Html, Css

In un file CSS ci riferiamo agli elementi scritti in Html nel seguente modo:

- Per gli **id**:

```
#par1 {
    color: red;
    text-align: center;
}
```

- Per le **classi**:

```
<!-- così ci riferiamo a tutti gli elementi della classe specif

.center {
    font-family: Arial;
    font-weight: bolder;
}

<!-- così ci riferiamo agli elementi di questo tipo appartenent

p.center {
    font-style: italic;
}
```

- Per ogni elemento della pagina:

```
* {
    text-align: left;
}
```

- Per i singoli elementi:

```
h1 {
  text-align: center;
  color: red;
}

<!-- o anche per più elementi insieme-->

h1, h2, p {
  text-align: center;
  color: red;
}
```

Background

Le proprietà background sono utilizzate per aggiungere effetti background agli elementi:

▼ background-color

```
<!-- specifica il colore di sfondo di un elemento -->

body {
    background-color: white;
}
```

▼ background-image

```
<!-- permette di utilizzare un'immagine come sfondo -->

body {
    background-image: url("desert.jpg");
```

▼ background-repeat

```
<!-- è una proprietà per definire la eventuale ripetizione de

body {
    background-image:
    background-repeat: repeat-x;
}

<!-- repeat-x : l'immagine viene solo ripetuta orizzontalment
evitare le ripetizioni -->
```

▼ background-attachment

```
<!-- specifica se l'immagine d background deve essere fissa

body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: fixed;       <!-- oppure scroll -->
}
```

▼ background-position

```
<!-- per posizionare l immagine correttamente all' interno de

body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

▼ background

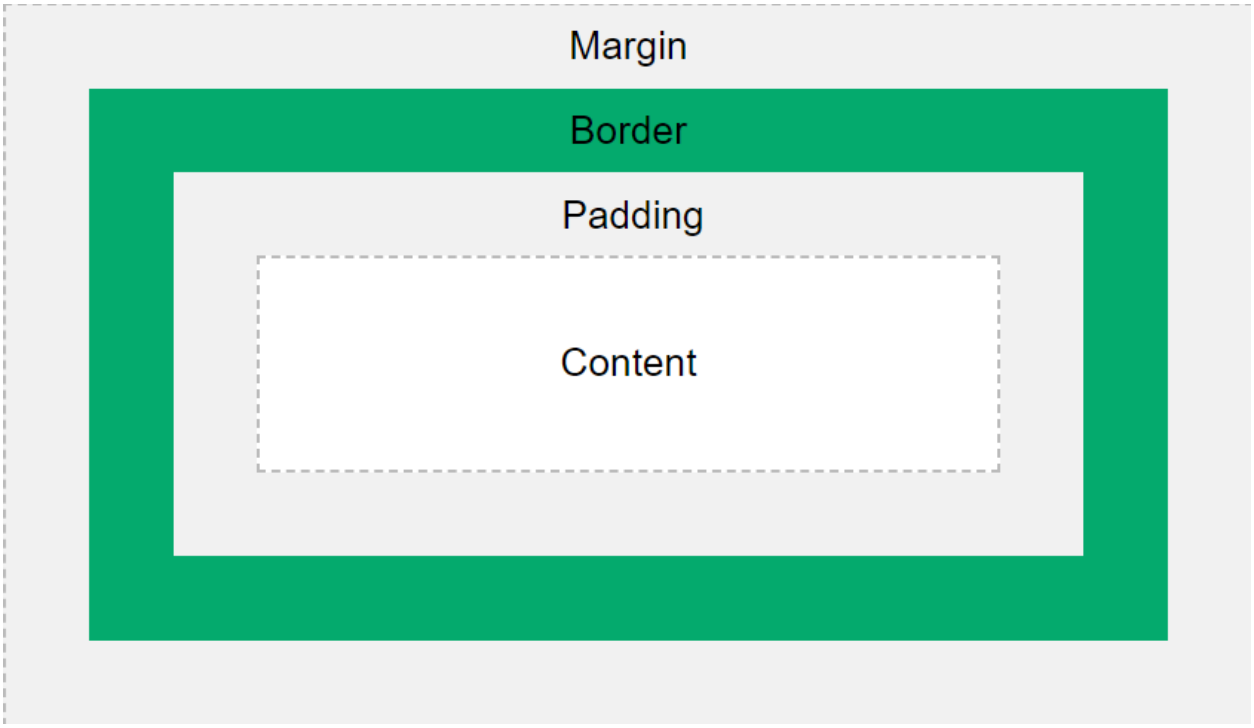
Oppure possiamo raggruppare tutte le proprietà appena viste sotto la specifica `background`:

```
body {
  background: #ffffff url("img_tree.png") no-repeat right to
}

<!-- devono essere in questo ordine, non importa se manca qu
```

La proprietà `opacity` specifica il livello di opacità/ trasparenza di un elemento. Prende valori da 0.0 a 1.0: più il valore è basso, più c'è trasparenza

Box Model



In CSS il termine “ box model” è utilizzato in riferimento al design ed al layout di una pagina.

▼ **Margini**

CSS `margin` serve per creare spazio intorno agli elementi., fuori dai bordi dell’elemento Possiamo sia regolarli manualmente tutti e quattro (*top right bottom left*) che dare un solo valore per tutti.

▼ **Padding**

CSS `padding` serve per creare spazio intorno al contenuto degli elementi, dentro i bordi. Come per `margin` possiamo scegliere se regolari manualmente tutti e 4 oppure assegnare un valore per tutti.

▼ **Height & Width**

CSS `height` and `width` servono per impostare l’altezza e la larghezza di un elemento

▼ **Border**

CSS `border` ci consente di specificare lo stile, lo spessore e il colore del bordo di un elemento. Di default il bordo non è visibile. *tra i tanti stili possibili, ricordiamo lo stile “solid” che crea una bella riga di separazione. per avere un bordo smussato ai lati utilizziamo la proprietà “border-radius”.*

Impaginazione

Iniziamo vedendo la proprietà
`position`: essa ci permette di specificare la posizione di un qualsiasi elemento all’interno della pagina html.

▼ **static**

E’ il valore di default: gli elementi appaiono per come sono stati dichiarati nel codice html.

▼ **relative**

Specifichiamo la nuova posizione dell’oggetto, a partire dalla posizione iniziale, andando ad agire su *top, bottom, left, right*.

▼ **absolute**

L’elemento che ha questa proprietà risponde come “elemento assoluto”, nel senso che non partecipa al flusso della pagina, ma è fisso, facendo riferimento alla sua posizione di default. Agendo su *top, bottom, left* e *right*, notiamo che è relativo al ”genitore”, cioè l’ultimo elemento di tipo relative che lo contiene. *fa sempre riferimento a qualcosa che lo contiene: ad esempio se ho un div position: relative; e un p all’interno position: absolute, quest’ultimo farà riferimento a div.*

▼ **sticky**

L’elemento scorre nella pagina, e si fissa nel momento in cui raggiunge l’offset da noi specificato. Rimane all’interno del suo genitore e , una volta che il genitore finisce la pagina, se ne va con lui.

▼ **fixed**

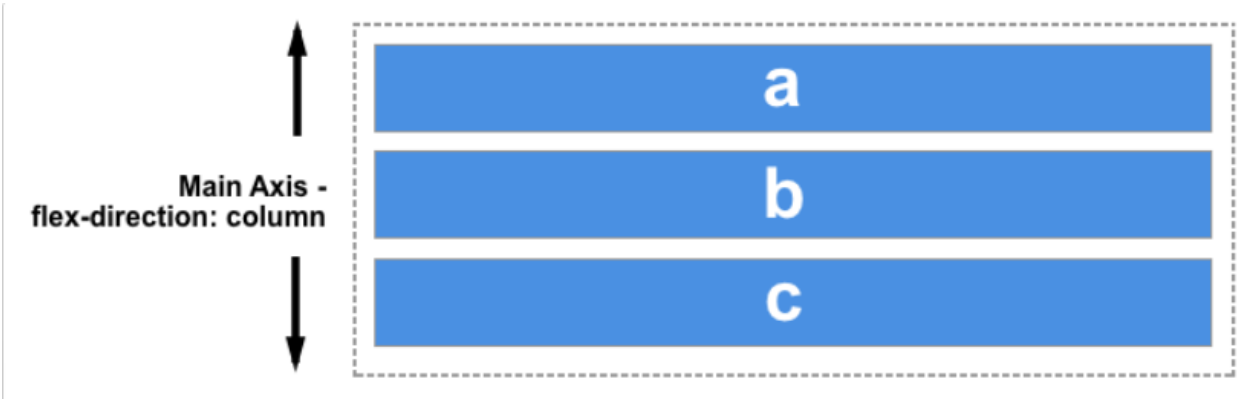
L’elemento rimane fisso nella pagina, sin dall’inizio, indipendentemente dall’offset specificato, per tutta la pagina. (*nel senso che si, rispetta l’offset, ma non si muove sin dal principio*)

Flexbox

E’ un modello di impaginazione monodimensionale, nel senso che lo utilizziamo per lavorare o a livello di riga o a livello di colonna.

Quando ragioniamo con le flexbox dobbiamo definire la **main area** tramite la proprietà `flex-direction`, e di conseguenza la **cross-area** che è, per definizione, perpendicolare alla main.

I possibili valori di `flex-direction` sono: *row, row-reverse, column* e *column-reverse*.



Un area di una pagina organizzata tramite flexbox è detta **flex container**. Per creare un flex container, settiamo la proprietà `display` come `flex` o `inline-flex`. Appena facio ciò, tutti gli elementi figli del container diventano **flex-items.**

Tramite la proprietà `flex-wrap` possiamo decidere se i **flex-items** possono fare wrap o meno. In particolare, settando questa proprietà, se le dimensioni dei flex-items eccedono la page, vengono posizionati su un'altra riga.

Possiamo combinare queste due proprietà appena viste, *flex-wrap* e *flex-direction*, in una sola: `flex-flow`.

Per avere più controllo sugli elementi possiamo inoltre utilizzare queste 3 proprietà:

- `flex-grow`: specifica quanto deve crescere l'elemento flessibile quando c'è spazio nel contenitore (*flessibile*)
- `flex-shrink`: specifica come deve restringersi l'elemento quando inizia a diminuire lo spazio nel contenitore (*flessibile*)
- `flex-basis`: specifica la dimensione iniziale prima che lo spazio residuo venga ridistribuito

Queste proprietà vanno applicate direttamente agli oggetti interessati: con queste andiamo a cambiare il modo in cui lo spazio disponibile è distribuito attorno ai nostri elementi.

Possiamo raggruppare queste proprietà con: `flex: <flex-grow> <flex-shrink> <flex-basis>`

La proprietà `align-items` allinea gli elementi sul cross-axis. Il valore di default è `stretch` ed è per questo che gli elementi si "allungano" su tutto il container, indipendentemente da "quanto occupano" (*pensiamo ad esempio a un p con una parola e un p con 2 righe: con align-items: stretch; entrambi i p avranno la stessa altezza*). I possibili valori sono *stretch* , *flex-start* (*allineamento a dimensione variabile che parte dall'alto*), *flex-end* (*allineamento a dimensione variabile che parte dal basso*) e *center* (*allineamento a dimensione variabile che parte dal centro*)

La proprietà `justify-content` è utilizzata per allineare gli elementi sul main-axis. Il valore iniziale è `flex-start` che posiziona gli elementi all'inizio del bordo del container. I valori sono diversi:

- `flex-start`: default
- `flex-end`: posiziona gli elementi alla fine del bordo del container
- `center`: posiziona gli elementi al centro
- `space-around` : determina uno spazio x uguale tra gli elementi e x/2 ai bordi
- `space-between`: determina uno spazio uguale tra gli elementi (*non ai bordi*)

- `space-evenly`: determina uno spazio uguale sia tra gli elementi che ai bordi

CSS Grid

Come suggerisce il nome, con la Grid dividiamo la pagina in varie regioni e definiamo una relazione in termini di dimensioni, posizioni e layout tra gli elementi. Gli elementi vengono allineati tramite righe e colonne.

Scriviamo `display: grid` per impostare questo tipo di layout. Possiamo utilizzare anche `display: inline-grip`. Qual è la differenza?? che `grid` crea un contenitore di blocco che occupa tutto lo spazio disponibile e va a capo dopo di se, mentre `inline-grid` crea un contenitore di linea, il che significa che non va a capo e può essere allineato orizzontalmente insieme ad altri elementi di linea. A questo punto utilizziamo tutte le varie proprietà del caso:

- `gap`: che può essere utilizzato così per specificare lo spazio tra gli elementi sia tra righe che colonne, oppure si "divide" in `column-gap` e `row-gap`.
- `grid-auto-columns`: specifica la dimensione di default delle colonne.
- `grid-auto-rows`: specifica la dimensione di default delle riga.
- `grid-column`: specifica dove inizia / finisce la l'elemento della griglia.
- `grid-row`: specifica dove inizia / finisce l'elemento della griglia.
- `grid-template-columns`: specifica la dimensione delle colonne, e quante ce ne sono.
- `grid-template-rows`: specifica la dimensione delle righe e quante ce ne sono.

Anche qui, come nelle flexbox, sono presenti le proprietà `align-items` `justify-content` che funzionano pressocché allo stesso modo.

C'è inoltre `align-content` che funziona come `align-items` ma per il contenitore (*l'intera griglia*) all'interno della pagina

Proprietà utili

Per rendere ancora più carina la pagina possiamo utilizzare proprietà “dinamiche” nella nostra pagina di stile. Ad esempio:

- **focus**

  Quando facciamo click col mouse su un elemento, ad esempio una casella di testo.

```
input[type=text]:focus {
  background-color: lightblue;
  border-color: blue;
}
<!-- in questo modo le caselle di testo cambiano colore quand
```

- **hover**

  Quando passiamo col mouse sopra a un certo elemento, ad esempio un bottone:

```
input[type=button]:hover {
  background-color: lightgreen;
  border-color: white;
}
<!-- in questo modo il bottone cambia colore quando ci passi
```