Опановування основами Go: Практичний посібник з освоєння мови Go

Глава 2: Типи даних та змінні

OP. Оператори та зарезервовані слова

Усі спеціальні символи, що використовуються в мові Go, використовуються як оператори або символи пунктуації.

```
+    &     +=    &=     &&    ==    !=    (    )
-    |     -=    |=     ||    <     <=    [    ]
*    ^     *=    ^=     <-    >     >=    {    }
/    <<    /=    <<=    ++    =     :=    ,    ;
%    >>    %=    >>=    --    !     ...   .    :
     &^          &^=
```

Усі ключові слова використані у мові Go.

> Жодне з цих слів не можна використовувати як назву функції або змінну

```
# --- initial keywords ---
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
# --- added after releases ---
any          delete       clean        min          max
```

2.1 Базові типи даних в Go

Go пропонує різноманітність базових типів даних, які є будівельними блоками будь-якої програми Go. Розуміння цих типів даних є важливим для ефективної роботи зі значеннями, змінними та константами. У цьому розділі ми досліджуємо найпоширеніші базові типи даних в Go.

2.1.1 Числові Типи Даних

Go має декілька числових типів даних, які можна поділити на цілі, з плаваючою комою та комплексні числа.

Типи цілих чисел: Цілі числа - це числа без десяткових знаків. Go має як знакові (позитивні або негативні) так і беззнакові (тільки позитивні) типи цілих чисел. Доступні розміри цілих чисел становлять 8, 16, 32 та 64 біти. Додатково, є тип "int" що залежить від архітектури, який є 32 або 64 біти, в залежності від платформи.

- Знакові цілі числа: `int8`, `int16`, `int32`, `int64` та `int`
- Беззнакові цілі числа: `uint8`, `uint16`, `uint32`, `uint64` та `uint`
- Спеціальні типи цілих чисел: `byte` (псевдонім для uint8) та `rune` (псевдонім для int32)

Типи з плаваючою комою: Числа з плаваючою комою використовуються для представлення дійсних чисел з десятковими знаками. Go має два типи з плаваючою комою: `float32` та `float64`. Ці типи представляють 32-бітні та 64-бітні числа з плаваючою комою відповідно.

Типи комплексних чисел: Go підтримує комплексні числа, які використовуються для представлення чисел з дійсною та уявною частинами. Існують два типи комплексних чисел: `complex64` (з `float32` дійсною та уявною частинами) та `complex128` (з `float64` дійсною та уявною частинами).

2.1.2 Тип даних "Рядок"

Тип даних `string` в Go представляє собою послідовність символів, які зазвичай використовуються для зберігання та маніпулювання текстом. Стрічки в Go за замовчуванням кодуються в UTF-8, що спрощує роботу з символами Unicode. Рядки є незмінними, тобто один раз створені, їх значення не можуть бути змінені. Замість цього, при маніпулюванні рядком, створюється новий рядок зі зміненим значенням.

Трохи про **обходження обмежень незмінності рядків**:

Зміна значення рядка в Go може бути дещо складною, оскільки рядки є незмінними. Проте, ви можете досягти цього, перетворивши рядок на зріз рун, змінивши бажані значення, а потім перетворивши зріз рун назад в рядок. Ось крок-за-кроком процес:

1. Перетворіть рядок на зріз рун:

```
originalString := "Hello, World!"
runeSlice := []rune(originalString)
```

2. Змініть бажані значення в зрізі рун:

Скажімо, ви хочете змінити символ "H" в "Hello, World!" на "J". Ви можете зробити це, призначивши нове значення руни відповідному індексу в зрізі:

```
runeSlice[0] = 'J'
```

3. Перетворіть зріз рун назад в рядок:

Нарешті, ви можете перетворити зріз рун назад в рядок, використавши приведення `string()`:

```
modifiedString := string(runeSlice)
```

Ось повний код:

```go
package main

import "fmt"

func main() {
    // Declare a string
    originalString := "Hello, World!"
    // Declare new slice of runes.Convert the string to a slice of runes
    runeSlice := []rune(originalString)
    // Change the first character "H" to "J"
    runeSlice[0] = 'J'
    //here we are changing the value of originalString
    originalString = string(runeSlice)

    fmt.Println(originalString) // Output: Jello, World!
}
```

2.1.3 Булевий Тип Даних

Булевий тип даних в Go використовується для представлення значень `true` або `false`. Булеві значення часто використовуються в умовних виразах та інших логічних операціях. Два можливі значення для булевого типу - це `true` та `false`.

У наступному розділі ми дізнаємося, як оголошувати та ініціалізувати змінні за допомогою цих базових типів даних. Ми також розглянемо, як виконувати операції над ними, такі як арифметичні операції, порівняння та конкатенація. Тому, пристігніть паски безпеки 😉 та зануримось глибше в світ Go! 🚀

2.2 Оператори Go

Оператори - це спеціальні символи, які дозволяють вам виконувати операції зі змінними та значеннями. Go має багатий набір операторів, які можна класифікувати наступними типами:

2.2.1 Арифметичні оператори

Арифметичні оператори використовуються для виконання математичних операцій, таких як додавання, віднімання, множення та ділення. Ось арифметичні оператори в Go:

- **" + "**: Додавання `sum := 7 + 5 // sum становить 12`
- **" - "**: Віднімання `difference := 7 - 5 // difference становить 2`
- **" * "**: Множення `product := 7 * 5 // product становить 35`
- **" / "**: Ділення `quotient := 7 / 5 // quotient становить 1`
- **" % "**: Модуль (остача від ділення) `remainder := 7 % 5 // remainder становить 2`

2.2.2 Оператори порівняння

Оператори порівняння використовуються для порівняння двох значень. Ці оператори повертають булеве значення (істину або брехню), засноване на результаті порівняння. Ось оператори порівняння

в Go:

- **==**: Рівне `fmt.Println(7 == 5) // Виведення: false`
- **!=**: Не рівне `fmt.Println(7 != 5) // Виведення: true`
- **<**: Менше `fmt.Println(7 < 5) // Виведення: false`
- **<=**: Менше або рівне `fmt.Println(7 <= 5) // Виведення: false`
- **>**: Більше `fmt.Println(7 > 5) // Виведення: true`
- **>=**: Більше або рівне `fmt.Println(7 >= 5) // Виведення: true`

2.2.3 Логічні оператори

Логічні оператори використовуються для комбінування булевих значень в умовних виразах. Ось логічні оператори в Go:

- **&&**`**: Логічне І (повертає true, якщо обидва операнди є true)` fmt.Println(true && true) // Вивід: true`
- **||**: Логічне АБО (повертає true, якщо хоча б один з операндів є true) `fmt.Println(true || false) // Вивід: true`
- **!**: Логічне НЕ (повертає true, якщо операнд є false, та false, якщо операнд є true) `fmt.Println(!true) // Вивід: false`

2.2.4 Бітові оператори

Бітові оператори використовуються для виконання операцій над двійковим представленням чисел. Ось бітові оператори в Go:

- **&**: Бітовий AND `bitwiseAnd := 7 & 5 // bitwiseAnd є 5 (binary 0111 & 0101 = 0101)`
- **|**: Бітовий OR `bitwiseOr := 7 | 5 // bitwiseOr є 7 (binary 0111 | 0101 = 0111)`
- **^**: Бітовий XOR (exclusive OR) `bitwiseXor := 7 ^ 5 // bitwiseXor є 2 (binary 0111 ^ 0101 = 0010)`
- **&^**: очищення біта (AND NOT) `bitClear := 7 &^ 5 // bitClear є 4 (binary 0111 &^ 0101 = 0100)`
- **<<**: зсув вліво `leftShift := 7 << 5 // leftShift є 224 (binary 0111 << 5 = 11100000)`
- **>>**: зсув вправо `rightShift := 7 >> 5 // rightShift є 0 (binary 0111 >> 5 = 00000000)`

2.2.5 Оператори присвоєння

Оператори присвоєння використовуються для присвоєння значень змінним. Go має основний оператор присвоєння `=` та складні оператори присвоєння, що виконують операцію та присвоєння за один крок. Ось оператори присвоєння в Go:

- `:=`: Декларація та присвоєння `x:= 5 //декларуємо та присвоюємо x значення 5`
- `=`: Присвоєння

```
x := 7 // декларуємо та присвоюємо x значення 7
x = 5 // x тепер 5
```

- `+=`: Додати та присвоїти

```
x := 7
x += 5 // x тепер 12 (еквівалентно x = x + 5)
```

- `-=`: Відняти та присвоїти

```
x := 7
x -= 5 // x тепер 2 (еквівалентно x = x - 5)
```

- `*=`: Помножити та присвоїти

```
x := 7
x *= 5 // x тепер 35 (еквівалентно x = x * 5)
```

- `/=`: Розділити та присвоїти

```
x := 7
x /= 5 // x тепер 1 (еквівалентно x = x / 5)
```

- `%=`: Модуль та присвоєння

```
x := 7
x %= 5 // x тепер 2 (еквівалентно x = x % 5)
```

- `&=`: Бітова `AND` оператор та присвоєння

```
x := 7
x &= 5 // x тепер 5 (еквівалентно x = x & 5)
```

- `|=`: Бітова `OR` оператор та присвоєння

```
x := 7
x |= 5 // x тепер 7 (еквівалентно x = x | 5)
```

- ^=: Бітова XOR оператор та присвоєння

```
x := 7
x ^= 5 // x тепер 2 (еквівалентно x = x ^ 5)
```

- <<=: Зсув вліво та присвоєння

```
x := 7
x <<= 5 // x тепер 224 (еквівалентно x = x << 5)
```

- >>=: Зсув вправо та присвоєння

```
x := 7
x >>= 5 // x тепер 0 (еквівалентно x = x >> 5)
```

- &^=: Очистити біт та присвоїти

```
x := 7
x &^= 5 // x тепер 2 (еквівалентно x = x &^ 5)
```

Тепер у вас є приклади для всіх операторів Go, які повинні допомогти вам зрозуміти, як використовувати кожен з них у ваших програмах Go.

Ці оператори становлять основу роботи з даними і логікою, дозволяючи вам створювати більш складні додатки і вирішувати широкий спектр проблем.

2.3 Декларація та ініціалізація змінних

Змінні є будівельними блоками будь-якої мови програмування, дозволяючи вам зберігати та маніпулювати даними. У Go ви можете декларувати та ініціалізувати змінні кількома способами. У цьому розділі ми розглянемо найпоширеніші методи декларації та ініціалізації змінних в Go.

2.3.1 Використання ключового слова var

Ви можете оголосити змінну, використовуючи ключове слово var, за яким слідує ім'я змінної та її тип:

```
var age int = 30
```

2.3.2 Визначення типу за допомогою :=

Go підтримує виведення типу, що означає, що ви можете опустити тип змінної, якщо надаєте початкове значення. Компілятор автоматично виведе тип на основі значення. Щоб оголосити та ініціалізувати змінну з використанням виведення типу, використовуйте оператор `:=`:

```
name := "John Doe"
```

У цьому прикладі, змінна `name` буде типу `string`, як і випливає з початкового значення "John Doe". Зверніть увагу, що цей скорочений синтаксис можна використовувати тільки при оголошенні та ініціалізації нової змінної всередині функції.

2.3.3 Оголошення кількох змінних

Ви можете оголошувати та ініціалізувати кілька змінних одночасно, використовуючи ключове слово var або висновок типу за допомогою `:=`:

```
var x, y int = 10, 20
```

```
x, y := 10, 20
```

У обох випадках, `x` та `y` оголошені як змінні типу `int` та ініціалізовані значеннями 10 та 20 відповідно.

2.3.4 Ключове слово const

Щоб оголосити константну змінну (тобто змінну, значення якої не може бути змінено після ініціалізації), використовуйте ключове слово `const`:

```
const pi float64 = 3.14159
```

```
/* Identifiers and literals example  */
const (
    age      = 42         // decimal
    userName = "username" // string
    userId   = 0b101010   // binary
    UserData = 0o52       // octal
    a1       = 0x2a       // hexadecimal
)
```

Тепер у вас є тверде розуміння того, як декларувати та ініціалізувати змінні в Go. З цією базою, ви можете почати досліджувати більш складні типи даних та операції, а також почати створювати свої власні програми Go.

2.4 Константи та переліки

Константи - це незмінні значення, які не можуть бути змінені під час виконання програми. Вони корисні, коли вам потрібно визначити значення, які будуть використовуватися по всьому вашому коду, і які не повинні бути змінені випадково або навмисно. Переліки - це спосіб визначення послідовності пов'язаних константних значень, які, як правило, використовуються для представлення набору різних станів або варіантів.

2.4.1 Декларування констант

В Go ви декларуєте константу за допомогою ключового слова const, за яким слідує ім'я константи та її значення:

```
const appName string = "My Go App"
```

Ви також можете використовувати виведення типів для констант, так само як зі змінними:

```
const appName = "My Go App" // Type inferred as string
```

Ви можете оголосити декілька констант одночасно, використовуючи ключове слово var або виведення типу за допомогою `:=`:

```
const x, y int64 = 10, 20
```

2.4.2 Константні вирази

Go дозволяє вам створювати константні вирази, які обчислюються на етапі компіляції. Ці вирази можуть включати арифметичні, бітові та порівняльні операції з константами:

```
const x = 10 + 20 // x is a constant with value 30
```

> Зверніть увагу, що ви не можете використовувати виклики функцій або змінні в константних виразах.

2.4.3 Переліки

Go не має вбудованого ключового слова enum як деякі інші мови програмування, але ви можете створювати переліки за допомогою комбінації `const` та `iota`.

Ключове слово `iota` представляє автоматично збільшуване цілочислове значення, що починається з 0 та збільшується на 1 кожен раз, коли воно використовується в декларації констант.

Ось приклад створення переліку для днів тижня:

```
type Weekday int

const (
Sunday Weekday = iota
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
)
```

У цьому прикладі, ми спочатку визначаємо новий тип `Weekday` як псевдонім для типу `int`. Потім ми використовуємо ключове слово const та дужки для оголошення послідовності констант для днів тижня. Першій константі `Sunday` присвоюється значення `iota`, яке стартує з `0`. Для кожної наступної константи, значення iota збільшується на `1`. Отже, `Monday` отримує значення `1`, `Tuesday` отримує значення `2`, і так далі.

Тепер ви можете використовувати тип Weekday та його константи у вашому коді, ось так:

```
package main

import "fmt"

type Weekday int

const (
    Sunday Weekday = iota
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
    Saturday
)

func main() {
    today := Monday
    fmt.Println("Today is:", today) // Output: Today is: 1
}
```

Маючи ці знання про константи та перерахування в Go, ви можете визначати незмінні значення та набори пов'язаних констант, щоб зробити ваш код більш надійним та виразним.

Давайте дослідимо більш складні випадки використання iota з декількома прикладами.

Бітовий Зсув:

Припустимо, ви хочете визначити константи, які представляють бітові прапорці, використовуючи степені 2. Ви можете використовувати iota з бітовим зрушенням для досягнення цього:

```
const (
Flag1 uint = 1 << iota
Flag2
Flag3
Flag4
)
```

У цьому прикладі, значення iota починається з 0 і збільшується з кожною константою. Константи отримують результат зсуву значення 1 вліво на позиції з iota:

- Flag1: 1 << 0 = 1
- Flag2: 1 << 1 = 2
- Flag3: 1 << 2 = 4
- Flag4: 1 << 3 = 8

Пропуск значень: Якщо ви хочете пропустити певні значення у переліку, ви можете використовувати пустий ідентифікатор _, щоб проігнорувати поточне значення iota. Наприклад:

```
type State int

const (
Active State = iota
Inactive
_ // Skipping the value 2
Deleted
)
```

У цьому прикладі, Active отримує значення 0, Inactive отримує значення 1, а Deleted отримує значення 3. Значення 2 пропускається.

Користувацькі вирази: Ви можете використовувати користувацькі вирази з iota для створення більш складних переліків. Наприклад, давайте визначимо перелік для геометричних форм і їхня кількість сторін:

```
type Shape int

const (
Circle    Shape = -1
Triangle        = 3 * (1 << iota)
Square
Pentagon
Hexagon
)
```

У цьому прикладі, константі Circle присвоєно значення -1, а значення iota починається з 1 для Triangle. Константам Triangle, Square, Pentagon та Hexagon присвоєно результат множення 3 на значення 1 << iota:

- Triangle: `3 * (1 << 1) = 6`
- Square: `3 * (1 << 2) = 12`
- Pentagon: `3 * (1 << 3) = 24`
- Hexagon: `3 * (1 << 4) = 48`

Ці приклади демонструють гнучкість та силу ключового слова iota в Go.

З невеликою креативністю, ви можете використовувати iota для створення широкого діапазону переліків та послідовностей констант для задоволення ваших специфічних потреб.