```
Clean Code {
  [ASW Grupo 5];
        < Rubén Rubio Del Castillo UO276255 >
        < Pablo Barrero Cruz U0289642 >
        < Alfonso González-Lamuño de la Calle U0276976 >
```



```
01-Definición.html
```

01-Contenido.html

```
Definición 'Contenido' {
        Facil de leer y de mantener;
                Robusto;
            02
                       Flexible;
12
```

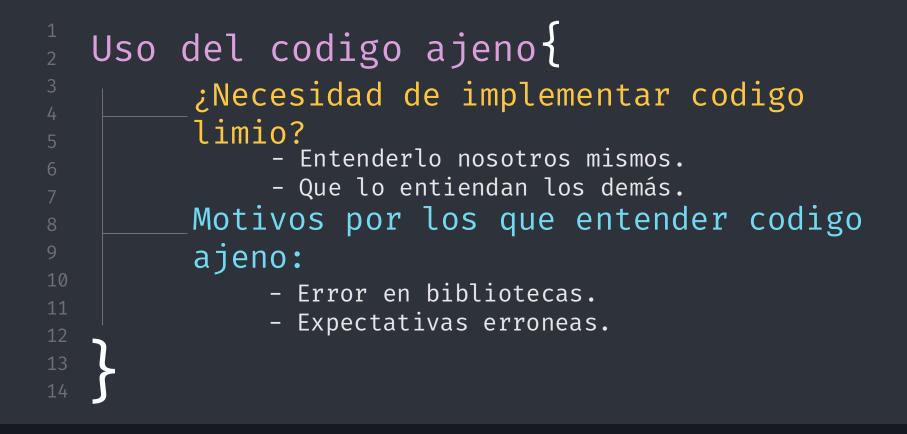
```
L Código limpio vs
Rendimiento ];
```

```
<sup>1</sup>Mas limpio no significa mejor </1>
      < Los principios del código limpio a veces
      pueden afectar al rendimiento >
         < Los programas no siempre estan limitados
         por la E/S >
```

La seguridad también puede ser un problema {

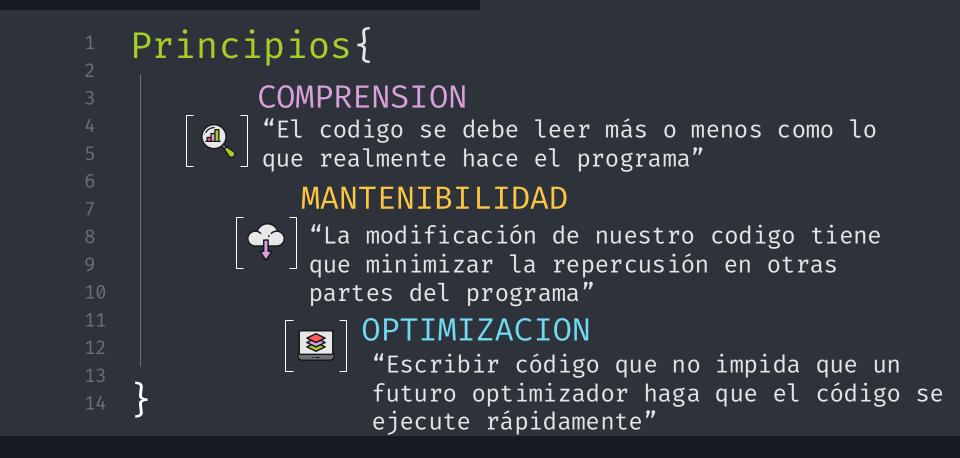
```
< Ocultar detalles puede dificultar la comprens
ión y el rendimiento >
  < Dar detalles de bibliotecas
    o frameworks ayuda a la optimización y a
    conocer el sistema >
```

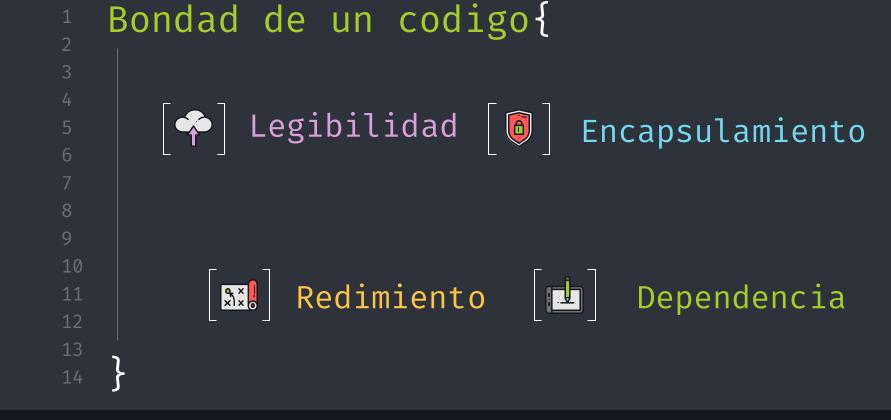
```
1. Foreing Code: {
            ¿Que necesidad tenemos de
   Step 01
            implementar codigo limpio?
               Motivos por los que necesitaríamos
      Step 02
               entender codigo ajeno.
         Step 03 Legibilidad vs Comprensibilidad
```



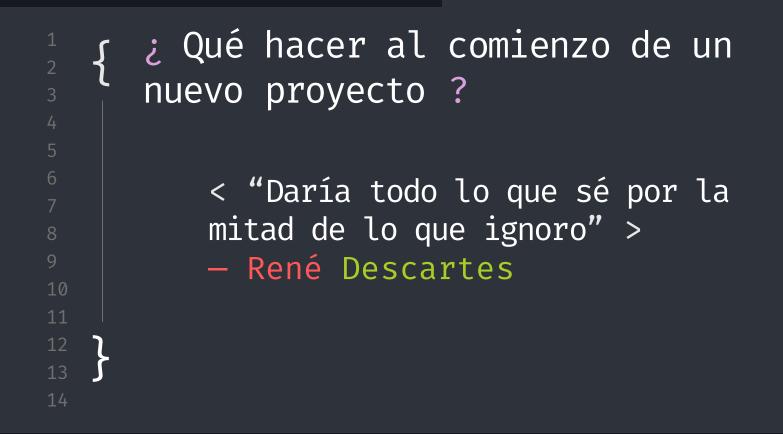
```
Legibilidad </1> {
       < Se refiere a la facilidad con la que una
       persona puede comprender y seguir el código
       fuente.
Comprensibilidad </2>
       < Se refiere a la capacidad de entender el
   👚 | propósito, la lógica y el funcionamiento del
       mismo. >
```







```
(CleanCode === Dinero)?
```



```
Ley de Conway {
< Las organizaciones que diseñan sistemas
se ven obligadas a producir diseños que
son copias de las estructuras de
comunicación de estas organizaciones >
Melvin Conway
                 Microservicios -> Solución
                  para la Ley de Conway
```

Medidas de calidad del código {

```
Rendimiento

Legibilidad

Velocidad a la que puede escribirse

Sigue los estándares
```

CLEAN CODE

```
Optimización {
  < Premature optimization is the</pre>
  root of all evil >
  - Sir Tony Hoare
        No quiere decir que no hace falta
        pensar en el rendimiento
             ¿Dónde optimizar?
```

Decisiones Meditadas {

2	Data Structure	Time Complexity								Space Complexity
3		Average				Worst				Worst
4		Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
5	Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	0(1)	0(n)	0(n)	0(n)	O(n)
6	Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	O(n)	0(1)	0(1)	0(n)
	Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	O(n)
7	Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	0(1)	0(1)	O(n)
8	Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	0(1)	0(1)	O(n)
9	Skip List	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	Θ(log(n))	O(n)	O(n)	0(n)	0(n)	O(n log(n))
10	Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	0(n)	0(n)	O(n)
	Binary Search Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	Θ(log(n))	Θ(log(n))	O(n)	O(n)	0(n)	0(n)	O(n)
12	Cartesian Tree	N/A	$\Theta(\log(n))$	Θ(log(n))	Θ(log(n))	N/A	O(n)	0(n)	0(n)	O(n)
13	B-Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	Θ(log(n))	Θ(log(n))	0(log(n))	O(log(n))	0(log(n))	O(log(n))	O(n)
1	Red-Black Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	Θ(log(n))	Θ(log(n))	0(log(n))	O(log(n))	0(log(n))	O(log(n))	O(n)
J	Splay Tree	N/A	$\Theta(\log(n))$	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
	AVL Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
CLEAN CODE	KD Tree	$O(\log(n))$	$\Theta(\log(n))$	Θ(log(n))	O(log(n))	0(n)	0(n)	0(n)	O(n)	0(n)