MODELO DE EXAMEN: SEMINARIO DE LENGUAJES - OPCIÓN GO
Nombre: _____ Fecha: _____
Instrucciones:
• Lee cada pregunta cuidadosamente.
• Responde de manera clara y concisa.
• Para las preguntas de código, escribe el código Go válido.
• Cita las fuentes si es relevante para tu respuesta (aunque no se pide en un examen real, aquí lo hago para reforzar el aprendizaje).

Sección 1: Fundamentos de Go (20 puntos)

1. Define qué es Go y menciona al menos tres de sus principales usos.

◦ **Go** es un lenguaje de programación multiplataforma de código abierto, compilado y fuertemente tipado. Fue desarrollado por Google y su sintaxis se basa en C/C++.
  ◦ 3 de sus principales usos:
    ▪ Desarrollo web (lado del servidor).
    ▪ Desarrollo de aplicaciones concurrentes.
    ▪ Desarrollo de aplicaciones multiplataforma.

2. Explica dos razones por las cuales se recomienda usar Go.

  ◦ 2 razones por las cuales se recomienda usar **Go**:
    ▪ Es fácil de aprender.
    ▪ Es portable a diferentes plataformas.

3. Considera el siguiente código Go:

  ◦ **¿A qué *package* debe pertenecer todo programa ejecutable en Go?**
    ▪ Todo ejecutable pertenece al *package* **"main"**.

  ◦ **Explica la diferencia en la declaración de `a` y `b`. ¿Cuándo se puede usar `:=`?**
    ▪ `var a int = 10` es una declaración de variable **explícita** que incluye el tipo (`int`) y la inicialización.
    ▪ `b := 20` es una declaración de variable de **asignación corta (short assignment)** donde el tipo (`int` en este caso) es **inferido** del valor de inicialización. El operador `:=` solo se puede usar **dentro de funciones**.

  ◦ **¿Cuál es la forma correcta de exportar un identificador (variable, función, etc.) en Go para que sea visible desde otros *packages*?**
    ▪ Los identificadores declarados en un *package* son **"exportados" (es decir, visibles desde afuera)** cuando **comienzan con mayúscula**.

4. Menciona el valor por defecto (zero value) para los siguientes tipos básicos en Go:
`bool, string, int, float64.`

- **`bool`: `false`**
- **`string`: `""` (cadena vacía)**
- **`int`: `0`**
- **`float64`: `0`**

Sección 2: Estructuras de Control y Funciones (25 puntos)

1. Escribe un ejemplo de un bucle `for` en Go que itere del 0 al 9 e imprima cada número. Luego, modifica el ejemplo para mostrar un bucle `for` sin una sentencia de inicialización o post-sentencia.

◦ Ejemplo básico:

```
fmt.Println("----Usando for----")

    for i := 0; i <= 9; i++ {
        fmt.Println(i)
    }
```

◦ Ejemplo sin sentencia de inicialización o post-sentencia (como un `while` en otros lenguajes):

```
    fmt.Println("----Usando ''While''----")

    for sigo {
        fmt.Println(i)
        if i == 9 {
            sigo = false
        }
        i = i + 1
    }
```

2. Explica la diferencia principal entre las funciones `fmt.Print()`, `fmt.Println()` y `fmt.Printf()` del paquete `fmt`.

- `fmt.Print()`: Imprime sus argumentos. Pone un espacio entre los argumentos, **excepto para las cadenas de texto** (strings). No añade una nueva línea al final.
- `fmt.Println()`: Imprime sus argumentos. Pone un espacio entre los argumentos, **incluso para las cadenas de texto**, y **agrega un "newline"** (nueva línea) al final.
- `fmt.Printf()`: Permite una **impresión formateada** utilizando "marcas" o "verbos" (por ejemplo, `%s`, `%d`, `%v`) que especifican cómo se deben formatear los valores. No añade una nueva línea a menos que se especifique (`\n`).

3. Considera la siguiente función Go:

```go
func operar(a, b int) (suma, producto int) {
    suma = a + b
    producto = a * b
    return // retorno implícito
}
```

- **¿Cómo se llaman los parámetros `suma` y `producto` en esta declaración de función?**
  - Son parámetros de retorno nombrados.

- **¿Qué significa su uso?**
  - El uso de parámetros de retorno nombrados permite declarar las variables de retorno directamente en la firma de la función.

Dentro del cuerpo de la función, se pueden asignar valores a estas variables, y cuando se ejecuta return (incluso sin argumentos), esos valores se retornan automáticamente.

4. Explica y da un ejemplo de un `switch` sin selector en Go. ¿Para qué tipo de situaciones es útil?

- Un `switch` sin selector en Go es similar a una cadena de sentencias `if-else if-else`. La sentencia `switch` ejecuta el primer `case` cuya expresión es verdadera.
- Es útil para **condiciones complejas que no se basan en un único valor de una variable**, sino en múltiples expresiones booleanas.

◦ Ejemplo:

```go
func clasificarEdad(edad int) {
    switch {
    case edad < 0:
        fmt.Println("Edad no válida")
    case edad < 13:
        fmt.Println("Niño")
    case edad < 18:
        fmt.Println("Adolescente")
    case edad < 65:
        fmt.Println("Adulto")
    default:
        fmt.Println("Adulto mayor")
    }
}
```

Sección 3: Colecciones (Arrays, Slices, Maps) (20 puntos)

1. Describe la diferencia fundamental entre un `Array` y un `Slice` en Go, en términos de su longitud y flexibilidad.

◦ **Array:** Es una secuencia indexada de elementos de un mismo tipo, con una **longitud fija** que es parte de su definición de tipo. Una vez declarado, su tamaño no puede cambiar.

◦ **Slice:** Es un "segmento" de un *array* subyacente. Son indexables y tienen una longitud, pero esta **longitud puede cambiar dinámicamente**. Los *Slices* son más frecuentes que los *Arrays* debido a su flexibilidad.