

GO-02 01: Основы языка.

Переменные, константы и типы данных

Описание:

В этом разделе вы познакомитесь с основами языка: переменными и константами, а также узнаете, какие типы данных реализованы в Go.

Названия переменных, констант, типов, функций и т.д могут содержать цифры и буквы, точнее, те символы, которые в Unicode считаются буквами. Символ `_` (нижнее подчеркивание) в Unicode считается буквой, поэтому может быть свободно использован в названии. Идентификатор всегда должен начинаться с буквы, в противном случае на этапе компиляции Go покажет ошибку. Также идентификатор не должен совпадать с зарезервированными словами, полный список которых можно найти по ссылке [Keywords](#). Go — регистрозависимый язык, поэтому `myVariable` и `myvariable` являются разными названиями.

В Go для именования сущностей принят `camelCase` стиль, поэтому при работе в IDE вы можете увидеть предупреждение, если используете `snake_case`.

Переменные

Чтобы объявить переменную, необходимо указать ее идентификатор, тип и значение.

Объявление начинается с ключевого слова `var`.

```
var myVar string = "Variable string" // (1)
var anotherVar string                // (2)
```

Особенностью языка Go является то, что переменная всегда имеет определенное значение (за исключением ссылочных типов, о которых поговорим далее). Таким образом можно инициализировать переменную без указания начального значения, как это сделано в строке (2). В этом случае Go присвоит значение по умолчанию для выбранного типа. Для строк значением по умолчанию является `""` (пустая строка), для числовых типов - 0, для булева типа - `false`, а для ссылочных типов, о которых поговорим позже, значением по умолчанию является `nil`.

При объявлении переменной можно не указывать тип. В этом случае Go сам определит тип согласно тому значению, которое указано при объявлении.

```
var noType = 100
```

Можно объявить сразу несколько переменных.

```
var a,b,c int // (3)
var d,e,f = "hello", 42, true // (4)
```

```

var (
    price      int
    qty        int
    isDeletable bool
)
// (5)

```

В строке (3) указанные переменные будут иметь значение по умолчанию, для типа `int` — это 0. В строке (4) Go установит нужные типы: `string`, `int` и `bool`. Начиная со строки (5), показан блок инициализации нескольких переменных.

Краткое объявление

Как правило, объявление через `var` используется тогда, когда для нас не важно начальное значение переменной, потому что оно будет присвоено в ходе работы программы как результат вызова функций.

Для объявления локальных переменных и присвоения им значения по месту используется синтаксис краткого объявления переменной.

```
pathToFile := getPath() // (6)
```

Таким же образом можно объявить сразу несколько переменных, как в случае с ключевым словом `var`.

```
str, number, isExist := "new string", 42, false
```

Для краткого объявления существует ограничение - в левой части выражения должна быть хотя бы одна новая (необъявленная ранее) переменная. Иначе вы получите ошибку компиляции.

```
str, number := "override string", 10 // no new variables on
left side of :=
```

Если вместо одной из двух переменных указать новый идентификатор, то программа выполнится. В этом случае для уже объявленных переменных краткое объявление работает как присваивание, а для новых — как объявление.

Этот прием часто используется. Например, когда вызываемая функция возвращает два и более значения (результат и ошибку). Также следует упомянуть о пустом идентификаторе `_` (символ нижнего подчеркивания), который используется в том случае, когда требуется использовать имя переменной, но в нашем коде эта переменная не нужна.

```
var path string = "/path/to/file"
f, err := os.Open(path)
```

```
str, _ := getParams()
```

Константы

Для объявления констант указывается ключевое слово `const` и значение. Особенностью констант в Go является то, что указание типа при объявлении не является обязательным. Также можно сразу в одном блоке объявить несколько констант.

```
const statusCode int = 200
const (
    orderStatusNew string = 'new'
    baseDiscount = 3.5
)
```

Если при объявлении константы не указать тип, то такая константа в терминах Go является нетипизированной. Так как нетипизированные константы не имеют привязки к конкретному типу, это позволяет Go работать с ними с повышенной точностью. Например, целочисленные константы представляются с точностью не менее 256 бит. Такие константы в процессе работы неявно преобразуются к нужному типу для выполнения вычисления, также при необходимости может потребоваться явное приведение к нужному типу.

```
const (
    untypedNum = 15
    typedNum int = 10
)
var c int32 = 30

fmt.Println(c + untypedNum) // 45
fmt.Println(c + typedNum) // invalid operation: c + typedNum
(mismatched types int32 and int)
```

При объявлении последовательности констант можно не указывать значения. В этом случае компилятор подставит константе значение предыдущей, то есть нужно указать значение только первой константе в блоке.

```
const (
    num1 = 21
    num2
    num3 = 10
    num4
)
fmt.Println(num1, num2, num3, num4) // 21 21 10 10
```

Генератор констант `iota`

При групповом объявлении связанных констант можно использовать генератор констант `iota`, значение которого увеличивается на единицу, начиная с 0. Несмотря на то, что `iota` выглядит как автоинкремент, его можно использовать в математических выражениях. Для пропуска ненужных значений можно использовать пустой идентификатор.

```

const (
    _ = 10 * iota
    speed10 = 10 * iota
    speed20 = 10 * iota
    speed30 = 10 * iota
    speed40 = 10 * iota
)

fmt.Println(speed10, speed20, speed30, speed40) // 10 20 30 40

```

Типы данных

Целые числа

Целые числа представлены в Go разной размерности, а также есть знаковые и беззнаковые типы. Размерность типа `int`, которую мы использовали в примерах выше, зависит от платформы и устанавливается автоматически компилятором.

```

var num int = 100

// int8, int16, int32, int64
var num8 int8 = 1 << 7 - 1

// uint8, uint16, uint32, uint64
var unsignedNum uint16 = 1 << 16 - 1

```

Числа с плавающей точкой

Вещественные числа в Go представлены двумя типами `float32` и `float64`. По умолчанию, если при объявлении переменной не указывать тип, то Go выберет `float64`.

```

f1 := 4.54
fmt.Printf("%T\n", f1) // float64

```

Булев тип

Как и в других языках программирования, булев тип имеет два значения `true` и `false`.

```

var b bool
var tr = true
isExist := true
fmt.Println(b, tr, isExist) // false true true

```

Строки

Строка объявляется в двойных кавычках и может содержать управляющие символы, такие как табуляция, перенос строки и т.д. Значением по умолчанию является пустая строка.

```

var str string // "" пустая строка

```

```
var newStr string = "New string"
```

В Go реализована поддержка UTF-8, поэтому свободно можно использовать различные языки.

Для работы с символами в Go реализованы два типа данных - byte и rune, которые представлены типами uint8 и uint32, соответственно. То есть, по сути своей они являются синонимами для указанных типов.

```
var strChar string = "!"  
rCode, _ := utf8.DecodeRuneInString(strChar)  
fmt.Printf("%U\n", rCode) // U+0021
```

```
var runeChar rune = '\x21'  
fmt.Println(string(runeChar)) // !
```

Конкатенация строк производится при помощи оператора + (плюс).

```
var hello = "Hello"  
fmt.Println(hello + " from RebrainMe!")
```

В Go строки представлены как последовательность байт, поэтому и работа со строками является по сути своей работой с байтами. Встроенная функция len() возвращает длину строки в байтах.

```
var greeting = "Привет!"  
fmt.Println(len(greeting)) // 13 байт  
fmt.Println(utf8.RuneCountInString(greeting)) // 7 символов (рун)
```

Можно обратиться к какому-то байту строки или получить подстроку при помощи оператора : (slice - срез). Как показано в примере ниже, если обратиться по индексу к одному байту, то вернется значение типа byte.

```
fmt.Println(greeting[3]) // 128  
fmt.Println(greeting[4:6]) // и
```

Используя оператор :, можно опустить одно или оба значения, в этом случае будут использованы значения по умолчанию - начало строки 0 и конец строки len(greeting).

```
fmt.Println(greeting[:6]) // При  
fmt.Println(greeting[:]) // Привет!
```

Строки в Go являются неизменяемой последовательностью байт, поэтому изменить строку нельзя. Чтобы изменить символ в строке, можно конвертировать ее в слайс рун, потом изменить и конвертировать в строку.

```
greeting[0] = "Л" // cannot assign to greeting[0]
```

```
var convGreeting = []rune(greeting)
convGreeting[4] = 'E'

fmt.Println(string(convGreeting)) // ПривЕт!
```

Поскольку строки являются неизменяемыми, то копирование и получение подстроки являются дешевыми, с точки зрения использования ресурсов, операциями, так как в этом случае не выделяется новая память.

Комплексные числа

В Go реализована поддержка комплексных чисел — `complex64` и `complex128`, составной частью которых являются `float32` и `float64`. Комплексное число можно объявить как литерал или воспользоваться встроенной функцией `complex()`.

```
var cmplx complex128 = 1.1 + 2.1i
cmplx3 := complex(2.1, 2)
```

```
fmt.Println(cmplx, cmplx3)
```

Для получения действительной и мнимой частей существуют встроенные функции `real` и `imag`. Для работы с комплексными числами можно воспользоваться пакетом `math/cmplx`.

Массивы

Массив является составным типом данных, который может содержать элементы одного типа. Особенностью массива в Go является его фиксированный размер, который нельзя изменить в процессе выполнения программы. Из-за этого массивы редко используются, так как в большинстве случаев мы можем не знать, сколько элементов будет в массиве. Узнать длину массива можно при помощи встроенной функции `len()`. При создании массива по умолчанию всем элементам задается нулевое значение заданного типа.

```
var defArr [3]string = [3]string{"one", "two", "three"}
fmt.Println(defArr) // [one two three]
```

```
var array [3]int
fmt.Println(array) // [0 0 0]
```

В случае, если потребуется объявить массив с заранее неизвестным количеством элементов, есть возможность определить это при объявлении при помощи оператора `....`

```
dynamic := [...]bool{4:true} // [false false false false true]
```

```
fmt.Println(dynamic)
fmt.Println(dynamic[2]) // false
```

В приведенном выше примере мы не только указали размер массива по фактическому количеству элементов, но и задали значение для элемента с индексом 4. Доступ к элементам массива осуществляется по индексу.

Массивы в Go передаются по значению, в отличие от других языков, в которых массивы неявно передаются по ссылке. Чтобы в Go передать массив по ссылке, надо явно передать указатель на массив. Также можно получить слайс элементов при помощи оператора `:`, который тоже будет ссылаться на элементы исходного массива.

```
arr := dynamic
arr[2] = true
fmt.Println(dynamic)           // [false false false false true]

arr2 := &dynamic
arr2[2] = true
fmt.Println(dynamic)           // [false false true false true]

sl := dynamic[:2]
sl[1] = true
fmt.Println(dynamic)           // [false true true false true]
```

Попытка получить элемент с индексом, который находится за пределами размера массива, приведет к ошибке компиляции программы.

```
fmt.Println(dynamic[7])        // invalid array index 7 (out of bounds
for 5-element array)
```

Определение типов

Как и в других типизированных языках, в Go есть возможность создавать новые типы, которые основываются на поддерживаемых базовых типах. Следует помнить, что в Go нет автоматического приведения типов. Поэтому значение базового типа необходимо самостоятельно явно конвертировать в созданный тип и наоборот.

```
type SettlementId string

var cityId SettlementId = "228edf0c-0c0d-4db6-bf0e-d508e68270a3"
var strCityId string = "228edf0c-0c0d-4db6-bf0e-d508e68270a3"

fmt.Println(cityId == SettlementId(strCityId))
```

Область видимости

Области видимости в Go ограничиваются лексическим блоком, в котором объявлена переменная. Выделяют всеобщий блок (universe block), блок пакета (о пакетах вы узнаете дальше), блок файла, блок функций, блок инструкций (if, for и т.д.). То есть переменная видна внутри блока и недоступна за пределами лексического блока.

Переменные могут перекрывать друг друга. Такие ситуации Go решает путем «поднятия» вверх от самого низкого по вложенности блока. Будет использована первая переменная с заданным именем, которую Go найдет по пути вверх.

Дальше вы узнаете о пакетах и более подробно - об экспорте методов, констант и переменных.

Полезные ссылки:

- [Типы в Go](#)
- [Go Type System Overview](#)
- [Конвертация типов данных в Go](#)

Задание:

1. Ознакомьтесь с математическими операциями в Go.
2. Ознакомьтесь с пакетом для преобразования типов `strconv`.
3. Создайте репозиторий с именем `module02` в gitlab.rebrainme.com в подгруппе `golang_users_repos/<your_gitlab_id>`, для работы над заданиями модуля.
4. Создайте новую директорию `module02` в `$GOPATH`, инициализируйте новый git-репозиторий и настройте работу с удаленным репозиторием проекта.
5. Создайте в репозитории новую ветку `01_task`.
6. Создайте новую директорию с файлом `main.go`. Напишите код, в котором:
 - объявите две переменные, первая - строка со значением `104`, вторая - целое число со значением `35`;
 - приведите строку к целому числу, а целое число - к строке;
7. Создайте новую директорию и новый файл `main.go`. Напишите код, в котором:
 - объявите два новых типа `AmericanVelocity` и `EuropeanVelocity`
 - выполните преобразование скорости `120.4` м/сек в км/ч и присвойте результат переменной с типом `EuropeanVelocity`;
 - выполните преобразование скорости `130` м/с в миль/ч и присвойте результат переменной с типом `AmericanVelocity`;
 - примечание: 1 миля = 1.609 км. Если потребуется, округлите значение до 2 знаков после запятой, для округления обратитесь к пакету `math`.
8. Создайте новый коммит с вашими решениями задач и отправьте в удаленный репозиторий в <https://gitlab.rebrainme.com/>.
9. В ответе пришлите ссылку на `merge request` в ветку `master` своего проекта ветки `01_task`.