Pour cela, le CSS définit un système de priorités.

REGLE 1 : Les règles qui sont les plus précises l'emportent sur les plus générales.

REGLE 2 : Si deux règles CSS ont la même spécificité, alors la dernière l'emporte.

Ainsi, la première règle a un sélecteur "p", ce qui est assez général. Les deux autres règles s'efforcent de restreindre le plus possible leur champs d'action, c'est donc l'une des deux dernières qui sera appliquée. Mais là encore, laquelle ?

Les styles restant en concurrence ont un degré de priorité variable, dépendant du sélecteur CSS utilisé et de sa syntaxe. Ce degré de priorité est calculé sous forme d'un nombre à 4 chiffres `ABCD` :

- `A` : `A`=1 s'il ne s'agit pas d'un sélecteur CSS mais d'un attribut style. Sinon, `A`=0. De cette manière, un style placé dans l'attribut `style` d'un élément HTML aura systématiquement un poids plus élevé qu'un style auteur placé dans une feuille de style (`ABCD` = 1000, la plus haute valeur possible) ;
- `B` : si `A` vaut 0, `B` est le nombre d'`id`s présents dans le sélecteur. Par exemple, `B`=2 pour le style `#conteneur p#special {...}` ;
- `C` : si `A` vaut 0, `C` est le nombre de classes (du type `.ma_classe)` et de pseudo-classes (du type `[hreflang=en]`) dans le sélecteur. Par exemple, `C`=2 pour le style `.article p a[hreflang=en] {...}` ;
- `D` : si `A` vaut 0, `D` est le nombre d'éléments HTML et de pseudo-éléments (du type `:before` et `:after`, `:first-line`, etc) dans le sélecteur. Par exemple, : `D`=2 pour le style `a:after {...}`.

Voici quelques exemples de sélecteurs et de règles CSS classés par ordre croissant de spécificité :

1. `* {...}` : 0000 (aucun idenfifiant, aucune classe, aucun élément) ;
2. `p {...}` : 0001 (aucun idenfifiant, aucune classe, un élément) ;
3. `blockquote p {...}` : 0002 (aucun idenfifiant, aucune classe, deux éléments) ;
4. `.class {...}` : 0010 (aucun idenfifiant, une classe, aucun élément) ;
5. `p.class {...}` : 0011 (aucun idenfifiant, une classe, un élément) ;
6. `blockquote p.class {...}` : 0012 (aucun idenfifiant, une classe, deux éléments) ;
7. `#id {...}` : 0100 (un idenfifiant, aucune classe, aucun élément) ;
8. `p#id {...}` : 0101 (un idenfifiant, aucune classe, un élément) ;
9. `blockquote p#id {...}` : 0102 (un idenfifiant, aucune classe, deux éléments) ;
10. `.class #id {...}` : 0110 (un idenfifiant, une classe, aucun élément) ;
11. `.class p#id {...}` : 0111 (un idenfifiant, une classe, un élément) ;
12. `blockquote.class p#id {...}` : 0112 (un idenfifiant, une classe, deux éléments) ;
13. `<p style="...">` : 1000 (attribut HTML `style` qui ne sera supplanté que par un style utilisateur normal) ;
14. `<p style="... !important">` : 1000 (attribut HTML `style` marqué `!important` qui ne sera supplanté que par un style utilisateur lui-même marqué `!important`).

Si nous revenons à notre exemple des styles de liens, nous pouvons calculer le degré de priorité des deux sélecteurs qui restaient en concurrence :

- `a` a un degré de priorité de 0001 (un nom d'élément) ;
- `#menu a` a un degré de priorité de 0101 (un identifiant et un élément).

Le deuxième sélecteur l'emporte, et les liens du menu seront donc en graisse normale.

Et en dernier ressort

Si, parvenu à ce stade, deux styles de même degré de priorité restent en concurrence, le dernier apparu dans l'ordre linéaire CSS-HTML l'emporte. Autrement dit, toutes choses étant égales par ailleurs :

- Les règles CSS située « *après* » dans une feuille de style l'emportent sur celles situées « *avant* » ;
- Les règles CSS contenues dans des feuilles de styles incorporées à la page Web « *après* » l'emportent sur celles incorporées « *avant* ».

Résultat de la priorité dans le code exemple

- Les styles de l'agent utilisateur n'agiront que dans les domaines où rien n'est spécifié par l'auteur ou l'utilisateur ;
- Les styles `!important` de l'utilisateur l'emportent sur tous autres styles, mais pas ses styles normaux : les auteurs ne devraient donc pas abuser de la règle `!important` ;
- Dans une combinaison de feuilles de style, `!important` donne plus de poids que n'importe quelle accumulation d'`id`, de classes ou de noms d'éléments ;

  Internet Explorer 5.x et 6.0 a cependant ici un bug portant sur la répétition d'une même propriété dans le même bloc, qui a donné lieu à un hack permettant de différencier les styles appliqués par IE de ceux appliqués par Firefox, Opera, Safari, etc :

```
foo {
color: red !important;
color: blue;
}
```

  Le comportement normal (respecté par Opera, Firefox,Safari, etc.) serait ici l'affichage en rouge, puisque le style `!important` est prioritaire, quel que soit l'ordre des propriétés. Mais Internet Explorer ignore dans ce cas `!important`, et *ne retient que la dernière propriété dans l'ordre du code* : le texte concerné par ce style sera donc affiché en bleu.

  Ce hack peut être utilisé en particulier pour indiquer à Internet Explorer des propriétés de dimensions compatibles avec son modèle de boîte CSS propriétaire et son implémentation défectueuse des propriétés de hauteur et de largeur minimales, etc. cependant, le risque potentiel est qu'une propriété utilisateur elle-même dénuée de `!important` ne sera alors pas prioritaire sur la propriété auteur ;

  - Un seul `id` donne à un sélecteur plus de poids que n'importe quelle accumulation de classes ou de noms d'éléments ;
  - Une seule classe donne à un sélecteur plus de poids que n'importe quelle accumulation de noms d'éléments ;
  - La présence de combinateurs + (sélecteurs d'éléments adjacents) ou > (sélecteurs d'enfants) n'a aucune influence sur la spécificité.

Pour terminer cette section sur les priorités des sélecteurs, répondront à la question existentielle posée un peu plus haut : de quelle couleur sera le paragraphe vert ? Faisons le calcul :

```
p          { font-weight: bold;   color: red;   } /* a=0 b=0 c=0 d=1 -> spécificité =    1 */

.important { font-size:   16px;   color: green; } /* a=0 b=0 c=1 d=0 -> spécificité =   10 */

div > p    { font-style:  italic; color: blue;  } /* a=0 b=0 c=0 d=2 -> spécificité =    2 */
```

Et oui, le texte sera vert. Le sélecteur ".important" étant plus précis que les deux autres.

Au final, pour ce paragraphe, et uniquement dans ce cas, les trois règles fusionnent pour donner ceci, comme si nous n'avions écrit que cette règle :

```
div > p.important {

  font-weight: bold;

  font-size: 16px;

  font-style: italic;

  color: green;

}
```

Notez enfin que si vous avez plusieurs sélecteurs séparés par des virgules, vous devez calculer la priorité de chacun de ces sélecteurs individuellement, comme s'ils avaient été définis séparément. Vous ne devez pas cumuler les priorités des sélecteurs séparés par des virgules.