->
  <div id="content">
    <article>
      <header>An article about HTML5</header>
      <nav>
        <a href="1.html">related link 1</a>
        <a href="2.html">related link 2</a>
      </nav>
      <p>here is the content of the article</p>
      <footer>This was an article by Ben Frain</footer>
    </article>
```

Come puoi vedere nel codice precedente, stiamo usando un <header>, <nav> e <footer> sia per la pagina che per l'articolo in essa contenuto. Modifichiamo la nostra area della barra laterale. Questo è ciò che abbiamo al momento nel markup HTML 4.01:

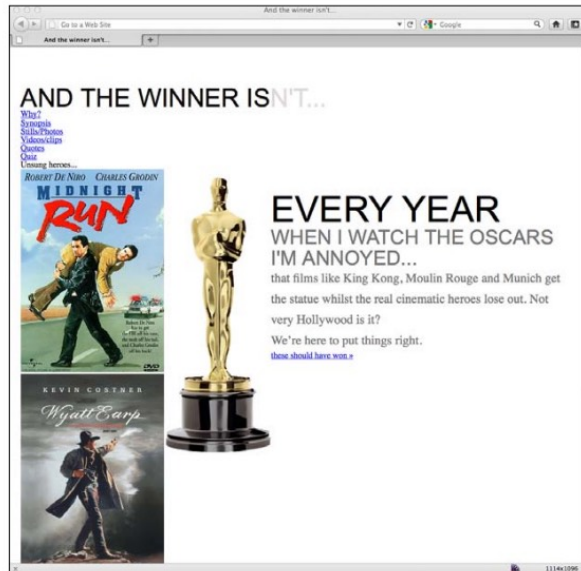
```
<!-- the sidebar -->
<div id="sidebar">
  <div class="sideBlock unsung">
    <h4>Unsung heroes...</h4>
    <a href="#">
  </a>
  <a href="#"></a>
```

```
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#">
</a>
<a href="#"></a>
</div>
</div>
```

Il nostro contenuto della barra laterale è sicuramente correlato al contenuto principale, quindi prima di tutto rimuoviamo <div id="sidebar"> e sostituiamolo con <aside>:

```
<!-- the sidebar -->
<aside>
<div class="sideBlock unsung">
<h4>Unsung heroes...</h4>
<a href="#">
</a>
<a href="#"></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#">
</a>
<a href="#"></a>
</div>
</aside>
```

Eccellente! Tuttavia, se diamo un'occhiata nel browser...



Ti sei accorto del problema, vero? Il motivo è che non abbiamo modificato il CSS per adattarlo ai nuovi elementi. Facciamolo ora prima di procedere, dobbiamo modificare tutti i riferimenti a #header in modo che siano semplicemente header, tutti i riferimenti a #navigation in modo che siano nav e tutti i riferimenti a #footer in modo che siano footer. Ad esempio, la prima regola CSS relativa all'intestazione cambierà da:

```
#header {
background-position: 0 top;
background-repeat: repeat-x;
background-image: url(../img/buntingSlice3Invert.png);
margin-right: 1.0416667%; /* 10 ÷ 960 */
margin-left: 1.0416667%; /* 10 ÷ 960 */
width: 97.9166667%; /* 940 ÷ 960 */
}
```

Per diventare:

```
header {
background-position: 0 top;
background-repeat: repeat-x;
background-image: url(../img/buntingSlice3Invert.png);
margin-right: 1.0416667%; /* 10 ÷ 960 */
margin-left: 1.0416667%; /* 10 ÷ 960 */
width: 97.9166667%; /* 940 ÷ 960 */
}
```

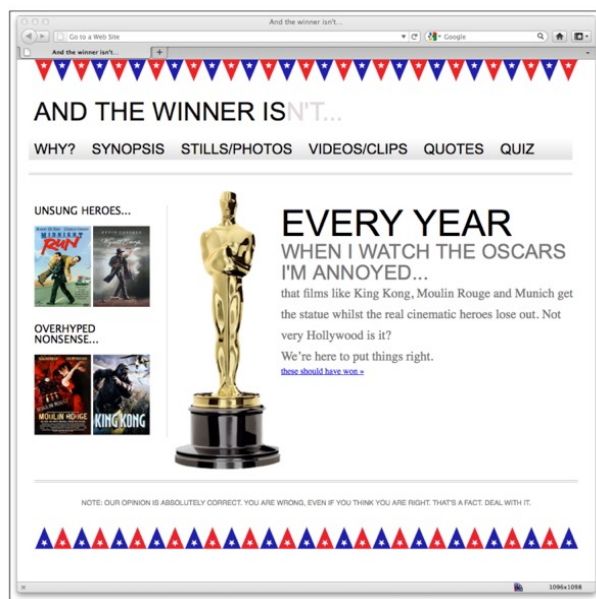
Ciò è stato particolarmente facile per l'intestazione, la navigazione e il piè di pagina poiché gli ID erano gli stessi dell'elemento per cui li stavamo cambiando: abbiamo semplicemente omesso il carattere iniziale "#". La barra laterale è leggermente diversa: dobbiamo invece cambiare i riferimenti da #sidebar a aside. Tuttavia, con "trova e sostituisci" nell'editor di codice di tua scelta, risolverai in qualche secondo questo problema. Per chiarire, una regola come la seguente:

```
#sidebar { }
```

Diventerà:

```
aside { }
```

Anche se hai scritto un enorme foglio di stile CSS, scambiare i riferimenti dagli ID HTML 4.01 agli elementi HTML5 è un compito abbastanza indolore. Tieni presente che con HTML5 possono esserci più elementi <header>, <footer> e <aside> all'interno di una pagina, quindi potrebbe essere necessario scrivere stili più specifici per singole istanze. Una volta che gli stili per "E il vincitore non è..." sono stati modificati di conseguenza, torniamo nel browser e vedremo:



Ora, anche se stiamo dicendo agli user agent quale sezione della pagina è aside, all'interno abbiamo due sezioni distinte, UNSUNG HEROES e OVERHYPED NONSENSE. Pertanto, nell'interesse di definire semanticamente tali aree, modifichiamo ulteriormente il nostro codice:

```
<!-- the sidebar -->
<aside>
<section>
<div class="sideBlock unsung">
<h4>Unsung heroes...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
</section>
<section>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
</section>
</aside>
```

La cosa importante da ricordare è che `<section>` non è inteso per scopi di stile, piuttosto per identificare un contenuto distinto e separato. Le sezioni normalmente dovrebbero avere anche intestazioni naturali, il che si adatta perfettamente alla nostra causa. Grazie all'algoritmo di struttura HTML5, possiamo anche modificare i nostri tag `<h4>` in tag `<h1>` producendo comunque una struttura accurata del nostro documento. E il contenuto principale del sito? Potrebbe sorprenderti che non ci sia un elemento distinto per contrassegnare il contenuto principale di una pagina. Tuttavia, la logica segue che poiché è possibile delimitare tutto il resto, ciò che rimane dovrebbe essere il contenuto principale della pagina.

Semantica a livello di testo HTML5

Oltre agli elementi strutturali che abbiamo esaminato, HTML5 rivede anche alcuni tag che venivano chiamati elementi inline. La specifica HTML5 ora fa riferimento a questi tag come semantica a livello di testo. Diamo un'occhiata ad alcuni esempi comuni.

Sebbene potremmo aver usato spesso l'elemento `` semplicemente come un gancio di stile, in realtà significava "rendi questo più evidente". Tuttavia, ora puoi usarlo ufficialmente semplicemente come gancio di stile nei CSS poiché la specifica HTML5 ora dichiara che `` è:

...un intervallo di testo su cui si attira l'attenzione per scopi utilitaristici senza trasmettere ulteriore importanza e senza implicazione di una voce o stato d'animo alternativo, come parole chiave nell'abstract di un documento, nomi di prodotti in una recensione, parole utilizzabili in testo interattivo.

OK, alzo la mano, ho usato spesso `` anche come gancio per lo styling. Ho bisogno di riparare i miei errori poiché in HTML5 è pensato per essere utilizzato per:

...sottolineare l'enfasi dei suoi contenuti.

Pertanto, a meno che tu non voglia effettivamente enfatizzare il contenuto racchiuso, considera l'utilizzo di un tag `` o, se pertinente, un tag `<i>`. La specifica HTML5 descrive il `<i>` come:

...un intervallo di testo con una voce o uno stato d'animo alternativo, o altrimenti sfalsato dalla normale prosa in un modo che indica una diversa qualità del testo.

Basti dire che non deve essere usato semplicemente per mettere in corsivo qualcosa. Diamo un'occhiata al nostro markup attuale per l'area del contenuto principale della nostra homepage e vediamo se possiamo migliorare il significato per gli user-agent. Questo è ciò che abbiamo attualmente:

```
<!-- the content -->
<div id="content">
  
  <h1>Every year <span>when I watch the Oscars I'm annoyed...</span>
</h1>
```

<p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>

<p>We're here to put things right. </p>

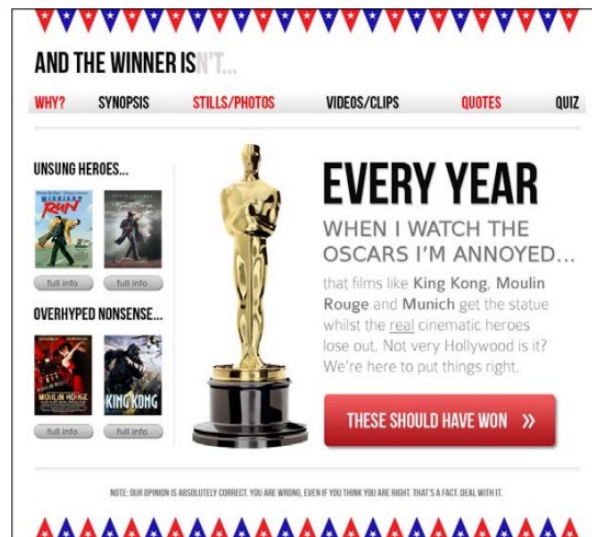
these should have won »

</div>

Possiamo sicuramente migliorare le cose e per cominciare, il tag all'interno del nostro tag headline <h1> è semanticamente privo di significato in quel contesto, quindi mentre stiamo cercando di aggiungere enfasi al nostro stile, facciamolo anche con il nostro codice:

<h1>Every year when I watch the Oscars I'm annoyed...</h1>

Diamo di nuovo un'occhiata al nostro design iniziale:



Abbiamo bisogno di dare uno stile ai nomi dei film in modo diverso, ma non è necessario che suggeriscano uno stato d'animo o una voce diversi. Sembra che il tag sia il candidato perfetto qui:

<p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?</p>

In tal caso, usiamo un tag <i>. Potresti obiettare che dovrei usare anche il tag che andrebbe bene anche in questo caso, ma vado con <i>. Quindi ecco! Questo sarebbe simile al seguente:

<p><i>We're here to put things right.</i></p>

Come ``, i browser impiegheranno in corsivo il tag `<i>` in modo che, se necessario, ridisegni lo stile se necessario. Quindi, ora abbiamo aggiunto alcune semantiche a livello di testo al nostro contenuto per dare maggiore significato al nostro markup. Ci sono molti altri tag semantici a livello di testo in HTML5; per il riepilogo completo, dai un'occhiata alla sezione pertinente della specifica al seguente URL: <http://dev.w3.org/html5/spec/Overview.html#text-level-semantics>. Tuttavia, con un piccolo sforzo extra possiamo fare un ulteriore passo avanti fornendo un significato aggiuntivo per gli utenti che ne hanno bisogno.

Aggiungere accessibilità al tuo sito

Lo scopo di WAI-ARIA è principalmente quello di risolvere il problema di rendere accessibili i contenuti dinamici di una pagina. Fornisce un mezzo per descrivere ruoli, stati e proprietà per i widget personalizzati (sezioni dinamiche nelle applicazioni Web) in modo che siano riconoscibili e utilizzabili dagli utenti di tecnologie assistive. Ad esempio, se un widget sullo schermo mostra un prezzo delle azioni in costante aggiornamento, come potrebbe saperlo un utente non vedente che accede alla pagina? WAI-ARIA tenta di risolvere questo problema.

L'implementazione completa di ARIA esula dallo scopo di questo libro (per informazioni complete, andare su <https://www.w3.org/WAI/intro/aria>). Tuttavia, ci sono alcune parti di ARIA molto facili da implementare che possiamo adottare per migliorare qualsiasi sito scritto in HTML5 per utenti di tecnologie assistive. Se hai il compito di creare un sito Web per un cliente, spesso non viene messo da parte tempo/denaro per aggiungere il supporto per l'accessibilità oltre le basi (purtroppo, spesso non ci si pensa affatto). Tuttavia, possiamo usare i ruoli fondamentali di ARIA per correggere alcune delle evidenti carenze nella semantica dell'HTML e consentire ai lettori di schermo che supportano WAI-ARIA di passare facilmente da una parte all'altra dello schermo. L'implementazione dei ruoli fondamentali di ARIA non è specifica per un web design reattivo. Tuttavia, poiché è relativamente semplice aggiungere un supporto parziale (che si convalida anche come HTML5 senza ulteriori sforzi), sembra poco utile lasciarlo fuori da qualsiasi pagina Web che scrivi in HTML5 da oggi in poi. Ora vediamo come funziona, considera la nostra nuova area di navigazione HTML5:

```
<nav>
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
```

```
</nav>
```

Possiamo rendere quest'area comprensibile per uno screen reader compatibile con WAI-ARIA aggiungendo un attributo del ruolo di riferimento, come mostrato nel seguente frammento di codice:

```
<nav role="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
```

Quanto è facile? Esistono ruoli fondamentali per le seguenti parti della struttura di un documento:

- **application**: questo ruolo viene utilizzato per specificare una regione utilizzata da un'applicazione Web.
- **banner**: questo ruolo viene utilizzato per specificare un'area dell'intero sito (piuttosto che specifica del documento).
L'intestazione e il logo di un sito, ad esempio.
- **complementary**: questo ruolo viene utilizzato per specificare un'area complementare alla sezione principale di una pagina. Nel nostro sito “E il vincitore non è...”, le aree **UNSUNG HEROES** e **OVERHYPED NONSENSE** sarebbero considerate complementari.
- **contentinfo**: questo ruolo dovrebbe essere utilizzato per informazioni sul contenuto principale. Ad esempio, per visualizzare le informazioni sul copyright nel piè di pagina di una pagina.
- **form**: hai indovinato, un modulo! Tuttavia, tieni presente che se il modulo in questione è un modulo di ricerca, utilizza invece il ruolo **search**.
- **main**: questo ruolo viene utilizzato per specificare il contenuto principale della pagina.

- navigation: questo ruolo viene utilizzato per specificare i collegamenti di navigazione per il documento corrente o i documenti correlati.
- search: questo ruolo viene utilizzato per definire un'area che esegue una ricerca.

Andiamo avanti ed estendiamo la nostra attuale versione HTML5 di “E il vincitore non è...” markup con i ruoli rilevanti di ARIA:

```
<body>
<div id="wrapper">
<!-- the header and navigation -->
<header role="banner">
<div id="logo">And the winner is<span>n't...</span></div>
<nav role="navigation">
<ul>
<li><a href="#">Why?</a></li>
<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</nav>
</header>
<!-- the content -->
<div id="content" role="main">

<h1>Every year <em>when I watch the Oscars I'm annoyed...</em>
</h1>
<p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and
<b>Munich</b> get the statue whilst the real cinematic heroes lose
out. Not very Hollywood is it?</p>
<p><i>We're here to put things right.</i></p>
<a href="#">these should have won &raquo;</a>
</div>
<!-- the sidebar -->
<aside>
```

```
<section role="complementary">
<div class="sideBlock unsung">
<h1>Unsung heroes...</h1>
<a href="#"></a>
<a href="#"></a>
</div>
</section>
<section role="complementary">
<div class="sideBlock overHyped">
<h1>Overhyped nonsense...</h1>
<a href="#"></a>
<a href="#"></a>
</div>
</section>
</aside>
<!-- the footer -->
<footer role="contentinfo">
<p>Note: our opinion is absolutely correct. You are wrong, even if
you think you are right. That's a fact. Deal with it.</p>
</footer>
</div>
</body>
```

Si spera che questa breve introduzione a WAI-ARIA abbia dimostrato quanto sia facile aggiungere un supporto parziale per coloro che usano la tecnologia assistiva e spero che tu la possa usare nel tuo prossimo progetto HTML5.

Incorporare media in HTML5

Per molti, HTML5 è entrato nel loro vocabolario per la prima volta quando Apple ha rifiutato di aggiungere il supporto per Flash nei propri dispositivi iOS. Flash aveva guadagnato il dominio del mercato (alcuni sostenebbero il controllo del mercato) come plug-in preferito per pubblicare video tramite un browser web. Tuttavia, invece di utilizzare la tecnologia proprietaria di Adobe, Apple ha deciso di affidarsi a HTML5 al posto di gestire il rendering rich media. Sebbene HTML5 stesse comunque facendo buoni progressi in quest'area, il supporto pubblico di Apple ha dato un grande vantaggio ad HTML5 e ha aiutato i suoi strumenti multimediali a ottenere maggiore successo nella comunità più ampia.

Come puoi immaginare, Internet Explorer 8 e versioni precedenti non supportano video e audio HTML5. Tuttavia, ci sono soluzioni alternative facili da implementare per i browser in difficoltà di Microsoft, di cui parleremo a breve. La maggior parte degli altri browser moderni (Firefox 3.5+, Chrome 4+, Safari 4, Opera 10.5+, Internet Explorer 9+, iOS 3.2+, Opera Mobile 11+, Android 2.3+) li gestiscono perfettamente. Aggiungere video e audio con HTML5 è semplice, ho sempre trovato l'aggiunta di media come video e audio in una pagina web è una vera seccatura in HTML 4.01. Non è difficile, solo disordinato. HTML5 rende le cose molto più facili. La sintassi è molto simile all'aggiunta di un'immagine:

```
<video src="myVideo.ogg"></video>
```

Una boccata d'aria fresca per la maggior parte dei web designer! Piuttosto che valanghe di codice attualmente necessarie per includere il video in una pagina, HTML5 consente a un singolo tag `<video></video>` (o `<audio></audio>` per l'audio) di fare tutto il lavoro sporco. È anche possibile inserire del testo tra il tag di apertura e quello di chiusura per informare gli utenti quando non utilizzano un browser compatibile con HTML5 e ci sono attributi aggiuntivi che normalmente vorresti aggiungere, come l'altezza e la larghezza. Aggiungiamo questi in:

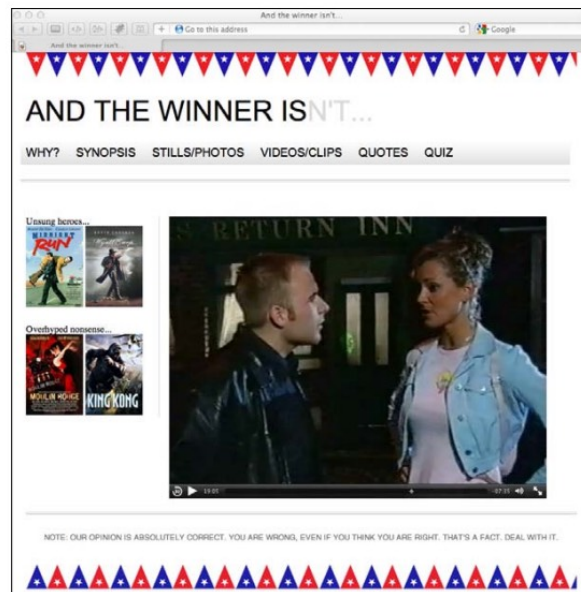
```
<video src="video/myVideo.mp4" width="640" height="480">What,  
do you mean you don't understand HTML5?</video>
```

Ora, se aggiungiamo lo snippet di codice precedente nella nostra pagina e lo guardiamo in Safari, apparirà ma non ci saranno controlli per la riproduzione. Per ottenere i controlli di riproduzione predefiniti è necessario

aggiungere l'attributo dei controlli. Potremmo anche aggiungere l'attributo di riproduzione automatica (non consigliato: è risaputo che tutti odiano i video che vengono riprodotti automaticamente). Ciò è dimostrato nel seguente frammento di codice:

```
<video src="video/myVideo.mp4" width="640" height="480" controls  
autoplay>What, do you mean you don't understand HTML5?</video>
```

Il risultato del frammento di codice precedente è mostrato nella schermata seguente:



Ulteriori attributi includono il precaricamento per controllare il precaricamento dei media (i primi utenti di HTML5 dovrebbero notare che il precaricamento sostituisce il buffer automatico), il ciclo per ripetere il video e il poster per definire un fotogramma poster del video. Ciò è utile se è probabile che si verifichi un ritardo nella riproduzione del video. Per utilizzare un attributo, aggiungilo semplicemente al tag. Ecco un esempio che include tutti questi attributi:

```
<video src="video/myVideo.mp4" width="640" height="480" controls  
autoplay preload="auto" loop poster="myVideoPoster.jpg">What, do you  
mean you don't understand HTML5?</video>
```

La specifica originale per HTML5 prevedeva che tutti i browser supportassero la riproduzione diretta (senza plug-in) di video e audio all'interno dei contenitori Ogg. Tuttavia, a causa di controversie all'interno

del gruppo di lavoro HTML5, l'insistenza sul supporto per Ogg (inclusi video Theora e audio Vorbis), come standard di base, è stata abbandonata dalle iterazioni più recenti della specifica HTML5. Pertanto, alcuni browser supportano la riproduzione di un set di file video e audio mentre altri supportano l'altro set. Ad esempio, Safari consente solo l'utilizzo di file multimediali MP4/H.264/AAC con gli elementi <video> e <audio> mentre Firefox e Opera supportano solo Ogg e WebM. Perché non possiamo andare tutti d'accordo?? Per fortuna, c'è un modo per supportare più formati all'interno di un tag. Tuttavia non ci preclude la necessità di creare più versioni dei nostri media. Incrociamo le dita affinché si risolva presto questa situazione, a tempo debito, nel frattempo, armati di più versioni del nostro file, contrassegnando il video come segue:

```
<video width="640" height="480" controls autoplay preload="auto"
loop
poster="myVideoPoster.jpg">
<source src="video/myVideo.ogv" type="video/ogg">
<source src="video/myVideo.mp4" type="video/mp4">
What, do you mean you don't understand HTML5?
</video>
```

Se il browser supporta la riproduzione di Ogg, utilizzerà quel file; in caso contrario, continuerà fino al tag <source> successivo. L'utilizzo del tag <source> in questo modo ci consente di fornire una serie di fallback, se necessario. Ad esempio, oltre a fornire entrambe le versioni MP4 e Ogg, se volessimo garantire un fallback adatto per Internet Explorer 8 e versioni precedenti, potremmo aggiungere un fallback Flash. Inoltre, se l'utente non disponeva di alcuna tecnologia di riproduzione adeguata, potremmo fornire collegamenti per il download ai file stessi:

```
<video width="640" height="480" controls autoplay preload="auto"
loop
poster="myVideoPoster.jpg">
<source src="video/myVideo.mp4" type="video/mp4">
<source src="video/myVideo.ogv" type="video/ogg">
<object width="640" height="480" type="application/x-
shockwaveflash" data="myFlashVideo.SWF">
<param name="movie" value="myFlashVideo.swf" />
<param name="flashvars"
value="controlbar=over&image=myVideoPo
```

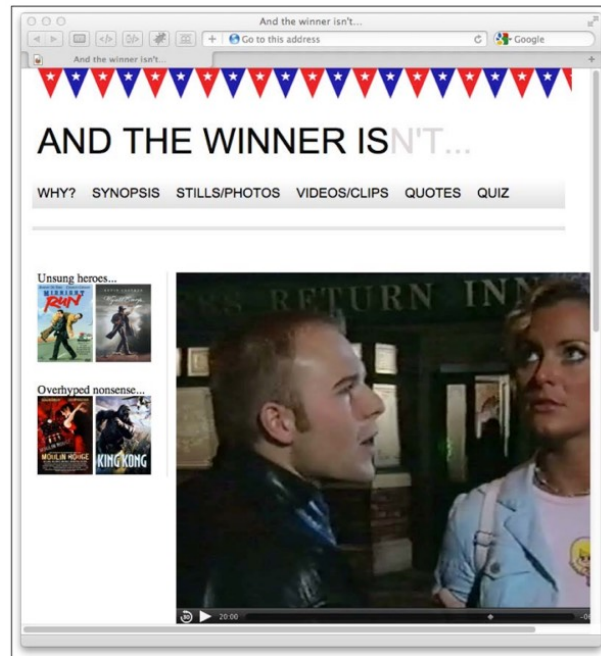
```
ster.jpg&file=video/myVideo.mp4" />

</object>
<p> <b>Download Video:</b>
MP4 Format: <a href="myVideo.mp4">"MP4"</a>
Ogg Format: <a href="myVideo.ogv">"Ogg"</a>
</p>
</video>
```

Il tag <audio> funziona secondo gli stessi principi con gli stessi attributi esclusi width, height e poster. In effetti, puoi anche usare i tag <video> e <audio> quasi in modo intercambiabile. La principale differenza tra i due è il fatto che <audio> non ha un'area di riproduzione per il contenuto visibile.

Video responsive

Abbiamo visto che, come sempre, il supporto dei browser più vecchi porta a un workaround nel codice. Ciò che era iniziato con il tag `<video>` costituito da una o due righe è finito per essere 10 o più righe (e un file Flash aggiuntivo) solo per rendere fruibili le versioni precedenti di Internet Explorer! Da parte mia, di solito rinuncio al fallback di Flash alla ricerca di un footprint di codice più piccolo, ma ogni caso d'uso è diverso. Ora, l'unico problema con la nostra adorabile implementazione video HTML5 è che non è reattiva. Giusto. Dai un'occhiata al seguente screenshot e fai del tuo meglio per trattenere le lacrime:



Per fortuna, per i video incorporati HTML5, la soluzione è semplice. Rimuovi semplicemente qualsiasi attributo di `width` e `height` nel markup (ad esempio, rimuovi `width="640" height="480"`) e aggiungi quanto segue nel CSS:

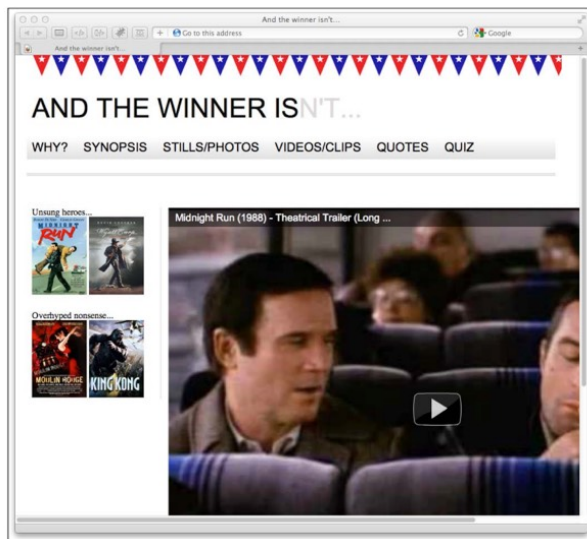
```
video { max-width: 100%; height: auto; }
```

Tuttavia, funziona bene per i file che potremmo ospitare localmente ma non risolve il problema dei video incorporati in un `iFrame` (YouTube,

Vimeo, e altri). Il codice seguente aggiunge un trailer del film per Midnight Run da YouTube:

```
<iframe width="960" height="720"
src="http://www.youtube.com/embed/B1_N28DA3gY" frameborder="0"
allowfullscreen></iframe>
```

Nonostante la mia precedente regola CSS, ecco cosa succede:



Sono sicuro che DeNiro non sarebbe troppo contento! Esistono diversi modi per risolvere il problema, ma di gran lunga il più semplice che ho incontrato è un piccolo plug-in jQuery chiamato FitVids. Vediamo com'è facile usare il plugin aggiungendolo al sito. Prima di tutto, avremo bisogno della libreria JavaScript jQuery. Caricala questo nel tuo elemento <head>, qui sto usando la versione della Content Delivery Network (CDN) di Google.

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js">
</script>
```

Scarica il plug-in FitVids da <http://fitvidsjs.com/> (maggiori informazioni sul plug-in sono disponibili su <http://daverupert.com/2011/09/responsive-video-embedswith-fitvids/>).

Ora, salva il file JavaScript FitVids in una cartella adatta (ho chiamato con fantasia il mio "js") e quindi collegalo al JavaScript FitVids nell'elemento <head>:


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

Infine, dobbiamo solo usare jQuery per indirizzare il particolare elemento contenente il nostro video di YouTube. Qui, ho aggiunto il mio video YouTube di Midnight Run all'interno del div #content:

```
<script>
$(document).ready(function(){
// Target your .container, .wrapper, .post, etc.
$("#content").fitVids();
});
</script>
```

Questo è tutto ciò che c'è da fare. Grazie al plug-in FitVid jQuery, ora ho un video YouTube completamente reattivo. (Nota: ragazzi, non fate attenzione al signor DeNiro; fumare fa male!).



Applicazioni Web offline

Sebbene ci siano molte interessanti funzionalità all'interno di HTML5 che non aiutano esplicitamente la nostra ricerca reattiva (l'API di geolocalizzazione, ad esempio), le applicazioni Web offline potrebbero potenzialmente interessarci. Poiché siamo consapevoli del numero crescente di utenti mobile che probabilmente accedono ai nostri siti, che ne dici di fornire loro un mezzo per visualizzare i nostri contenuti senza nemmeno essere connessi a Internet? La funzionalità delle applicazioni Web offline HTML5 offre questa possibilità. Tale funzionalità è di utilità più ovvia per le applicazioni web (stranamente; mi chiedo come abbiano inventato il titolo). Immagina un'applicazione web per prendere appunti online. Un utente potrebbe essere a metà del completamento di una nota quando la connessione al cellulare si interrompe. Con le applicazioni Web offline HTML5, potrebbero continuare a scrivere la nota mentre sono offline e i dati potrebbero essere inviati una volta che la connessione è nuovamente disponibile. La cosa fantastica degli strumenti delle applicazioni Web offline HTML5 è che sono troppo facili da configurare e utilizzare. Qui li useremo in modo semplice, per creare una versione offline del nostro sito. Ciò significa che se gli utenti vogliono guardare il nostro sito mentre non hanno una connessione di rete, possono farlo. Le applicazioni Web offline funzionano in base a ciascuna pagina che deve essere utilizzata offline, puntando a un file di testo noto come file .manifest. Questo file elenca tutte le risorse (HTML, immagini, JavaScript e così via) necessarie alla pagina se non è in linea. Un browser abilitato per l'applicazione Web offline (Firefox 3+, Chrome 4+, Safari 4+, Opera 10.6+, iOS 3.2+, Opera Mobile 11+, Android 2.1+, Internet Explorer 10+) legge il file .manifest, scarica le risorse elencate e li memorizza nella cache in locale in caso di interruzione della connessione. Semplice, eh? Nel tag HTML di apertura, indichiamo un file .manifest:

```
<html lang="en" manifest="/offline.manifest">
```

Puoi chiamare questo file come vuoi, ma si consiglia che l'estensione del file utilizzata sia .manifest. Se il tuo server web funziona su Apache, probabilmente dovrai modificare il file .htaccess con la seguente riga:

```
AddType text/cache-manifest .manifest
```

Ciò consentirà al file di avere il tipo MIME corretto, ovvero text/cachemanifest. Mentre siamo nel file .htaccess, aggiungi anche quanto segue:

```
<Files offline.manifest>
ExpiresActive On
ExpiresDefault "access"
</Files>
```

L'aggiunta delle righe di codice precedenti impedisce al browser di memorizzare la cache nella cache. Sì, avete letto bene. Poiché il file offline.manifest è un file statico, per impostazione predefinita il browser memorizzerà nella cache il file offline.manifest. Quindi, questo dice al server di dire al browser di non farlo! Ora dobbiamo scrivere il file offline.manifest. Questo indicherà al browser quali file rendere disponibili offline. Ecco il contenuto dell'offline.manifest per il sito “E il vincitore non è...”

```
CACHE MANIFEST
#v1
CACHE:
basic_page_layout_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png
NETWORK:
```



FALLBACK:

```
//offline.html
```

Il file manifest deve iniziare con CACHE MANIFEST. La riga successiva è semplicemente un commento, che indica il numero di versione

del file manifest. Ne parleremo a breve. La sezione CACHE: elenca i file di cui abbiamo bisogno per l'uso offline. Questi dovrebbero essere relativi al file offline.manifest, quindi potrebbe essere necessario modificare i percorsi a seconda delle risorse che richiedono la memorizzazione nella cache. È anche possibile utilizzare URL assoluti, se necessario. La sezione NETWORK: elenca tutte le risorse che non devono essere memorizzate nella cache. Pensala come una "lista bianca online". Qualunque cosa sia elencata qui ignorerà sempre la cache se è disponibile una connessione di rete. Se vuoi rendere disponibile il contenuto del tuo sito dove è disponibile una rete (piuttosto che cercare solo nella cache offline), il carattere * lo consente. È noto come flag jolly della whitelist online. La sezione FALLBACK: utilizza il carattere / per definire un pattern URL. Fondamentalmente chiede "questa pagina è nella cache?", se trova la pagina lì, ottimo, la visualizza. In caso contrario, mostra all'utente il file specificato: offline.html.

A seconda delle circostanze, esiste un modo ancora più semplice per impostare un file offline. file manifest. Qualsiasi pagina che punta a un file manifest offline (ricorda che lo facciamo aggiungendo manifest="/offline.manifest" nel nostro tag di apertura <html>) viene automaticamente aggiunta alla cache quando un utente la visita. Questa tecnica aggiungerà alla cache tutte le pagine del tuo sito visitate da un utente in modo che possano visualizzarle nuovamente offline. Ecco come dovrebbe essere il manifest:

```
CACHE MANIFEST
# Cache Manifest v1
FALLBACK:
//offline.html
NETWORK:
```

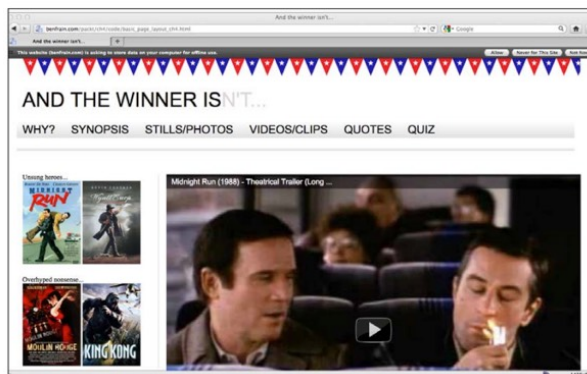


UN PUNTO DA notare quando si opta per questa tecnica è che verrà scaricato e memorizzato nella cache solo l'HTML della pagina visitata. Non le immagini/JavaScript e altre risorse che possono contenere e a cui collegarsi. Se questi sono essenziali, specificali in una sezione CACHE: come già descritto in precedenza nella sezione Comprensione del file

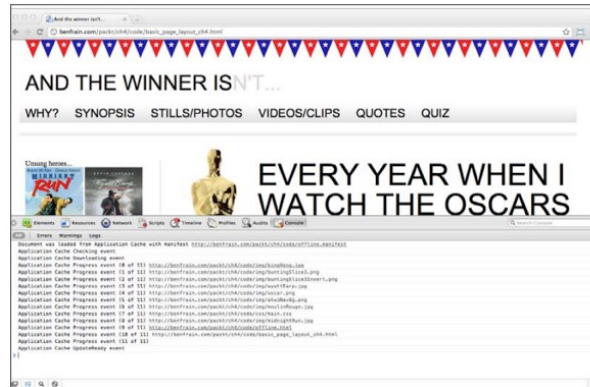
manifest. A proposito di quella versione commento Quando apporti modifiche al tuo sito o a una qualsiasi delle sue risorse, devi modificare in qualche modo il file offline.manifest e ricaricarlo. Ciò consentirà al server di fornire il nuovo file al browser, che riceverà le nuove versioni dei file e avvierà nuovamente il processo offline. Seguo l'esempio di Nick Pilgrim (dall'ottimo Dive into HTML5) e aggiungo un commento all'inizio del file offline.manifest che incremento ad ogni modifica:

```
# Cache Manifest v1
```

Ora è il momento di testare il nostro lavoro manuale. Visita la pagina in un browser compatibile con l'applicazione Web offline. Alcuni browser avviseranno della modalità offline (ad esempio Firefox, nota la barra in alto) mentre Chrome non ne fa menzione:



Ora, stacca la spina (o spegni il WiFi, che semplicemente non suonava così drammatico come "staccare la spina") e aggiorna il browser. Si spera che la pagina si aggiorni come se fosse connessa, ma non lo è. Quando ho problemi a far funzionare correttamente i siti in modalità offline, tendo a utilizzare Chrome per risolvere i problemi. Gli strumenti per sviluppatori integrati hanno una pratica sezione Console (accedi facendo clic sul logo della chiave inglese a destra della barra degli indirizzi e quindi vai su Strumenti | Strumenti per sviluppatori e fai clic sulla scheda Console) che segnala il successo o il fallimento della cache offline e spesso fa notare cosa stai sbagliando. Nella mia esperienza, di solito sono problemi di percorso; ad esempio, non indirizzare le mie pagine alla posizione corretta del file manifest.



Abbiamo trattato molto in questo capitolo. Tutto, dalle basi per creare una pagina valida come HTML5, per consentire alle nostre pagine di funzionare offline quando gli utenti non dispongono di una connessione Internet. Abbiamo anche affrontato l'incorporamento di rich media (video) nel nostro markup e assicurato che si comporti in modo reattivo per diverse viewport. Sebbene non sia specifico per i design reattivi, abbiamo anche spiegato come possiamo scrivere codice semanticamente ricco e significativo e fornire anche aiuto agli utenti che si affidano alle tecnologie assistive. Tuttavia, il nostro sito deve ancora affrontare alcune gravi carenze. Senza esagerare, sembra piuttosto squallido. Il nostro testo non ha uno stile e ci mancano completamente dettagli come i pulsanti visibili nella composizione originale. Finora abbiamo evitato di caricare il markup con le immagini per risolvere questi problemi con una buona ragione. Non abbiamo bisogno di loro! Invece, nei prossimi capitoli abbracceremo la potenza e la flessibilità di CSS3 per creare un design reattivo più veloce e manutenibile.

INTRODUZIONE A CSS

"Un minuto per imparare... Una vita per diventare maestri". Quella frase potrebbe sembrare un po' banale di questi tempi, ma mi piace comunque. È stata resa popolare recentemente per essere lo slogan di un gioco da tavolo. Come in tutti i giochi, ci sono delle regole da seguire e solitamente non sono troppo complesse. Allo stesso modo, non è particolarmente difficile imparare le regole dei CSS. Scrivi un selettore e abbinai gli elementi, quindi scrivi coppie chiave/valore che danno uno stile a quegli elementi. Anche i principianti non hanno molti problemi a capire questa sintassi di base. Il trucco per diventare bravi con i CSS, come in tutti i giochi, è sapere esattamente quando fare cosa. CSS è uno dei linguaggi del web, ma non è proprio un linguaggio di programmazione. I CSS hanno poco o niente in termini di logica e cicli loop. La parte matematica è stata limitata a una singola funzione e solo di recente sono state aggiunte le variabili. Raramente è necessario considerare l'aspetto relativo alla sicurezza, sei libero di fare ciò che ti piace con i CSS. Non ci sarà alcun errore e non vedrai parti di codice non compilato. Il viaggio per diventare bravi con i CSS implica l'apprendimento di tutto ciò di cui sono capaci i CSS. Più sai, più naturale inizia a sembrarti. Più ti eserciti, più facilmente il tuo cervello raggiungerà quel metodo di layout perfetto. Più leggi, più ti sentirai sicuro di affrontare qualsiasi progetto. Gli sviluppatori CSS davvero bravi non sono scoraggiati da alcun design anzi, cercano sempre nuove sfide per creare qualcosa di sorprendente e funzionale. Ogni lavoro diventa un'opportunità per diventare intelligenti, un enigma da risolvere. Gli sviluppatori CSS davvero bravi hanno quello spettro completo e ampio di conoscenza di ciò di cui sono capaci i CSS. Questo libro fa parte del tuo viaggio per diventare un ottimo sviluppatore CSS, otterrai lo spettro di conoscenze necessarie per arrivarci. Se permetti un'altra metafora, nonostante i CSS abbiano un paio di decenni di vita, è un po' come il selvaggio Far West. Puoi fare qualsiasi cosa tu voglia, purché faccia quello che vuoi. Non ci sono regole rigide, ma poiché sei da solo, senza delle linee guida per dirti se stai facendo un buon lavoro o meno, dovrai prestare molta attenzione. Piccoli cambiamenti possono avere effetti enormi, infatti, un foglio di stile può crescere sempre più, diventando ingombrante. Puoi iniziare ad avere paura dei tuoi stili! Ti aiuterò a diventare uno sviluppatore

CSS e a domare questo selvaggio Far West. Ti immergerai profondamente nel linguaggio stesso, imparando di cosa sono capaci i CSS. Quindi, in modo altrettanto importante, imparerai concetti sul linguaggio che ti faranno salire di livello in altri modi. Sarai più bravo a scrivere codice duraturo, comprensibile e performante. Anche gli sviluppatori esperti ne trarranno vantaggio, infatti, se ti ritrovi a leggere qualcosa che già conosci, rafforzerai le tue abilità, rivedrai le tue conoscenze e troverai delle chicche che ti sorprenderanno ed estenderanno le conoscenze pregresse. CSS è stato proposto nel 1994 e implementato (parzialmente) per la prima volta da Internet Explorer 3 nel 1996. Fu in quel periodo che scoprii il meraviglioso pulsante "Visualizza sorgente" e realizzai che tutti i segreti di una pagina web erano lì e dovevano essere decifrati. Ho imparato HTML e CSS giocando in un editor di testo e vedendo in che modo funzionava. Era una scusa divertente per passare più tempo possibile su Internet ma, nel frattempo, avevo bisogno di trovare una vera carriera. Ho continuato con la laurea in Informatica ma non sapevo che le strade si sarebbero intrecciate negli anni 2000 quando è emerso il concetto di "sviluppatore web". Sono stato in sintonia con i CSS sin dall'inizio e anche quando lavoro, mi sembra di giocare. Ho lavorato sia sul back-end e sul front-end, ma mi sono sempre trovato ad essere l'esperto CSS di ogni team di cui ho fatto parte. Spesso è la parte più trascurata dello stack web. Ma una volta che sei stato su un progetto con CSS pulito, non vorrai più farne a meno, dopo averlo visto in azione, anche gli sviluppatori web esperti chiedono: "Come faccio a imparare i CSS?" Non c'è una risposta concisa e diretta a questa domanda. Non si tratta di imparare uno o due comandi rapidi, piuttosto, devi capire tutte le parti disparate del linguaggio e come possono combaciare. Alcuni libri sono una buona introduzione ai CSS per principianti, ma molti sviluppatori hanno già una conoscenza di base. Alcuni libri insegnano molti trucchi utili ma presuppongono che il lettore abbia padronanza del linguaggio. Allo stesso tempo, i CSS cambiano velocemente e il design reattivo è ora lo standard de facto. Nel 2016 abbiamo assistito all'ascesa di flexbox e nel 2017 è iniziata l'ascesa di qualcosa chiamato layout a griglia. Le modalità di fusione, le ombre dei riquadri, le trasformazioni, le transizioni e le animazioni sono tutti concetti nuovi. Le nuove funzionalità continuano ad essere implementate, man mano che i browser si aggiornano automaticamente alla versione più recente. Bisogna sempre stare al passo e, indipendentemente dal fatto che tu sia relativamente nuovo nel settore o che

tu abbia bisogno di migliorare o aggiornare le tue skills CSS, ho scritto questo libro per aggiornarti. Tutto in questo libro è per uno dei tre motivi:

1. È essenziale. Ci sono molti fondamenti del linguaggio che, purtroppo, molti sviluppatori non comprendono appieno. Ciò include la cascata, il comportamento dei float e il posizionamento. Li esaminerò a fondo, spiegando come funzionano.
2. È nuovo. Molte nuove funzionalità sono emerse negli ultimi anni o stanno emergendo solo ora. Tratterò gli ultimi miglioramenti di CSS ed alcune cose che sono proprio dietro l'angolo. Questo è un libro lungimirante. Indicherò problemi di compatibilità con le versioni precedenti, ove pertinente, ma sono sfacciatamente ottimista sul presente e sul futuro dello sviluppo cross-browser.
3. Non è incluso nella maggior parte dei libri CSS. Il mondo dei CSS è enorme. Esistono importanti best practice e approcci comuni nel mondo moderno dello sviluppo di applicazioni web. Questi non sono strettamente parte del linguaggio CSS, ma piuttosto parte della sua cultura. E sono vitali per lo sviluppo web moderno.

Allora, come si imparano i CSS? Questo libro è un tentativo di rispondere a questa domanda, per le persone che sanno di averne più bisogno. Il mondo dei CSS sta maturando. Sempre più sviluppatori web nel settore si stanno rendendo conto che credono di "conoscere" i CSS ma non lo conoscono così profondamente come probabilmente dovrebbero. Negli ultimi anni, il linguaggio si è evoluto, quindi anche quegli sviluppatori che una volta erano esperti in CSS potrebbero trovare una serie completamente nuova di abilità su cui recuperare. Questo libro mira a soddisfare entrambe queste esigenze: fornire una profonda padronanza del linguaggio e aggiornarti sui recenti sviluppi e sulle nuove funzionalità dei CSS. Laddove i concetti siano difficili o comunemente fraintesi, spiegherò in dettaglio come funzionano e perché si comportano in quel modo. In altri capitoli, potrei non esaurire l'argomento, ma ti darò abbastanza conoscenze per poter lavorare efficacemente e ti indirizzerò nella giusta direzione se desideri approfondire le tue conoscenze. In tutto, questo libro colmerà le tue lacune.

Alcuni degli argomenti potrebbero giustificare interi libri da soli: animazione, tipografia, persino flexbox e layout della griglia. Il mio obiettivo è arricchire le tue conoscenze, aiutarti a rafforzare i tuoi punti deboli e farti innamorare di questo potente linguaggio. Innanzitutto, questo libro è per gli sviluppatori che sono stanchi di combattere con i CSS e sono pronti a capire davvero come funziona. Potresti essere un principiante o potresti avere quindici anni di esperienza. Mi aspetto che tu abbia una conoscenza superficiale di HTML, CSS e, in alcuni punti, JavaScript. Finché avrai familiarità con la sintassi di base dei CSS, probabilmente sarai in grado di seguire questo libro. Ma è scritto principalmente per gli sviluppatori che hanno passato del tempo con i CSS, si sono imbattuti nei limiti e ne sono usciti frustrati. Nei punti in cui utilizzo JavaScript, l'ho mantenuto il più semplice possibile; quindi, dovresti essere in grado di seguirmi. Se invece sei un designer che cerca di entrare nel mondo del web design, sospetto che anche tu imparerai molto, anche se non l'ho scritto pensando a te. Il libro può fornire alcune informazioni sul punto di vista degli sviluppatori con cui lavorerai ed è diviso in 3 capitoli. Partiremo dalle basi, concentrandoci su alcuni dettagli che probabilmente ti sei perso la prima volta:

- Il capitolo 1 copre la cascata e l'eredità. Questi concetti controllano quali stili vengono applicati a quali elementi della pagina.
- Il capitolo 2 discute le unità relative, con un'enfasi su em e rem. Le unità relative sono strumenti versatili e importanti nei CSS e questo capitolo ti farà acquisire familiarità con il lavoro con esse.
- Il capitolo 3 tratta il box model. Ciò comporta il controllo della dimensione degli elementi sulla pagina e la quantità di spazio tra di loro.

Mi sono impegnato molto per descrivere al meglio CSS in questo libro. Si parte dall'essenziale che è da lì, si costruiscono gli argomenti l'uno sull'altro. In molti punti, mi riferisco a concetti precedenti per collegarli in modo pertinente. Questo libro contiene molti esempi di codice sorgente, sia in elenchi numerati che in linea con il testo normale. In entrambi i casi, il codice sorgente è formattato in un font a larghezza fissa come questo per

separarlo dal testo normale. A volte il codice è anche in grassetto per evidenziare il codice che è stato modificato rispetto ai passaggi precedenti del capitolo, ad esempio quando una nuova funzionalità viene aggiunta a una riga di codice esistente. In molti casi, il codice sorgente originale è stato riformattato; ho aggiunto interruzioni di riga e rielaborato il rientro per adattare lo spazio disponibile della pagina nel libro. Inoltre, i commenti nel codice sorgente sono stati spesso rimossi dagli elenchi quando il codice è descritto nel testo. Le annotazioni del codice accompagnano molti degli elenchi, evidenziando concetti importanti. CSS è pensato per essere accoppiato con HTML; fornisco sempre un elenco di codici per l'HTML e un altro per i CSS. Nella maggior parte dei capitoli, riutilizzo lo stesso HTML per più elenchi CSS. Ti guiderò attraverso la modifica di un foglio di stile in molte fasi e ho cercato di chiarire come mi aspetto che tu modifichi il tuo foglio di stile da un elenco CSS all'altro. Il test cross-browser è una parte importante dello sviluppo web. La maggior parte del codice in questo libro è supportata in IE 10 e 11, Microsoft Edge, Chrome, Firefox, Safari, Opera e la maggior parte dei browser mobile. Le funzionalità più recenti potrebbero non funzionare in tutti questi browser; in tal caso sarà indicato. Solo perché una funzione non è supportata in un particolare browser non significa che non puoi usarla. Spesso puoi fornire un comportamento di fallback per i browser meno recenti come compromesso accettabile, verranno mostrati esempi di questo tipo in diversi casi. Se stai seguendo gli esempi di codice sul tuo computer, ti consiglio di utilizzare l'ultima versione di Firefox o Chrome.

LE BASI DI CSS

In questo libro esamineremo in modo approfondito le parti più essenziali dei CSS: la cascata, le unità relative e il box model. Questi fondamenti controllano quali stili vengono applicati agli elementi sulla pagina e come vengono determinate le dimensioni di tali elementi. Una comprensione completa di questi argomenti è fondamentale per capire le quattro parti che compongono il modello a cascata, la differenza tra cascata ed ereditarietà, come controllare quali stili si applicano a quali elementi ed evitare fraintendimenti comuni. CSS non è un linguaggio di programmazione, in senso stretto, ma richiede un pensiero astratto. Non è solo uno strumento di progettazione, ma richiede un po' di creatività. Fornisce una sintassi dichiarativa ingannevolmente semplice, ma se ci hai lavorato su progetti di grandi dimensioni, sai che può diventare di una ingombrante complessità. Quando devi imparare a fare qualcosa nella programmazione convenzionale, di solito puoi capire cosa cercare (ad esempio, "Come faccio a trovare elementi di tipo x in un array?"). Con i CSS, non è sempre facile porre il problema in una singola domanda. Anche quando puoi, la risposta è spesso "dipende". Il modo migliore per realizzare qualcosa dipende spesso dai tuoi vincoli progettuali e dalla precisione con cui vorrai gestire vari casi limite. Sebbene sia utile conoscere alcuni "trucchi" o ricette utili che puoi seguire, padroneggiare i CSS richiede la comprensione dei principi che rendono possibili queste pratiche. Questo libro è pieno di esempi, ma è principalmente un libro di principi. La maggior parte degli sviluppatori web conosce il modello a cascata e il box model, conoscono l'unità pixel e potrebbero aver sentito dire che "dovrebbero invece usare gli ems". La verità è che c'è molto altro in questi argomenti e una comprensione superficiale di essi ti porta solo fino a un certo punto. Se vuoi padroneggiare i CSS, devi prima conoscere i fondamenti e bisogna conoscerli fino a fondo. Esaminerò rapidamente le basi, che probabilmente conosci già, e poi approfondirò ogni argomento. Il mio obiettivo è rafforzare le basi su cui è costruito il resto del tuo CSS. In questo capitolo, iniziamo con la C nei CSS, la cascata. Articolero come funziona, quindi ti mostrerò come lavorarci praticamente. In seguito, esamineremo un argomento correlato, l'ereditarietà. Lo sviscererò con uno sguardo alle proprietà abbreviate e ad alcuni malintesi comuni. Insieme,

questi argomenti riguardano l'applicazione degli stili che desideri agli elementi che desideri. Ci sono molti "trucchi" che spesso fanno inciampare gli sviluppatori e una buona comprensione di questi argomenti ti darà un migliore controllo su come fare in modo che il tuo CSS faccia ciò che vuoi che faccia. Con un po' di fortuna, ti divertirai a lavorare con i CSS.

La cascata

Fondamentalmente, i CSS riguardano la dichiarazione di regole: in varie condizioni, vogliamo che accadano determinate cose. Se questa classe viene aggiunta a quell'elemento, allora applica questi stili. Se l'elemento X è figlio dell'elemento Y, applica quegli altri stili. Il browser quindi prende queste regole, determina quali si applicano e dove, infine le utilizza per eseguire il rendering della pagina. Quando guardi piccoli esempi, questo processo è generalmente molto semplice. Ma man mano che il tuo foglio di stile cresce o aumenta il numero di pagine a cui lo applichi, il tuo codice può diventare complesso in modo sorprendentemente rapido. Ci sono spesso diversi modi per ottenere la stessa cosa nei CSS. A seconda della soluzione che utilizzi, potresti ottenere risultati molto diversi quando la struttura dell'HTML cambia o quando gli stili vengono applicati a pagine diverse. Una parte fondamentale dello sviluppo dei CSS consiste nello scrivere le regole in modo che siano prevedibili. Il primo passo è capire esattamente in che modo il browser dà un senso alle tue regole. Ogni regola può essere semplice di per sé, ma cosa succede quando due regole forniscono informazioni contrastanti su come definire lo stile di un elemento? Potresti scoprire che una delle tue regole non fa quello che ti aspetti perché è in conflitto con un'altra. Prevedere come si comportano le regole richiede una comprensione della cascata. Per illustrare ciò, creerai un'intestazione di pagina di base come quella che potresti vedere nella parte superiore di una pagina Web:

Wombat Coffee Roasters

[Home](#) [Coffees](#) [Brewers](#) [Specials](#)

Figure 1.1 Page heading and navigation links

Ha il titolo del sito Web in cima a una serie di link di navigazione di colore verde acqua. L'ultimo link è colorato in arancione per farlo risaltare come una sorta di collegamento in primo piano. Creando questa intestazione di pagina, probabilmente avrai familiarità con la maggior parte dei CSS coinvolti. Questo ci consentirà di concentrarci su aspetti dei CSS che potresti dare per scontati o comprendere solo parzialmente. Per iniziare,

crea un documento HTML e un foglio di stile denominato styles.css. Aggiungi il codice nel listato seguente all'HTML:

```
<!doctype html>
<head>
<link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
<header class="page-header">
<h1 id="page-title" class="title">Wombat Coffee Roasters</h1>
<nav>
<ul id="main-nav" class="nav">
<li><a href="/">Home</a></li>
<li><a href="/coffees">Coffees</a></li>
<li><a href="/brewers">Brewers</a></li>
<li><a href="/specials" class="featured">Specials</a></li>
</ul>
</nav>
</header>
</body>
```

Quando due o più regole hanno come target lo stesso elemento sulla tua pagina, le regole possono fornire dichiarazioni contrastanti. Il prossimo elenco mostra come ciò sia possibile. Lo snippet mostra tre set di regole, ognuno dei quali specifica uno stile di carattere diverso per il titolo della pagina. Il titolo non può avere tre caratteri diversi contemporaneamente. Quale sarà? Aggiungi questo al tuo file CSS per vedere cosa cambia.

```
h1 {
font-family: serif;
}
#page-title {
font-family: sans-serif;
}
.title {
font-family: monospace;
}
```

I set di regole con dichiarazioni in conflitto possono apparire uno dopo l'altro o possono essere sparsi nel foglio di stile. Ad ogni modo, dato il tuo HTML, puntano tutti allo stesso elemento. Tutti e tre i set di regole tentano

di impostare una famiglia di caratteri diversa per questa intestazione. Quale vincerà? Per determinare la risposta, il browser segue una serie di regole, quindi il risultato è prevedibile. In questo caso le regole prevedono che vinca la seconda dichiarazione, che ha un selettore ID; il titolo avrà un font sans-serif. La cascata è il nome di questo insieme di regole e determina come vengono risolti i conflitti, è una parte fondamentale del funzionamento del linguaggio. Sebbene gli sviluppatori più esperti abbiano un'idea generale della cascata, a volte alcune parti vengono fraintese.

Wombat Coffee Roasters

- [Home](#)
- [Coffees](#)
- [Brewers](#)
- [Specials](#)

Analizziamo la cascata: quando le dichiarazioni sono in conflitto, la cascata considera tre cose per risolvere la differenza:

1. Origine del foglio di stile: da dove provengono gli stili. I tuoi stili vengono applicati insieme agli stili predefiniti del browser.
2. Specificità del selettore: quali selettori hanno la precedenza.
3. Ordine di origine: ordine in cui gli stili sono dichiarati nel foglio di stile.

Le regole della cascata sono considerate in questo ordine. Queste regole consentono ai browser di comportarsi in modo prevedibile quando risolvono qualsiasi ambiguità nel CSS. Esaminiamoli uno alla volta ma prima dobbiamo fare una precisazione. A seconda di dove hai imparato i CSS, potresti avere o meno familiarità con tutti i nomi delle varie parti della sintassi CSS. Poiché userò questi termini in tutto il libro, è meglio essere chiari sul loro significato. La riga seguente si chiama dichiarazione:

color: black;

Questa dichiarazione è composta da una proprietà (color) e da un valore (black). Le proprietà non devono essere confuse con gli attributi, che fanno parte della sintassi HTML. Ad esempio, nell'elemento ``, href è un attributo del tag a. Un gruppo di dichiarazioni racchiuse tra parentesi

graffe è chiamato blocco di dichiarazione. Un blocco di dichiarazione è preceduto da un selettore (in questo caso `body`):

```
body { color: black; font-family: Helvetica; }
```

Insieme, il selettore e il blocco di dichiarazione sono chiamati set di regole. Un set di regole è anche chiamato regola, anche se la mia osservazione è che la regola è usata raramente in modo così preciso e di solito è usata al plurale per riferirsi a un insieme più ampio di stili. Infine, le regole `@` sono costrutti linguistici che iniziano con un simbolo "at", come regole `@import` o `@media query`.

I fogli di stile che aggiungi alla tua pagina web non sono gli unici applicati dal browser. Esistono diversi tipi o origini, di fogli di stile. I tuoi sono chiamati stili d'autore; ci sono anche stili di user agent, che sono gli stili predefiniti del browser. Gli stili user agent hanno una priorità più bassa, quindi i tuoi stili li sovrascrivono. Nota bene: alcuni browser consentono agli utenti di definire un foglio di stile utente. Questa è considerata una terza origine, con una priorità tra user agent e stili dell'autore. Gli stili utente sono usati raramente e sfuggono al tuo controllo, li ho tralasciati per semplicità. Gli stili degli user agent variano leggermente da browser a browser, ma generalmente fanno le stesse cose: ai titoli (da `<h1>` a `<h6>`) e ai paragrafi (`<p>`) viene assegnato un margine superiore e inferiore, agli elenchi (`` e ``) viene assegnato un riempimento sinistro e vengono impostati i colori dei link e le dimensioni dei caratteri predefinite.

Esaminiamo nuovamente la pagina di esempio: il titolo è sans-serif a causa degli stili che hai aggiunto. Un certo numero di altre cose è determinato dagli stili user agent: l'elenco ha un riempimento sinistro e un `list-style-type: disc` per produrre i puntini dell'elenco. I link sono blu e sottolineati così l'intestazione e l'elenco avranno margini superiore e inferiore. Dopo aver considerato gli stili dello user agent, il browser applica i tuoi stili: gli stili dell'autore. Ciò consente alle dichiarazioni specificate di sovrascrivere quelle impostate dal foglio di stile dello user agent. Se colleghi più fogli di stile nel tuo HTML, hanno tutti la stessa origine: l'autore. Gli stili dello user agent impostano le cose che in genere desideri; quindi, non fanno nulla di completamente inaspettato. Quando non ti piace quello che fanno a una determinata proprietà, imposta il tuo valore nel tuo foglio di stile. Facciamolo ora. Puoi sovrascrivere alcuni degli stili di user agent che non sono quelli che desideri in modo che la tua pagina faccia quello che desideri. Nell'elenco seguente, ho rimosso le dichiarazioni della

famiglia di caratteri in conflitto dall'esempio precedente e ne ho aggiunte di nuove per impostare i colori e sovrascrivere i margini e il riempimento dell'elenco e i punti dell'elenco. Modifica il foglio di stile in modo che corrisponda a queste modifiche:

```
h1 {  
  color: #2f4f4f;  
  margin-bottom: 10px;  
}  
#main-nav {  
  margin-top: 10px;  
  list-style: none;  
  padding-left: 0;  
}  
#main-nav li {  
  display: inline-block;  
}  
#main-nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 5px;  
  border-radius: 2px;  
  text-decoration: none;  
}
```

Se hai lavorato a lungo con i CSS, probabilmente sei abituato a sovrascrivere gli stili di user agent. Quando lo fai, stai usando la parte di origine della cascata. I tuoi stili sovrascriveranno sempre gli stili user agent perché le origini sono diverse. Potresti notare che ho usato selettori ID in questo codice. Ci sono diverse ragioni per evitare questo approccio, lo approfondirò a breve. C'è un'eccezione alle regole di origine dello stile: le dichiarazioni contrassegnate come importanti. Una dichiarazione può essere contrassegnata come importante aggiungendo !important alla fine della dichiarazione, prima del punto e virgola:

```
color: red !important;
```

Le dichiarazioni contrassegnate con !important sono trattate come un'origine a priorità più alta, quindi l'ordine di preferenza generale, in ordine decrescente, è questo:

1. Important dell'autore
2. Autore
3. User agent

La cascata risolve in modo indipendente i conflitti per ogni proprietà di ogni elemento della pagina. Ad esempio, se imposti un carattere in grassetto su un paragrafo, il margine superiore e inferiore del foglio di stile dello user agent si applicano ancora (a meno che non li sostituisca esplicitamente). Il concetto di origine dello stile entrerà in gioco quando userai transizioni e animazioni perché introducono più origini a questo elenco. L'annotazione `!important` è un'interessante stranezza dei CSS, su cui torneremo a breve. Se le dichiarazioni contrastanti non possono essere risolte in base alla loro origine, il browser cerca quindi di risolverle osservando la loro specificità. Comprendere la specificità è essenziale. Puoi fare molta strada senza comprendere l'origine del foglio di stile perché il 99% degli stili sul tuo sito Web proviene dalla stessa origine. Ma se non capisci la specificità, perderai molto tempo inutilmente. Purtroppo, è un concetto che viene poco trattato.

Il browser valuta la specificità in due parti: stili applicati in linea nell'HTML e stili applicati utilizzando un selettore. Se utilizzi un attributo di stile HTML per applicare gli stili, le dichiarazioni vengono applicate solo a quell'elemento. Si tratta, in effetti, di dichiarazioni "con ambito", che sovrascrivono qualsiasi dichiarazione applicata dal foglio di stile o da un tag `<style>`. Gli stili in linea non hanno un selettore perché vengono applicati direttamente all'elemento a cui mirano. Nella tua pagina, vuoi che il link `Specials` sia in evidenza nel menu di navigazione con il colore arancione, come mostrato nella figura. Ecco diversi modi in cui puoi farlo, a cominciare dagli stili inline.



Per vederlo nel tuo browser, modifica la tua pagina in modo che corrisponda al codice qui fornito:

```
<li>
```

```
<a href="/specials" class="featured" style="background-color:
orange;">
  Specials
</a>
</li>
```

Per sovrascrivere le dichiarazioni inline nel tuo foglio di stile, dovrai aggiungere un `!important` alla dichiarazione, spostandolo in un'origine con priorità più alta. Se gli stili in linea sono contrassegnati come importanti, nulla può sovrascriverli. È preferibile farlo dall'interno del foglio di stile. Annulla questa modifica ed esaminiamo approcci migliori. La seconda parte della specificità è determinata dai selettori. Ad esempio, un selettore con due nomi di classe ha una specificità maggiore rispetto a un selettore con uno solo. Se una dichiarazione imposta uno sfondo arancione, ma un'altra con una specificità (maggiore) lo imposta su verde acqua, il browser applicherà il colore verde acqua. Per verificare, vediamo cosa succede quando proviamo a trasformare in arancione il collegamento in primo piano con un semplice selettore di classe. Aggiorna la parte finale del tuo foglio di stile in modo che corrisponda al codice qui fornito:

```
#main-nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}
.featured {
  background-color: orange;
}
```

Non funziona! Tutti i collegamenti rimangono verde acqua. Come mai? Il primo selettore qui è più specifico del secondo. È composto da un ID e un nome di tag, mentre il secondo è composto da un nome di classe. Diversi tipi di selettori hanno anche specificità diverse. Un selettore ID ha una specificità maggiore rispetto a un selettore di classe, ad esempio. In effetti, un singolo ID ha una specificità maggiore rispetto a un selettore con un numero qualsiasi di classi. Allo stesso modo, un selettore di classe ha una specificità maggiore rispetto a un selettore di tag (chiamato anche selettore di tipo). Le regole esatte di specificità sono:

- Se un selettore ha più ID, vince (cioè è più specifico)
- In caso di parità, vince il selezionatore con il maggior numero di classi.
- In caso di parità, vince il selezionatore con il maggior numero di nomi di tag.

Considera i selettori mostrati nell'elenco seguente (ma non aggiungerli alla tua pagina). Questi sono scritti in ordine di specificità crescente.

```
html body header h1 {  
  color: blue;  
}  
body header.page-header h1 {  
  color: orange;  
}  
.page-header .title {  
  color: green;  
}  
#page-title {  
  color: red;  
}
```

Il selettore più specifico qui è `page-title`, con un ID, quindi la sua dichiarazione con colore rosso sarà applicata al titolo. Il prossimo specifico è `.page-header .title`, con due nomi di classe. Ciò si applicherebbe se il selettore ID fosse assente. Il selettore `.page-header .title` ha una specificità maggiore del selettore `body header.page-header h1`, nonostante la sua lunghezza: due classi sono più specifiche di una classe. Infine, `html body header h1` è il meno specifico, con quattro tipi di elementi (ovvero i nomi dei tag) ma senza ID o classi. I selettori di pseudo-classi (ad esempio, `:hover`) e i selettori di attributi (ad esempio, `[type="input"]`) hanno ciascuno la stessa specificità di un selettore di classe. Il selettore universale (`*`) e i combinatori (`>`, `+`, `~`) non hanno alcun effetto sulla specificità.

Se aggiungi una dichiarazione al tuo CSS e sembra non avere alcun effetto, spesso è perché una regola più specifica la sovrascrive. Molte volte gli sviluppatori scrivono selettori utilizzando gli ID, senza rendersi conto che ciò crea una specificità più elevata, difficile da ignorare in seguito. Se devi sovrascrivere uno stile applicato utilizzando un ID, devi utilizzare un altro ID. È un concetto semplice, ma se non capisci la specificità, puoi

impazzire cercando di capire perché una regola funziona e un'altra no. Un modo comune per indicare la specificità è in forma numerica, spesso con virgole tra ogni numero. Ad esempio, "1,2,2" indica una specificità di un ID, due classi e due tag. Gli ID con la priorità più alta vengono elencati per primi, seguiti dalle classi, quindi dai tag. Il selettore `#page-header #page-title` ha due ID, nessuna classe e nessun tag. Possiamo dire che questo ha una specificità di 2,0,0. Il selettore `ul li`, con due tag ma senza ID o classi, ha una specificità di 0,0,2. La tabella mostra i selettori della lista:

Selector	IDs	Classes	Tags	Notation
<code>html body header h1</code>	0	0	4	0,0,4
<code>body header.page-header h1</code>	0	1	3	0,1,3
<code>.page-header .title</code>	0	2	0	0,2,0
<code>#page-title</code>	1	0	0	1,0,0

Ora diventa una questione di confrontare i numeri per determinare quale selettore è più specifico. Una specificità di 1,0,0 ha la precedenza su una specificità di 0,2,2 e anche su 0,10,0 (anche se non consiglio di scrivere selettori lunghi quanto uno con 10 classi), perché il primo numero (ID) ha la priorità più alta. Occasionalmente, si usa usano una notazione a quattro numeri con uno 0 o 1 nella cifra più significativa per rappresentare se una dichiarazione viene applicata tramite stili inline. In questo caso, uno stile inline ha una specificità di 1,0,0,0. Ciò sovrascriverebbe gli stili applicati tramite selettori, che potrebbero essere indicati come aventi specificità di 0,1,2,0 (un ID e due classi) o qualcosa di simile.

Torniamo alla pratica, quando hai provato ad applicare lo sfondo arancione utilizzando il selettore `.featured`, non ha funzionato. Il selettore `#main-nav` a ha un ID che sovrascrive il selettore di classe (specificità 1,0,1 e 0,1,0). Per correggere questo problema, hai alcune opzioni da considerare. La soluzione più rapida consiste nell'aggiungere un `!important` alla dichiarazione che si desidera favorire. Modificare la dichiarazione in modo che corrisponda a quella fornita qui:

```
#main-nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 5px;  
  border-radius: 2px;  
  text-decoration: none;
```



```
}  
.featured {  
background-color: orange !important;  
}
```

Funziona perché l'annotazione !important eleva la dichiarazione a un'origine con priorità più alta. Certo, è facile, ma è anche una soluzione ingenua. Potrebbe essere adatta ora, ma può causare problemi lungo la strada. Se inizi ad aggiungere !important a più dichiarazioni, cosa succede quando devi dare priorità a qualcosa già impostato come important? Quando assegni a più dichiarazioni un !important, le origini corrispondono e si applicano le regole di specificità regolari. Questo alla fine ti lascerà al punto in cui hai iniziato; una volta introdotto un !important, è probabile che ne seguiranno altri. Troviamo un modo migliore. Invece di cercare di aggirare le regole della specificità del selettore, proviamo a farle funzionare. E se aumentassi la specificità del tuo selettore? Aggiorna i set di regole nel tuo CSS in modo che corrispondano a questo elenco:

```
#main-nav a {  
color: white;  
background-color: #13a4a4;  
padding: 5px;  
border-radius: 2px;  
text-decoration: none;  
}  
#main-nav .featured {  
background-color: orange;  
}
```

Anche questa correzione funziona. Ora, il tuo selettore ha un ID e una classe, dandogli una specificità di 1,1,0, che è maggiore di #main-nav a (che ha una specificità di 1,0,1), quindi viene applicato il colore di sfondo arancione all'elemento. Puoi ancora renderlo migliore, però. Invece di aumentare la specificità del secondo selettore, vediamo se possiamo abbassare la specificità del primo. L'elemento ha anche una classe: <ul id="main-nav" class="nav">, quindi puoi cambiare il tuo CSS per indirizzare l'elemento in base al nome della sua classe anziché al suo ID. Cambia #main-nav in .nav nei tuoi selettori come mostrato qui:

```
.nav {  
margin-top: 10px;
```

```
list-style: none;
padding-left: 0;
}
.nav li {
display: inline-block;
}
.nav a {
color: white;
background-color: #13a4a4;
padding: 5px;
border-radius: 2px;
text-decoration: none;
}
.nav .featured {
background-color: orange;
}
```

Hai abbassato la specificità dei selettori. Come puoi vedere da questi esempi, la specificità tende a diventare una sorta di corsa alle armi, questo è particolarmente vero per i grandi progetti. In genere è meglio mantenere bassa la specificità quando puoi, quindi quando devi sovrascrivere qualcosa.

Il terzo e ultimo passaggio per risolvere la cascata è l'ordine di origine. Se l'origine e la specificità sono le stesse, la dichiarazione che appare più avanti nel foglio di stile, o appare in un foglio di stile incluso più avanti nella pagina, ha la precedenza. Ciò significa che puoi manipolare l'ordine di origine per definire lo stile del tuo link in primo piano. Se rendi uguali nella specificità i due selettori in conflitto, vince l'ultimo che appare. Consideriamo la quarta opzione mostrata nell'elenco seguente:

```
.nav a {
color: white;
background-color: #13a4a4;
padding: 5px;
border-radius: 2px;
text-decoration: none;
}
a.featured {
background-color: orange;
}
```

In questa soluzione, le specificità sono uguali. L'ordine di origine determina quale dichiarazione viene applicata al tuo link, risultando in un pulsante arancione in primo piano. Questo risolve il tuo problema ma, potenzialmente, ne introduce anche uno nuovo: sebbene un pulsante in primo piano all'interno del nav sembri corretto, cosa succede se vuoi usare la classe featured su un altro link altrove nella pagina, al di fuori del tuo nav? Otterrai una strana combinazione di stili: lo sfondo arancione, ma non il colore del testo, il riempimento o il raggio del bordo dei link di navigazione.



Il codice seguente mostra il markup che crea questo comportamento. Ora c'è un elemento preso di mira solo dal secondo selettore, ma non dal primo, che produce un risultato indesiderato. Dovrai decidere se vuoi che questo stile di pulsante arancione funzioni al di fuori del nav e, in tal caso, dovrai assicurarti che tutti gli stili desiderati si applichino anche ad esso.

```
<header class="page-header">
  <h1 id="page-title" class="title">Wombat Coffee Roasters</h1>
  <nav>
    <ul id="main-nav" class="nav">
      <li><a href="/">Home</a></li>
      <li><a href="/coffees">Coffees</a></li>
      <li><a href="/brewers">Brewers</a></li>
      <li><a href="/specials" class="featured">Specials</a></li>
    </ul>
  </nav>
</header>
<main>
  <p>
    Be sure to check out
    <a href="/specials" class="featured">our specials</a>.
  </p>
```

</main>

Idealmente sul tuo sito web, sarai in grado di fare ipotesi plausibili, forse sai che avrai bisogno di un link featured in altri posti. Molto spesso nei CSS, come ho detto prima, la risposta migliore è "dipende". Ci sono molti percorsi per raggiungere lo stesso risultato finale. Vale la pena considerare diverse opzioni e pensare alle ramificazioni di ciascuna. Quando affronto un problema di stile, lo affronto spesso in due fasi: in primo luogo capisco quali dichiarazioni lo faranno sembrare giusto. In secondo luogo, penso ai possibili modi per strutturare i selettori e scelgo quello che meglio si adatta alle esigenze.

Quando hai iniziato a studiare CSS, potresti aver appreso che i tuoi selettori per i collegamenti di stile dovrebbero essere scritti in un certo ordine. Questo perché l'ordine di origine influisce sulla cascata. Questo elenco mostra gli stili per i collegamenti su una pagina nell'ordine "corretto".

```
a:link {  
  color: blue;  
  text-decoration: none;  
}  
a:visited {  
  color: purple;  
}  
a:hover {  
  text-decoration: underline;  
}  
a:active {  
  color: red;  
}
```

La cascata è la ragione per cui questo ordine è importante: data la stessa specificità, gli stili successivi prevalgono sugli stili precedenti. Se due o più di questi stati sono veri per un elemento contemporaneamente, l'ultimo può sovrascrivere gli altri. Se l'utente passa il mouse su un link visitato, gli stili al passaggio del mouse hanno la precedenza. Se l'utente attiva il link (ovvero, fa clic su di esso) mentre ci passa sopra, gli stili attivi hanno la precedenza. Un utile mnemonico per ricordare questo ordine è LoVe/HAtE: link, visit, hover, active. Nota che se modifichi uno dei selettori in modo

che abbia una specificità diversa rispetto agli altri, potresti ottenere risultati imprevisti.

Il browser segue questi tre passaggi: origine, specificità e ordine di origine per risolvere ogni proprietà per ogni elemento della pagina. Una dichiarazione che "vince" la cascata è chiamata valore in cascata. C'è al massimo un valore a cascata per proprietà per elemento. Un particolare paragrafo (<p>) sulla pagina può avere un margine superiore e un margine inferiore, ma non può avere due margini superiori diversi o due margini inferiori diversi. Se il CSS specifica valori diversi per una proprietà, la cascata ne sceglierà solo uno durante il rendering dell'elemento. Questo è il valore in cascata. Se una proprietà non viene mai specificata per un elemento, non ha alcun valore in cascata per tale proprietà. Lo stesso paragrafo, ad esempio, potrebbe non avere un bordo o un riempimento specificato.

Come forse saprai, ci sono due regole pratiche comuni per lavorare con la cascata. Poiché possono essere utili, ecco un promemoria:

1. Non utilizzare gli ID nel selettore. Anche un solo ID aumenta molto la specificità. Quando è necessario sovrascrivere il selettore, spesso non si dispone di un altro ID significativo da utilizzare; quindi, si finisce per dover copiare il selettore originale e aggiungere un'altra classe per distinguerla da quella che si sta tentando di sovrascrivere.
2. Non usare !important. Questo è ancora più difficile da ignorare rispetto a un ID e, una volta utilizzato, dovrai aggiungerlo ogni volta che desideri ignorare la dichiarazione originale e quindi devi fare di nuovo i conti con la specificità.

Queste due regole possono essere un buon consiglio, ma non ti aggrappare per sempre ad esse. Ci sono eccezioni in cui possono andare bene, ma non usarle mai in una reazione istintiva per vincere una battaglia di specificità. Negli ultimi anni è emersa una serie di metodologie pratiche per aiutare a gestire la specificità del selettore. Ma ora che hai chiaro come si comporta la cascata, possiamo andare avanti.

Una nota importante sull'importanza: se stai creando un modulo JavaScript per la distribuzione (come un pacchetto NPM), ti consiglio vivamente di non applicare stili in linea tramite JavaScript se può essere

evitato. Se lo fai, stai costringendo gli sviluppatori che usano il tuo pacchetto ad accettare esattamente i tuoi stili o ad usare `!important` per ogni proprietà che vogliono cambiare. Invece, includi un foglio di stile nel tuo pacchetto. Se il tuo componente ha bisogno di apportare modifiche allo stile in modo dinamico, è quasi sempre preferibile utilizzare JavaScript per aggiungere e rimuovere classi agli elementi. Quindi gli utenti possono utilizzare il tuo foglio di stile e hanno la possibilità di modificarlo come preferiscono senza combattere la specificità.

Ereditarietà

C'è un ultimo modo in cui un elemento può ricevere stili: l'ereditarietà. La cascata è spesso confusa con il concetto di ereditarietà. Sebbene i due argomenti siano correlati, dovresti capirli individualmente. Se un elemento non ha un valore in cascata per una determinata proprietà, può ereditarne uno da un elemento antenato. È comune applicare una famiglia di caratteri all'elemento `<body>`. Tutti gli elementi predecessori all'interno ereditano quindi questo carattere; non è necessario applicarlo esplicitamente a ciascun elemento della pagina. Tuttavia, non tutte le proprietà vengono ereditate, per impostazione predefinita, solo alcune sono ereditate. In generale, queste sono le proprietà che vorresti ereditare e sono principalmente proprietà relative al testo: `color`, `font`, `font-family`, `font-size`, `font-weight`, `font-variant`, `font-style`, `line-height`, `letter-spacing`, `text-align`, `text-indent`, `text-transform`, `white-space` e `word-spacing`. Anche altre vengono ereditate, come le proprietà della lista: `list-style`, `list-style-type`, `list-style-position` e `list-style-image`. Vengono ereditate anche le proprietà del bordo della tabella, `border-collapse` e `border-spacing`; nota che queste controllano il comportamento dei bordi delle tabelle, non le proprietà più comunemente utilizzate per specificare i bordi per elementi non di tabella. Questo non è un elenco completo, ma è abbastanza esaustivo. Puoi usare l'ereditarietà a tuo favore sulla tua pagina applicando un font all'elemento `body`, permettendo ai suoi elementi discendenti di ereditare quel valore. Aggiungi questo codice nella parte superiore del tuo foglio di stile per applicare questo principio alla tua pagina:

```
body {  
  font-family: sans-serif;  
}
```

Questo viene applicato all'intera pagina aggiungendolo al `body`. Ma puoi anche scegliere come target un elemento specifico nella pagina, l'ereditarietà passerà da un elemento all'altro finché non viene sovrascritta da un valore in cascata.

Un complicato nido di valori che si ereditano e si scavalcano a vicenda può diventare rapidamente difficile da gestire. Se non hai già familiarità con gli strumenti di sviluppo del tuo browser, inizia ad usarli. DevTools fornisce visibilità esattamente su quali regole si applicano a quali elementi e perché.

La cascata e l'ereditarietà sono concetti astratti; DevTools è il modo migliore che conosco per orientarmi. Facendo clic con il pulsante destro del mouse su un elemento e scegliendo "Ispeziona" o "Ispeziona elemento" dal menu di scelta rapida, avrai una panoramica completa. Verrà mostrato ogni selettore che punta all'elemento ispezionato, ordinato per specificità e sotto ci saranno tutte le proprietà ereditate. Questo mostra a colpo d'occhio tutta la cascata e l'ereditarietà per l'elemento. Gli stili più vicini alla parte superiore hanno la precedenza su quelli sotto e gli stili sovrascritti sono barrati. Il foglio di stile e il numero di riga per ogni set di regole sono mostrati a destra, quindi puoi trovarli nel tuo codice sorgente. Questo ti dice esattamente quale elemento ha ereditato quali stili e da dove hanno avuto origine. Puoi anche digitare qualcosa nella casella "Filtro" per nascondere tutto tranne un determinato insieme di dichiarazioni.

Valori speciali

Ci sono due valori speciali che puoi applicare a qualsiasi proprietà per aiutare a manipolare la cascata: `inherit` e `initial`. Diamo un'occhiata a questi. A volte, vorrai forzare l'ereditarietà quando un valore a cascata la impedisce. Per fare ciò, puoi utilizzare la parola chiave `inherit`. Puoi sovrascrivere un altro valore con questo e farà sì che l'elemento erediti quel valore dal suo genitore. Supponi di aggiungere un piè di pagina grigio chiaro alla tua pagina. Nel footer potrebbero esserci dei collegamenti, ma non vuoi che risaltino troppo perché il piè di pagina non è una parte importante della pagina. Quindi renderai i collegamenti nel piè di pagina in grigio scuro. Aggiungi questo markup alla fine della tua pagina:

```
<footer class="footer">
&copy; 2022 Wombat Coffee Roasters &mdash;
<a href="/terms-of-use">Terms of use</a>
</footer>
```

In genere, avrai un set di colori del carattere per tutti i collegamenti sulla pagina (e in caso contrario, ne impostano uno gli stili dello user agent) e quel colore viene applicato anche al link per i termini di utilizzo. Per rendere grigio il collegamento nel piè di pagina, dovrai sovrascriverlo. Aggiungi questo codice al tuo foglio di stile per farlo:

```
a:link {
color: blue;
}
...
.footer {
color: #666;
background-color: #ccc;
padding: 15px 0;
text-align: center;
font-size: 14px;
}
.footer a {
color: inherit;
text-decoration: underline;
}
```

Il terzo set di regole qui sovrascrive il colore del link blu, dando al link nel footer un valore a cascata `inherit`. Pertanto, eredita il colore dal suo genitore, `<footer>`. Il vantaggio qui è che il collegamento del piè di pagina cambierà insieme al resto del piè di pagina se qualcosa lo altera. Se, ad esempio, il testo del piè di pagina su alcune pagine è di un grigio più scuro, il collegamento cambierà in modo pertinente. Puoi anche utilizzare la parola chiave `inherit` per forzare l'ereditarietà di una proprietà normalmente non ereditata, come il bordo o il riempimento. Ci sono pochi usi pratici, ma vedrai un caso utile nel capitolo 3 quando esamineremo il box model.

A volte scoprirai di avere un elemento con stili che desideri rimuovere o annullare. Puoi farlo specificando la parola chiave `initial`. Ogni proprietà CSS ha un valore `initial` o `inherit`. Se si assegna il valore `initial` a tale proprietà, viene effettivamente ripristinato il valore predefinito, è come un hard reset di quel valore. Nota bene: la parola chiave `initial` non è supportata in nessuna versione di Internet Explorer o Opera Mini. Funziona in tutti gli altri principali browser, incluso Edge, il successore di Microsoft di IE11.

Poiché il nero è il valore iniziale per la proprietà `color` nella maggior parte dei browser, `color: initial` è equivalente a `color:black`, ecco il codice CSS:

```
.footer a { color: initial; text-decoration: underline; }
```

Il vantaggio di questo è che non devi pensarci molto. Se vuoi rimuovere un bordo da un elemento, imposta `border: initial`. Se vuoi ripristinare un elemento alla sua larghezza predefinita, imposta `width: initial`. Potresti avere l'abitudine di utilizzare il valore `auto` per eseguire questo tipo di "reset". In effetti, puoi usare `width: auto` per ottenere lo stesso risultato. Questo perché il valore predefinito di larghezza è `auto`. È importante notare, tuttavia, che `auto` non è il valore predefinito per tutte le proprietà. Non è nemmeno valido per molte proprietà; ad esempio, `border-width: auto` e `padding: auto` non sono validi e quindi non ha alcun effetto. Potresti prenderti del tempo per scovare il valore iniziale per queste proprietà, ma spesso è più facile usare `initial`.

Proprietà abbreviate

Le proprietà abbreviate sono proprietà che consentono di impostare i valori di diverse altre proprietà contemporaneamente. Ad esempio, `font` è una proprietà abbreviata che consente di impostare diverse proprietà dei caratteri. Questa dichiarazione specifica lo stile del carattere, il peso del carattere, la dimensione del carattere, l'altezza della linea e la famiglia dei caratteri:

`font: italic bold 18px/1.2 "Helvetica", "Arial", sans-serif;`

Allo stesso modo:

- `background` è una proprietà abbreviata per più proprietà di sfondo: `background-color`, `background-image`, `background-size`, `background-repeat`, `background-position`, `background-origin`, `background-clip` e `background-attachment`
- `border` è una scorciatoia per `border-width`, `border-style` e `border-color` che a loro volta sono anche proprietà abbreviate.
- `border-width` è un'abbreviazione per le larghezze del bordo `top`, `right`, `bottom` e `left`.

Le proprietà abbreviate sono utili per mantenere il codice conciso e chiaro, ma ci sono alcune stranezze che non sono immediatamente evidenti. La maggior parte delle proprietà abbreviate ti consente di omettere determinati valori e specificare solo ciò che ti interessa. È importante sapere, tuttavia, che in questo modo si impostano comunque i valori omessi; verranno impostati implicitamente al loro valore iniziale. Se, ad esempio, dovessi utilizzare la proprietà `font` per il titolo della pagina senza specificare `font-weight`, sarebbe comunque impostato un `font-weight:normal`. Aggiungi il codice da questo elenco al tuo foglio di stile per vedere come funziona:

```
h1 { font-weight: bold; }
```

```
.title { font: 32px Helvetica, Arial, sans-serif; }
```

A prima vista, può sembrare che `<h1 class="title">` sia un'intestazione in grassetto, ma non è così. Questi stili sono equivalenti a questo codice:

```
h1 {  
  font-weight: bold;
```

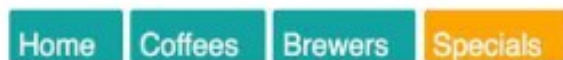
```
}  
.title {  
font-style: normal;  
font-variant: normal;  
font-weight: normal;  
font-stretch: normal;  
line-height: normal;  
font-size: 32px;  
font-family: Helvetica, Arial, sans-serif;  
}
```

Ciò significa che l'applicazione di questi stili a `<h1>` comporta uno spessore del carattere normale e non grassetto. Può anche sovrascrivere altri stili di carattere che sarebbero altrimenti ereditati da un elemento antenato. Di tutte le proprietà abbreviate, `font` è il più eclatante per causare problemi proprio perché imposta una gamma così ampia di proprietà. Per questo motivo evito di usarlo se non per impostare stili generici sull'elemento `<body>`. Puoi ancora riscontrare questo problema con altre proprietà abbreviate, quindi tieni presente questa possibilità.

Le proprietà abbreviate cercano di essere indulgenti quando si tratta dell'ordine dei valori specificati. Puoi impostare `border: 1px solid red` o `border: red 1px solid` ed entrambi funzioneranno a dovere. Questo perché è chiaro al browser quale valore specifica la larghezza, quale specifica il colore e quale specifica lo stile del bordo. Ma ci sono molte proprietà in cui i valori possono essere più ambigui. In questi casi, l'ordine dei valori è significativo. È importante comprendere questo ordine per le proprietà abbreviate che utilizzi.

In particolare, gli sviluppatori devono stare attenti quando si tratta di proprietà come `margin` e `padding`, o alcune delle proprietà del bordo che specificano i valori per ciascuno dei quattro lati di un elemento. Per queste proprietà, i valori sono in senso orario, iniziando dall'alto. Ricorda questo ordine per risparmiare tempo ed evitare problemi. In effetti, la parola inglese `TRouBL` è un mnemonico che puoi usare per ricordare l'ordine: in alto, a destra, in basso, a sinistra. Puoi usare questo mnemonico per impostare il `padding` sui quattro lati di un elemento. I collegamenti mostrati nell'immagine seguente hanno un `padding` superiore di 10 px, un riempimento a destra di 15 px, un riempimento in basso di 0 e un

riempimento a sinistra di 5 px. Questo sembra irregolare, ma illustra il principio.



Questo elenco mostra il CSS per questi collegamenti:

```
.nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 10px 15px 0 5px;  
  border-radius: 2px;  
  text-decoration: none;  
}
```

Le proprietà i cui valori seguono questo modello supportano anche le notazioni troncate. Se la dichiarazione termina prima che a uno dei quattro lati venga assegnato un valore, quel lato prende il suo valore dal lato opposto. Specificando tre valori, il lato sinistro e destro utilizzeranno entrambi il secondo valore specificato. Specificando due valori, la parte superiore e quella inferiore utilizzeranno il primo. Specificando un solo valore, verrà applicato a tutti e quattro i lati. Pertanto, le seguenti dichiarazioni sono tutte equivalenti:

```
padding: 1em 2em;  
padding: 1em 2em 1em;  
padding: 1em 2em 1em 2em;  
Anche questi sono equivalenti tra loro:  
padding: 1em;  
padding: 1em 1em;  
padding: 1em 1em 1em;  
padding: 1em 1em 1em 1em;
```

Per molti sviluppatori, il più problematico di questi è quando specificati solo tre valori. Ricorda, questo specifica la parte superiore, destra e inferiore. Poiché non viene fornito alcun valore a sinistra, assumerà lo stesso valore di destra; il secondo valore verrà applicato a entrambi i lati sinistro e destro. Pertanto, il `padding: 10px 15px 0` applica un riempimento di 15 px a entrambi i lati sinistro e destro, mentre il riempimento in alto è

10 px e il riempimento in basso è 0. Molto spesso, tuttavia, avrai bisogno di due valori. In particolare, sugli elementi più piccoli, spesso è meglio avere più padding ai lati che in alto e in basso. Questo approccio è adatto ai pulsanti o, nella tua pagina, ai link di navigazione:



Utilizza le proprietà per applicare prima il riempimento verticale, quindi quello orizzontale:

```
.nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 5px 15px;  
  border-radius: 2px;  
  text-decoration: none;  
}
```

Poiché così tante proprietà comuni seguono questo schema, vale la pena memorizzare questo ordine. Il mnemonico TRouBLe si applica solo alle proprietà che si applicano individualmente a tutti e quattro i lati dell'elemento. Altre proprietà supportano un massimo di due valori, parliamo di proprietà come background-position, box-shadow e text-shadow (sebbene queste non siano proprietà abbreviate, in senso stretto). Rispetto alle proprietà a quattro valori come padding, l'ordine di questi valori è invertito. Considerando che il padding: 1em 2em specifica prima i valori verticale superiore/inferiore, seguiti dai valori orizzontali destra/sinistra, background-position: 25% 75% specifica prima i valori orizzontali destra/sinistra, seguiti dai valori verticali superiore/inferiore. La ragione di ciò è semplice: i due valori rappresentano una griglia cartesiana. Le misurazioni della griglia cartesiana sono generalmente fornite nell'ordine x, y (orizzontale e poi verticale). Se, ad esempio, si desidera applicare un'ombra, bisogna specificare prima il valore x (orizzontale):

```
.nav .featured {  
  background-color: orange;  
  box-shadow: 10px 2px #6f9090;  
}
```

Il primo valore (maggiore) si applica all'offset orizzontale, mentre il secondo valore (minore) si applica a quello verticale. Se stai lavorando con una proprietà che specifica due misure a partire da un angolo, pensa all'asse cartesiano. Se stai lavorando con uno che specifica le misure per ciascun lato tutto intorno a un elemento, pensa ad un orologio.

UNITÀ RELATIVE

Quando si tratta di specificare i valori, CSS offre un'ampia gamma di opzioni tra cui scegliere. Uno dei più familiari e probabilmente più semplice con cui lavorare è il pixel. Si tratta, infatti, di unità assolute cioè 5 px significano hanno sempre lo stesso significato. Altre unità, come em e rem, non sono assolute, ma relative. Il valore delle unità relative cambia, in base a fattori esterni; ad esempio, il significato di 2 em cambia a seconda dell'elemento (e talvolta anche della proprietà) su cui lo stai utilizzando. Naturalmente, questo rende più difficile lavorare con le unità relative. Gli sviluppatori, anche sviluppatori esperti in CSS, spesso non amano lavorare con unità relative, incluse le famigerate em. Il modo in cui il valore di un em può cambiare lo fa sembrare imprevedibile e meno nitido del pixel. In questo capitolo farò luce sul mistero che circonda le unità relative. Innanzitutto, spiegherò il valore unico che apportano ai CSS, quindi ti aiuterò a dargli un senso. Spiegherò come funzionano e ti mostrerò come domare la loro natura apparentemente imprevedibile. Puoi fare in modo che i valori relativi, se usati correttamente, rendano il tuo codice più semplice, più versatile e più facile.

Il potere dei valori relativi

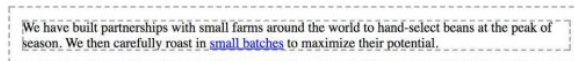
CSS usa un approccio lazy nei confronti degli stili nella pagina web: il contenuto e i suoi stili non vengono messi insieme fino al termine della creazione di entrambi. Ciò aggiunge un livello di complessità al processo di progettazione che non esiste in altri tipi di progettazione grafica, ma fornisce anche più potenza: un foglio di stile può essere applicato a centinaia, persino migliaia, di pagine. Inoltre, il rendering finale della pagina può essere modificato direttamente dall'utente, il quale, ad esempio, può modificare la dimensione del carattere di default o ridimensionare la finestra del browser. All'inizio dello sviluppo di applicazioni per computer (così come nell'editoria tradizionale), gli sviluppatori (o gli editori) conoscevano i limiti esatti del loro mezzo. Una particolare finestra del programma potrebbe essere larga 400 px e alta 300 px, oppure una pagina potrebbe essere larga 20 centimetri e alta 40 centimetri. Di conseguenza, quando gli sviluppatori hanno deciso di disporre i pulsanti e il testo dell'applicazione, sapevano esattamente quanto potevano essere grandi quegli elementi e quanto spazio avrebbero avuto per lavorare con altri elementi sullo schermo. Sul web non è così.

Nell'ambiente web, l'utente può avere la finestra del browser con diverse dimensioni e il CSS deve applicarsi ad essa. Inoltre, gli utenti possono ridimensionare la pagina dopo che è stata aperta e il CSS deve adattarsi a nuovi vincoli. Ciò significa che gli stili non possono essere applicati quando crei la tua pagina; il browser deve calcolarli quando la pagina viene visualizzata sullo schermo. Questo aggiunge un livello di astrazione ai CSS. Non possiamo modellare un elemento secondo un contesto ideale; dobbiamo specificare regole che funzioneranno in qualsiasi contesto in cui tale elemento potrebbe essere posizionato. Oggi con il Web, la tua pagina potrà essere visualizzata su un dispositivo con display da 4 pollici, così come su un monitor da 30 pollici o una TV da 55 pollici. Per molto tempo, i designer hanno mitigato questa complessità concentrandosi su design "perfetti al pixel". Si trattava di creare un contenitore ben definito, spesso una colonna centrata di circa 800 px di larghezza quindi, con questi vincoli, iniziavano a progettare più o meno come facevano i loro predecessori con applicazioni native o pubblicazioni cartacee.

Man mano che la tecnologia è migliorata e i produttori hanno introdotto monitor a risoluzione più elevata, l'approccio "pixel-perfect" ha iniziato lentamente a fallire. All'inizio degli anni 2000, si discuteva molto sul fatto che noi sviluppatori potessimo progettare in sicurezza per schermi di 1.024 px di larghezza invece di 800 px di larghezza. Ci siamo davanti ad una decisione da prendere. Era meglio rendere il nostro sito troppo ampio per i computer più vecchi o troppo stretto per quelli nuovi? Quando sono emersi gli smartphone, gli sviluppatori sono stati costretti a smettere di fingere che tutti potessero avere la stessa esperienza sui loro siti. Indipendentemente dal fatto che lo amassimo o lo odiassimo, abbiamo dovuto abbandonare le colonne con numero fisso di pixel e iniziare a pensare al design reattivo. Non potevamo più nasconderci dall'astrazione che deriva dai CSS, l'astrazione aggiuntiva significa ulteriore complessità. Se imposto ad un elemento una larghezza di 800 px, come apparirà in una finestra più piccola? Come apparirà un menu orizzontale se non si adatta interamente ad una riga? Mentre scrivi il tuo CSS, devi essere in grado di pensare contemporaneamente in termini specifici, oltre che in generale. Quando hai più modi per risolvere un problema particolare, dovrai favorire la soluzione che funziona in circostanze multiple e diverse. Le unità relative sono uno degli strumenti forniti dai CSS per lavorare con questo livello di astrazione. Invece di impostare una dimensione del carattere a 14 px, puoi far in modo che venga ridimensionata proporzionalmente alla dimensione della finestra. In alternativa, puoi impostare la dimensione di tutto il resto sulla pagina rispetto alla dimensione del carattere di base, quindi ridimensionare l'intera pagina con una singola riga di codice. Diamo un'occhiata a ciò che fornisce CSS per rendere possibile questo tipo di approccio.

Em e rem

Em, l'unità di lunghezza relativa più comune, è una misura utilizzata in tipografia, in riferimento a una dimensione del carattere specificata. In CSS, 1 em indica la dimensione del carattere dell'elemento corrente; il suo valore esatto varia a seconda dell'elemento a cui lo stai applicando. L'immagine seguente mostra un div con 1 em di riempimento:



Il codice per produrlo è mostrato nello snippet seguente. Il set di regole specifica una dimensione del carattere di 16 px, che diventa la definizione locale dell'elemento per 1 em. Quindi il codice usa ems per specificare il padding dell'elemento. Aggiungi questo codice ad un nuovo foglio di stile e inserisci del testo in un `<div class="padded">` per vedere l'effetto nel tuo browser:

```
.padded { font-size: 16px; padding: 1em; }
```

Questo padding ha un valore specifico di 1em, viene moltiplicato per la dimensione del carattere, producendo un padding renderizzato di 16 px. Questa è la parte importante: i valori dichiarati utilizzando le unità relative vengono valutati dal browser in base a un valore assoluto, chiamato valore calcolato. In questo esempio, modificando il padding a 2 em produrrebbe un valore calcolato di 32 px. Se un altro selettore ha come target lo stesso elemento e lo sovrascrive con una dimensione del carattere diversa, cambierà il valore locale di em e il riempimento calcolato cambierà per riflettere la modifica. L'uso di ems può essere conveniente quando si impostano proprietà come padding, height, width o border-radius perché queste si ridimensioneranno in modo uniforme con l'elemento se eredita caratteri di dimensioni diverse o se l'utente modifica le impostazioni del carattere. L'immagine seguente mostra due box di dimensioni diverse. Le proprietà font size, padding e border-radius in ciascuno non sono equivalenti.



È possibile definire gli stili per questi box specificando il riempimento e il raggio del bordo utilizzando l'unità di misura em. Dando a ciascuno un riempimento e un raggio del bordo di 1 em, puoi specificare una dimensione del carattere diversa per ciascun elemento e le altre proprietà verranno ridimensionate insieme al font. Nel tuo HTML, crea due caselle come mostrato di seguito e aggiungi le classi box-small e box-large a ciascuna:

```
<span class="box box-small">Small</span>
```

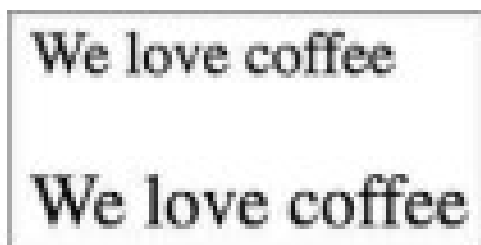
```
<span class="box box-large">Large</span>
```

Ora aggiungi gli stili mostrati al tuo foglio di stile, questo definisce un box usando em. Definisce due classi, ognuna delle quali specifica una diversa dimensione del font:

```
.box {  
padding: 1em;  
border-radius: 1em;  
background-color: lightgray;  
}  
.box-small {  
font-size: 12px;  
}  
.box-large {  
font-size: 18px;  
}
```

Quella appena usata è una potente caratteristica degli em infatti, puoi definire la dimensione di un elemento e quindi ridimensionare il tutto in alto o in basso con un'unica dichiarazione che cambia la dimensione del carattere. Tra poco costruirai un altro esempio simile, ma prima parliamo di em e dimensioni dei caratteri.

Quando si tratta della proprietà font-size, gli ems si comportano in modo leggermente diverso. Come ho detto, gli ems sono definiti dalla dimensione del carattere dell'elemento corrente. Ma se dichiarassi font-size: 1.2em, avrebbe senso? Una dimensione del carattere non può essere uguale 1,2 volte a sé stessa. Invece, le dimensioni del carattere derivano dalla dimensione del carattere ereditata. Per un esempio di base, vedere l'immagine seguente, mostra due parti di testo, ciascuna con una dimensione del carattere diversa.



Modifica la tua pagina in modo che corrisponda al seguente snippet. La prima riga di testo si trova all'interno del tag <body>, quindi verrà visualizzata alla dimensione del carattere del body. La seconda parte, lo slogan, eredita quella dimensione del carattere.

```
<body>
We love coffee
<p class="slogan">We love coffee</p>
</body>
```


Il CSS nell'elenco successivo specifica la dimensione del carattere del body. Ho usato i pixel in questo caso, per chiarezza. Successivamente, utilizzerai ems per aumentare le dimensioni dello slogan.

```
body {
font-size: 16px;
}
.slogan {
font-size: 1.2em;
}
```

La dimensione del carattere specificata per lo slogan è 1,2 em. Per determinare il valore in pixel calcolato, dovrai fare riferimento alla dimensione del carattere ereditata di 16 px: 16 per 1,2 equivale a 19,2,

quindi la dimensione del carattere calcolata è 19,2 px. Ecco un suggerimento, se conosci la dimensione del carattere basata sui pixel che desideri, ma desideri specificare la dichiarazione in ems, ecco una semplice formula: dividi la dimensione del pixel desiderata per la dimensione del pixel principale (ereditato). Ad esempio, se desideri un carattere da 10 px e il tuo elemento eredita un carattere da 12 px, $10 / 12 = 0,8333$ em. Se vuoi un carattere da 16 px e il carattere principale è 12 px, $16 / 12 = 1.3333$ em. Faremo questo calcolo più volte nel corso di questo capitolo. È utile sapere che, per la maggior parte dei browser, la dimensione del carattere predefinita è 16 px. Tecnicamente, è il valore della parola chiave medium calcolato in 16 px.

A questo punto hai definito in ems per la dimensione del carattere (basata su una dimensione del carattere ereditata) e hai definito in ems altre proprietà come padding e border-radius (in base alla dimensione del carattere dell'elemento corrente). Ciò che rende gli ems complicati è quando li usi sia per la dimensione del carattere che per qualsiasi altra proprietà sullo stesso elemento. Quando si esegue questa operazione, il browser deve prima calcolare la dimensione del carattere; quindi, utilizza quel valore per calcolare gli altri valori. Entrambe le proprietà possono avere lo stesso valore dichiarato, ma avranno valori calcolati diversi. Nell'esempio precedente, abbiamo calcolato la dimensione del carattere in 19,2 px (16 px dimensione del carattere ereditata per 1,2 em). L'immagine seguente mostra lo stesso elemento dello slogan, ma con un padding aggiuntivo di 1,2 em e uno sfondo grigio per rendere più evidente la dimensione del padding. Questo riempimento è leggermente più grande della dimensione del carattere, anche se entrambi hanno lo stesso valore dichiarato.



We love coffee

Quello che sta succedendo qui è che il paragrafo eredita una dimensione del carattere di 16 px dal body, producendo una dimensione del carattere calcolata di 19,2 px. Ciò significa che 19,2 px è ora il valore locale per un em e quel valore viene utilizzato per calcolare il padding. Aggiorna il tuo foglio di stile per vedere l'effetto nella tua pagina di test:

```
body {  
  font-size: 16px;
```

```

}
.slogan {
font-size: 1.2em;
padding: 1.2em;
background-color: #ccc;
}

```

In questo esempio, il padding ha un valore specificato di 1,2 em, moltiplicato per 19,2 px (la dimensione del carattere dell'elemento corrente) produce un valore calcolato di 23,04 px. Anche se la dimensione del carattere e il riempimento hanno lo stesso valore specificato, i loro valori calcolati sono diversi.

L'unità di misura Ems può produrre risultati inaspettati quando li usi per specificare le dimensioni dei caratteri di più elementi nidificati. Per conoscere il valore esatto di ciascun elemento, devi conoscere la dimensione del carattere ereditato, che, se definita sull'elemento padre in ems, richiede la conoscenza della dimensione ereditata dell'elemento padre e così via nell'albero. Questo aspetto diventa subito evidente quando si utilizzano gli ems per la dimensione del carattere degli elenchi puntati, annidando gli elenchi a diversi livelli di profondità. Quasi tutti gli sviluppatori web ad un certo punto della loro carriera caricano la propria pagina e vedono qualcosa di simile:

- Top level
 - Second level
 - Third level
 - Fourth level
 - Fifth level

Il testo si sta restringendo! Questo è esattamente il tipo di problema che spaventa gli sviluppatori e li fa desistere dall'uso di ems. Questo effetto si verifica quando si annidano elenchi in profondità su diversi livelli e si applica una dimensione del carattere basata su em a ciascun livello. Nel codice seguente vedrai che il selettore ha come target ogni sulla pagina; quindi quando questi elenchi ereditano la dimensione del carattere da altri elenchi, l'ems viene aggiornato.

```

body {
font-size: 16px;

```



```
}  
ul {  
font-size: .8em;  
}
```

La pagina HTML è composta da:

```
<ul>  
<li>Top level  
<ul>  
<li>Second level  
<ul>  
<li>Third level  
<ul>  
<li>Fourth level  
<ul>  
<li>Fifth level</li>  
</ul>  
</li>  
</ul>  
</li>  
</ul>  
</li>  
</ul>  
</li>  
</ul>
```

Ogni elenco ha una dimensione del carattere 0,8 volte quella del suo genitore. Ciò significa che il primo elenco ha una dimensione del carattere di 12,8 px, ma per il successivo è di 10,24 px ($12,8 \text{ px} \times 0,8$), per il terzo livello è di 8,192 px e così via. Allo stesso modo, se hai specificato una dimensione maggiore di 1 em, il testo aumenterebbe continuamente. Quello che vogliamo è specificare il carattere al livello superiore, quindi mantenere la stessa dimensione del carattere fino in fondo. Nel codice seguente andremo ad impostare la dimensione del carattere del primo elenco su .8 em come prima, mentre il secondo selettore nell'elenco punterà a tutti gli elenchi non ordinati all'interno di un elenco non ordinato, tutti tranne il livello superiore. Gli elenchi nidificati ora avranno una dimensione del font uguale ai loro genitori.

```
ul {
```

```
font-size: .8em;
}
ul ul {
font-size: 1em;
}
```

Questo risolve il problema, anche se non è la soluzione ideale; in pratica stai impostando un valore e lo stai sovrascrivendo immediatamente con un'altra regola. Sarebbe meglio evitare di sovrascrivere le regole aumentando la specificità dei selettori. A questo punto, dovrebbe essere chiaro che gli ems possono creare problemi se non stai attento. Sono utili per il padding, i margini e il dimensionamento degli elementi, ma quando si tratta di dimensioni del carattere, possono complicare il tutto. Per fortuna, c'è un'opzione migliore: rems.

Quando il browser analizza un documento HTML, crea una rappresentazione in memoria di tutti gli elementi della pagina. Questa rappresentazione è chiamata DOM (Document Object Model). È una struttura ad albero, in cui ogni elemento è rappresentato da un nodo. L'elemento `<html>` è il nodo di primo livello (o radice). Sotto di esso ci sono i suoi nodi figli, `<head>` e `<body>`. E sotto di loro ci sono i loro figli, i figli dei figli e così via. Il nodo radice è l'antenato di tutti gli altri elementi nel documento. Ha uno speciale selettore di pseudo-classi (`:root`) che puoi usare e ciò equivale a utilizzare il selettore di tipo `html` con la specificità di una classe anziché di un tag. Rem è l'abbreviazione di root em. Invece di essere relativi all'elemento corrente, i rem sono relativi all'elemento radice. Indipendentemente da dove lo applichi nel documento, 1,2 rem ha lo stesso valore calcolato: 1,2 volte la dimensione del carattere dell'elemento radice. Il codice seguente stabilisce la dimensione del carattere principale e quindi utilizza rems per definire la dimensione del carattere per gli elenchi non ordinati relativi.

```
:root {
font-size: 1em;
}
ul {
font-size: .8rem;
}
```

In questo esempio, la dimensione del carattere radice è quella predefinita del browser di 16 px (1em sull'elemento radice è relativa

all'impostazione predefinita del browser). Gli elenchi non ordinati hanno una dimensione del carattere specificata di 0,8 rem, che è calcolata pari a 12,8 px. Poiché relativa alla radice, la dimensione del carattere rimarrà costante, anche se tu annidassi gli elenchi. I rems semplificano molte delle complessità legate agli ems, in effetti, offrono una buona via di mezzo tra pixel ed ems fornendo i vantaggi delle unità relative, ma è più semplice usarli. Questo significa che dovresti usare rem ovunque e abbandonare le altre opzioni? No. Nei CSS, ancora una volta, la risposta è spesso "dipende". I rem sono solo uno strumento nella tua borsa degli attrezzi. Una parte importante della padronanza dei CSS è imparare quando usare quale strumento. Preferisco usare rems per le dimensioni dei caratteri, pixel per i bordi ed ems per la maggior parte delle altre misure, in particolare padding, margini e raggio del bordo (sebbene preferisca l'uso di percentuali per le larghezze del box contenitore quando necessario). In questo modo, le dimensioni dei caratteri sono prevedibili, ma otterrai comunque il potere degli ems di ridimensionare il riempimento e i margini, qualora altri fattori alterassero la dimensione del font di un elemento. I pixel hanno senso per i bordi, in particolare quando vuoi una bella linea sottile. Queste sono le mie unità di riferimento per le varie proprietà, ma ancora una volta sono strumenti e, in alcune circostanze, uno strumento è più adatto di un altro e fa meglio il suo lavoro. In caso di dubbio, utilizza rems per la dimensione del carattere, pixel per i bordi ed ems per la maggior parte delle altre proprietà.

Alcuni browser forniscono all'utente due modi per personalizzare la dimensione del testo: lo zoom e una dimensione del carattere predefinita. Premendo Ctrl-più (+) o Ctrl-meno (–), l'utente può ingrandire o ridurre la pagina. Questo ridimensiona visivamente tutti i caratteri e le immagini, rendendo generalmente tutto più grande o più piccolo sulla pagina. In alcuni browser, questa modifica viene applicata solo alla scheda corrente ed è temporanea, il che significa che non viene trasferita alle nuove schede. L'impostazione di una dimensione del carattere predefinita funziona in modo leggermente diverso. Non solo è più difficile trovare dove impostarlo (di solito nella pagina delle impostazioni del browser), ma le modifiche a questo livello rimangono permanenti, fino a quando l'utente non ritorna e modifica nuovamente il valore. Il problema è che questa impostazione non ridimensiona i caratteri definiti utilizzando pixel o altre unità assolute. Poiché si tratta di una dimensione del carattere predefinita, è vitale per

alcuni utenti, in particolare per quelli con problemi di vista, pertanto, dovrete sempre specificare le dimensioni del carattere con unità o percentuali relative.

Non pensare in pixel

Un pattern, o meglio, anti-pattern, che è stato comune negli ultimi anni è quello di reimpostare la dimensione del carattere alla radice della pagina a .625 em o 62,5%.

```
html { font-size: .625em; }
```

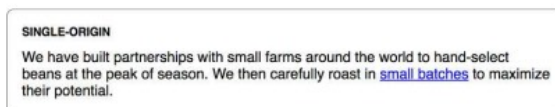
Non lo consiglio affatto. Questa dichiarazione prende la dimensione del carattere predefinita del browser, 16 px, e la ridimensiona a 10 px. Questa pratica semplifica la matematica: se il tuo cliente ti dice di rendere il carattere pari a 14 px, puoi facilmente dividere per 10 e digitare 1.4 rem, il tutto mentre usi ancora le unità relative. Inizialmente, questo può essere conveniente, ma ci sono due problemi con questo approccio. Innanzitutto, ti costringe a scrivere molti stili duplicati. Dieci pixel sono troppo piccoli per la maggior parte del testo, quindi dovrai sovrascriverlo in tutta la pagina. Ti ritroverai a impostare i paragrafi a 1.4 rem, aside a 1.4 rem, link di navigazione a 1.4 rem e così via. Ciò introduce più punti di errore, più punti da aggiornare nel codice quando deve essere modificato e aumenta le dimensioni del foglio di stile. Il secondo problema è che quando lo fai, stai ancora pensando in pixel. Potresti digitare 1.4 rem nel tuo codice, ma nella tua mente stai ancora pensando "14 pixel". Su un Web reattivo, dovresti sentirti a tuo agio con i valori "sfocati". Non importa quanti pixel 1.2 em restituisce; tutto ciò che devi sapere è che è un po' più grande della dimensione del carattere ereditata. E, se non appare come lo vuoi sullo schermo, lo cambi. Questo richiede alcuni tentativi ed errori, ma in realtà funziona anche con i pixel. Quando si lavora con ems, è facile rimanere ossessionati dall'esatta quantità di pixel valutati degli elementi, in particolare riguardo le dimensioni dei caratteri. Resterai legato a dividere e moltiplicare i valori man mano che procedi. Invece, ti sfido a prendere l'abitudine di usare prima gli ems. Se sei abituato a utilizzare i pixel, l'utilizzo dei valori em può richiedere pratica, ma ne vale la pena. Questo non vuol dire che non dovrai mai lavorare con i pixel. Se stai lavorando con un designer, probabilmente avrai bisogno di parlare di numeri di pixel concreti, e va bene. All'inizio di un progetto, dovrai stabilire una dimensione del carattere di base (e spesso alcune dimensioni comuni per intestazioni e note a piè di pagina). I valori assoluti sono più facili da usare nella fase iniziale e la conversione in rem comporterà l'uso dell'aritmetica;

quindi, tieni una calcolatrice a portata di mano. L'inserimento di una dimensione del carattere radice definisce una rem. Da quel momento in poi, lavorare in pixel dovrebbe essere l'eccezione, non la norma. Continuerò a menzionare i pixel in tutto questo capitolo e questo mi aiuterà a ribadire perché le unità relative si comportano in tal modo, oltre ad aiutarti ad abituarti al calcolo degli ems.

Diciamo che vuoi che la dimensione del carattere predefinita sia 14 px. Invece di impostare un valore predefinito di 10 px e quindi sovrascriverlo in tutta la pagina, imposta il valore alla radice. Il valore desiderato diviso il valore ereditato, in questo caso l'impostazione predefinita del browser, è 14/16, che equivale a 0,875. Aggiungi il seguente snippet all'inizio di un nuovo foglio di stile, poiché sarà la nostra nuova base da cui partire. Questo imposta il carattere predefinito alla radice (<html>):

```
:root { font-size: 0.875em; }
```

Ora la dimensione del carattere desiderata viene applicata all'intera pagina. Non sarà necessario specificarlo altrove. Dovrai solo cambiarlo nei punti in cui il design non segue queste indicazioni, come i titoli. Creiamo il pannello mostrato nell'immagine seguente, in base alla dimensione del carattere di 14 px, utilizzando misurazioni relative.



Il markup per questo risultato è il seguente:

```
<div class="panel">
```

```
<h2>Single-origin</h2>
```

```
<div class="panel-body">
```

```
We have built partnerships with small farms around the world to  
hand-select beans at the peak of season. We then carefully roast  
in <a href="/batch-size">small batches</a> to maximize their  
potential.
```

```
</div>
```

```
</div>
```

L'elenco successivo mostra gli stili. Utilizzerai ems per il riempimento e il raggio del bordo, rem per la dimensione del carattere dell'intestazione e

px per il bordo. Aggiungi questi al tuo foglio di stile.

```
.panel {  
padding: 1em;  
border-radius: 0.5em;  
border: 1px solid #999;  
}  
.panel > h2 {  
margin-top: 0;  
font-size: 0.8rem;  
font-weight: bold;  
text-transform: uppercase;  
}
```

Questo codice disegna un bordo sottile attorno al pannello e modella l'intestazione. Ho optato per un'intestazione più piccola ma in grassetto e in maiuscolo. (Puoi rendere questo carattere più grande o diverso se vuoi modificarlo.) Il carattere `>` nel secondo selettore è un combinatore discendente diretto. Mira a un `h2` che è un elemento figlio di un elemento `.panel`. Nel codice HTML appena visto, ho aggiunto la classe `panel-body` al corpo principale del pannello per chiarezza, ma noterai che non avevi bisogno di usarla nel tuo CSS poiché questo elemento eredita già la dimensione del carattere principale, infatti, appare già come vuoi che appaia.

Andiamo un po' oltre. È possibile utilizzare alcune media query per modificare la dimensione del carattere di base, a seconda delle dimensioni dello schermo. Ciò renderà il pannello di dimensioni diverse in base alle dimensioni dello schermo dell'utente.



Per vedere questo risultato, modifica questa parte del foglio di stile in modo che corrisponda a questo snippet:

```
:root {  
font-size: 0.75em;  
}
```

```
@media (min-width: 800px) {  
  :root {  
    font-size: 0.875em;  
  }  
}  
  
@media (min-width: 1200px) {  
  :root {  
    font-size: 1em;  
  }  
}
```

Questo primo set di regole specifica una piccola dimensione predefinita per il font. Questa è la dimensione del carattere che vogliamo applicare su schermi più piccoli. Per questo motivo hai utilizzato le media query per sovrascrivere quel valore con dimensioni dei caratteri sempre più grandi su schermi con una larghezza di 800 px e 1.200 px o oltre tale dimensione. Applicando queste dimensioni dei caratteri alla radice della tua pagina, hai ridefinito in modo reattivo il significato di em e rem in tutta la pagina. Ciò significa che il pannello ora è reattivo, anche se non hai apportato modifiche direttamente. Su uno schermo piccolo, come uno smartphone, il carattere sarà reso più piccolo (12 px); allo stesso modo, il riempimento e il raggio del bordo saranno più piccoli per rendere gli elementi coerenti. Su schermi più grandi, ovvero con larghezza superiore a 800 px e 1.200 px, il componente viene ridimensionato rispettivamente fino a una dimensione del carattere di 14 px e 16 px. Prova a ridimensionare la finestra del browser per vedere queste modifiche all'opera. Se sei abbastanza attento da modellare l'intera pagina con unità relative come questa, l'intera pagina verrà ridimensionata in base alle dimensioni del viewport. Questa può essere una parte fondamentale della tua strategia reattiva, infatti, le due media query nella parte superiore del tuo foglio di stile possono eliminare la necessità di dozzine di media query nel resto del tuo CSS. Tutto questo, però, non funziona se definisci i tuoi valori in pixel. Allo stesso modo, se il tuo capo o il tuo cliente decide che i caratteri sul sito che hai creato sono troppo piccoli o troppo grandi, puoi cambiarli globalmente toccando solo una riga di codice. La modifica si diffonderà in tutto il resto della pagina, senza alcuno sforzo.

Puoi anche utilizzare ems per ridimensionare un singolo componente nella pagina. A volte potresti aver bisogno di una versione più grande della

stessa parte dell'interfaccia in alcune parti della pagina. In tal caso, potresti aggiungere una classe grande al pannello: `<div class="panel large">`. L'effetto è simile ai pannelli reattivi, ma entrambe le dimensioni possono essere utilizzate contemporaneamente sulla stessa pagina. Apportiamo una piccola modifica al modo in cui hai definito le dimensioni dei caratteri del pannello. Utilizzerai comunque le unità relative, ma regolerai a cosa sono relative. Innanzitutto, aggiungi la dichiarazione `font-size: 1rem` all'elemento padre di ogni pannello. Ciò significa che ogni pannello stabilirà una dimensione del carattere prevedibile per se stesso, indipendentemente da dove è posizionato sulla pagina. In secondo luogo, ridefinisci la dimensione del carattere dell'intestazione usando l'unità `ems` anziché `rem` per renderla relativa alla dimensione del carattere del genitore che hai appena stabilito a 1 `rem`. Aggiorna il tuo foglio di stile in modo che corrisponda a questo snippet:

```
.panel {  
  font-size: 1rem;  
  padding: 1em;  
  border: 1px solid #999;  
  border-radius: 0.5em;  
}  
.panel > h2 {  
  margin-top: 0;  
  font-size: 0.8em;  
  font-weight: bold;  
  text-transform: uppercase;  
}
```

Questa modifica non ha alcun effetto sull'aspetto del pannello, ma ora ti consente di creare la versione più grande con una singola riga di CSS. Tutto quello che devi fare è sovrascrivere 1 `rem` dell'elemento genitore con un altro valore. Poiché tutte le misurazioni del componente sono relative, l'override ridimensionerà l'intero pannello. Aggiungi il CSS seguente al tuo foglio di stile per definire una versione più grande del pannello:

```
.panel.large {  
  font-size: 1.2rem;  
}
```

Ora puoi usare `class="panel"` per un pannello normale e `class="panel large"` per un pannello dalle dimensioni più grandi. Allo stesso modo, puoi

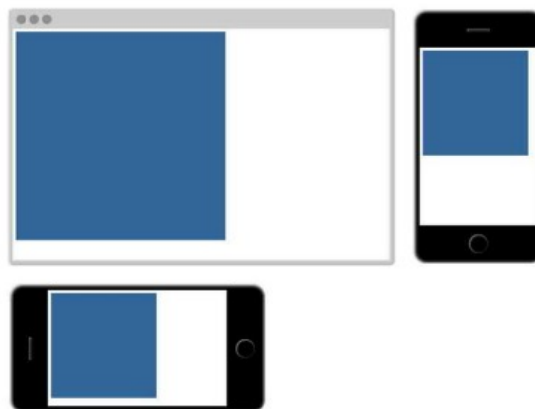
definire una versione più piccola del pannello impostando una dimensione del font minore. Se il pannello fosse un componente più complicato, con più font o padding, sarebbe utile solo questa dichiarazione per ridimensionarli, purché tutto all'interno sia definito usando ems.

Unità relative al viewport

Hai imparato che ems e rem sono definiti in relazione alla dimensione del carattere, ma questi non sono l'unico tipo di unità relative. Ci sono anche unità relative al viewport per definire le dimensioni relative al viewport del browser. Se non hai familiarità con le unità relative al viewport, ecco una breve spiegazione.

- **vh**—1/100 dell'altezza del viewport
- **vw**—1/100 della larghezza del viewport
- **vmin**—1/100 della dimensione minore, altezza o larghezza (IE9 supporta **vm** invece di **vmin**)
- **vmax**—1/100 della dimensione maggiore, altezza o larghezza (non supportate in IE e in Edge)

Ad esempio, 50 vw equivale a metà della larghezza della finestra e 25 vh equivale al 25% dell'altezza della finestra. **vmin** si basa sull'elemento più piccolo tra i due (altezza o larghezza), è utile per garantire che un elemento si adatti allo schermo indipendentemente dal suo orientamento: se lo schermo è orizzontale, sarà basato sull'altezza; se verticale, sarà basato sulla larghezza. L'immagine seguente mostra un elemento quadrato così come appare in diverse finestre con diverse dimensioni dello schermo. È definito sia con un'altezza che con una larghezza di 90 vmin, che equivale al 90% della più piccola delle due dimensioni: il 90% dell'altezza sugli schermi orizzontali o il 90% della larghezza su verticale.



Vediamo quali sono gli stili per questo elemento utili a produrre un grande quadrato che si adatta sempre alla finestra, indipendentemente dalle dimensioni del browser. Puoi aggiungere un `<div class= "square">` alla tua pagina per vederlo.

```
.square {  
width: 90vmin;  
height: 90vmin;  
background-color: #369;  
}
```

Le lunghezze relative al viewport sono ottime in alcuni casi, ad esempio, fare in modo che l'immagine di un super-eroe riempia lo schermo. L'immagine può trovarsi all'interno di un contenitore lungo, ma impostando l'altezza dell'immagine su 100 vh, diventa esattamente l'altezza del viewport. Le unità relative al viewport sono una funzionalità recente per la maggior parte dei browser, quindi potresti incontrare qualche problema usandole con altri stili. Ti consiglio di visitare il sito <https://caniuse.com/#feat=viewport-units> per maggiori informazioni.

Un'applicazione per le unità relative al viewport che potrebbero non essere immediatamente evidenti è la dimensione del carattere. In effetti, ho notato che questo uso è più pratico rispetto all'applicazione di vh e vw alle altezze o alle larghezze degli elementi. Mi spiego meglio, cosa accadrebbe se applicassi font-size: 2vw a un elemento? Su un monitor desktop a 1.200 px, questo corrisponde a 24 px (2% di 1.200). Su un tablet con una larghezza dello schermo di 768 px, restituisce circa 15 px (2% di 768). La cosa interessante è che l'elemento scala senza problemi tra le due dimensioni, ciò significa che non ci sono cambiamenti improvvisi del punto di interruzione; passa in modo incrementale al variare delle dimensioni della finestra. Sfortunatamente, 24 px sono un po' troppo grandi su uno schermo grande. Purtroppo, la dimensione diventa 7,5 px su un iPhone 6, sarebbe bello avere questo effetto di ridimensionamento, ma un po' meno estremo. Puoi ottenere questo risultato con la funzione calc() di CSS. La funzione calc() consente di eseguire operazioni aritmetiche di base con due o più valori. Ciò è particolarmente utile per combinare valori misurati in unità diverse. Questa funzione supporta addizione (+), sottrazione (-), moltiplicazione (*) e divisione (/). Gli operatori di addizione e sottrazione devono essere circondati da spazi bianchi, quindi suggerisco di aggiungere sempre uno spazio prima e dopo ogni operatore; ad esempio, calc(1em +

10px). Utilizzerai `calc()` per combinare `ems` con unità `vw`, rimuovi la dimensione del carattere di base precedente (e le relative media query) dal tuo foglio di stile e aggiungi questo al suo posto:

```
:root { font-size: calc(0.5em + 1vw); }
```

Ora apri la pagina e ridimensiona lentamente il tuo browser. Vedrai la dimensione del carattere ridimensionarsi senza intoppi. Lo 0,5 em qui funziona come una sorta di dimensione minima del carattere e 1 vw aggiunge uno scalare reattivo. Questo ti darà una dimensione del carattere di base che scala da 11,75 px su un iPhone 6 fino a 20 px in una finestra del browser con larghezza pari a 1.200 px. Puoi regolare questi valori a tuo piacimento e ora hai realizzato gran parte della tua strategia reattiva senza una singola media query. Invece di tre o quattro punti di interruzione a codice, tutto nella tua pagina verrà ridimensionato in modo fluido in base al viewport.

Numeri senza unità e line-height

Alcune proprietà consentono valori senza unità (ovvero un numero senza alcuna unità specificata). Le proprietà che lo supportano sono line-height, z-index e font-weight (700 equivale a grassetto; 400 equivale al carattere normale e così via). Puoi anche utilizzare il valore senza unità 0 ovunque sia richiesta un'unità di lunghezza (come px, em o rem) perché, in questi casi, l'unità non ha importanza: 0 px è uguale a 0%, che è uguale a 0 em. Nota bene: uno 0 senza unità può essere utilizzato solo per valori di lunghezza e percentuali, ad esempio in spaziature interne, bordi e larghezze. Non può essere utilizzato per valori angolari, come gradi o valori basati sul tempo come secondi. La proprietà line-height è insolita in quanto accetta sia valori con unità che senza unità. In genere dovresti usare numeri senza unità perché sono ereditati in modo diverso. Mettiamo il testo nella pagina e vediamo come si comporta, aggiungi il codice seguente al tuo foglio di stile:

```
<body>
<p class="about-us">
We have built partnerships with small farms around the world to
hand-select beans at the peak of season. We then carefully roast in
small batches to maximize their potential.
</p>
</body>
```

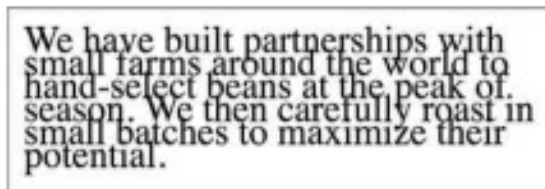
Specifichiamo line-height per l'elemento body, in modo da consentire l'ereditarietà dal resto del documento. Funzionerà proprio come ci aspettiamo, indipendentemente dalle dimensioni dei caratteri nella pagina.

We have built partnerships with small farms around the world to hand-select beans at the peak of season. We then carefully roast in small batches to maximize their potential.

Aggiungi il seguente codice:

```
body {  
  line-height: 1.2;  
}  
.about-us {  
  font-size: 2em;  
}
```

Il paragrafo eredita un'altezza di riga di 1.2. Poiché la dimensione del carattere è 32 px (2 em × 16 px, impostazione predefinita del browser), l'altezza della linea viene calcolata localmente a 38,4 px (32 px × 1,2). Ciò lascerà una quantità adeguata di spazio tra le righe di testo. Se invece specifichi l'altezza della linea utilizzando un'unità, potresti riscontrare risultati imprevisti, come quello mostrato nell'immagine seguente dove le righe di testo si sovrappongono. Ecco l'immagine e il CSS che ha generato la sovrapposizione:



```
body {  
  line-height: 1.2em;  
}  
.about-us {  
  font-size: 2em;  
}
```

Questi risultati sono dovuti a una particolarità dell'ereditarietà: quando un elemento ha un valore definito utilizzando una lunghezza (px, em, rem e così via), il suo valore calcolato viene ereditato dagli elementi figli. Quando vengono specificate unità come ems per line-height, il loro valore viene calcolato e quel valore calcolato viene passato a tutti i figli ereditari. Con la proprietà line-height, ciò può causare risultati imprevisti se l'elemento figlio ha una dimensione del carattere diversa, ecco il perché del testo sovrapposto. Quando si utilizza un numero senza unità, il valore dichiarato

viene ereditato, il che significa che il suo valore calcolato viene ricalcolato per ogni elemento figlio ereditante. Questo sarà quasi sempre il risultato che desideri. L'uso di un numero senza unità ti consente di impostare l'altezza della linea sul body e poi dimenticartene per il resto della pagina, a meno che non ci siano punti particolari in cui desideri fare un'eccezione.

Proprietà custom (Variabili CSS)

Questa specifica, apparsa per la prima volta nel 2015, ha introdotto il concetto di variabili nel linguaggio e ha consentito un nuovo livello di stili dinamici basati sul contesto. In sostanza, puoi dichiarare una variabile e assegnarle un valore; quindi puoi fare riferimento a questo valore in tutto il tuo foglio di stile. Puoi usarle per ridurre le ripetizioni nel tuo foglio di stile, ma anche per alcune altre applicazioni utili come vedrai a breve. Al momento, il supporto per le proprietà personalizzate è stato implementato in tutti i principali browser tranne IE. Per informazioni aggiornate sui browser meno conosciuti, controlla su <https://caniuse.com/#feat=css-variables>. Se ti capita di utilizzare un preprocessore CSS che supporta le proprie variabili, come Sass o Less, potresti essere tentato dall'ignorare le variabili CSS ma non farlo. Le variabili CSS sono di natura diversa e sono molto più versatili di qualsiasi cosa un preprocessore possa realizzare. Tendo a chiamarle "proprietà custom" piuttosto che variabili per enfatizzare questa distinzione. Per definire una proprietà custom, ti basta usare la stessa sintassi delle altre proprietà CSS. Crea una nuova pagina e un nuovo foglio di stile e aggiungi questo CSS:

```
:root { --main-font: Helvetica, Arial, sans-serif; }
```

Questo codice definisce una variabile denominata `--main-font` e ne imposta il valore su un insieme di caratteri sans-serif. Il nome deve iniziare con due trattini (`--`) per distinguerlo dalle proprietà CSS, seguito dal nome che desideri utilizzare. Le variabili devono essere dichiarate all'interno di un blocco di dichiarazione. Ho usato il selettore `:root` qui, che imposta la variabile per l'intera pagina, lo spiegherò a breve. Di per sé, questa dichiarazione di variabile non fa nulla finché non viene usata.

La funzione chiamata `var()` consente l'uso di variabili. Utilizzerai questa funzione per fare riferimento alla variabile `--main-font` appena definita. Aggiungi il set di regole mostrato per utilizzare la variabile:

```
:root {  
  --main-font: Helvetica, Arial, sans-serif;  
}  
p {  
  font-family: var(--main-font);  
}
```

Le proprietà custom (o personalizzate) ti consentono di definire un valore in un'unica posizione, come "unica fonte di verità" e di riutilizzare quel valore in tutto il foglio di stile. Ciò è particolarmente utile per valori ricorrenti come i colori. Andiamo ad aggiungere una proprietà personalizzata per il colore del marchio, puoi usare questa variabile dozzine di volte nel tuo foglio di stile, ma se vuoi cambiarla, devi solo modificarla in un posto.

```
:root {
  --main-font: Helvetica, Arial, sans-serif;
  --brand-color: #369;
}
p {
  font-family: var(--main-font);
  color: var(--brand-color);
}
```

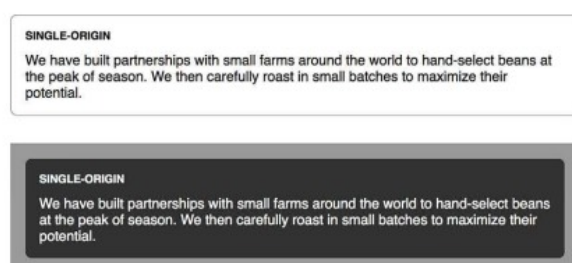
La funzione var() accetta un secondo parametro, che specifica un valore di fallback. Se la variabile specificata nel primo parametro non è definita, viene utilizzato il secondo valore.

```
:root {
  --main-font: Helvetica, Arial, sans-serif;
  --brand-color: #369;
}
p {
  font-family: var(--main-font, sans-serif);
  color: var(--secondary-color, blue);
}
```

Questo codice specifica i valori di fallback in due diverse dichiarazioni. Nella prima regola, --main-font è definito come Helvetica, Arial, sans-serif, quindi viene utilizzato questo valore. Nella seconda, --secondary-color è una variabile non definita, quindi viene utilizzato il valore di fallback blue. Se una funzione var() restituisce un valore non valido, la proprietà verrà impostata sul suo valore iniziale. Ad esempio, se la variabile in padding: var(--brand -color) restituisse un colore, sarebbe un valore di padding non valido. In tal caso, il riempimento verrebbe invece impostato su 0.

Negli esempi mostrati fino ad ora, le proprietà personalizzate sono semplicemente una bella comodità; possono evitare molte ripetizioni nel tuo codice. Ma ciò che le rende particolarmente interessanti è che le

dichiarazioni di proprietà personalizzate si sovrappongono ed ereditano: puoi definire la stessa variabile all'interno di più selettori e la variabile avrà un valore diverso per varie parti della pagina. È possibile definire una variabile come nera, ad esempio, e quindi ridefinirla come bianca all'interno di un determinato contenitore. Quindi, tutti gli stili basati su quella variabile si risolveranno dinamicamente in nero se si trovano all'esterno del contenitore e in bianco quando si trovano all'interno del contenitore. Usiamo questo esempio per ottenere un risultato come quello mostrato nell'immagine seguente.



L'HTML per questo esempio è definito di seguito, ha due istanze del pannello: una all'interno del body e una all'interno di una sezione scura. Aggiorna il tuo HTML in modo che corrisponda a questo:

```
<body>
<div class="panel">
<h2>Single-origin</h2>
<div class="body">
We have built partnerships with small farms
around the world to hand-select beans at the
peak of season. We then careful roast in
small batches to maximize their potential.
</div>
</div>
<aside class="dark">
<div class="panel">
<h2>Single-origin</h2>
<div class="body">
We have built partnerships with small farms
around the world to hand-select beans at the
```

peak of season. We then careful roast in
small batches to maximize their potential.

</div>

</div>

</aside>

</body>

Ridefiniamo il pannello per utilizzare le variabili relative al testo e al colore dello sfondo. Questo codice imposta il colore dello sfondo a bianco con il testo nero, spiegherò come funziona prima di aggiungere gli stili per la variante scura.

```
:root {  
  --main-bg: #fff;  
  --main-color: #000;  
}  
.panel {  
  font-size: 1rem;  
  padding: 1em;  
  border: 1px solid #999;  
  border-radius: 0.5em;  
  background-color: var(--main-bg);  
  color: var(--main-color);  
}  
.panel > h2 {  
  margin-top: 0;  
  font-size: 0.8em;  
  font-weight: bold;  
  text-transform: uppercase;  
}
```

Ancora una volta, hai definito le variabili all'interno di un set di regole con il selettore `:root`. Questo è significativo perché significa che questi valori sono impostati per tutto nell'elemento radice (l'intera pagina). Quando un elemento discendente della radice utilizza le variabili, questi sono i valori in cui si risolveranno. In questo caso hai due pannelli, ma sembrano sempre uguali. Definiamo di nuovo le variabili, ma questa volta con un selettore diverso. Impostiamo uno sfondo grigio scuro sul contenitore, oltre a una piccolo padding e un margine. Inoltre, ridefiniamo entrambe le variabili. Aggiungi questo al tuo foglio di stile:

```
.dark {  
  margin-top: 2em;  
  padding: 1em;  
  background-color: #999;  
  --main-bg: #333;  
  --main-color: #fff;  
}
```

Ricarica la pagina e il secondo pannello avrà uno sfondo scuro e testo bianco. Questo perché quando il pannello utilizza queste variabili, si risolveranno ai valori definiti sul contenitore dark, piuttosto che sulla radice. Nota che non hai dovuto fare un restyling del pannello o applicare classi aggiuntive. In questo esempio, hai definito le proprietà personalizzate due volte: prima sulla radice (dove il `--main-color` è nero) e poi sul contenitore scuro (dove il `--main-color` è bianco). Le proprietà personalizzate si comportano come una sorta di variabile con ambito (scope) perché i valori vengono ereditati da elementi discendenti. All'interno del contenitore scuro, `--main-color` è bianco; altrove nella pagina, è nero.

È anche possibile accedere e manipolare le proprietà personalizzate nel browser utilizzando JavaScript. Poiché questo non è un libro su JavaScript, ti mostrerò l'essenziale per familiarizzare con il concetto. Il codice seguente mostra come accedere a una proprietà su un elemento pertanto aggiungi lo script alla pagina, che registra il valore della proprietà `--main-bg` dell'elemento radice:

```
<script type="text/javascript">  
var rootElement = document.documentElement;  
var styles = getComputedStyle(rootElement);  
var mainColor = styles.getPropertyValue('--main-bg');  
console.log(String(mainColor).trim());  
</script>
```

Poiché puoi specificare al volo nuovi valori per le proprietà personalizzate, puoi utilizzare JavaScript per impostare un nuovo valore per `--main-bg` in modo dinamico. Il codice nell'elenco successivo imposta un nuovo valore su `--main-bg` sull'elemento radice. Aggiungilo alla fine del tag `<script>`:

```
var rootElement = document.documentElement;  
rootElement.style.setProperty('--main-bg', '#cdf');
```

Se esegui questo script, tutti gli elementi che ereditano la proprietà `--main-bg` verranno aggiornati per utilizzare questo nuovo valore. Sulla tua pagina, questo cambia lo sfondo del primo pannello in un colore azzurro. Il secondo pannello rimane invariato, poiché sta ancora ereditando la proprietà dal contenitore scuro. Con questa tecnica, puoi utilizzare JavaScript per modificare il tema del tuo sito, dal vivo nel tuo browser. Oppure, potresti evidenziare alcune parti della pagina o apportare al volo un numero qualsiasi di altre modifiche. Utilizzando solo poche righe di JavaScript, puoi apportare delle modifiche che influiscono su un gran numero di elementi della pagina.

Le proprietà personalizzate sono un'area completamente nuova dei CSS che gli sviluppatori stanno iniziando ad esplorare. Poiché il supporto del browser è limitato, non è ancora stato utilizzato intensamente. Sono sicuro che nel tempo vedrai emergere best practice e nuovi usi, sicuramente è qualcosa da tenere d'occhio. Sperimenta con le proprietà custom e guarda cosa puoi inventare. Tieni presente che qualsiasi dichiarazione che utilizza `var()` verrà ignorata dai vecchi browser che non la conoscono. Fornisci un comportamento alternativo per quei browser quando possibile:

```
color: black;
```

```
color: var(--main-color);
```

Ciò non sarà sempre possibile, tuttavia, data la natura dinamica delle proprietà personalizzate tieni d'occhio il sito <https://caniuse.com> per maggiori informazioni.

PADRONEGGIARE IL BOX MODEL

Quando si tratta di disporre gli elementi sulla pagina, troverai molte cose da imparare e molti modi in cui impaginare. In un sito complesso, potresti avere float, elementi posizionati in modo assoluto e altri elementi di varie dimensioni. Potresti anche avere alcuni layout che utilizzano costrutti CSS più recenti, come un flexbox o un layout a griglia. Ci sono diversi elementi di cui tenere traccia e imparare tutto ciò che riguarda il layout può essere sconcertante. E' importante avere una solida conoscenza dei fondamenti su come il browser ridimensiona e posiziona gli elementi. Gli argomenti più avanzati del layout sono costruiti su concetti come il flusso del documento e il box model; queste sono le regole base che determinano la posizione e la dimensione degli elementi nella pagina. In questo capitolo creerai un layout di pagina a due colonne. Potresti avere familiarità con un classico esercizio come questo, ma ti guiderò attraverso in un modo da evidenziare diverse sfumature di layout spesso trascurate. Esamineremo alcuni casi limite del box model e ti darò consigli pratici per dimensionare e allineare gli elementi. Affronteremo anche due dei problemi più noti nei CSS: centrare in modo verticale e avere colonne di uguale altezza.

Difficoltà con width

In questo capitolo creerai una semplice pagina con un'intestazione in alto e due colonne poste al di sotto. Alla fine del capitolo, la tua pagina apparirà come quella mostrata nell'immagine seguente. Ho intenzionalmente creato il design della pagina un po' "a blocchi", in modo da poter vedere facilmente le dimensioni e la posizione di tutti gli elementi.



Crea una nuova pagina e un foglio di stile vuoto, quindi collegali. Aggiungi il markup mostrato accanto alla tua pagina, avrà un'intestazione, oltre a un elemento main e una barra laterale che formeranno le due colonne della tua pagina. Un contenitore avvolge le due colonne:

```
<body>
<header>
<h1>Franklin Running Club</h1>
</header>
<div class="container">
<main class="main">
<h2>Come join us!</h2>
<p>
The Franklin Running club meets at 6:00pm every Thursday
at the town square. Runs are three to five miles, at your
own pace.
</p>
</main>
<aside class="sidebar">
<div class="widget"></div>
<div class="widget"></div>
</aside>
</div>
</body>
```

Cominciamo con alcuni degli stili più ovvi. Imposta il carattere per la pagina, quindi i colori dello sfondo per la pagina e ciascuno dei contenitori principali. Questo ti aiuterà a vedere la posizione e le dimensioni di ciascuno mentre procedi. Dopo averlo fatto, la tua pagina apparirà così:



Per alcuni progetti, il colore dello sfondo di diversi contenitori potrebbe essere trasparente. In questo caso, potrebbe essere utile applicare temporaneamente un colore di sfondo al contenitore fino a quando non viene ridimensionato e posizionato di conseguenza. Attualmente, la barra laterale è vuota, quindi, per impostazione predefinita, non ha altezza. Aggiungi un padding per dargli un po' di altezza, gli altri contenitori avranno bisogno di un po' di padding alla fine, ma su questo torneremo in seguito. Per ora, aggiungi questo codice al tuo foglio di stile:

```
body {
  background-color: #eee;
  font-family: Helvetica, Arial, sans-serif;
}
header {
  color: #fff;
  background-color: #0072b0;
  border-radius: .5em;
}
main {
  display: block;
}
.main {
  background-color: #fff;
  border-radius: .5em;
}
.sidebar {
  padding: 1.5em;
  background-color: #fff;
  border-radius: .5em;
```

```
}
```

Quindi, mettiamo a posto le tue due colonne. Per iniziare, utilizzerai un layout basato su float. Farai "fluttuare" la barra principale e quella laterale a sinistra e darai loro larghezze rispettivamente del 70% e del 30%. Aggiorna il tuo foglio di stile in modo che corrisponda al CSS mostrato qui:

```
.main {  
  float: left;  
  width: 70%;  
  background-color: #fff;  
  border-radius: .5em;  
}  
.sidebar {  
  float: left;  
  width: 30%;  
  padding: 1.5em;  
  background-color: #fff;  
  border-radius: .5em;  
}
```

Puoi vedere il risultato nell'immagine seguente, ma non è proprio quello che volevi:



Invece delle due colonne fianco a fianco, sono una sotto l'altra. Anche se hai specificato larghezze del 70% e del 30%, in totale le colonne occupavano più del 100% dello spazio disponibile. Ciò è dovuto al comportamento predefinito del box model. Quando imposti la larghezza o l'altezza di un elemento, stai specificando la larghezza o l'altezza del suo contenuto; qualsiasi riempimento, bordo e margine vengono quindi aggiunti a quella larghezza. Questo comportamento significa che un elemento con una larghezza di 300 px, un riempimento di 10 px e un bordo di 1 px ha una larghezza di rendering di 322 px (larghezza più riempimento sinistro e destro più bordo sinistro e destro). Questo diventa ancora più confuso quando le unità non sono tutte uguali. In questo caso hai la barra laterale che ha una larghezza del 30% più 1,5 em di riempimento sinistro e destro

mentre il contenitore principale ha solo una larghezza del 70%. Questo porta il totale delle due colonne a 100% più 3 ems. Per adattarsi, i contenitori devono usare una nuova riga.

La soluzione più ingenua consiste nel ridurre la larghezza di una delle colonne (la barra laterale, per esempio). Sul mio schermo, una larghezza pari al 26% per la barra laterale risolve il problema, ma questo metodo non è affidabile. In tal caso, il 26% è conosciuto come numero magico. Invece di utilizzare un valore desiderato, l'ho trovato apportando modifiche casuali ai miei stili fino a ottenere il risultato desiderato. Per la programmazione in generale, i numeri magici non sono consigliati infatti spesso è difficile spiegare perché un numero magico funziona. Se non capisci da dove viene il numero, non potrai capire come si comporterà in circostanze diverse. Il mio schermo è largo 1440 px, quindi nelle finestre più piccole, la barra laterale continuerà ad andare a capo. Sebbene ci sia spazio per tentativi ed errori nei CSS, in genere bisogna affidarsi a scelte di natura stilistica e non forzare le cose per potersi adattare. Un'alternativa a questo numero magico è lasciare che il browser faccia i calcoli. In questo caso, le colonne sono 3 em più larghe (a causa del riempimento), quindi puoi usare la funzione `calc()` per ridurre esattamente la larghezza di quel tanto che basta. Una larghezza della barra laterale pari a `calc(30% - 3em)` ti dà esattamente ciò di cui hai bisogno. Ma c'è ancora un modo migliore.

A causa dei problemi che hai appena riscontrato, il box model predefinito non è quello che vorresti usare normalmente. Immagino tu voglia che le larghezze specificate includano anche il padding e i bordi. CSS ti consente di regolare il comportamento del box model con la sua proprietà `box-sizing`. Per impostazione predefinita, `box-sizing` è impostata sul valore `content-box`. Ciò significa che qualsiasi altezza o larghezza specificata imposta solo la dimensione della casella del contenuto. Puoi invece assegnare il valore `border-box` e, in questo modo, le proprietà di altezza e larghezza impostano la dimensione del contenuto, spaziatura interna e bordo, che è esattamente ciò che desideri in questo esempio. Con questo modello, l'imbottitura non allarga un elemento; riduce il contenuto interno per adattarlo e fa lo stesso anche per l'altezza. Se aggiorni questi elementi per utilizzare il `border-box`, si adatteranno sulla stessa linea, indipendentemente dal padding sinistro e destro. Ecco cosa abbiamo adesso:



Per modificare il box model per i due elementi, main e barra laterale, aggiorna il foglio di stile in modo che corrisponda a questo:

```
.main {  
  box-sizing: border-box;  
  float: left;  
  width: 70%;  
  background-color: #fff;  
  border-radius: .5em;  
}  
.sidebar {  
  box-sizing: border-box;  
  float: left;  
  width: 30%;  
  padding: 1.5em;  
  background-color: #fff;  
  border-radius: .5em;  
}
```

Usando `box-sizing: border-box`, i due elementi si sommano fino ad una larghezza pari al 100%. Le loro larghezze del 70% e del 30% sono ora comprensive della loro imbottitura, quindi si adattano alla stessa linea.

Hai reso il dimensionamento della "scatola" più intuitivo per questi due elementi, ma sicuramente ti imbatteai in altri elementi con lo stesso problema. Sarebbe bello risolverlo una volta per tutte, in modo universale e valido per tutti gli elementi, per non dover più pensare a questo problema. Puoi farlo con il selettore universale (*), che ha come target tutti gli elementi della pagina. Ho aggiunto anche i selettori per indirizzare ogni pseudo-elemento sulla pagina, copia questo codice in cima al tuo foglio di stile:

```
*,  
::before,  
::after {  
  box-sizing: border-box;  
}
```

Dopo averlo applicato alla pagina, altezza e larghezza specificheranno sempre l'altezza e la larghezza effettive di un elemento, l'imbottitura non li cambierà. L'aggiunta di questo frammento all'inizio del foglio di stile è diventata una pratica comune. Se, tuttavia, aggiungi componenti di terze parti con il proprio CSS alla tua pagina, potresti vedere alcuni layout non funzionare correttamente per quei componenti, soprattutto se il loro CSS non è stato scritto tenendo in mente questa correzione. Poiché questo codice mira ad ogni elemento del componente con il selettore universale, la correzione può creare problemi. Dovresti scegliere come target ogni elemento all'interno del componente per ripristinare il dimensionamento del content-box. Puoi renderlo più semplice con una versione leggermente modificata della correzione e dell'ereditarietà. Aggiorna questa parte del tuo foglio di stile in modo che corrisponda al seguente codice:

```
:root {  
  box-sizing: border-box;  
}  
*,  
::before,  
::after {  
  box-sizing: inherit;  
}
```

Il box-sizing non è normalmente una proprietà ereditata, ma utilizzando la parola chiave inherit, puoi forzare l'ereditarietà. In questo modo, puoi convertire un componente di terze parti in un content-box quando necessario, scegliendo come target il suo contenitore di primo livello. Quindi tutti gli elementi all'interno del componente ereditano il box sizing:

```
.third-party-component { box-sizing: content-box; }
```

Ora, ogni elemento del tuo sito avrà un box model più prevedibile. Ti consiglio di aggiungere il codice visto precedentemente al tuo CSS ogni volta che crei un nuovo sito; ti farà risparmiare un sacco di problemi sul lungo termine. Tuttavia, può essere un po' problematico in un foglio di stile esistente, soprattutto se hai già scritto molti stili basati sul modello predefinito. Se lo aggiungi a un progetto esistente, esaminalo in modo approfondito per scongiurare eventuali bug. Da questo momento in poi, ogni esempio in questo libro presupporrà che questa correzione si trovi all'inizio del foglio di stile.

Spesso è visivamente più attraente avere un piccolo spazio tra le colonne. A volte puoi ottenere questo risultato aggiungendo il riempimento ad una colonna; ma in alcuni casi, questo approccio non funziona. Se entrambe le colonne hanno un colore di sfondo o un bordo, come con la tua pagina di esempio, vorrai che il tutto appaia in modo congruente per i due elementi. In questo caso, presta attenzione allo spazio grigio tra i due sfondi bianchi. Puoi ottenere questo aspetto in una manciata di modi. Diamo un'occhiata:



Innanzitutto, puoi aggiungere un margine a una delle colonne e regolare le larghezze dei tuoi elementi per tenere conto dello spazio aggiunto. Ecco come sottrarre l'1% dalla larghezza della colonna della barra laterale e spostarla sul margine, aggiorna il tuo CSS in modo che corrisponda a:

```
.main {  
  float: left;  
  width: 70%;  
  background-color: #fff;  
  border-radius: .5em;  
}  
.sidebar {  
  float: left;  
  width: 29%;  
  margin-left: 1%;  
  padding: 1.5em;  
  background-color: #fff;  
  border-radius: .5em;  
}
```

Questo aggiunge uno spazio, ma la sua larghezza si basa sulla larghezza del contenitore esterno: la percentuale è relativa all'intera larghezza del genitore. Cosa succede se si desidera specificare la dimensione in unità diverse da una percentuale? Puoi farlo con `calc()`, al posto di spostare l'1% dalla larghezza al margine, puoi spostarti di 1,5 em. Questo codice mostra

come `calc()` rende possibile tutto ciò. Modifica di nuovo il tuo CSS in modo che corrisponda a questo codice:

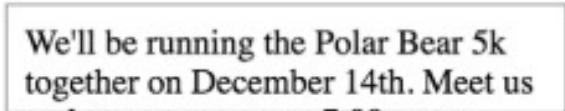
```
.main {  
  float: left;  
  width: 70%;  
  background-color: #fff;  
  border-radius: .5em;  
}  
.sidebar {  
  float: left;  
  width: calc(30% - 1.5em);  
  margin-left: 1.5em;  
  padding: 1.5em;  
  background-color: #fff;  
  border-radius: .5em;  
}
```

Non solo questo ti consente di usare `ems` piuttosto che le percentuali ma ha anche il vantaggio di essere un po' più esplicito nel codice. Quando andrai a rivedere il codice in un secondo momento, potrebbe non essere chiaro da dove provenga una percentuale specifica, ma `30% - 1,5 em` fornisce un indizio che stai facendo qualcosa legato al valore 30%.

Difficoltà con height

Lavorare con l'altezza dell'elemento è diverso dal lavorare con la larghezza dell'elemento. Le correzioni della border-box che hai apportato finora sono ancora valide e possono essere utili ma, in genere è meglio evitare di impostare altezze esplicite sugli elementi. Il normale document flow è progettato per funzionare con una larghezza limitata e un'altezza illimitata. Il contenuto riempie la larghezza della finestra e poi la linea va a capo se necessario. Per questo motivo, l'altezza di un contenitore è determinata organicamente dal suo contenuto, non dal contenitore stesso.

Quando si imposta esplicitamente l'altezza di un elemento, si corre il rischio che il suo contenuto "trabocchi" ovvero vada fuori dal contenitore. Ciò si verifica quando il contenuto non soddisfa il vincolo specificato e viene visualizzato al di fuori dell'elemento padre.



We'll be running the Polar Bear 5k
together on December 14th. Meet us
at the town square at 7:00am to
carpool. Wear blue!

Puoi controllare il comportamento esatto del contenuto con la proprietà `overflow`, che supporta quattro valori:

- `visible` (valore predefinito): tutto il contenuto è visibile, anche quando eccede i bordi del contenitore.
- `hidden`: il contenuto che trabocca dal bordo di riempimento del contenitore viene ritagliato e non sarà visibile.
- `scroll`: vengono aggiunte le scrollbar al contenitore in modo che l'utente possa scorrere per visualizzare il contenuto rimanente. Su alcuni sistemi operativi vengono aggiunte barre di scorrimento sia orizzontali che verticali, anche se tutto il contenuto è visibile. In questo caso, le barre di scorrimento saranno disabilitate (in grigio).

- auto: vengono aggiunte le scrollbar al contenitore solo se il contenuto è traboccato.

In genere, preferisco auto piuttosto che scroll perché, nella maggior parte dei casi, non voglio che le barre di scorrimento appaiano se non necessario. Presta attenzione all'uso delle barre di scorrimento. I browser inseriscono una barra di scorrimento per lo scorrimento della pagina e l'aggiunta di aree scorrevoli nidificate all'interno della pagina può essere frustrante per gli utenti. Se un utente utilizza la rotellina del mouse per scorrere la pagina verso il basso e il cursore raggiunge un'area scorrevole più piccola, la rotellina smetterà di far scorrere la pagina e farà scorrere invece la casella più piccola.

Specificare l'altezza utilizzando una percentuale è problematico. La percentuale si riferisce alla dimensione del blocco contenitore di un elemento; l'altezza di quel contenitore, tuttavia, è tipicamente determinata dall'altezza dei suoi figli. Questo produce una definizione circolare che il browser non può risolvere, quindi semplicemente ignorerà la dichiarazione. Affinché le altezze basate sulla percentuale funzionino a dovere, il genitore deve avere un'altezza definita in modo esplicito. Uno dei motivi per cui i programmatori cercano di utilizzare le altezze in base alla percentuale è fare in modo che un contenitore riempi lo schermo. Un approccio migliore consiste nell'usare le unità vh relative al viewport, che abbiamo esaminato nel capitolo precedente. Un'altezza di 100 vh è esattamente l'altezza del viewport. L'uso più comune, tuttavia, è creare colonne di uguale altezza e anche questo può essere risolto senza una percentuale.

Il problema di avere colonne di uguale altezza è una debolezza che ha afflitto i CSS sin dall'inizio. All'inizio degli anni 2000, i CSS hanno soppiantato l'uso delle tabelle HTML per la disposizione dei contenuti. All'epoca, le tabelle erano l'unico modo per produrre due colonne di uguale altezza, o, più specificamente, colonne della stessa altezza senza specificarne esplicitamente l'altezza. Puoi facilmente impostare tutte le colonne a un'altezza di 500 px o un altro valore arbitrario. Ma se consentissi alle colonne di determinare le loro altezze in modo naturale, ogni elemento valuterebbe un'altezza diversa, in base al suo contenuto. Ci sono voluti alcuni workaround per aggirare il problema ma, con l'evoluzione dei CSS, sono emerse soluzioni che coinvolgono pseudo-elementi o margini negativi. Se stai ancora utilizzando uno di questi metodi complicati, è tempo di

cambiare. I browser moderni rendono tutto molto più semplice: supportano le tabelle CSS. Ad esempio, IE8+ supporta `display: table` e IE10+ consente un box flessibile, o `flexbox`, che, per impostazione predefinita, producono colonne di uguale altezza. Quando dico browser moderni, intendo versioni recenti di browser con aggiornamento automatico. Questi includono Chrome, Firefox, Edge, Opera e, nella maggior parte dei casi, Safari. Internet Explorer è la preoccupazione maggiore; se dico che qualcosa è supportato in IE10+, ciò implica generalmente che anche i browser "evergreen" lo supportino. Alcuni modelli comuni richiedono colonne di uguale altezza, come nel caso della tua pagina a due colonne, un ottimo esempio. Sarebbe più opportuno allineare le altezze della colonna principale e della barra laterale e man mano che il contenuto in una delle colonne cresce, ogni colonna cresce secondo necessità, in modo che siano sempre allineate.



Potresti ottenere ciò impostando un'altezza arbitraria su entrambe le colonne, ma quale valore sceglieresti? Con un valore troppo grande avresti un tanto spazio vuoto sul fondo dei tuoi contenitori; con un valore troppo piccolo potresti far traboccare il contenuto. La soluzione migliore è che le colonne si dimensionino da sole in modo naturale, allungando la colonna più corta in modo che la sua altezza sia uguale all'altezza di quella più alta. Ti mostrerò come farlo usando entrambi i layout di tabella basati su CSS e un `flexbox`.

In primo luogo, utilizzerai un layout di tabella basato su CSS. Al posto di usare `float`, renderai il contenitore un `display: table` e ogni colonna un `display: table-cell`. Aggiorna i tuoi stili in modo che corrispondano al codice seguente:

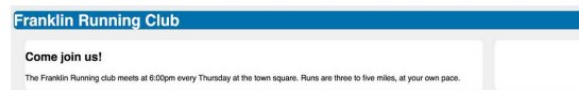
```
.container {  
  display: table;  
  width: 100%;  
}  
.main {  
  display: table-cell;
```

```

width: 70%;
background-color: #fff;
border-radius: .5em;
}
.sidebar {
display: table-cell;
width: 30%;
margin-left: 1.5em;
padding: 1.5em;
background-color: #fff;
border-radius: .5em;
}

```

Per impostazione predefinita, un elemento con `display: table` non si espanderà a una larghezza del 100% come farebbe con `block`, quindi dovrai dichiarare la larghezza in modo esplicito. Questo codice ti avvicina all'obiettivo ma non lo centra. Questo perché i margini non possono essere applicati agli elementi della `table-cell`. Dovrai apportare più modifiche per ottenere esattamente come lo desideri. Per definire lo spazio tra le celle di una tabella, puoi utilizzare la proprietà `border-spacing`. Questa proprietà accetta due valori di lunghezza: uno per la spaziatura orizzontale e uno per la spaziatura verticale. (Puoi anche specificare un solo valore da applicare a entrambi.) Puoi aggiungere `border-spacing: 1.5em 0` al tuo contenitore, ma questo ha un effetto collaterale particolare: quel valore viene applicato anche ai bordi esterni della tabella. Ora le tue due colonne non sono più allineate con l'intestazione sui bordi sinistro e destro:



Puoi risolvere questo problema con l'uso intelligente di un margine negativo, ma questo deve andare su un nuovo contenitore, ecco come. Aggiungi un `<div class="wrapper">` attorno al contenitore e applica un margine sinistro e destro di `-1.5em` per contrastare la spaziatura del bordo di `1.5em` sulle barre laterali. Questa parte del tuo foglio di stile dovrebbe assomigliare a questa:

```

.wrapper {

```

```

margin-left: -1.5em;
margin-right: -1.5em;
}
.container {
display: table;
width: 100%;
border-spacing: 1.5em 0;
}
.main {
display: table-cell;
width: 70%;
background-color: #fff;
border-radius: .5em;
}
.sidebar {
display: table-cell;
width: 30%;
padding: 1.5em;
background-color: #fff;
border-radius: .5em;
}

```

Mentre i margini positivi spingono verso l'interno i bordi del contenitore, il margine negativo tira fuori i bordi. In combinazione con border-spacing, i bordi esterni della colonna ora si allineano con i bordi del <body> (il riquadro contenente il wrapper). Ora hai il layout che desideri: due colonne di uguale altezza, spazio di 1,5 em e bordi esterni allineati con l'intestazione:



I margini negativi hanno alcuni usi interessanti che esamineremo tra un po'.

Abbiamo parlato di tabelle per il layout, se hai lavorato per un po' nello sviluppo web, probabilmente hai sentito dire che è una cattiva pratica usare le tabelle HTML per i layout. Molti progettisti di siti Web all'inizio degli

anni 2000 hanno strutturato i propri siti utilizzando elementi `<table>`. Spesso era più facile disporre le pagine usando le tabelle invece di combattere con i float (l'unica alternativa praticabile all'epoca). Alla fine, c'è stato un forte contraccolpo contro l'uso delle tabelle per i layout perché ciò significava utilizzare HTML non semantico. Al posto dei tag HTML che rappresentano il contenuto, stavano sfruttando il layout, qualcosa di cui i CSS dovrebbero essere responsabili. I browser ora supportano la visualizzazione delle tabelle per tutti i tipi di elementi diversi da `<table>`, così puoi goderti i vantaggi dei layout delle tabelle e mantenere il markup semantico. Tuttavia, non è una soluzione risolutiva perché gli attributi della tabella HTML `colspan` e `rowspan` non hanno equivalenti e float, flexbox e inline-block possono disporre il contenuto in modi impossibili per le tabelle.

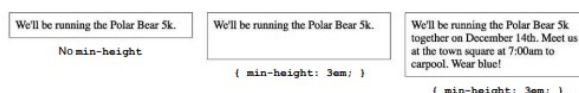
Flexbox

La realizzazione di un layout a due colonne con colonne di uguale altezza può essere eseguita anche con un flexbox. In particolare, un flexbox non richiede l'uso di un div wrapper aggiuntivo. Per impostazione predefinita, l'utilizzo di un flexbox produce elementi di uguale altezza; non dovrai preoccuparti dei margini negativi. Rimuovi il div wrapper che hai aggiunto al layout della tabella e aggiorna il foglio di stile in modo che corrisponda al seguente snippet. Se non conosci flexbox, questa sarà un'introduzione delicata.

```
.container {  
  display: flex;  
}  
.main {  
  width: 70%;  
  background-color: #fff;  
  border-radius: 0.5em;  
}  
.sidebar {  
  width: 30%;  
  padding: 1.5em;  
  margin-left: 1.5em;  
  background-color: #fff;  
  border-radius: .5em;  
}
```

Applicando la dichiarazione `display: flex` al contenitore, si ottiene un contenitore flessibile. I suoi elementi figlio diventeranno della stessa altezza per impostazione predefinita. Puoi impostare larghezze e margini sugli oggetti e, anche se questo andrebbe oltre il 100%, il flexbox risolve questo problema. Questo codice crea pixel per pixel lo stesso del layout della tabella, non ha bisogno del wrapper extra e il CSS è un po' più semplice. Un flexbox offre molte opzioni e questo esempio mostra tutto ciò di cui hai bisogno per costruire il tuo primo layout basato su flexbox. Non impostare mai esplicitamente l'altezza di un elemento a meno che tu non abbia altra scelta. Cerca sempre un approccio alternativo e ricorda che impostare un'altezza porta inevitabilmente a ulteriori complicazioni.

Due proprietà che possono essere estremamente utili sono `min-height` e `max-height`. Invece di definire in modo esplicito un'altezza, puoi utilizzare queste proprietà per specificare un valore minimo o massimo, consentendo all'elemento di ridimensionarsi naturalmente entro quei limiti. Supponiamo di voler posizionare l'immagine del tuo eroe dietro un paragrafo di testo più grande e sei preoccupato che vada fuori dal contenitore. Invece di impostare un'altezza esplicita, puoi specificare un'altezza minima con `min-height`. Ciò significa che l'elemento sarà almeno alto quanto specificato e, se il contenuto non si adatta, il browser consentirà all'elemento di crescere naturalmente per prevenire l'overflow. Nell'immagine seguente sono mostrati tre elementi.



L'elemento a sinistra non ha un'altezza minima, quindi la sua altezza è determinata naturalmente, mentre gli altri due hanno un'altezza minima pari a 3 em. L'elemento al centro avrebbe un'altezza naturale inferiore a quella, ma il valore `min-height` ha portato l'altezza a 3 em. L'elemento a destra ha abbastanza testo, tanto da aver superato 3 em ma il contenitore è cresciuto naturalmente per contenere il testo in modo appropriato.

Allo stesso modo, `max-height` consente a un elemento di ridimensionarsi in modo naturale, fino a un certo punto. Se viene raggiunta quella dimensione, l'elemento non diventa più alto e il contenuto traboccherà. In modo simile, `min-width` e `max-width` vincolano la larghezza di un elemento.

La centratura verticale nei CSS è un altro problema noto. Storicamente ci sono stati diversi modi per ottenere la centratura verticale, ognuno dei quali funzionava solo in determinate circostanze. Con i CSS, la risposta a un problema è spesso "dipende" e questo può certamente essere il caso adatto per questa risposta. Molti dei problemi derivano dall'impostazione dell'altezza di un contenitore su un valore costante e dal tentativo di centrare un contenuto di dimensioni dinamiche al suo interno. Quando possibile, cerca di ottenere l'effetto desiderato consentendo al browser di determinare le altezze in modo naturale.

Ti sei chiesto perché vertical-align non funziona? Gli sviluppatori sono spesso frustrati quando applicano vertical-align: middle a un elemento di blocco, aspettandosi che centri il contenuto del blocco. Invece, questa dichiarazione viene ignorata dal browser. Una dichiarazione vertical-align ha effetto solo sugli elementi inline e table-cell. Con gli elementi inline, controlla l'allineamento tra gli altri elementi sulla stessa linea. Puoi usarlo per controllare un'immagine in linea con il testo vicino, ad esempio. Con gli elementi table-cell, invece, vertical-align controlla l'allineamento dei contenuti all'interno della cella. Se un layout di tabella CSS funziona correttamente per la tua pagina, puoi eseguire la centratura verticale con vertical-align.

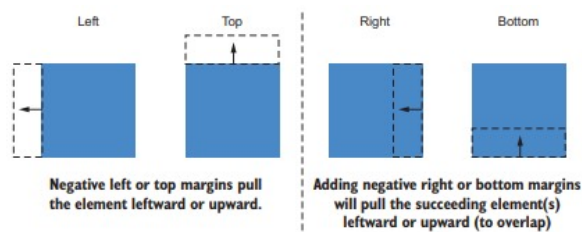
Ecco il modo più semplice per centrare verticalmente in CSS: dai a un contenitore un padding uguale in alto e in basso e lascia che siano il contenitore e il suo contenuto a determinare la loro altezza in modo naturale. Puoi aggiungere temporaneamente questo codice al tuo foglio di stile per visualizzarlo sulla tua pagina (assicurati di rimuoverlo in seguito, poiché non fa parte del tuo design):

```
header {  
  padding-top: 4em;  
  padding-bottom: 4em;  
  color: #fff;  
  background-color: #0072b0;  
  border-radius: .5em;  
}
```

Questo approccio funziona indipendentemente dal fatto che il contenuto all'interno del contenitore sia inline, block o di qualsiasi altro valore. A volte, tuttavia, potrebbe essere necessario impostare una certa altezza sul contenitore. Questo è un problema comune che si presenta con colonne di uguale altezza, in particolare se si utilizza una tecnica precedente con i float. Fortunatamente, sia le tabelle CSS che flexbox semplificano la centratura. Se utilizzi una delle tecniche precedenti, dovrai trovare un altro modo per centrare il contenuto.

Margini negativi

A differenza del riempimento e della larghezza del bordo, puoi assegnare un valore negativo ai margini. Questo ha alcuni usi peculiari, come consentire agli elementi di sovrapporsi o allungarsi più larghi dei loro contenitori. Il comportamento esatto di un margine negativo dipende dal lato dell'elemento a cui lo si applica. Puoi vederlo illustrato nell'immagine seguente:



Se applicato a sinistra o in alto, il margine negativo sposta l'elemento rispettivamente verso sinistra o verso l'alto. Ciò può far sì che l'elemento si sovrapponga a un altro elemento che lo precede nel document flow. Se applicato sul lato destro o inferiore, un margine negativo non sposta l'elemento; invece, si riferisce a qualsiasi elemento successivo. Dare a un elemento un margine inferiore negativo non è diverso dal dare agli elementi sottostanti un margine superiore negativo. Quando un elemento block non ha una larghezza specificata, riempie naturalmente la larghezza del suo contenitore. Un margine destro negativo, però, può cambiare questo comportamento: finché non viene specificata alcuna larghezza, tira il bordo dell'elemento a destra, portandolo fuori dal contenitore. Unisci questo con un margine sinistro negativo uguale ed entrambi i lati dell'elemento verranno estesi all'esterno del contenitore. Questa stranezza è ciò che ti ha permesso di ridimensionare il layout della tabella per riempire la larghezza del `<body>`, nonostante la spaziatura del bordo.

L'utilizzo di margini negativi per sovrapporre gli elementi può rendere alcuni elementi non selezionabili se vengono spostati sotto altri elementi. I margini negativi potrebbero non essere usati spesso ma sono utili in alcune circostanze. In particolare, sono utili quando si costruiscono layout di colonne. Assicurati di non usarli troppo frequentemente, tuttavia, o potresti

scoprire rapidamente di perdere traccia di ciò che sta accadendo sulla pagina.

Margini collassati

Dai un'altra occhiata alla tua pagina. Noti qualcosa di strano sui margini? Non hai applicato alcun margine all'intestazione o al contenitore, tuttavia c'è uno spazio vuoto tra di loro. Perché c'è quello spazio?

Quando i margini superiore e/o inferiore sono adiacenti, si sovrappongono, combinandosi per formare un unico margine. Questo è indicato come collasso. Lo spazio sotto l'intestazione è il risultato di margini compressi o collassati. Diamo un'occhiata a come funziona. Il motivo principale per i margini compressi ha a che fare con la spaziatura dei blocchi di testo. I paragrafi (<p>), per impostazione di default, hanno un margine superiore di 1 em e un margine inferiore di 1 em. Questo viene applicato dal foglio di stile dello user agent ma quando si impilano due paragrafi, uno dopo l'altro, i loro margini non si sommano a uno spazio di 2 em. In realtà collassano, sovrapponendosi per produrre solo 1 em di spazio tra i due paragrafi. Puoi vedere questo tipo di margine compresso nella colonna di sinistra della pagina. Nota come i margini di ogni elemento occupino lo stesso spazio sulla pagina.



La dimensione del margine compresso è uguale al più grande dei margini uniti. In questo caso, l'intestazione ha un margine inferiore di 19,92 px (dimensione carattere 24 px \times 0,83 em) e il paragrafo ha un margine superiore di 16 px (dimensione carattere 16 px \times 1 em margine). Il più grande di questi, 19,92 px, è la quantità di spazio renderizzata tra i due elementi.

Gli elementi non devono essere fratelli adiacenti per far collassare i loro margini. Anche se avvolgi il paragrafo all'interno di un div aggiuntivo, come nel codice seguente, il risultato visivo sarà lo stesso. In assenza di qualsiasi altro CSS che interferisca, tutti i margini superiore e inferiore adiacenti collasseranno.

```
<main class="main">
```

<h2>Come join us!</h2>

<div>

<p>

The Franklin Running club meets at 6:00pm
every Thursday at the town square. Runs
are three to five miles, at your own pace.

</p>

</div>

</main>

In questo caso, ci sono tre diversi margini che collassano insieme: il margine inferiore di <h2>, il margine superiore di <div> e il margine superiore di <p>. I valori calcolati di questi sono rispettivamente 19,92 px, 0 px e 16 px, quindi lo spazio tra gli elementi è ancora 19,92 px, il più grande dei tre. In effetti, puoi annidare il paragrafo all'interno di più div e il rendering sarà comunque lo stesso: tutti i margini si comprimono insieme. In breve, tutti i margini superiori e inferiori adiacenti collasseranno insieme. Se aggiungi un div vuoto e senza stile (senza altezza, bordo o riempimento) alla pagina, i suoi margini superiore e inferiore verranno compressi. I margini sinistro e destro non collassano. I margini compressi agiscono come una sorta di "bolla dello spazio personale". Se due persone in piedi a una fermata dell'autobus si sentono a proprio agio con 3 metri di spazio tra loro, staranno felicemente a 3 metri di distanza. Non hanno bisogno di stare a 6 metri di distanza per essere soddisfatti entrambi. Questo comportamento in genere significa che puoi modellare i margini su vari elementi senza preoccuparti di ciò che potrebbe apparire sopra o sotto di essi. Se applichi un margine inferiore di 1,5 em alle intestazioni, puoi aspettarti la stessa spaziatura dopo le intestazioni, sia che l'elemento successivo sia un <p> con un margine superiore di 1 em o un div senza margine superiore. Il margine compresso tra gli elementi appare più grande solo se l'elemento successivo richiede più spazio.

Il modo in cui tre margini consecutivi collassano potrebbe coglierti alla sprovvista. Il collasso del margine di un elemento all'esterno del suo contenitore produce in genere un effetto indesiderato se il contenitore ha uno sfondo. Nell'immagine precedente, il titolo della pagina è un <h1>, con un margine inferiore di 0,67 em (21,44 px) applicato dagli stili dello user agent. Quel titolo è all'interno di un <header> senza margini e i margini inferiori di entrambi gli elementi sono adiacenti, quindi collassano,

risultando in un margine inferiore di 21,44 px sull'intestazione. La stessa cosa accade anche con i margini superiori dei due elementi. Questo è un po' strano. In questo caso, vuoi che il margine di `<h1>` rimanga all'interno dell'<intestazione>. I margini non collassano sempre esattamente nel punto desiderato. Per fortuna, ci sono diversi modi per prevenire questo comportamento. Questo perché i margini degli elementi flexbox non si comprimono e abbiamo disposto quella parte della pagina usando un flexbox. Il padding fornisce un'altra soluzione: se aggiungi il riempimento superiore e inferiore all'intestazione, i margini al suo interno non collasseranno verso l'esterno. Già che ci sei, aggiorniamo l'intestazione in modo che assomigli all'immagine seguente e applichiamo anche il riempimento sinistro e destro. Per fare ciò, aggiorna il tuo foglio di stile, noterai che ora non c'è margine tra l'intestazione e il contenuto principale:

```
header {  
  padding: 1em 1.5em;  
  color: #fff;  
  background-color: #0072b0;  
  border-radius: .5em;  
}
```

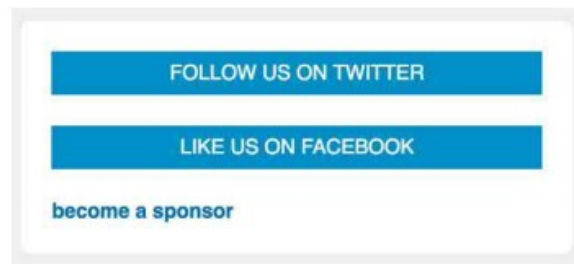
Di seguito sono riportati i modi per impedire il collasso dei margini:

- Usando `overflow: auto` (o qualsiasi valore diverso da `visible`) sul contenitore impedisce che i margini all'interno del contenitore si comprimano con quelli all'esterno del contenitore. Questa è spesso la soluzione meno invadente.
- Aggiungendo un bordo o un riempimento tra due margini si impedisce il collasso
- I margini non collassano all'esterno di un contenitore flottante, ovvero un blocco in linea o che ha una posizione assoluta o fissa.
- Quando si utilizza un flexbox, i margini non collassano tra gli elementi che fanno parte del layout flessibile. Questo vale anche per il layout della griglia.
- Gli elementi con `display: table-cell` non hanno un margine, quindi non si comprimeranno. Questo vale anche per `table-row` e per la maggior parte degli altri tipi di visualizzazione delle tabelle. Le eccezioni sono `table`, `table-inline` e `table-caption`.

Molti di questi cambiano il comportamento del layout dell'elemento, quindi probabilmente non vorrai applicarli a meno che non producano il layout che stai cercando.

Creare spazio tra gli elementi in un container

L'interazione tra il padding di un contenitore e i margini del suo contenuto può essere difficile da gestire. In fondo, si tratta di mettere alcuni elementi nella barra laterale e risolvere i problemi che potrebbero sorgere. Alla fine, ti mostrerò una tecnica utile che può semplificare notevolmente le cose. Aggiungerai due pulsanti che conducono alle pagine dei social media e un altro collegamento meno importante alla barra laterale. Il tuo obiettivo è che la barra laterale assomigli a questa:



Cominciamo con i due link social. Aggiungili alla barra laterale come mostrato nell'elenco seguente. La classe `button-link` sarà un buon obiettivo per il tuo selettore CSS:

```
<aside class="sidebar">
  <a href="/twitter" class="button-link">
    follow us on Twitter
  </a>
  <a href="/facebook" class="button-link">
    like us on Facebook
  </a>
</aside>
```

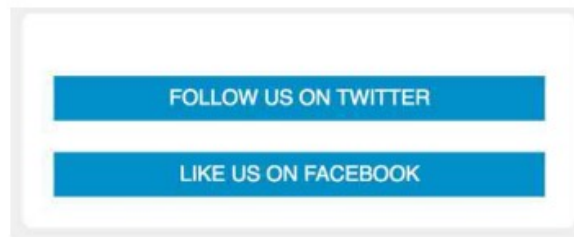
Successivamente, applicherai gli stili per l'aspetto generale dei pulsanti. Farai in modo che blocchino gli elementi in modo che riempiano la larghezza del contenitore e ognuno apparirà su una propria riga. Aggiungi questo CSS al tuo foglio di stile:

```
.button-link {
  display: block;
  padding: 0.5em;
```



```
color: #fff;  
background-color: #0090C9;  
text-align: center;  
text-decoration: none;  
text-transform: uppercase;  
}
```

Ora i collegamenti hanno lo stile corretto, ma devi ancora capire la spaziatura tra di loro. Senza margini, si accumuleranno direttamente l'uno sull'altro, come fanno ora. Hai alcune opzioni: potresti dare loro margini superiore e inferiore separatamente o insieme, dove si verificherebbe il collasso del margine tra i due pulsanti. Indipendentemente dall'approccio scelto, tuttavia, ti troverai comunque davanti ad un problema: il margine deve funzionare insieme al padding della barra laterale. Se aggiungi `margin-top: 1.5em`, otterrai il risultato mostrato:



Ora avrai spazio extra nella parte superiore del contenitore. Il margine superiore del primo pulsante più il riempimento superiore del contenitore producono una spaziatura non uniforme con gli altri tre lati del contenitore. Puoi risolvere questo problema in diversi modi. Ecco una delle soluzioni più semplici. Utilizza il combinatore di fratelli adiacenti (+) per indirizzare solo i collegamenti a pulsanti che seguono immediatamente altri collegamenti a pulsanti come fratelli sotto lo stesso elemento padre. Ora il margine appare solo tra due pulsanti.

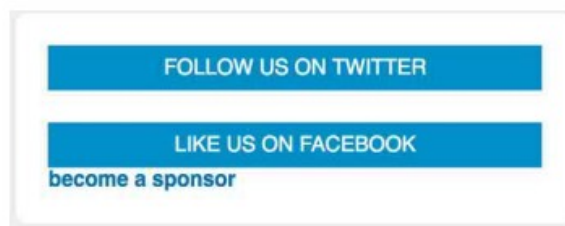
```
.button-link {  
display: block;  
padding: .5em;  
color: #fff;  
background-color: #0090C9;  
text-align: center;  
text-decoration: none;
```

```
text-transform: uppercase;
}
.button-link + .button-link {
margin-top: 1.5em;
}
```

Questo approccio sembra funzionare infatti il primo pulsante non ha più un margine superiore, quindi la spaziatura è uniforme. Sei sulla strada giusta, ma il problema della spaziatura si ripresenta non appena aggiungi più contenuti alla barra laterale. Aggiungi il terzo link alla tua pagina, come mostrato nel codice seguente. Questo ha una classe `sponsor-link` in modo da poter applicare stili diversi al link.

```
<aside class="sidebar">
<a href="/twitter" class="button-link">
follow us on Twitter
</a>
<a href="/facebook" class="button-link">
like us on Facebook<
/a>
<a href="/sponsors" class="sponsor-link">
become a sponsor
</a>
</aside>
```

Lo potrai modellare, ma ancora una volta dovrai affrontare la questione della spaziatura tra questo e gli altri pulsanti. L'immagine mostra come apparirà il collegamento prima di correggere il margine:



Probabilmente sei tentato dall'aggiungere anche un margine superiore al link; tieni duro per ora. Ti mostrerò un'alternativa interessante in seguito.

```
.sponsor-link {
display: block;
```

```
color: #0072b0;  
font-weight: bold;  
text-decoration: none;  
}
```

Potresti aggiungere un margine superiore e sembrerebbe giusto ma devi fare una considerazione: l'HTML ha la brutta abitudine di cambiare. Ad un certo punto, il mese prossimo o l'anno prossimo, qualcosa in questa barra laterale dovrà essere spostato o sostituito. Forse il link di sponsorizzazione dovrà essere spostato nella parte superiore della barra laterale. Oppure, forse dovrai aggiungere un widget per iscriverti a una newsletter via e-mail. Ogni volta che le cose cambiano, dovrai rivedere la questione di questi margini. Dovrai assicurarti che ci sia spazio tra ogni articolo, ma nessuno spazio estraneo nella parte superiore (o inferiore) del contenitore.

Il web designer Heydon Pickering una volta ha detto, riferendosi ai margini, che sono "come applicare la colla su un lato di un oggetto prima di aver determinato se vuoi attaccarlo a qualcosa". Invece di fissare i margini per il contenuto della pagina corrente, sistemiamolo in un modo che funzioni indipendentemente da come la pagina venga ristrutturata. Ecco cosa serve e si presenta così: * + *. Questo è un selettore universale (*) che prende di mira tutti gli elementi, seguito da un combinatore fratello adiacente (+), seguito da un altro selettore universale. Si guadagna il nome di gufo proprio perché ricorda lo sguardo vacuo di un gufo. Il gufo non è diverso dal selettore che hai usato prima: .social-button + .social-button tranne per il fatto che, invece di puntare ai pulsanti che seguono immediatamente altri pulsanti, punta a qualsiasi elemento che segue immediatamente qualsiasi altro elemento. Cioè, seleziona tutti gli elementi sulla pagina che non sono il primo figlio del loro genitore. Usiamo il gufo per aggiungere margini superiori agli elementi della pagina. In questo modo distanzierai uniformemente ogni elemento nella barra laterale. Questo indirizzerà anche il contenitore principale perché un fratello segue immediatamente l'intestazione, fornendo anche lo spazio che desideri lì. Ho incluso il body all'inizio del selettore e ciò limita il selettore a scegliere come target solo gli elementi all'interno del body. Se usi il gufo da solo, prenderà di mira l'elemento <body> perché è un fratello adiacente dell'elemento <head>.



```
body * + * { margin-top: 1.5em; }
```

Potresti essere preoccupato per le implicazioni sulle prestazioni del selettore universale (*). In IE6 era incredibilmente lento, quindi gli sviluppatori evitavano di usarlo. Oggi questo non è più un problema perché i browser moderni lo gestiscono in modo efficiente. Inoltre, il suo utilizzo riduce potenzialmente il numero di selettori nel foglio di stile, poiché risolve globalmente la maggior parte degli elementi. In effetti, potrebbe essere più performante, a seconda dei dettagli del tuo foglio di stile. Il margine superiore del gufo ha un effetto collaterale indesiderato sulla barra laterale. Poiché la barra laterale è una sorella adiacente della colonna principale, anch'essa riceve un margine superiore. Dovrai ripristinarlo a zero. Dovrai anche aggiungere padding alle colonne principali perché non l'hai ancora fatto. Aggiorna la parte corrispondente del tuo foglio di stile in modo che corrisponda a ciò che è mostrato qui:

```
.main {  
width: 70%;  
padding: 1em 1.5em;  
background-color: #fff;  
border-radius: .5em;  
}  
.sidebar {  
width: 30%;  
padding: 1.5em;  
margin-top: 0;  
margin-left: 1.5em;  
background-color: #fff;  
border-radius: .5em;  
}
```

Questi sono gli ultimi ritocchi per la tua pagina. Ora dovrebbe assomigliare a questo:



Usare il gufo in questo modo è un compromesso. Semplifica molti margini in tutta la pagina, ma dovrai sovrascriverlo nei punti in cui non desideri che venga applicato. Questo sarà generalmente vero solo nei punti in cui hai elementi fianco a fianco, come con i layout a più colonne. A seconda del tuo design, dovrai anche impostare i margini desiderati su paragrafi e intestazioni. Userò il gufo in diversi esempi per aiutarti a farti un'idea dei compromessi coinvolti. Il gufo potrebbe non essere la soluzione corretta per ogni progetto ed è difficile aggiungerlo a un progetto esistente senza creare problemi al layout, ma consideralo la prossima volta che avvii un nuovo sito Web o un'applicazione Web. Il foglio di stile completo è il seguente:

```
:root { box-sizing: border-box; }
*,
::before,
::after {
  box-sizing: inherit;
}
body {
  background-color: #eee;
  font-family: Helvetica, Arial, sans-serif;
}
body * + * {
  margin-top: 1.5em;
}
header {
  padding: 1em 1.5em;
  color: #fff;
  background-color: #0072b0;
  border-radius: .5em;
}
.container {
  display: flex;
```

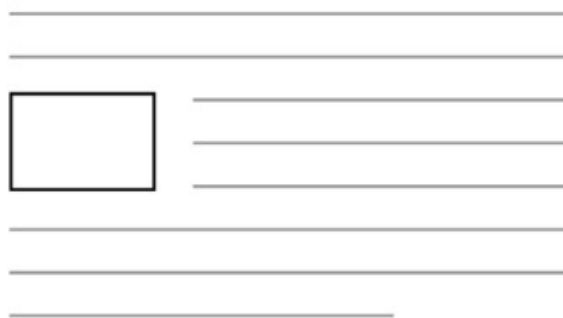
```
}  
.main {  
width: 70%;  
padding: 1em 1.5em;  
background-color: #fff;  
border-radius: .5em;  
}  
.sidebar {  
width: 30%;  
padding: 1.5em;  
margin-top: 0;  
margin-left: 1.5em;  
background-color: #fff;  
border-radius: .5em;  
}  
.button-link {  
display: block;  
padding: .5em;  
color: #fff;  
background-color: #0090C9;  
text-align: center;  
text-decoration: none;  
text-transform: uppercase;  
}  
.sponsor-link {  
display: block;  
color: #0072b0;  
font-weight: bold;  
text-decoration: none;  
}
```


FLOATS

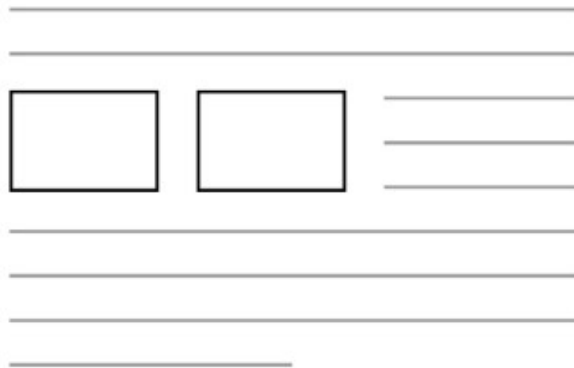
Abbiamo già trattato alcuni concetti fondamentali del dimensionamento e della spaziatura degli elementi. Adesso ci baseremo su questi concetti esaminando più da vicino i metodi principali per la disposizione della pagina. Esamineremo i metodi più importanti per alterare il flusso del documento: float e flexbox. Quindi esamineremo il posizionamento, che viene utilizzato principalmente per impilare gli elementi uno di fronte all'altro. I layout flexbox e grid sono entrambi nuovi per CSS e si stanno rivelando strumenti davvero essenziali. Sebbene i float e il posizionamento non siano nuovi, sono spesso fraintesi. In questo capitolo, esamineremo prima i float. Sono il metodo più antico per creare una pagina web e per molti anni sono stati l'unico modo. Sono un po' strani, tuttavia. Ti mostrerò come affrontare alcune delle loro stranezze, incluso uno strumento chiamato clearfix. Questo darà un contesto al loro comportamento. Man mano che procediamo, imparerai anche due modelli che potresti vedere spesso nei layout di pagina: il modello a doppio contenitore e l'oggetto media. Per concludere, metterai al lavoro le tue conoscenze per costruire un sistema a griglia, che è uno strumento versatile per strutturare una pagina.

Lo scopo dei float

Sebbene i float non fossero originariamente destinati a costruire layout di pagina, hanno svolto bene e per diverso tempo quel compito. Per dare un senso ai float, tuttavia, dobbiamo prima tenere a mente il loro scopo originale. Un float trascina un elemento (spesso un'immagine) su un lato del suo contenitore, consentendo al flusso di documenti di avvolgerlo. Questo layout è comune in giornali e riviste, quindi i float sono stati aggiunti ai CSS per ottenere questo effetto. Questa illustrazione mostra un elemento trascinato a sinistra, ma puoi anche fluttuare un elemento a destra.



Un elemento flottante viene rimosso dal normale flusso di documenti e trascinato verso il bordo del contenitore. Il flusso del documento riprende quindi, ma avvolgerà lo spazio in cui ora risiede l'elemento mobile. Se fai fluttuare più elementi nella stessa direzione, si accatastano l'uno accanto all'altro, come mostrato nella figura seguente:



Se scrivi CSS da un po', questo comportamento probabilmente non è nuovo per te. Ma la cosa importante da notare è questa: non usiamo sempre i float in questo modo, anche se è il loro scopo originale. Agli albori dei CSS, gli sviluppatori si sono resi conto che potevano utilizzare questo semplice strumento per spostare sezioni della pagina per creare tutti i tipi di layout. Non doveva essere uno strumento di impaginazione, ma da quasi due decenni lo usiamo come tale. Lo abbiamo fatto perché era la nostra unica opzione. Infine, è emersa la possibilità di utilizzare display: inline-block o display: table, che offriva alternative, anche se limitate. Fino all'aggiunta del flexbox e dei layout a griglia negli ultimi anni, i float sono rimasti il nostro grosso battitore per il layout di pagina. Diamo un'occhiata a come funzionano.



Supponiamo di voler costruire l'immagine qui sopra. Negli esempi in questo capitolo, utilizzerai i float per posizionare ciascuna delle quattro caselle grigie. All'interno delle caselle, farai quindi scorrere le immagini accanto al testo. Crea una pagina vuota e collegala a un nuovo foglio di stile, quindi aggiungi il codice in questo elenco alla tua pagina.

```
<body>
<div class="container">
<header>
<h1>Franklin Running Club</h1>
</header>
<main class="main clearfix">
<h2>Running tips</h2>
<div>
<div class="media">

<div class="media-body">
<h4>Strength</h4>
<p>
Strength training is an important part of
injury prevention. Focus on your core&mdash;
especially your abs and glutes.
</p>
</div>
</div>
<div class="media">

<div class="media-body">
<h4>Cadence</h4>
<p>
Check your stride turnover. The most efficient
runners take about 180 steps per minute.
</p>
</div>
</div>
<div class="media">

<div class="media-body">
<h4>Change it up</h4>
<p>
Don't run the same every time you hit the
road. Vary your pace, and vary the distance
of your runs.
```

```

</p>
</div>
</div>
<div class="media">

<div class="media-body">
<h4>Focus on form</h4>
<p>
Run tall but relaxed. Your feet should hit
the ground beneath your hips, not out in
front of you.
</p>
</div>
</div>
</div>
</main>
</div>
</body>

```

Questo elenco ti dà la struttura della pagina: un'intestazione e un elemento principale che conterrà il resto della pagina. All'interno dell'elemento principale c'è il titolo della pagina, seguito da un div anonimo (cioè un div senza classe o ID). Questo serve a raggruppare i quattro elementi multimediali grigi, ognuno dei quali contiene un'immagine e un elemento del corpo. Di solito è più semplice disporre prima le grandi aree di una pagina, quindi procedere verso gli elementi più piccoli all'interno. Prima di iniziare a far fluttuare gli elementi, posiziona la struttura esterna della pagina. Aggiungi l'elenco seguente al tuo foglio di stile:

```

:root {
  box-sizing: border-box;
}

```

*,

```

::before,
::after {
  box-sizing: inherit;
}

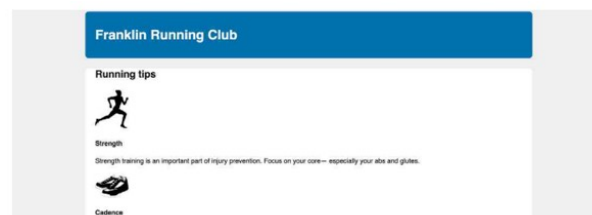
```

```

body {
background-color: #eee;
font-family: Helvetica, Arial, sans-serif;
}
body * + * {
margin-top: 1.5em;
}
header {
padding: 1em 1.5em;
color: #fff;
background-color: #0072b0;
border-radius: .5em;
margin-bottom: 1.5em;
}
.main {
padding: 0 1.5em;
background-color: #fff;
border-radius: .5em;
}

```

Questo imposta alcuni stili di base per la pagina, tra cui una correzione per il ridimensionamento della scatola e un gufo (vedi capitolo precedente). Successivamente, vorrai limitare la larghezza del contenuto della pagina, mostrato nell'immagine seguente. Devi fare in modo che i margini grigio chiaro su entrambi i lati e come sia l'intestazione che il contenitore principale abbiano la stessa larghezza all'interno.



Questo layout è comune per centrare il contenuto su una pagina. Puoi ottenerlo posizionando il tuo contenuto all'interno di due contenitori nidificati e quindi impostando i margini sul contenitore interno per posizionarlo all'interno di quello esterno.



Lo sviluppatore Web Brad Westfall lo chiama il modello a doppio contenitore.

Nel nostro esempio, `<body>` funge da contenitore esterno. Per impostazione predefinita, questo è già al 100% della larghezza della pagina, quindi non dovrai applicarvi nuovi stili. Al suo interno, hai racchiuso l'intero contenuto della pagina in un `<div class="container">`, che funge da contenitore interno. A ciò applicherai una larghezza massima e margini automatici per centrare i contenuti. Aggiungi questo al tuo foglio di stile:

```
.container { max-width: 1080px; margin: 0 auto; }
```

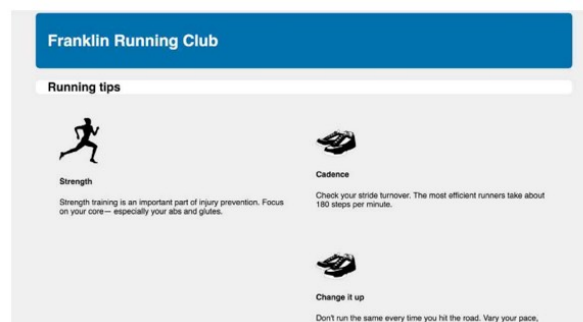
Usando `max-width` invece di `width`, l'elemento si riduce a meno di 1080 px se il viewport dello schermo è più piccolo di quel valore. Vale a dire, nelle finestre più piccole, il contenitore interno riempirà lo schermo, ma in quelle più grandi si espanderà a 1080 px. Questo è importante per evitare lo scorrimento orizzontale su dispositivi con schermi più piccoli.

Hai ancora bisogno di sapere come usare i float? Flexbox sta rapidamente soppiantando l'uso dei float per il layout della pagina. Il suo comportamento è semplice e spesso più prevedibile per i nuovi sviluppatori. Potresti trovarti a chiederti se hai bisogno di sapere qualcosa riguardo i float. CSS li ha superati? Con i browser moderni, puoi sicuramente andare molto oltre senza float rispetto a quanto potresti fare in passato. Probabilmente puoi cavartela del tutto senza float. Ma se hai bisogno di supportare Internet Explorer, potresti averne ancora bisogno per il momento. Flexbox è supportato in IE 10 e 11 e anche in questo caso presenta alcuni bug. Se non vuoi preoccuparti dei bug del browser o devi supportare browser meno recenti, i float potrebbero essere un'opzione migliore. Se stai supportando una base di codice obsoleta, probabilmente utilizza float; dovrai sapere come funzionano per la manutenzione. Inoltre, i layout basati su float spesso richiedono meno markup, mentre i metodi più recenti richiedono l'aggiunta di elementi contenitori. Se hai un controllo limitato sul markup che stai disegnando, i float potrebbero essere in grado di fare ciò di cui hai bisogno. E i float sono ancora l'unico modo per

spostare un'immagine sul lato della pagina e consentire al testo di avvolgerla.

Collapse e clearfix

In passato, i bug del browser hanno afflitto il comportamento dei float, anche se principalmente in IE 6 e 7. È quasi certo che non hai più bisogno di supportare questi browser; quindi, non devi preoccuparti di quei bug. Ora puoi fidarti che i browser gestiranno i float in modo coerente. Tuttavia, alcuni comportamenti dei float potrebbero prenderti alla sprovvista. Questi non sono bug, ma piuttosto float che si comportano esattamente come dovrebbero comportarsi. Diamo un'occhiata a come funzionano e come puoi modificare il loro comportamento per ottenere il layout che desideri. Sulla tua pagina, facciamo fluttuare le quattro caselle multimediali a sinistra. I problemi diventeranno immediatamente evidenti:

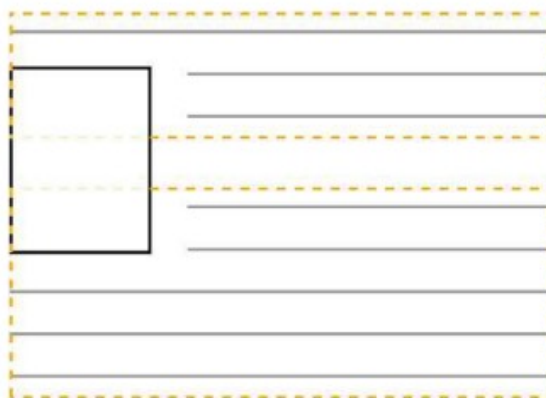


Che fine ha fatto lo sfondo bianco? Lo vediamo dietro il titolo della pagina ("Suggerimenti per la corsa"), ma si ferma lì invece di estendersi fino a comprendere i riquadri multimediali. Per vederlo sulla tua pagina, aggiungi le seguenti regole al tuo foglio di stile. Quindi vedremo perché questo accade e come puoi risolverlo.

```
.media {  
  float: left;  
  width: 50%;  
  padding: 1.5em;  
  background-color: #eee;  
  border-radius: 0.5em;  
}
```

Hai impostato uno sfondo grigio chiaro su ogni media box, aspettandoti di vedere lo sfondo bianco del contenitore dietro (o meglio, intorno) di loro.

Invece, lo sfondo bianco si è fermato sopra la riga superiore dei riquadri multimediali. Perché è successo ciò? Il problema è che, a differenza degli elementi nel normale flusso di documenti, gli elementi mobili non aggiungono altezza ai loro elementi principali. Questo può sembrare strano, ma risale allo scopo originale dei float. Come hai appreso all'inizio di questo capitolo, i float hanno lo scopo di consentire al testo di avvolgerli. Quando fai fluttuare un'immagine all'interno di un paragrafo, il paragrafo non cresce per contenere l'immagine. Ciò significa che, se l'immagine è più alta del testo del paragrafo, il paragrafo successivo inizierà immediatamente sotto il testo del primo e il testo in entrambi i paragrafi si avvolgerà attorno al float. Ciò è illustrato nell'immagine seguente:



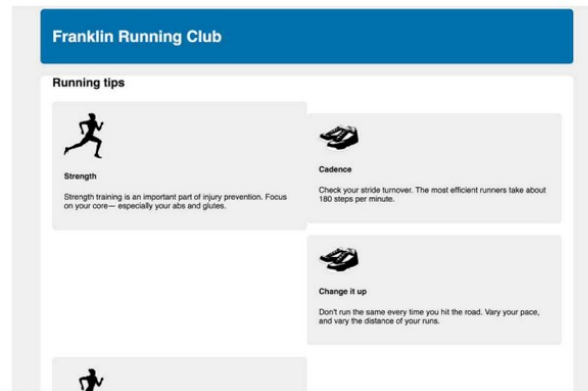
Nella tua pagina, tutto all'interno dell'elemento principale è mobile tranne il titolo della pagina, quindi solo il titolo della pagina contribuisce all'altezza del contenitore, lasciando tutti gli elementi multimediali mobili che si estendono sotto lo sfondo bianco. Questo non è il comportamento che vogliamo, quindi risolviamolo. L'elemento principale dovrebbe estendersi verso il basso per contenere i riquadri grigi. Un modo per correggere questo è con la proprietà `clear`. Se si posiziona un elemento all'estremità del contenitore principale e si usa `clear`, il contenitore si espande fino al fondo dei float. Il codice nel prossimo snippet mostra, in linea di principio, cosa vogliamo fare. Puoi aggiungerlo temporaneamente alla tua pagina per vedere come funziona.

```
<main class="main">
```

```
...
```

```
<div style="clear: both"></div>
```

</main>

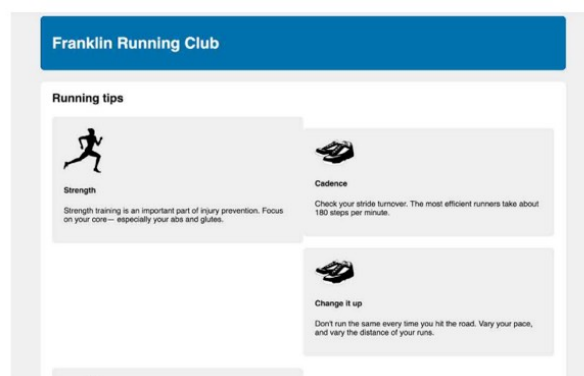


La dichiarazione `clear: both` fa sì che questo elemento si sposti sotto il fondo degli elementi `float`, piuttosto che accanto ad essi. È possibile assegnare a questa proprietà il valore `left` o `right` per cancellare solo gli elementi fluttuanti rispettivamente a sinistra o a destra. Poiché questo div vuoto non è `float`, il contenitore si estenderà fino a racchiuderlo, contenendo così anche i `float` sopra di esso. Questo ridimensiona il contenitore come desideri, ma è piuttosto un trucco; stai aggiungendo markup indesiderato al tuo HTML per fare il lavoro che dovrebbe essere fatto dal CSS. Elimina quel div vuoto e diamo un'occhiata ad un modo in cui puoi farlo esclusivamente tramite il tuo CSS. Invece di aggiungere un div extra al tuo markup, utilizzerai uno pseudo-elemento. Utilizzando `::after` come selettore di pseudo-elementi, puoi inserire efficacemente un elemento nel DOM alla fine del contenitore, senza aggiungerlo al markup. Vediamo un approccio comune al problema del contenimento dei `float`, chiamato `clearfix`. (Alcuni sviluppatori amano abbreviare il nome della classe in `cf`, che è anche un'abbreviazione per "contain floats.") Aggiungi questo al tuo foglio di stile:

```
.clearfix::after {  
  display: block;  
  content: " ";  
  clear: both;  
}
```

È importante sapere che il `clearfix` viene applicato all'elemento che contiene i `float`; un errore comune è applicarlo all'elemento sbagliato, come i `float` o il contenitore dopo quello che li contiene. Il `clearfix` ha subito

dozzine di modifiche nel corso degli anni, alcune più complicate di altre. Molte versioni avevano sfumature per correggere vari bug del browser. La maggior parte delle soluzioni alternative non è più necessaria, sebbene in questo esempio sia presente una soluzione alternativa: lo spazio nel valore del contenuto. Anche una stringa vuota ("") può funzionare, ma il carattere spazio risolve un oscuro bug nelle vecchie versioni di Opera. Tendo a lasciare questa correzione perché è discreta. Rimane un'incoerenza con questo clearfix: i margini degli elementi flottanti all'interno non collassano all'esterno del contenitore clearfix; ma i margini degli elementi non mobili collassano normalmente. Puoi vederlo nella tua pagina dove la voce “Suggerimenti per la corsa” è schiacciata direttamente contro la parte superiore del <main> bianco; il suo margine è crollato fuori dal contenitore.



Alcuni sviluppatori preferiscono utilizzare una versione modificata del clearfix che conterrà tutti i margini perché può essere leggermente più prevedibile. L'aggiunta di questa versione alla tua pagina impedirà al margine superiore del titolo della pagina di collassare al di fuori del principale, come mostrato nella figura sopra, lasciando una spaziatura appropriata sopra l'intestazione. Per la versione modificata, aggiorna il clearfix nel tuo foglio di stile in modo che corrisponda a questo elenco.

```
.clearfix::before,  
.clearfix::after {  
display: table;  
content: " ";  
}  
.clearfix::after {  
clear: both;
```

}

Questa versione utilizza `display: table` anziché `display: block`. Applicando questo a entrambi gli pseudo-elementi `::before` e `::after`, conterrai i margini di qualsiasi elemento figlio sia nella parte superiore che in quella inferiore del contenitore. Questa versione di `clearfix` funge anche da metodo utile per evitare il collasso dei margini dove non lo vogliamo. Quale versione di `clearfix` utilizzare nei tuoi progetti dipende da te. Alcuni sviluppatori sostengono che il “far collassare i margini” è una caratteristica fondamentale dei CSS, quindi preferiscono non contenere i margini. Ma, poiché nessuna delle versioni contiene i margini degli elementi mobili, altre preferiscono il comportamento più coerente della versione modificata. Ogni argomento ha il suo merito.

Clearfix e display: table

L'utilizzo di `display: table` nel clearfix comporta dei margini a causa di alcune particolarità dei CSS. La creazione di un elemento `display: table` (o, in questo caso, pseudo-elemento) crea implicitamente una riga di tabella all'interno dell'elemento e una cella di una tabella al suo interno. Poiché i margini non collassano attraverso gli elementi della cella della tabella (come menzionato precedentemente), non collassano nemmeno attraverso uno pseudo elemento della tabella di visualizzazione. Potrebbe sembrare, quindi, che tu possa usare `display: table-cell` con lo stesso effetto. Tuttavia, la proprietà `clear` funziona solo se applicata agli elementi a livello di blocco. Una tabella è un elemento a livello di blocco, ma una cella di tabella non lo è; quindi, la proprietà `clear` non può essere applicata insieme a `display: table-cell`. Pertanto, è necessario utilizzare `display: table` per cancellare i float e la sua cella di tabella implicita per contenere i margini.

Per molti sviluppatori web, CSS è un linguaggio intimidatorio. Ha un piede nel mondo del design e un altro nel mondo del codice. Alcune parti del linguaggio non sono intuitive, soprattutto se sei un autodidatta nella materia. Spero che questo libro ti abbia aiutato a trovare la tua strada. Abbiamo esaminato a fondo le parti fondamentali del linguaggio e alcune delle parti più confuse del layout di pagina. Abbiamo approfondito molti argomenti, dall'organizzazione dei CSS per una più semplice manutenzione del codice ai metodi di layout più recenti. Ci siamo avventurati nel mondo del design e abbiamo costruito un'interfaccia non solo utile, ma anche intuitiva e divertente. Il mio ultimo consiglio per voi è di rimanere curiosi. Ti ho mostrato una gamma di strumenti nel set di strumenti CSS ma i modi in cui questi strumenti possono essere combinati e abbinati sono infiniti. Quando incontri una pagina web che ti stupisce, apri il DevTools del tuo browser e prova a capire come funziona, come è fatta e se, in qualche modo, potrebbe essere migliorata. Segui sviluppatori e designer online che forniscono demo creative o offrono tutorial interessanti, puoi farlo via Github, Instagram e tanti altri social, i canali non mancano. Prova nuovi strumenti, nuovi approcci di programmazione. E continua ad imparare!