Estructura de inicio

Todos los programas en Go inician por la función `main` en el paquete `main`. Además no es necesario terminar las sentencias con `;` (punto y coma).

```
package main

import "fmt"

func main() {

	fmt.Printf("hello, world\n")

}
```

En el programa anterior, podemos ver que la primera sección del programa compuesta por la primera línea contiene la declaración del `package` actual, que en nuestro caso es `main`. La segunda sección del programa compuesta por la tercera línea contiene la declaración de las rutas de importación que utiliza la palabra reservada `import`. Las rutas de importación cargan basicamente librerías (clases) propias del core o de terceros. La tercera parte del programa compuesta por las tres últimas líneas define la función `main` la cual es el punto de inicio del programa. Para agregar más de una ruta de importación se puede agregar las líneas que sean necesarias.

```
import "fmt"

import "math/rand"
```

O bien utilizar la sintaxis recomendada (factorizada).

```
import (

	"fmt"

	"math/rand"

)
```

Variables

La estructura de inicio vista anteriormente es el andamio inicial para la creación de programas en Go. Sin embargo, un programa sin variables es poco útil. Las variables en Go pueden ser definidas tanto a nivel de paquete como a nivel de función.

```
package main

import "fmt"

var c, python, java bool

func main() {

	var i int

	fmt.Println(i, c, python, java)

}
```

Las variables pueden ser declaradas e inicializadas en una misma sentencia como se muestra a continuación.

```
var i, j int = 1, 2
```

Cuando se utiliza el inicializador se puede omitir el tipo de variable quedando de la siguiente manera:

```
var i, j = 1, 2
```

También es posible declarar e inicializar variables de distinto tipo.

```
var c, python, java = true, false, "no!"
```

Si la declaración de variables depende de una expresión o se compone de muchas sentencias se puede utilizar la forma factorizada o de bloque de la siguiente manera:

```
var (
	isSomething  bool       = false
	MaxInt       uint64     = 1<<64 - 1
	z            complex128 = cmplx.Sqrt(-5 + 12i)
)
```

Finalmente, dentro de una función puede evitarse el uso de la palabra reservada `var` en la definición de variables al combinarlo con la declaración de asignación corta `:=`.

```
k := 3

c, python, java := true, false, "no!"
```