

# Bootcamp Java Developer

Fase 2 - Java Web Developer  
Módulo 14

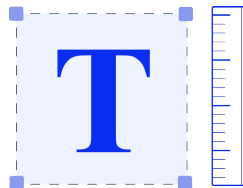
# Funciones en CSS

## Uso de Clamp()

A veces el trabajo con tipografías fluidas utilizando la regla `@media`, tampoco es del todo óptima y sigue generando problemas en ciertos *breakpoints* intermedios.

Es allí cuando aparece una tercera opción de trabajo: la función `clamp()`. Permite hacer un trabajo más amplio donde **se utiliza el tamaño mínimo, máximo y el preferido** al momento de trabajar, como se muestra a la derecha. En la pantalla siguiente encontrarás un diagrama que explica la sintaxis de la función `clamp()`.

```
h2 {  
  font-variation-settings: 'wght' 200;  
  text-transform: capitalize;  
  font-size: clamp(1rem, 3vw, 2.25rem);  
}
```



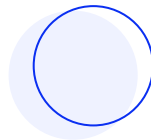
# Función Calc()

## Implementación

Al inicio de este curso, mencionamos la función **calc()** pero no la vimos en profundidad. **Esta función permite** (sin necesidad aún de trabajar con SASS o LESS), **integrar cálculos a la hoja de estilo**. De esta manera podemos realizar una suma, resta, multiplicación o división en nuestro propio CSS.

Supongamos que tenemos una estructura que queremos dividir en tres partes, como en el ejemplo de la derecha.

```
div {  
  background-color: blue;  
  width: calc( 100% / 3) ;}
```



## Sintaxis de la función clamp()

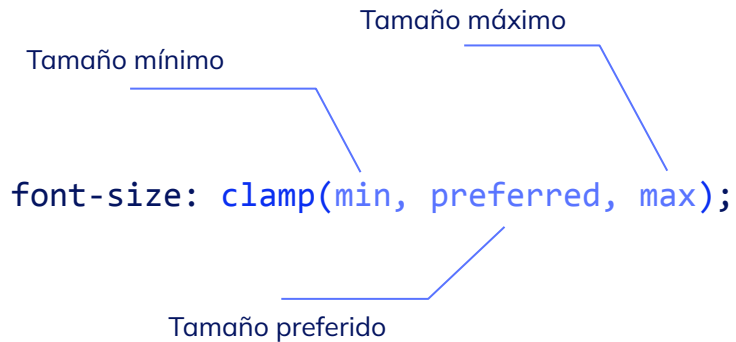


Diagram illustrating the syntax of the `clamp()` function:

```
font-size: clamp(min, preferred, max);
```

The arguments are labeled as follows:

- Tamaño mínimo (Minimum size) points to `min`.
- Tamaño máximo (Maximum size) points to `max`.
- Tamaño preferido (Preferred size) points to `preferred`.

## Para tener en cuenta...

Existen muchas páginas que pueden ayudarnos a evaluar el comportamiento de nuestra tipografía, por ejemplo [modern-fluid-typography.vercel.app/](https://modern-fluid-typography.vercel.app/)

Allí, podemos ver un gráfico con variaciones para determinar cómo se comportará nuestra tipografía en diferentes tamaños o **breakpoints**.



Si consideramos un sistema de grilla con 20px de espacio entre columna, si bien podríamos trabajar con **flex-grow** y **gap**, también es posible hacerlo con **calc()** así:

```
index.html X # estilos.css X
css > # estilos.css > ...
1  * { box-sizing: border-box;}
2
3  main { display: flex;
4         height: 50vh;
5         justify-content: space-between;
6
7
8  }
9
10 div {
11     background-color: blue;
12     width: calc(( 100% / 3) - 20px) ;}
13
```



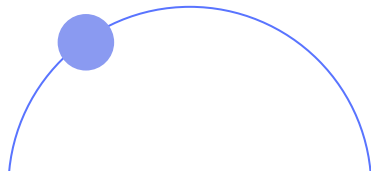
## Uso en tipografía

**calc()** también es útil para trabajar con la tipografía.

Por ejemplo, queremos que nuestra tipografía se adapte al **viewport (usamos vw)**, pero queremos fijar un mínimo y no hacerlo a través de **@media**.

Entonces podemos realizar el siguiente cálculo para que nunca sea menor a 0.7rem.

```
<> index.html # estilos.css X
css > # estilos.css > main
1  p{
2
3      font-size: calc(0.7rem + 1vw);
4
5  }
6  }
```





El uso anterior, incluso, se puede apreciar en el trabajo de uno de los frameworks más conocidos: **Bootstrap**, que fija un mínimo pero adapta la tipografía al ancho de la pantalla del **viewport** como metodología responsiva.



```
Styles  Computed  Layout  >>
Filter  :hov .cls + [icon] [icon]
element.style {
}
.h2, h2 {                                     _rfs.scss:298
☒ font-size: calc(1.325rem + .9vw);
}
```

## Transiciones + calc()

**calc()** puede ser una herramienta muy útil en la creación de transiciones CSS porque permite calcular movimientos o cambios en sus elementos mediante sencillas operaciones.

Estas operaciones se realizan en base, por ejemplo, al total de ancho de cierto contenedor menos o más el ancho del propio contenedor en movimiento para así lograr mejores y más precisos resultados.

```
#box2 {  
  background: ■ rgba(236, 252, 100, .75);  
  left: calc(50% - 6.25em);  
  top: calc(50% - 12.5em);  
}
```

De esta forma, **un elemento con posicionamiento absoluto puede trasladarse, en una transición, de un lugar a otro, de forma más exacta** que si indicamos un valor específico.

Por ejemplo, quiero trasladarlo al 50% de la pantalla pero restarle el tamaño del elemento, entonces.

```
left: calc(100% - 150px);
```

De esta forma el proceso que podría resultar engorroso se realiza de manera muy sencilla.



**¡Sigamos  
trabajando!**