¿Que es “Go” y para que se utiliza?

Go es un lenguaje de programación multiplataforma de código abierto, desarrollado por Google en 2007. Es un lenguaje compilado y fuertemente tipado, con una sintaxis basada en C/C++.
Se utiliza principalmente para el desarrollo web (back-end), aplicaciones en red, aplicaciones multiplataforma, desarrollo nativo en la nube y aplicaciones concurrentes. Es valorado por su facilidad de aprendizaje, rapidez en tiempo de ejecución y compilación, soporte para concurrencia y genéricos, gestión eficiente de la memoria y portabilidad a diferentes plataformas (Windows, Mac, Linux).

¿Cuáles son las estructuras básicas de un programa?

Un programa Go se organiza en paquetes (packages). Todo programa ejecutable debe pertenecer a el paquete “main”. Las librerías necesarias se importan usando la palabra clave import. La ejecución del programa comienza en la función main(). Go ignora las líneas en blanco y cada sentencia se separa con un salto de línea o un punto y coma. Los identificadores (variables, funciones, etc.) que comienzan con mayúscula son “exportados” y visibles desde otros paquetes, ósea son públicos, y los que no empiezan con mayúscula son privados. Go soporta comentarios de una línea (//) y de varias (/* … */).

¿Cómo se manejan las variables, constantes y tipos de datos en Go?

Las variables se declaran con una palabra clave var, y pueden incluir inicialización. Go permite inferir el tipo de variable si inicializa, usando la sintaxis corta := dentro de una función. Los nombres de las variables son sensibles a mayúsculas y minúsculas (case sensitive). Las constantes se declaran con const y pueden ser de tipos carácter, string boolean o numéricos y también pueden ser sin tipo. Los tipos básicos de Go incluyen bool, string, enteros (int, uint y sus variantes con tamaño especifico), byte, rune, flotantes (float32, float64) y complejos (comlex64, complex128). La conversión de tipos se realiza explícitamente usando T(v), donde T es el tipo deseado y v el valor a convertir. Ej: float64(num), en donde num es un entero y lo quiero pasar a float64.

- ¿Cómo se implementan las estructuras de control en Go?

Go ofrece varias estructuras de control para manejar el flujo del programa:

- Secuencia: Las sentencias se ejecutan en el orden en que aparecen.
- Iteración (for): Go solo tiene la palabra clave for para bucles. Puede funcionar como un bucle while (con o sin condición inicial y final), un bucle for tradicional con inicialización, condición e incremento, o un bucle infinito que se rompe con break. También se puede iterar sobre colecciones con for...range.
- Selección (if, switch): if: Permite la ejecución condicional de código. Puede incluir una sentencia de inicialización opcional antes de la condición, cuya variable estará disponible dentro del bloque if y else. switch: Se utiliza para la selección de casos. Puede tener un selector explícito o implícito (evaluando true). También puede incluir una sentencia de inicialización y no necesita break al final de cada case, ya que solo ejecuta el bloque coincidente y sale.

- ¿Qué son los Arrays, Slices y Maps en Go?

- Arrays: Son secuencias indexadas de elementos del mismo tipo, con una longitud fija y el primer índice en cero. Su longitud es arte de su definición de tipo.
- Slices: son más flexibles que los Arrays. Son "segmentos" de un array subyacente (son como Arrays dinámicos). Tienen una longitud(len), que es la cantidad de elementos (dimensión lógica) y una capacidad(cap), que es cuanto puede crecer antes de tener que crear un nuevo array subyacente (dimensión física). Son indexables y tienen una longitud que puede cambiar. Los Slices son referencias a los Arrays subyacentes, lo que significa que modificar un slice puede afectar al array original y a otros Slices que referencien a la misma porción del array. Se pueden crear Slices a partir de Arrays existentes, con la función make o mediante literales. La función append permite agregar elementos a un slice y copy permite copiar elementos de un slice a otro.
- Maps: Son colecciones no ordenadas de pares clave-valor. No permiten claves duplicadas. Las claves deben ser de tipos que soporten comparación por igualdad (como números, cadenas, Arrays, etc., pero no Slices, Maps o funciones). Los Maps son referencias, lo que implica que la

asignación de un map a otra variable crea otra referencia al mismo map subyacente. Se pueden agregar, modificar (usando make o sintaxis literal) y eliminar elementos (usando delete).

- ¿Cómo se implementan los Punteros, Structs e Interfaces en Go?

- Punteros: Un puntero almacena la dirección de memoria de un valor de un tipo específico. El valor cero de un puntero es nil. Se pueden obtener la dirección de una variable con "&" y acceder al valor apuntado con "*". La función new(T) asigna memoria para un valor de tipo T y retorna un puntero a esa ubicación. Los parámetros de funciones se pasan "por valor", incluso los punteros, aunque sus contenidos si pueden ser modificados.
  Go pasa todos los argumentos por valor, lo que incluye punteros. Cuando se pasa un puntero, lo que se copia es la dirección de memoria, por lo que tanto la función como el llamador pueden acceder y modificar el mismo valor en memoria.
- Structs: Son colecciones de campos(variables) de distintos tipos, agrupados bajo un único nombre. Permiten encapsular datos relacionados. Los Structs son comparables si todos sus campos lo son y pueden usarse como claves en Maps.
- Interfaces: Una interfaz define un conjunto de firmas de métodos. Cualquier tipo que implemente todos los métodos de una interfaz se considera que "implementa" esa interfaz, sin necesidad de una declaración explicita. Un valor de interfaz se puede concebir como un par (valor, tipo), donde el tipo es un tipo concreto y el valor es un valor de ese tipo. La "Interfaz vacía" (interface{}) puede contener valores de cualquier tipo lo que la hace muy útil para funciones genéricas como fmt.Print.

- ¿Cómo se maneja la Concurrencia en Go?

Go fomenta la concurrencia a través de Goroutines y Channels.

- Goroutines: Son funciones que se ejecutan concurrentemente con otras funciones. Son ligeras y se inician con la palabra clave "Go". "sync.WaitGroup" es una forma de esperar a que varias Goroutines terminen su ejecución.

- Channels: Son el mecanismo principal para la comunicación y sincronización entre Goroutines, se crean con make (ch:=make(chan int))
  canal <- valor (Manda valor hacia el canal)
  valor := <- canal (Saca un valor desde el canal).
  Son conductos tipados a través de los cuales las Goroutines envían y reciben datos. Por defecto, las operaciones de envío y recepción en un channel son bloqueantes, osea que cuando alguien envía se queda esperando a que alguien reciba y cuando alguien recibe se queda esperando a que alguien envíe. Los canales pueden ser unidireccionales y pueden tener un búfer que sirve para permitir envíos y recepciones sin que se bloqueen hasta que el búfer se llene. Los canales se pueden cerrar (close(chan)).
- Select: Permite a una goroutine esperar por múltiples operaciones de channel, ejecutando la primera que esté lista. Puede incluir una clausula default para operaciones no bloqueantes. Es algo similar a Switch pero para canales.
- Exclusión Mutua: Para proteger recursos compartidos y evitar condiciones de carrera, Go proporciona sync.Mutex (semáforo binario) y sync.RWMutex(semáforo de "un escritor – múltiples lectores").
  Lock() y Unlock() controlan el acceso a la sección critica.

– ¿Qué son los Genéricos en Go?

Go introdujo los genéricos para permitir escribir funciones y tipos que operan con cualquier tipo, sin perder la seguridad de tipos, es decir poder crear código reutilizable y más limpio sin usar interface{}.

- Funciones Genéricas: permiten definir funciones que aceptan parámetros de tipo, haciendo el código más reutilizable. Los parámetros de tipo tienen "restricciones de tipo" que especifican las propiedades que deben tener los tipos permitidos, si el tipo necesita hacer comparaciones, hay que poner explícitamente que el tipo es "comparable", sino simplemente se pone "any", que se refiere a cualquier tipo.
- Tipos Genéricos: Permiten definir estructuras de datos (como listas o arboles) que puedan almacenar elementos de cualquier tipo especificado por el parámetro de tipo.