Practica 8

Ejercicio 1: Una sentencia puede ser simple o compuesta, ¿Cuál es la diferencia?

Ejercicio 1- Una sentencia simple realiza una sola accion, es decir no tiene sentencias dentro, y no puede ser divisible en mas sentencias. En cambio una compuesta, agrupa varias sentencias simples y/o compuestas dentro de un bloque y sirve para ejecutar múltiples acciones como una unidad.

Ejercicio 2: Analice como C implementa la asignación.

Ejercicio 2- Una asignacion en C se implementa con el operador “=” y su lógica es igual al de los otros lenguajes.

Ejercicio 3: ¿Una expresión de asignación puede producir efectos laterales que afecten al
resultado final, dependiendo de cómo se evalúe? De ejemplos.

Ejercicio 3-

Sí, en muchos lenguajes de programación, una expresión de asignación puede tener efectos laterales que afectan el resultado final, y el orden de evaluación puede cambiar el comportamiento del programa. Por ejemplo, cuando una operación modifica el estado del programa además de producir un valor.

Ejemplos comunes:

- Cambiar el valor de una variable
- Modificar estructuras de datos
- Escribir en archivos, consola, memoria, etc.
- Llamar funciones que tienen sus propios efectos

⚠️ ¿Cómo el orden de evaluación puede afectar?

Cuando una expresión:

- Tiene **más de un acceso o modificación a la misma variable**
- Llama a funciones con efectos laterales

- Involucra operadores como `++`, `-`, o asignaciones múltiples

... el **resultado puede variar dependiendo del orden en que se evalúen los componentes** de esa expresión.

Ejercicio 4: Qué significa que un lenguaje utilice circuito corto o circuito largo para la evaluación
de una expresión. De un ejemplo en el cual por un circuito de error y por el otro no.

Ejercicio 4-

- **Circuito corto (short-circuit evaluation)**:

  Un lenguaje **detiene la evaluación** de una expresión lógica tan pronto como el resultado se puede determinar.

- **Circuito largo (long-circuit evaluation)**:

  El lenguaje **evalúa todas las partes** de la expresión lógica, **aunque el resultado ya sea conocido**.

```
   x = 0
   if (x != 1 && 10 / x > 1) {
// ...
}
```

  En el ejemplo por circuito corto la primera condicion es falsa por lo que “10 / x “ no se ejecuta. En cambio con circuito largo se ejecuta la condicion “10 / x “ y la misma produce un error por que hace una division por 0.

Ejercicio 5: ¿Qué regla define Delphi, Ada y C para la asociación del else con el if
correspondiente? ¿Cómo lo maneja Python?

Ejercicio 5-

🔷 Delphi → **usa bloques delimitados por `begin` y `end`**, por lo que **no hay ambigüedad**.

- Cada `else` se asocia claramente con su correspondiente `if` según la estructura de bloques.

🔷 Ada → **requiere que los bloques estén claramente cerrados** usando `end if;`.

- Cada `if` debe terminarse explícitamente, lo que **elimina cualquier ambigüedad**.

🔷 C → **usa la regla de "else se asocia con el if más cercano no emparejado"**.

- Esto **puede causar ambigüedad** si no se usan llaves `{}`

🔷Python en cambio usa la indentación obligatoria para definir bloques, lo que elimina completamente el problema.

- El `else` se asocia con el `if` al mismo nivel de indentación.

Ejercicio 6: ¿Cuál es la construcción para expresar múltiples selección que implementa C?
¿Trabaja de la misma manera que la de Pascal, ADA o Python?

Ejercicio 6-

🔷 C: utiliza `switch` evalúa la **expresión** y ejecuta el código del `case` que coincida.

- El `break` es **necesario** para evitar que la ejecución caiga en los siguientes casos (lo que se llama *fall through*).
- El `default` es opcional y se ejecuta si ningún `case` coincide.

🔷 Pascal utiliza el case que es muy parecido al switch pero no utiliza el sentencia break, y no permite caídas entre casos.

🔷 ADA es similar a Pascal

- Requiere que todos los casos posibles estén cubiertos (ya sea con valores explícitos o con `others`).
- No hay caída entre casos.

🔷 Python agrega la sentencia en la version 3.10

- `match-case` se parece a `switch`, pero es más poderoso: permite **estructuras de patrones**, no solo valores simples.
- **No hay caída entre casos**, como en Pascal o Ada.

Ejercicio 7: Sea el siguiente código:

```
.....
var i, z:integer;
Procedure A;
begin
  i:= i +1;
end;
begin
z:=5;
```

```
for i:=1..5 do
 begin
  z:=z*5;
  A;
  z:=z + i;
 end;
end;
```

a- Analice en las versiones estándar de ADA y Pascal, si este código puede llegar a traer problemas. Justifique la respuesta.
b- Comente qué sucedería con las versiones de Pascal y ADA, que Ud. utilizó.

a-

✅ En Pascal (estándar):

- En Pascal, **la variable de control de un `for` loop (como `i`) no debería modificarse dentro del bucle**, ya sea directamente o indirectamente (como ocurre aquí a través de la llamada a la `Procedure A`).
- **Este código es legal en muchos compiladores de Pascal**, pero:
  - **Es peligroso** porque **el valor de `i` es modificado dentro del bucle**, lo que puede **causar un comportamiento inesperado**.
  - La mayoría de compiladores **no interrumpen el bucle**, pero **el incremento manual (`i := i + 1`) puede desincronizar la variable de control**, haciendo que el bucle:
    - Termine antes de tiempo
    - Se repita más veces
    - O entre en un bucle infinito (dependiendo del rango y valor de `i`)

✅ En Ada:

- Ada **prohíbe explícitamente modificar la variable de control de un bucle `for`**.
- El intento de hacerlo, incluso de forma indirecta desde una subrutina, como se hace en `Procedure A`, genera un **error en tiempo de compilación**.

- La variable del bucle en Ada se considera **constante dentro del bloque del** `for`.

Ejercicio 8 - Sea el siguiente código en Pascal:
var puntos: integer;
begin
...
case puntos
1..5: write(“No puede continuar”);
10:write(“Trabajo terminado”)
end;
..
Analice, si esto mismo, con la sintaxis correspondiente, puede trasladarse así a los lenguajes
ADA, C. ¿Provocaría error en algún caso? Diga cómo debería hacerse en cada lenguaje y
explique el por qué. Codifíquelo.

Ejercicio 8- En C no seria posible ya que este denomina la sentencia con switch, cada caso se llama case, utiliza el break en cada case y ademas no se puede utilizar rangos para estos, por lo que no se podria trasladar asi como esta.

En cambio en ADA, no coincide la estrtuctura semantica del switch pero permite rangos dentro de los mismos.

Ejercicio 9: Qué diferencia existe entre el generador YIELD de Python y el return de una
función. De un ejemplo donde sería útil utilizarlo.

Ejercicio 9-

La diferencia es que el yield en Python se utiliza para devolver generadores de funciones. Ademas, nos permiten pausar una funcion y devolver un valor determinado, as su vez, luego podemos retomar la funcion desde el punto donde quedo formando asi una secuencia de valores. En cambio, el return se utiliza para devolver un valor determinado de una funcion, siendo este el valor final.

```
def pares_hasta(n):
    for i in range(n):
        if i % 2 == 0:
            yield i

# Usando el generador
for numero in pares_hasta(10):
    print(numero)
```

Este ejemplo simple sirve para generar los primeros n numeros pares.

Ejercicio 10: Describa brevemente la instrucción map en javascript y sus alternativas.

Ejercicio 10-
La función
`map()` es un método de los arrays en JavaScript que permite **transformar** cada elemento del arreglo original y **devolver un nuevo arreglo** con los resultados sin modificar el arreglo original.

Alternativas:

Alternativa	¿Qué hace?	¿Reemplaza a `map`?
`forEach()`	Ejecuta una función por cada elemento (sin devolver)	❌ No. No devuelve nuevo array
`for` / `for...of`	Estructura de bucle tradicional	✅ Sí, pero más código
`reduce()`	Acumula valores en una sola salida	✅ Sí, pero más complejo
`filter()`	Devuelve elementos que cumplen una condición	❌ No transforma, solo filtra

Ejercicio 11: Determine si el lenguaje que utiliza frecuentemente implementa instrucciones para
el manejo de espacio de nombres. Mencione brevemente qué significa este concepto y enuncie
la forma en que su lenguaje lo implementa. Enuncie las características más importantes de este
concepto en lenguajes como PHP o Python.

Ejercicio 11-

Sí, Java **implementa manejo de espacio de nombres** usando **paquetes** (`packages`).

✅ ¿Cómo lo hace?

- Se agrupan clases dentro de **paquetes** (carpetas).
- Se usa la palabra clave `package` al inicio del archivo.
- Se accede a otras clases usando `import`.

El
espacio de nombres es un mecanismo que **organiza y agrupa identificadores** (como clases, funciones, variables) para **evitar conflictos de nombres**.

🟪 En Python

- Usa **módulos** y **paquetes** (archivos `.py` y carpetas con `__init__.py`).
- Se importa con `import` o `from ... import`.
- Los módulos se pueden anidar fácilmente.
- Muy flexible pero menos estricta que Java.

🟥 En PHP

- Desde PHP 5.3, se introdujo `namespace`.

```
php
CopiarEditar
namespace MiApp\Modelo;

class Usuario {}
```

- Se accede con `use` o el nombre completo:

```
php
CopiarEditar
```

```
use MiApp\Modelo\Usuario;
```

✔️ Similar al enfoque de Java, pero con más libertad estructural.