**Golang – GO**

Lenguaje compilado open source, desarrollado por **Google**

Para chequear si se instaló correctamente, en cmd:

**go version**

Genera también variables de entorno para un uso más cómodo

Instalar en **Visual Studio Code** la extensión Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

* Se ejecuta:

Con el comando **go run archivo.go**

En caso de necesitar ejecutar varios archivos

**go run archivo1.go archivo2.go**

**go build archivo.go** 🡪 para compilar y generar el ejecutable, luego podremos hacer archivo.exe

Si tenemos problemas para ejecutar y depurar desde el visual code, usar este comando 

**Lenguaje de tipado estático** 🡪 los tipos de datos deben estar definidos y mantenerse a lo largo de la ejecución del programa

**Tipos de datos numéricos**

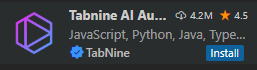
Enteros 🡪 **int**

Enteros sin signo (no permite negativos) 🡪 **uint**

Flotantes 🡪 **float32 / float64**

No puedo sumar un entero con un flotante

Extensión de Visual Studio Code que autocompleta el código



La , me permite concatenar strings con números, el signo + permite solo concatenar entre strings. Además, la coma (,) me agrega un espacio entre los objetos concatenados.

**Creación de tipos propios dentro de GO**

Texto

Descripción generada automáticamentesimilar a la definición de una clase en otros lenguajes

Texto

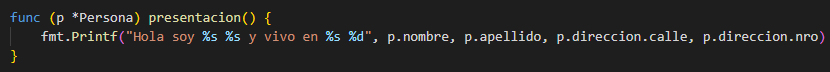
Descripción generada automáticamenteproceso similar a la instanciación de una clase (generación de un objeto de tipo persona)

Podemos incluir mas tipos propios dentro del creado

Dentro de un Printf podemos incluir variables en las comillas de la siguiente forma:

%s 🡪 strings %d 🡪 int %T 🡪 tipo de dato %v 🡪 variable

Definición de un método para la clase Persona

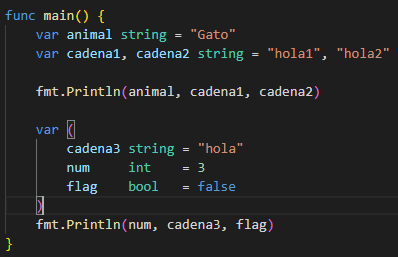


Código completo

Texto

Descripción generada automáticamente

**Definición de variables**



de esta forma el mismo compilador asume cual es el tipo que posee la variable

**Valores por defecto que toman las variables**

Numéricos 🡪 toman 0 como valor por predeterminado

Booleanos 🡪 false

Strings 🡪 cadena vacía

**Alcance o scope de las variables**

Global: todo el código

Local: dentro de una función (cuidado con las llaves)

**Constantes**

Valores que se van a mantener a lo largo de la ejecución del programa

También tienen un scope, como las variables

**Punteros**

Ubicación en memoria de una variable

Con le ampersand podemos acceder a ella 🡪 &var

Paso por valor 🡪 copia del valor

Paso por referencia 🡪 ubicación de memoria

**Funciones**

Función main ( ) es el punto de partida de ejecución del programa, no puede llevar parámetros ni devolver valores.

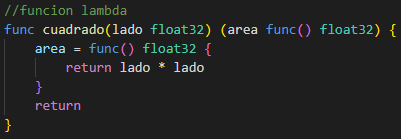
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Funciones lambda 🡪 funciones anónimas





**Operadores**

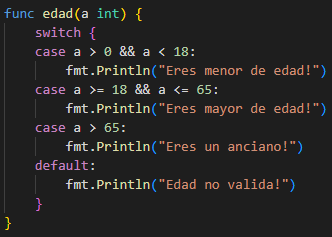
Comparación 🡪 ==, !=, >, <, etc.

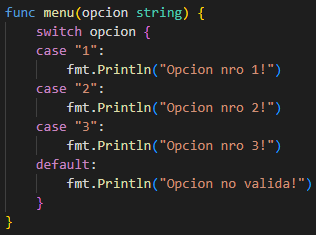
Aritméticos 🡪 con el modulo Math tendremos mayor cantidad de operadores

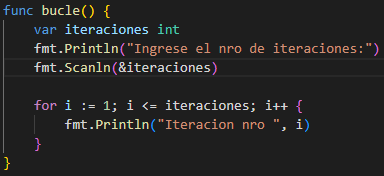
Lógicos 🡪 &&, ||, !

Texto

Descripción generada automáticamente







Texto

Descripción generada automáticamente

**Slice**

No necesito saber el tamaño al momento de definirlo, le defino un tamaño pero puedo modificarlo fácilmente. No puedo eliminar un elemento del slice, son inmutables. Para eliminar un elemento debo reconstruir el slice

Texto

Descripción generada automáticamente





arranca desde el índice 0 y toma dos elementos

 arranca desde el índice 0 hasta el final





Interfaz de usuario gráfica, Texto

Descripción generada automáticamenteentonces si modifico un slice también se modifica el otro

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Mapas**

Accedemos a los elementos a través de sus keys

Tiene una estructura key:value

Texto

Descripción generada automáticamenteen este caso es una key de string y un value int. Se puede usar cualquier combinación

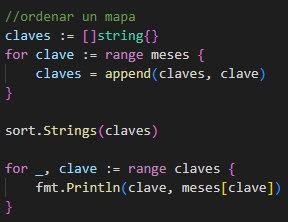
Texto

Descripción generada automáticamenterecorrer un mapa

Podemos agregar fácilmente elementos

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza baja

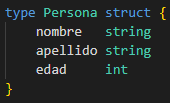


**Estructuras**

Campos públicos 🡪 accesible desde cualquier parte de la aplicación – mayúsculas. Permite ser exportado

Campos privados 🡪 solo se puede acceder desde el propio paquete – minúsculas. No permite ser exportado

Se establece con el primer caracter del nombre de la función, atributo, estructura o lo que fuere.

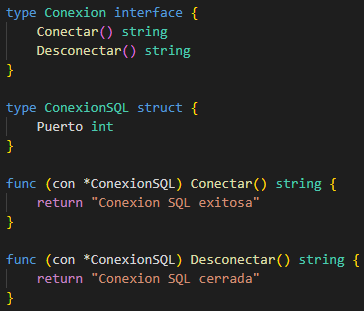
la estructura es Publica, pero los atributos son privados

Si usamos punteros (&, \*) para asignar el valor, si modifico uno se modifica el otro.

Texto

Descripción generada automáticamente

**Interfaces 🡪** plantilla de métodos para implementarlos en otra clase

****

**Texto

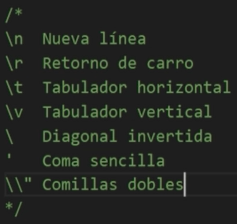
Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

**Practica con strings**

**Rune 🡪 código para indicar algo dentro de un string**

****

**Texto multilíneaTexto

Descripción generada automáticamente**

**Concatenar strings 🡪 +**

**Texto

Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

**Librerías como strings, strconv, fmt**

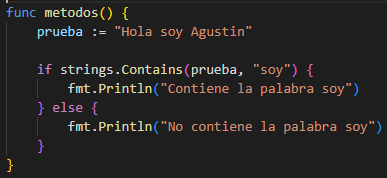
**Conversión de tipos**

**Texto

Descripción generada automáticamente** convierte de int a string

FormatInt 🡪 convierte a binario, Hexa, base10, etc.

fmt.Sprint() 🡪 convierte a string

**** es sensitive case

compare(“cadena1”, “cadena2”) 🡪 compara las cadenas y da 0, 1 o -1 según resultado

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamentedevuelve -1 si no encuentra

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**Errores en Go 🡪** un error viene como último retorno de la función que lo generó

**Manejo de errores**

**Texto

Descripción generada automáticamente**

**Tipo error**

Texto

Descripción generada automáticamentecreamos un nuevo objeto de tipo error que nos sea útil. Necesitamos importar la librería “errors”

Texto

Descripción generada automáticamente

**Panic**

Función en GO que nos permite detener por completo el flujo de ejecución del programa

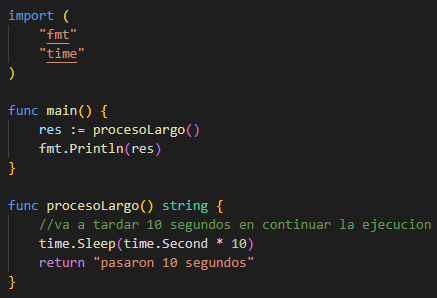
Texto

Descripción generada automáticamente

**Concurrencia**

Ejecutar en orden parcial, de a pedazos, en desorden

**Funciones y bloqueos**



**Go-rutinas**

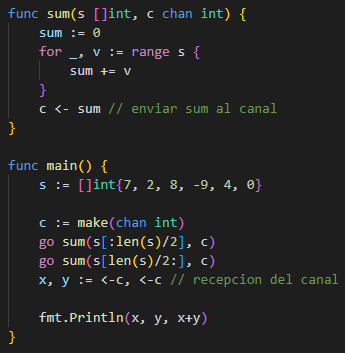
Ejecutar de forma concurrente varios procesos

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

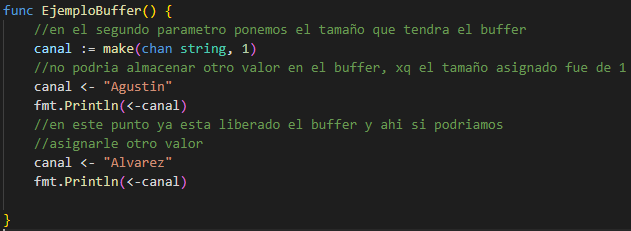
**Canales / Channels**

Permiten a los datos moverse dentro y fuera de las rutinas y facilitar la comunicación entre ellas. En GO se comparte la memoria para esto

****

**Buffer**

Mantiene el canal abierto hasta que el receptor está disponible



**Iterar un canal**

Recorrer los valores que va tomando el canal. Para que no de un mensaje de warning debemos cerrar el canal.

Texto

Descripción generada automáticamente

Podemos definir canales como de solo lectura, solo escritura o lectura escritura

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**Canales múltiples y timeouts**

Permite por ejemplo que podamos capturar al primer proceso que termina de ejecutarse. Además, el timeout permite ponerle un limite de tiempo a una rutina

Texto

Descripción generada automáticamente

**Creación de paquetes**