Práctica 4

Ejercicio 1: a) Tome una de las variables de la línea 3 del siguiente código e indique y defina cuáles son sus atributos:

```
1. Procedure Practica4();
2. var
3. a,i:integer
4. p:puntero
5. Begin
6. a:=0;
7. new(p);
8. p:= ^i
9. for i:=1 to 9 do
10.a:=a+i;
11.end;
12....
13.p:= ^a;
14....
15.dispose(p);
16.end;
```

Atributos de la variable a.

Nombre: a

Alcance: Estático, de la línea 3 hasta línea 16

Tipo: Predefinido Integer, ligadura estática.

L-Valor: Alocacion dinámica automática.

R-Valor: Dinámico 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

b) Compare los atributos de la variable del punto a) con los atributos de la variable de la línea 4. ¿Qué dato contiene esta variable?

Atributos de la variable p.

Nombre: p

Alcance: Estático, de la línea 4 hasta línea 16

Tipo: Tipos definidos por el usuario Puntero, ligadura estática.

L-Valor: Alocación dinámica explícita.

R-Valor: Dinámico l-valor de i (línea 8), l-valor de a (línea 13).

Ejercicio 2:

a. Indique cuales son las diferentes formas de inicializar una variable en el momento de la declaración de la misma.

- Ignorar el problema: la inicializo con lo que haya en memoria
- Estrategia de inicialización:
    - Inicialización por defecto: Enteros se inicializan en 0, los caracteres en blanco, etc.
    - Inicialización en la declaración: C int i =0, j= 1

b. Analice en los lenguajes: Java, C, Phyton y Ruby las diferentes formas de inicialización de variables que poseen. Realice un cuadro comparativo de esta característica.

Java	- Declaración e inicialización en la misma línea: int num = 10; - Declaración e inicialización separadas: int num; num = 10; - Inicialización por defecto: int num; (inicializada a 0 automáticamente)
C	- Declaración e inicialización en la misma línea: int num = 10; - Declaración e inicialización separadas: int num; num = 10; - Las variables globales y estáticas se inicializan por defecto a 0 si no se les asigna un valor explícitamente. - Las variables locales no tienen un valor por defecto y su valor inicial es impredecible, a menos que se les asigne un valor explícitamente.
Python	- Declaración e inicialización en la misma línea: num = 10 - No es necesario declarar el tipo de variable antes de su inicialización
Ruby	- Declaración e inicialización en la misma línea: num = 10 - No es necesario declarar el tipo de variable antes de su inicialización

En resumen, Java y C tienen formas similares de inicialización de variables, mientras que Python y Ruby tienen una sintaxis más flexible y sencilla. En Python y Ruby no es necesario declarar el tipo de variable antes de su inicialización, lo que puede hacer que el código sea más fácil de escribir y entender. Por otro lado, Java y C no permiten la asignación de variables sin declarar su tipo previamente. Además, Java tiene la opción de inicialización por defecto, que es útil en ciertos casos, como cuando se desea inicializar un array.

Ejercicio 3: Explique (de al menos un ejemplo de cada uno) los siguientes conceptos asociados al atributo l-valor de una:

El atributo l-valor de una variable hace referencia a su capacidad de ser referenciada o manipulada mediante una expresión de asignación. En otras palabras, una variable con l-valor puede ser asignada a otra variable o valor.

a. Variable estática.

Una variable estática tiene un ámbito de visibilidad limitado a la función o archivo en el que se declara y su valor se mantiene constante durante toda la ejecución del programa. Es decir, es una variable que existe y se inicializa en tiempo de compilación y su valor persiste en memoria durante toda la ejecución del programa.

Ejemplo en C:

```c
#include <stdio.h>

void static_example() {
  static int num = 0;
  printf("El valor de num es: %d\n", num);
  num++;
}

int main() {
  for (int i = 0; i < 5; i++) {
    static_example();
  }
  return 0;
}
```

En este ejemplo, la variable num es una variable estática dentro de la función static_example().

La salida del programa sería:

```
El valor de num es: 0
El valor de num es: 1
El valor de num es: 2
El valor de num es: 3
El valor de num es: 4
```

b. Variable automática o semiestática.

Una variable automática o semiestática se declara dentro de un bloque y su ámbito de visibilidad se limita a ese bloque. Su valor se inicializa al entrar al bloque y se destruye al salir de él. Estas variables también se llaman locales, ya que solo son visibles dentro de la función en la que se declaran.

Ejemplo en Python:

```python
def automatic_example():
  num = 0
  print("El valor de num es: ", num)
  num += 1

```

```
for i in range(5):
  automatic_example()
```

En este ejemplo, la variable num es una variable automática dentro de la función automatic_example().

La salida del programa sería:

```
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0
El valor de num es: 0
```

c. Variable dinámica.

Una variable dinámica es una variable que se crea y se destruye en tiempo de ejecución. En la mayoría de los lenguajes de programación, estas variables se crean utilizando funciones o métodos específicos como malloc() o new. El valor de una variable dinámica puede cambiar a lo largo de la ejecución del programa.

Ejemplo en Python:

```
def dynamic_example():
  num = input("Ingrese un número: ")
  print("El valor ingresado es:", num)

for i in range(5):
  dynamic_example()
```

En este ejemplo, la variable num es una variable dinámica que se inicializa mediante la entrada de usuario en cada llamada a la función dynamic_example().

La salida del programa dependerá de lo que ingrese el usuario.

d. Variable semidinámica.

Una variable semidinámica es una variable que se inicializa en tiempo de ejecución pero que no cambia de tamaño después de ser creada.

```
def semidynamic_example():
    num = [1, 2, 3]
    print("El valor de num es:", num)
    num[1] = 5
    print("El valor modificado de num es:", num)

semidynamic_example()
```

En este ejemplo, la variable num es una variable semidinámica, ya que se crea en tiempo de ejecución como una lista con tres elementos y su tamaño no cambia después de ser creada. Sin embargo, su valor puede ser modificado, como se ve en la línea num[1] = 5.

La salida del programa sería:

```
El valor de num es: [1, 2, 3]
El valor modificado de num es: [1, 5, 3]
```

Investigue sobre que tipos de variables respecto de su l-valor hay en los lenguajes C y Ada.

En el lenguaje de programación C, se pueden clasificar las variables según su l-valor de la siguiente manera:

a. Variable estática: se define usando la palabra clave "static" y se mantiene en memoria durante toda la ejecución del programa. Estas variables se inicializan automáticamente a cero si no se les asigna un valor explícitamente.

b. Variable automática o semiestática: se definen dentro de una función y se eliminan de la memoria cuando la función termina. Estas variables no se inicializan automáticamente, por lo que su valor es indeterminado hasta que se les asigna uno.

c. Variable dinámica: se asignan en tiempo de ejecución usando funciones como malloc() y calloc(). Estas variables se mantienen en la memoria hasta que se libera explícitamente usando la función free().

d. Variable semidinámica: se crean usando un tamaño fijo en tiempo de compilación y su valor puede cambiar en tiempo de ejecución. Por ejemplo, un arreglo puede ser una variable semidinámica.

En el lenguaje de programación Ada, las variables se clasifican según su l-valor de la siguiente manera:

a. Variable estática: se definen usando la palabra clave "constant" o "static" y se mantiene en memoria durante toda la ejecución del programa.

b. Variable automática o semiestática: se definen dentro de un bloque y se eliminan de la memoria cuando el bloque termina.

c. Variable dinámica: se asignan en tiempo de ejecución usando la palabra clave "new". Estas variables se mantienen en la memoria hasta que se libera explícitamente usando la palabra clave "delete".

d. Variable semidinámica: no hay una clasificación explícita para las variables semidinámicas en Ada, pero se pueden crear utilizando arreglos de tamaño fijo. El tamaño del arreglo es fijo en tiempo de compilación, pero el valor de sus elementos puede cambiar en tiempo de ejecución.

Ejercicio 4:

a. ¿A qué se denomina variable local y a qué se denomina variable global?

Una variable local es una variable declarada dentro de una función o bloque de código y su ámbito de alcance se limita a esa función o bloque de código. Por otro lado, una variable global es una variable declarada fuera de cualquier función o bloque de código y su ámbito de alcance se extiende a todo el programa.

b. ¿Una variable local puede ser estática respecto de su l-valor? En caso afirmativo dé un ejemplo

Sí, una variable local puede ser estática respecto a su l-valor. En algunos lenguajes de programación, como C y C++, se pueden definir variables locales como estáticas para mantener su valor entre llamadas sucesivas a una función.

Un ejemplo en C podría ser:

```c
#include <stdio.h>

void ejemploFuncion() {
  static int variableLocal = 0;
  variableLocal++;
  printf("El valor de la variable estática es %d\n", variableLocal);
}

int main() {
  ejemploFuncion(); // Imprime "El valor de la variable estática es 1"
  ejemploFuncion(); // Imprime "El valor de la variable estática es 2"
  ejemploFuncion(); // Imprime "El valor de la variable estática es 3"
  return 0;
}
```

En este ejemplo, la variable variableLocal se declara como static dentro de la función ejemploFuncion(). Esto significa que su valor se mantiene entre llamadas sucesivas a la función. En cada llamada a la función, el valor de variableLocal se incrementa en uno y se imprime en pantalla. Como la variable es estática, su valor se mantiene entre llamadas y no se pierde cuando la función termina su ejecución.

c. Una variable global ¿siempre es estática? Justifique la respuesta.

No necesariamente. La estática es una propiedad que puede tener una variable independientemente de su alcance. En algunos lenguajes de programación, como C, las variables globales son, por defecto, estáticas en términos de su l-valor, lo que significa que su valor se mantiene durante toda la vida útil del programa y no puede ser modificado por otras funciones o bloques de código. Sin embargo, en otros lenguajes de programación como Java y Python, las variables globales no son estáticas en términos de su l-valor, ya que pueden ser modificadas en tiempo de ejecución y su valor puede ser accedido y modificado por cualquier función o bloque de código en el programa.

En resumen, la naturaleza de las variables globales respecto a su l-valor no depende del hecho de que sean globales o no, sino que depende del lenguaje de programación utilizado y de cómo

se manejan las variables en ese lenguaje. Por lo tanto, es importante conocer las características del lenguaje de programación que se está utilizando para comprender el comportamiento de las variables globales en términos de su l-valor.

d. Indique qué diferencia hay entre una variable estática respecto de su l-valor y una constante

La diferencia principal entre una variable estática respecto a su l-valor y una constante es que el valor de una constante no puede ser modificado una vez que se ha definido, mientras que el valor de una variable estática puede ser modificado durante la ejecución del programa.

Una constante se define en el código del programa y su valor es fijo durante toda la ejecución. Es decir, el valor de una constante se define una sola vez y no se puede modificar en tiempo de ejecución. Las constantes se usan para representar valores fijos como pi, el número de días en una semana, entre otros.

Por otro lado, una variable estática se define en el código del programa y su valor se mantiene en memoria durante toda la ejecución del programa, pero el valor de la variable se puede modificar en cualquier momento. Las variables estáticas se utilizan para mantener un valor a lo largo de la vida útil del programa, como contar el número de veces que se llama a una función.

En resumen, la diferencia fundamental entre una variable estática y una constante es que el valor de una constante no puede ser modificado después de ser definido, mientras que el valor de una variable estática puede ser modificado en tiempo de ejecución.

Ejercicio 5:

a. En Ada hay dos tipos de constantes, las numéricas y las comunes. Indique a que se debe dicha clasificación.

La clasificación de las constantes en Ada en numéricas y comunes se debe a que las constantes numéricas son aquellas que se refieren a valores numéricos, mientras que las constantes comunes son aquellas que pueden contener caracteres y símbolos, como cadenas de texto.

b. En base a lo respondido en el punto a), determine el momento de ligadura de las constantes del siguiente código:

H: constant Float:= 3,5;

I: constant:= 2;

K: constant float:= H*I;

El momento de ligadura de las constantes en Ada es durante la compilación del programa, es decir, en tiempo de compilación. En el código proporcionado, el momento de ligadura de la constante "H" es cuando se le asigna el valor 3.5, el momento de ligadura de la constante "I" es cuando se le asigna el valor 2, y el momento de ligadura de la constante "K" es cuando se calcula su valor mediante la multiplicación de "H" e "I". Por lo tanto, el momento de ligadura de "K" es después del momento de ligadura de "H" e "I" durante el proceso de compilación. Como

todas las constantes se definen en tiempo de compilación, su valor se conoce antes de que se ejecute el programa.

Ejercicio 6: Sea el siguiente archivo con funciones de C:

Archivo.c

```
{ int x=1; (1)
int func1();{
int i;
for (i:=0; i < 4; i++) x=x+1;
}
int func2();{
int i, j;
/*sentencias que contienen declaraciones y
sentencias que no contienen declaraciones*/
.....
for (i:=0; i < 3; i++) j=func1 + 1;
}
}
```

Analice si llegaría a tener el mismo comportamiento en cuanto a alocación de memoria, sacar la declaración (1) y colocar dentro de func1() la declaración static int x =1;

Si, llegaría a tener el mismo comportamiento ya que si se tiene 'static int x=1' en la func1(), la memoria se asignaría 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.

Ejercicio 7: Sea el siguiente segmento de código escrito en Java, indique para los identificadores si son globales o locales.

```
Clase Persona {
        public long id
        public string nombreApellido
        public Domicilio domicilio
        private string dni;
        public string fechaNac;
        public static int cantTotalPersonas;

        //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;
                public string fN =
this.getFechaNac();
                        ...
                        ...
          return edad;
        }
}
Clase Domicilio {
        public long     id;
        public static int nro
        public string calle
        public Localidad loc;

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

Los identificadores se pueden clasificar de la siguiente manera:

Globales (variables estáticas de clase, aquellos que se declaran a nivel de la clase y pueden ser accedidos sin necesidad de crear una instancia de la clase):

- cantTotalPersonas (estático) en la clase Persona
- nro (estático) en la clase Domicilio

Variables de instancia (no estáticas, son variables de instancia que se declaran dentro de la clase y solo pueden ser accedidas a través de una instancia de la clase):

- id en las clases Persona y Domicilio
- nombreApellido en la clase Persona
- domicilio en la clase Persona
- dni en la clase Persona
- fechaNac en la clase Persona
- calle en la clase Domicilio
- loc en la clase Domicilio

Locales (son aquellos que se declaran dentro de un método y solo pueden ser accedidos dentro del mismo):

- edad en el método getEdad() de la clase Persona
- fN en el método getEdad() de la clase Persona

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

de cada objeto y duran tanto como el objeto exista. Por último, las variables locales solo son visibles en el método o bloque de código en el que se declaran y tienen una duración limitada a la ejecución de ese método o bloque.

Ejercicio 8: Sea el siguiente ejercicio escrito en Pascal

1- Program Uno;

2- type tpuntero= ^integer;

3- var mipuntero: tpuntero;

4- var i:integer;

5- var h:integer;

6- Begin

7- i:=3;

8- mipuntero:=nil;

9- new(mipuntero);

10- mipunterno^:=i;

11- h:= mipuntero^+i;

12- dispose(mipuntero);

13- write(h);

14- i:= h- mipuntero;

15- End.

a) Indique el rango de instrucciones que representa el tiempo de vida de las variables i, h y mipuntero.

i: línea 1 – línea 15

h: línea 1- línea 15

mipuntero: línea 9 – línea 12

b) Indique el rango de instrucciones que representa el alcance de las variables i, h y mipuntero.

i: línea 4 – línea 15

h: línea 5 – línea 15

mipuntero: línea 3 – línea 15

c) Indique si el programa anterior presenta un error al intentar escribir el valor de h. Justifique

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) Indique si el programa anterior presenta un error al intentar asignar a i la resta de h con mipuntero. Justifique

Si, se presenta un error porque a h se le intenta restar la dirección de memoria del valor al que apunta mipuntero, y ese valor es nil ya que antes se efectuo 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) Determine si existe otra entidad que necesite ligar los atributos de alcance y tiempo de vida para justificar las respuestas anteriores. En ese caso indique cuál es la entidad y especifique su tiempo de vida y alcance.

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 el 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) Especifique el tipo de variable de acuerdo a la ligadura con el l-valor de las variables que encontró en el ejercicio.

i es de tipo entero, h es de tipo entero y mi puntero es de tipo tpuntero (puntero a entero).

Ejercicio 9: Elija un lenguaje y escriba un ejemplo:

a. En el cual el tiempo de vida de un identificador sea mayor que su alcance

Ejemplo en C:

```c
#include <stdio.h>

void suma(){
    static int num = 0;
    num++;
    printf ("El valor de num es: %d\n",num);
}

int main(void) {
    int i;
    for (i = 1; i <= 3; i++) {
        suma();
    }
    return 0;
}
```

Se imprime en consola:

```
El valor de num es: 1
El valor de num es: 2
El valor de num es: 3
```

En el anterior programa, el alcance de num es solo dentro de la función suma, pero su tiempo de vida es desde la primera vez que se ejecuta la función suma hasta que finaliza el programa ya que se trata de una variable estática.

b. En el cual el tiempo de vida de un identificador sea menor que su alcance

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

c. En el cual el tiempo de vida de un identificador sea igual que su alcance

Ejemplo en Python:

```
1  def ejercicioC():
2      x = 5
3      print(x)
4      y = 10
5      if x == 5:
6          z = 15
7          print(y, z)
8      print(y)
9
10 ejercicioC()
```

En el anterior programa, el alcance de x, y, y z está limitado a la función ejercicioC(). Además, su tiempo de vida coincide con el tiempo de ejecución de la función (desde que se ejecuta la función hasta que termina la función). Por lo tanto, el alcance y el tiempo de vida de x, y, y z son iguales.

Ejercicio 10: Si tengo la siguiente declaración al comienzo de un procedimiento:

int c; en C

var c:integer; en Pascal

c: integer; en ADA

Y ese procedimiento NO contiene definiciones de procedimientos internos. ¿Puedo asegurar que el alcance y el tiempo de vida de la variable "c" es siempre todo el procedimiento en donde se encuentra definida? Analícelo y justifique la respuesta, para todos los casos.

Sí, en los tres lenguajes se puede asegurar que la variable c tiene alcance y tiempo de vida limitados al procedimiento en el que se encuentra definida, ya que no hay definiciones de procedimientos internos que puedan afectar su alcance o tiempo de vida.

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) Escriba la definición correcta de tipo de dato de una variable

- Definición:
    - Conjunto de valores
    - Conjunto de las operaciones
- Antes de que una variable pueda ser referenciada debe ligársele un tipo
    - Protege a las variables de operaciones no permitidas
- Chequeo de tipos: verifica el uso correcto de las variables

Ejercicio 12: Sea el siguiente programa en ADA, completar el cuadro siguiente indicando para cada variable de que tipo es en cuanto al momento de ligadura de su l-valor, su r-valor al momento de alocación en memoria y para todos los identificadores cuál es su alcance y cual

es su el tiempo de vida. Indicar para cada variable su r-valor al momento de alocación en memoria

```
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 (línea 4)	automática	basura	4-14	1-14

Ejercicio 13: Justifique la respuesta. El nombre de una variable puede condicionar:

a) Su tiempo de vida: El nombre de una variable puede condicionar su tiempo de vida, ya que 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 período de tiempo durante el cual el uso de ese nombre es válido.

b) Su alcance: El nombre una variable puede condicionar su alcance de, ya que el alcance de una variable es el rango de instrucciones en el que se conoce el nombre.

c) Su r-valor: El nombre de una variable no condiciona su r-valor, ya que el r-valor 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.

d) Su tipo: El nombre de una variable no condiciona directamente su tipo, ya que el tipo se define explícitamente en la declaración de la variable. Sin embargo, es importante elegir un

nombre descriptivo para la variable que refleje el tipo de dato que almacena, lo que puede hacer que sea más fácil para los programadores entender y trabajar con el código que utiliza la variable.

Ejercicio 14: Sean los siguientes archivos en C, los cuales se compilan juntos Indicar para cada variable de que tipo es en cuanto al momento de ligadura de su l-valor. Indicar para cada identificador cuál es su alcance y cuál es su el tiempo de vida. Indicar para cada variable su r-valor al momento de alocación en memoria

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.

Ejercicio 15: Para javascript investigue la diferencia semántica para declarar una variable utilizando los modificadores const, var, let y la ausencia de cualquiera de estos. Compárelo con un lenguaje de su preferencia.

HOISTING: es un mecanismo de JavaScript donde las variables y las declaraciones (SOLO DECLARACIONES, NO INICIALIZACION) de funciones se mueven a la parte superior de su alcance antes de la ejecución del código

- var
    - El alcance es global cuando una variable var se declara fuera de una función. Esto significa que cualquier variable que se declare con var fuera de un bloque de funciones está disponible para su uso en todo el programa.
    - var tiene un alcance de función cuando se declara dentro de una función. Esto significa que está disponible y se puede acceder solo dentro de esa función.
    - Las variables declaradas con var se pueden volver a declarar y actualizar
    - Hoisting de var:
        - Las variables var se elevan a la parte superior de su alcance y se inicializan con un valor de indefinido.
- let
    - Una variable declarada en un bloque con let solo está disponible para su uso dentro de ese bloque (Un bloque es un fragmento de código delimitado por {}. Un bloque vive entre llaves. Cualquier cosa entre llaves es un bloque)
    - let se puede actualizar, pero no volver a declarar (si la misma variable se define en diferentes ámbitos, no habrá error).
    - Hoisting de let:
        - Las variables let se elevan a la parte superior de su alcance pero no se inicializan.
- const
    - Las variables declaradas con const mantienen valores constantes.
    - Solo se puede acceder a las declaraciones const dentro del bloque en el que se declararon.
    - const no se puede actualizar o volver a declarar.
    - Cada declaración con const debe inicializarse en el momento de la declaración.
    - Hoisting de const:
        - Las variables const se elevan a la parte superior de su alcance pero no se inicializan.

Si se declara una variable sin utilizar const, let o var, el motor de JavaScript va a intentar buscar la variable en un contexto superior de forma recursiva. Para ello, primero busca en el ámbito local, si no la encuentra, se intenta en su contexto inmediatamente superior, y así hasta que se llegue al contexto global. Si no se ha encontrado en ningún contexto, esa variable se crea automáticamente y su comportamiento es similar al de var en cuanto a su alcance y mutabilidad. El problema es si esa variable se encuentra en uno de ese contexto superior que se han ido comprobando, ya que tomara su valor.

```
1  // VARIABLES GLOBALES
2  var primer = 'Ana';
3  var edad = 30;
4  var sexo = 'F';
5
6
7  function funcionConVar() {
8    // VARIABLES LOCALES (TIENEN VAR)
9    var primer = 'Juan';
10   var edad = 20;
11   var sexo = 'M';
12   console.log('(LOCAL) El nombre es: '+ primer + " y tengo " + edad );
13 }
```

```
14
15 function funcionSinVar() {
16   // NO SE INDICA VAR, HAY QUE BUSCARLAS PRIMERO EN CONTEXTOS SUPERIORES
17   primer = 'Juan';
18   edad = 20;
19   sexo = 'M';
20   console.log('(LOCAL) El nombre es: '+ primer + " y tengo " + edad );
21 }
22
23 console.log('(GLOBAL) El nombre es: '+ primer + " y tengo " + edad );
24 funcionConVar();
25 console.log('(GLOBAL) El nombre es: '+ primer + " y tengo " + edad );
26 funcionSinVar();
27 console.log('(GLOBAL) El nombre es: '+ primer + " y tengo " + edad );
```

```
1 (GLOBAL) El nombre es: Ana y tengo 30
2 (LOCAL) El nombre es: Juan y tengo 20
3 (GLOBAL) El nombre es: Ana y tengo 30
4 (LOCAL) El nombre es: Juan y tengo 20
5 (GLOBAL) El nombre es: Juan y tengo 20
```

En este caso tenemos tres variables que están en un contexto superior (Ana 30 F) a las dos funciones funcionConVar y funcionSinVar.

Si ejecutas la funcionConVar, las variables primer, edad y sexo son locales a esa función, es decir son copias nuevas y lo que hagas con ellas no afectan a las del contexto superior. Es decir que si le pones Juan 20 M, el usuario Ana 30 F no se ve modificado.

Sin embargo, si ejecutas la funcionSinVar, al no tener var, se intenta buscar variables con el mismo nombre en contextos superiores y sí que se se encuentra a Ana 30 F. Por lo que ahora esas variables no son locales y si las modificas con Juan 20 M, modificas indirectamente a Ana 30 F.

Si ejecutas el código que te he puesto puedes ver que Ana 30 F queda modificado indirectamente.

Se debe intentar poner var, const o let a las variables locales para evitar este tipo de comportamientos dependientes del contexto superior.

Con var lo que consigues es que se use la variable local con precedencia en caso de que ya exista esa variable en otro contexto superior.

En comparación, en Python, no se utilizan modificadores de este tipo para declarar variables. En Python, las variables se declaran simplemente asignándoles un valor. Además, el alcance de las variables se define por la identación, es decir, por la estructura del código. Por ejemplo, una variable declarada dentro de una función solo es accesible dentro de esa función. Si se intenta acceder a ella fuera de la función, se producirá un error.