Práctica 4 CPLP

1)a. La variable a:

Nombre: a

Alcance: estático, de la línea 3 a la 16.

Tipo: predefinido integer, ligadura estática.

L-Valor: alocación dinámica automática.

R-Valor: Dinámico indefinido.

b. variable P

Nombre: P

Alcance: estático, de la linea 4 a la 16.

Tipo: tipo definido por el usuario puntero, ligadura estática.

L-Valor: alocación dinámica explícita.

R-Valor: dinámico L-Valor de i(linea 8) y L-Valor de a (línea 13)

2)a.

- Ignorar el problema: lo inicializo con lo que haya en memoria.
- Estrategia de inicialización:

_ Inicialización por defecto: enteros se inicializan en 0, los caracteres en blanco, etc.

_ Inicializacion en la declaración: C int i=0, j=1.

b. A continuación, se presenta un cuadro comparativo de las diferentes formas de inicialización de variables en los lenguajes Java, C, Python y Ruby:

Lenguaje	Declaración	Inicialización implícita	Inicialización explícita	Ejemplo
Java	Tipo variable;	No	Sí	`int x = 10;`
C	Tipo variable;	No	Sí	`int x = 10;`
Python	Variable = valor	Sí	Sí	`x = 10`
Ruby	Variable = valor	Sí	Sí	`x = 10`

En Java y C, se declara la variable indicando su tipo y nombre, y se puede inicializar explícitamente en la misma línea de código. No hay inicialización implícita en estos lenguajes.

En Python y Ruby, la variable se puede inicializar implícitamente en el momento en que se asigna un valor. También es posible inicializar explícitamente en una línea de código separada.

En resumen, todos estos lenguajes permiten la inicialización explícita de variables, pero Python y Ruby ofrecen la opción adicional de inicialización implícita. La elección entre estas opciones depende de la situación particular y las preferencias del programador.

3) El atributo l-valor (l-value en inglés) de una variable se refiere a su capacidad de aparecer en la parte izquierda de una asignación, es decir, su capacidad de ser un destino válido para una asignación. Los cuatro conceptos asociados al atributo l-valor son:

a. Variable estática: Una variable estática es aquella cuyo espacio de almacenamiento se reserva durante toda la ejecución del programa. Su atributo l-valor es constante durante toda su vida útil, es decir, siempre se puede asignar un valor a la variable. Un ejemplo de variable estática es una variable global en C:

```
int count = 0; // variable global estática en
```

b. Variable automática o semiestática: Una variable automática es aquella cuyo espacio de almacenamiento se reserva en la pila de ejecución y se libera automáticamente cuando la función que la declaró sale de su ámbito. Su atributo l-valor es constante durante la ejecución de la función que la declaró, es decir, siempre se puede asignar un valor a la variable mientras la función esté en ejecución. Un ejemplo de variable automática es una variable local en C:

```
void foo
  int x; // variable local automática en C
  // ...
}
```

c. Variable dinámica: Una variable dinámica es aquella cuyo espacio de almacenamiento se reserva en tiempo de ejecución y se libera manualmente mediante el uso de la función de liberación de memoria. Su atributo l-valor es variable, es decir, no siempre se puede asignar un valor a la variable si no se ha reservado previamente su espacio de almacenamiento. Un ejemplo de variable dinámica es una variable creada con la función **`malloc()`** en C:

```
int *p = malloc(sizeof(int)); //
```

d. Variable semidinámica: Una variable semidinámica es aquella que combina características de las variables estáticas y dinámicas. Su espacio de almacenamiento se reserva en tiempo de ejecución, pero su liberación se realiza automáticamente al finalizar el ámbito en el que se declaró. Su atributo l-valor es variable, es decir, no siempre se puede asignar un valor a la

variable si no se ha reservado previamente su espacio de almacenamiento. Un ejemplo de variable semidinámica es una variable local declarada con el modificador **`static`** en C:

```
void foo() {
  static int x;
  // ...
}
```

En el lenguaje C, existen cuatro tipos de variables en términos de su l-valor:

1. Variables locales automáticas: Son variables creadas automáticamente cuando se entra en una función y se destruyen automáticamente cuando se sale de ella. Su l-valor es temporal y se encuentra en la pila de llamadas. Estas variables deben ser inicializadas explícitamente, ya que si no lo son, su valor será indeterminado y puede causar errores en el programa.
2. Variables locales estáticas: Son variables que se mantienen en memoria durante toda la ejecución del programa y su l-valor es permanente. Su alcance está limitado a la función donde se declaran y su valor se mantiene entre llamadas a la función. Estas variables son inicializadas automáticamente con el valor cero, pero pueden ser inicializadas explícitamente con otro valor.
3. Variables globales: Son variables que se declaran fuera de cualquier función y su l-valor es permanente. Su alcance es en todo el programa y su valor se mantiene durante toda la ejecución del mismo. Estas variables son inicializadas automáticamente con el valor cero, pero también pueden ser inicializadas explícitamente con otro valor.
4. Variables dinámicas: Son variables que se crean y se destruyen dinámicamente durante la ejecución del programa mediante el uso de funciones como malloc() y free(). Su l-valor es temporal y se encuentra en el heap de memoria. Estas variables deben ser inicializadas explícitamente y liberadas cuando ya no se necesitan.

En el lenguaje Ada, existen tres tipos de variables en términos de su l-valor:

5. Variables locales: Son variables que se declaran dentro de una unidad (función, procedimiento, etc.) y su l-valor es temporal. Estas variables deben ser inicializadas explícitamente.
6. Variables globales: Son variables que se declaran fuera de cualquier unidad y su l-valor es permanente. Su alcance es en todo el programa y su valor se mantiene durante toda la ejecución del mismo. Estas variables son inicializadas automáticamente con el valor cero, pero también pueden ser inicializadas explícitamente con otro valor.
7. Variables por referencia: Son variables que se declaran mediante el uso del operador "&" y su l-valor es el mismo que la variable referenciada. Estas variables no tienen un valor propio, sino que hacen referencia a otra variable y su valor cambia cuando cambia el valor de la variable referenciada.

4) a. Una variable local es aquella que se define dentro de una función o bloque de código y solo está disponible dentro de ese contexto. Es decir, su alcance está limitado a la función o bloque de código en el que fue definida. Las variables locales no pueden ser accedidas desde fuera de la función o bloque en el que fueron declaradas.

Por otro lado, una variable global es aquella que se define fuera de una función o bloque de código y puede ser accedida y modificada desde cualquier parte del programa. Su alcance es global y no está limitado por una función o bloque en particular.

El uso de variables globales debe ser cuidadoso, ya que su acceso desde cualquier parte del programa puede hacer que sean modificadas accidentalmente, lo que puede causar errores difíciles de detectar. Por lo general, es preferible utilizar variables locales para evitar problemas de este tipo.

b. Sí, una variable local puede ser estática respecto de su l-valor en algunos lenguajes de programación, como C y C++.

En estos lenguajes, se puede declarar una variable local como estática mediante la palabra clave "static". Una variable estática mantiene su valor entre llamadas a una función, lo que significa que el valor se conserva entre diferentes invocaciones de la función.

Por ejemplo, en el siguiente código en C, la variable "contador" se declara como estática dentro de la función "suma":

```c
#include <stdio.h>

int suma(int num) {
    static int contador = 0;
    contador += num;
    return contador;
}

int main ) {
    printf("%d\n", suma(10));
    printf("%d\n", suma(20));
    printf("%d\n", suma(30));
    return 0;
}
```

En este caso, la variable "contador" es una variable local a la función "suma", pero al declararla como estática, su valor se mantiene entre las diferentes llamadas a la función. Cada vez que se llama a la función, se incrementa el valor de "contador" con el valor de "num" y se devuelve el valor actualizado.

c. No necesariamente, una variable global no siempre es estática. La variable global puede ser estática o dinámica, según la forma en que se declare.

En el caso de las variables globales estáticas, su valor se mantiene durante toda la duración del programa, es decir, su valor se inicializa una sola vez y se conserva hasta el final del programa. Además, las variables globales estáticas solo están visibles dentro del archivo en el que se declaran, a menos que se declare como extern en otro archivo.

Por otro lado, las variables globales dinámicas no tienen un tiempo de vida fijo y su valor puede ser cambiado durante la ejecución del programa. A diferencia de las variables globales estáticas, las variables globales dinámicas son visibles desde cualquier parte del programa.

En resumen, una variable global puede ser tanto estática como dinámica, dependiendo de cómo se declare.

d. Una variable estática respecto a su l-valor es aquella que mantiene su valor durante la ejecución del programa, es decir, su valor se inicializa una sola vez y se mantiene constante durante toda la duración del programa. En cambio, una constante es una entidad que también mantiene su valor constante, pero a diferencia de una variable estática, su valor no se puede modificar en tiempo de ejecución.

En términos de uso, las variables estáticas se utilizan para almacenar información que necesita persistir a lo largo de la ejecución del programa, como contadores o banderas, mientras que las constantes se utilizan para definir valores que no cambian a lo largo del programa, como por ejemplo el valor de PI.

5)a. En Ada, las constantes numéricas son aquellas que tienen un valor numérico fijo, como por ejemplo el valor de PI o la velocidad de la luz. Estas constantes se definen utilizando la palabra clave "constant" y su valor se debe especificar en tiempo de compilación.

Por otro lado, las constantes comunes son aquellas que tienen un valor no numérico fijo, como por ejemplo un mensaje de error o una dirección de memoria. Estas constantes se definen utilizando la palabra clave "pragma" y su valor se puede especificar en tiempo de compilación o en tiempo de ejecución, dependiendo de la situación.

La clasificación de las constantes en numéricas y comunes se debe a que cada tipo de constante tiene un uso diferente en el programa y requiere una sintaxis y un tratamiento diferente en el lenguaje. En general, las constantes numéricas se utilizan para definir valores matemáticos y físicos que se utilizan en cálculos y operaciones numéricas, mientras que las constantes comunes se utilizan para definir valores fijos que se utilizan en todo el programa y que pueden cambiar en tiempo de ejecución en algunos casos.

b. En Ada, las constantes tienen ligadura estática. En el ejemplo dado, la constante **H** se define como **`Float`** y se le asigna el valor **`3,5`** en el momento de su declaración. La constante **`I`** se define como tipo **`constant`**, lo que significa que es una constante común, y se le asigna el valor **2** en el momento de su declaración. La constante **K** se define como **`float`** y se le asigna el resultado de la multiplicación de **H** e **`I`**, que se realiza en tiempo de

compilación, ya que ambos son constantes y su valor es conocido en ese momento. Por lo tanto, la ligadura de estas constantes es estática, ya que su valor se conoce en tiempo de compilación y no cambia durante la ejecución del programa.

6)Si, llegaria a tener el mismo comportamiento ya que si se tiene 'static int= 1' en la func1(), la memoria se asignaria una sola vez cuando se llama por primera vez a la función y esta memoria se mantiene asignada durante toda la vida del programa, al igual que tener 'int x=1' como variable global , por lo tanto, el comportamiento en cuanto a la asignación de memoria seria el mismo.

7)

```
Clase Persona {
        public long id     Variable de instancia
        public string nombreApellido Var de instancia
        public Domicilio domicilio  Var de instancia
        private string dni;    Variable de instancia
        public string fechaNac;  Variable de instancia
        public static int cantTotalPersonas;  Global

        //Se tienen los getter y setter de cada una
de las variables
        //Este método calcula la edad de la persona
a partir de la fecha de nacimiento
```

```
public int getEdad(){
                public int edad=0;   Local
                public string fN =    Local
this.getFechaNac();
                        ...
                        ...
          return edad;
        }
}
Clase Domicilio {
        public long     id;     Variable de instancia
        public static int nro   Global
        public string calle     Variable de instancia
        public Localidad loc;   Variable de instancia

        //Se tienen los getter y setter de cada una
de las variables
}
```

La diferencia entre variables globales, variables de instancia y variables locales en Java se basa en su alcance y su duración. Las variables globales son visibles en toda la clase y tienen una duración de toda la vida de la aplicación, mientras que las variables de instancia son específicas del objeto y duran tanto como el objeto exista. Por último, las variables locales solo son visibles al método o bloque de código en el que se declaran y tienen una duración limitada a la ejecución de ese método o bloque.

8) a.Tiempo de vida:

i : de la línea 1 a la 15

h: línea 1 - línea 15

mipuntero: línea 1 -línea 15

mipuntero^ : línea 9 – línea 15

b. Alcance:

I: linea 4 – linea 15

H: linea 5 – linea 15

Mipuntero: linea 3 – linea 15

mipuntero^ : linea 3 – linea 15

c.No porque a h se le asigna el valor apuntado por mipuntero + i antes de realizar el dispose (mipuntero), es decir, antes de que se libere la memoria.

d. Si, se presenta un error porque a h se le intenta restar lo que se encuentra en la dirección de memoria de mipuntero y ese valor es nil porque anteriormente se hizo un dispose(mipuntero) y no se puede realizar una resta entre un entero(h) y un puntero nulo(nil), por lo tanto se genera un error en tiempo de ejecución.

e.Si, la otra entidad que existe es el programa principal, el cual es una entidad que necesita ligar los atributos de alcance y tiempo de vida para justificar las respuestas anteriores. En este caso de Pascal, el programa principal tiene alcance global y su tiempo de vida es desde su declaración hasta el final del programa (línea 6 - línea 15)

f. i es de tipo entero, h es de tipo entero y mipuntero es de tipo tpuntero(puntero a un integer)

9)a. En el lenguaje de programación **C++**, podemos declarar una variable local como estática para que su tiempo de vida sea mayor que su alcance. Un ejemplo sería el siguiente:

```
#include <iostream>

using namespace std;

void contador ) {
  static int count = 0; // declarar una variable local como estática
  count++;
  cout << "El contador es " << count << endl;
}

int main ) {
  for (int i = 0; i < 5; i++)
    contador(); // llamamos a la función contador varias veces
  }
  return 0;
}
```

En este ejemplo, la variable **`count`** se declara como estática dentro de la función **`contador()`**. Esto significa que su tiempo de vida se extiende más allá del alcance de la función y se conserva entre las llamadas a la función. En cada llamada a **`contador()`**, el

valor de **count** se incrementa y se imprime por pantalla, dando como resultado la cuenta de 1 a 5. Si la variable **count** no se hubiera declarado como estática, su valor se reiniciaría a cero cada vez que se llama a la función.

b. Ejemplo en Pascal:

```
1 program ejemploB;
2
3 type tpuntero = ^integer;
4
5 var
6 mipuntero: tpuntero;
7 begin
8       new (mipuntero);
9       mipuntero^ := 10;
10      dispose(mipuntero);
11      writeln(mipuntero^);
12 end.
```

En el anterior programa, el alcance de mipuntero es desde la línea 6(su declaración) hasta la línea 12, pero su tiempo de vida es desde la 8 hasta la línea 10(termina por el dispose). Despues de liberar la memoria asignada al puntero, el puntero deja de existir y no se puede acceder al entero apuntado por él, como se intenta hacer en la línea 11.

c.Ejemplo en Python

```python
def suma(a, b):
    resultado = a + b
    print(resultado)

    return resultado

x = 5
y = 3
suma(x, y)
```

En este ejemplo, la función **suma** recibe dos parámetros **a** y **b**, los cuales son variables locales y su tiempo de vida comienza al entrar a la función. Dentro de la función, se crea la variable **resultado** y se le asigna el valor de la suma de **a** y **b**. Luego se imprime el resultado y se retorna el valor.

La variable **x** y **y** son globales y su tiempo de vida comienza al inicio del programa y termina cuando el programa finaliza.

En este caso, el tiempo de vida de las variables locales **a**, **b** y **resultado** es igual al alcance de la función **suma**. Una vez que la función termina, estas variables locales dejan de existir. El alcance de las variables globales **x** e **y** es el programa completo, por lo que su tiempo de vida es igual al alcance.

10) Si, en los tres lenguajes se puede asegurar que el tiempo de vida y el alcance de la variable estan limitados al procedimiento en el que se encuentra definida, ya que no hay definiciones ni procedimientos internos que puedan afectar con su tiempo de vida o alcance.

11)

Ejercicio 11: a) Responda Verdadero o Falso para cada opción. El tipo de dato de una variable es?
I) Un string de caracteres que se usa para referenciar a la variable y operaciones que se pueden realizar sobre ella. F
II) Conjunto de valores que puede tomar y un rango de instrucciones en el que se conoce el nombre. F
III) Conjunto de valores que puede tomar y lugar de memoria asociado con la variable. F
IV) Conjunto de valores que puede tomar y conjunto de operaciones que se pueden realizar sobre esos valores. V

b. El tipo de dato de una variable se refiere a la naturaleza del valor que se almacena en esa variable. En términos más simples, es la forma en que se interpreta y se manipula la información almacenada en la variable. Los tipos de datos pueden incluir enteros, números de punto flotante, caracteres, cadenas de caracteres, valores booleanos, entre otros. La definición correcta de un tipo de dato de una variable depende del lenguaje de programación utilizado, pero en general se refiere a la forma en que se especifica y se utiliza ese tipo de dato en el código fuente del programa.

Antes de que una variable sea referenciada debe ser ligada a su tipo(protege a las variables de operaciones no permitidas).

Chequeo de tipos: verifica el uso correcto de las variables.

12)

```
1.        with text_io; use text_io;
2.        Procedure Main is;
3.        type vector is array(integer range <>);
4.        a, n, p:integer;
5.        v1:vector(1..100);
6.        c1: constant integer:=10;
7.        Procedure Uno is;
1.    type puntero is access integer;
2.    v2:vector(0..n);
3.    c1, c2: character;
4.    p,q: puntero;
5.    begin
        7.5.1. n:=4;
        7.5.2. v2(n):= v2(1) + v1(5);
        7.5.3. p:= new puntero;
        7.5.4. q:= p;
        7.5.5. .......
        7.5.6. free p;
        7.5.7. ......
        7.5.8. free q;
        7.5.9. ......
  7.6. end;
8. begin
9.   n:=5;
10.  .....
11.  Uno;
12.  a:= n + 2;
13.  .....
14. end
```

Ident.	Tipo	r-valor	Alcance	T.V.
a (linea 4)	automática	basura	4-14	1-14
n(linea 4)	automática	basura	4-14	1-14
p (linea 4)	automática	basura	4-14	1-14
v1(linea 5)	automática	basura	5-14	1-14
c1(linea 6)	automática	10	6-14	1-14
Uno() (linea 7)			7-14	7-7.6
v2 (linea 2)	semidinámica	basura	7.2-7.6	7 - 7.6
c1(linea 3)	automática	basura	7.3 -7.6	7 - 7.6
c2 (linea 3)	automática	basura	7.3-7.6	7 - 7.6
p (linea 7.4)	automática	basura	7.4- 7.6	7- 7.6
^p	dinámica	nil	7.4-7.6	7.5.3 -7.5.6
q (linea 7.4)	automática	basura	7.4-7.6	7-7.6
^q	dinámica	nil	7.4-7.6	7.5.4-7.5.8
Main() (linea 2)			3-29	2-29

Aclaración:

Ident.= Identificador

T.V. = Tiempo de Vida

r-valor debe ser tomado al momento de la alocación en memoria

El **alcance** de los identificadores debe indicarse desde la línea siguiente a su declaración.

13)El nombre de la variable puede condicionar:

a. **Su tiempo de vida:** la vida útil de una variable está determinada por el ámbito en el que se declara la variable y se le da un nombre, por lo tanto, el tiempo de vida es el periodo de tiempo por lo cual el uso de esa variable es válido.

b. **Su alcance:** ya que el alcance es el rango de instrucciones en el que se conoce el nombre de la variable.

c. **Su R-Valor:** el nombre de la variable no condiciona a su R-Valor, ya que este mismo se determina por la asignación de un valor a la variable (al nombre). Lo que si permite el nombre es poder acceder, mediante ese nombre, a la variable y por lo tanto a su R-Valor.

d. **Su tipo:** el nombre de la variable no condiciona a su tipo, ya que el tipo se define explícitamente en la declaración de la variable. Lo que es importante realizar, es dar un nombre descriptivo que refleje que tipo de variable es, lo que puede hacer más fácil al programador entender y trabajar con el código que utiliza la variable.

14)

ARCHIVO1.C

```
1.    int v1;
2.    int *a;
3.    Int fun2 ()
4.    {  int  v1, y;
5.       for(y=0; y<8; y++)
6.       {  extern int v2;
7.       ...}
8.    }
9.    main()
10.   {static int var3;
11.   extern int v2;
12.   int  v1, y;
13.   for(y=0; y<10; y++)
14.   { char var1='C';
15.    a=&v1;}
16.   }
```

ARCHIVO2.C

```
17.   static int aux;
18.   int v2;
19.   static int fun2( )
20.   { extern int v1;
21.     aux=aux+1;
22.     …
23.   }
24.   int fun3( )
25.   { int aux;
26.     aux=aux+1;
27.     …
28.   }
```

Ident.	Tipo	r-valor	Alcance	T.V.
v1(linea 1)	automática	0	2- 4,9-12, 21-23	1-28
*a(linea 2)	dinámica	indef	3-16	15-16
a(linea 2)	automática	null	3-16	1-28
int fun2()			4-16	3-8
v1'(linea 4)	automática	indef	5-8	3-8
y (linea 4)	automática	indef	5-8	3-8
main()			10-16	9-16
var3(linea 10)	estática	0	11-16	<1-28>
v1(linea 12)	automática	indef	13-16	9-16
y(linea 12)	automática	indef	13-16	9-16
var1(linea 14)	automática	'C'	15	13-16
aux(linea 17)	estática	0	18-25	<1-28>
v2(linea 18)	automática	indef	12-16,19-28	1-28
fun2()			20-28	19-23
fun3()			25-28	24-28
aux(linea 25)	automática	indef	26-28	24-28

Aclaración:

Ident.= Identificador

T.V. = Tiempo de Vida

r-valor debe ser tomado al momento de la alocación en memoria

El **alcance** de los identificadores debe indicarse desde la línea siguiente a su declaración

15) En JavaScript, existen tres formas de declarar variables: **`var`**, **`let`**, y **`const`**. La diferencia semántica entre ellos radica en su alcance y mutabilidad:

- **`var`**: Es el modificador original para declarar variables en JavaScript. La variable declarada con **`var`** tiene un alcance de función, lo que significa que su alcance se limita al bloque de función en el que se define, o al alcance global si se define fuera de una función. Además, las variables declaradas con **`var`** son mutables, lo que significa que se pueden reasignar en cualquier momento.
- **`let`**: Fue introducido en ES6 como una alternativa a **`var`**. Las variables declaradas con **`let`** tienen un alcance de bloque, lo que significa que su alcance se limita al bloque de código en el que se define. Además, las variables declaradas con **`let`** son mutables, lo que significa que se pueden reasignar en cualquier momento.
- **`const`**: También fue introducido en ES6 como una forma de declarar variables inmutables. Las variables declaradas con **`const`** tienen un alcance de bloque, al igual que las variables declaradas con **`let`**. Sin embargo, a diferencia de **`let`**, las variables declaradas con **`const`** son inmutables, lo que significa que su valor no se puede reasignar una vez que se ha inicializado.

Si se declara una variable sin usar ningún modificador, el comportamiento por defecto dependerá del modo estricto en el que se esté ejecutando el código. Si se está ejecutando en modo estricto, una variable sin modificador será tratada como una variable de alcance de bloque y generará un error si se intenta asignar sin haber sido declarada. Si no se está en modo estricto, una variable sin modificador se comportará como una variable de alcance global si se declara fuera de una función, o de alcance de función si se declara dentro de una función.

Comparado con el lenguaje de programación Python, la diferencia semántica en la declaración de variables es la siguiente:

- **`var`**, **`let`** y **`const`** no existen en Python. En su lugar, las variables se declaran simplemente asignándoles un valor. La declaración de una variable se hace en la misma línea en la que se le asigna un valor, y su alcance depende de dónde se declare. Si se declara fuera de una función, la variable tendrá un alcance global; si se declara dentro de una función, tendrá un alcance local a esa función. Python no tiene un modificador de variable inmutable, pero se pueden simular mediante convenciones de nomenclatura y programación defensiva.

El hoisting en JavaScript es un comportamiento en el cual las declaraciones de variables (ya sea con la palabra clave **`var`**, **`let`** o **`const`**) y funciones se mueven al inicio del ámbito en el que se encuentran, antes de que se ejecute el código. Es decir, aunque una variable o función sea declarada después de su uso en el código, el intérprete de JavaScript la moverá al inicio de su ámbito correspondiente.

Por ejemplo, en el siguiente código:

```
console.log(x);
var x = 5;
```

El resultado impreso en la consola sería **undefined**, ya que la declaración de la variable **x** se mueve al inicio del ámbito, pero su asignación no. Por lo tanto, al momento de imprimir la variable, su valor aún no ha sido asignado.

Es importante tener en cuenta que el hoisting solo aplica a la declaración de variables y funciones, no a su inicialización o asignación. Además, el hoisting solo ocurre en el ámbito de función o global, no en el ámbito de bloque introducido por las llaves **{}**.

En comparación con otros lenguajes como Java o C, el hoisting es un comportamiento específico de JavaScript que puede llevar a confusiones y errores si no se comprende adecuadamente. En general, otros lenguajes no tienen un comportamiento similar al hoisting y las variables y funciones deben ser declaradas antes de su uso en el código.