

Go

Начальный курс по Go

ПЕРЕМЕННЫЕ

Переменные

Синтаксис:

имя переменной

`var count int`

ключевое слово

тип

Переменные

Синтаксис (объявление + присваивание):

```
var count int = 10
```

```
var count = 10
```

```
count := 10
```

} стараться использовать этот вариант

short version

```
a := int8(4)
```

Приведение типов

запомнить

В Go нет неявного приведения типов, если встречаются в одном выражении два разных типа, то приводить нужно вручную через выражения `int8(x)`, `int16(x)` и т.д.

var

var (

 a int = 1

 b int = 2

)

var

```
var a, b, c int
```



у всех будет тип int

```
var a, b, c = 1, 2, 3
```


short version

a, b, c := 1, 2, 3




слева должно быть
хотя бы одно новое имя

Для тех имён, которые уже определены,
работает как =

swap

a, b = b, a



меняем местами a и b

УСЛОВИЯ

if, else

```
if <expr> { ... }
```

```
if <expr> { ... } else { ... }
```

Важно:

1. <expr> всегда должно быть boolean!
2. Всегда ставьте { }* (кроме одного случая)

Ключевые операторы

`==` - проверка на равенство

`!=` - проверка на неравенство

`>` - больше

`<` - меньше

`>=` - больше или равно

`<=` - меньше или равно

Важно, результат всегда **boolean**!

Ключевые операторы

! - инвертирование результата выражения

&& - объединение нескольких выражений (И)

|| - объединение нескольких выражений (ИЛИ)

Важно: результат всегда **boolean**!

Выдача кредита

Написать функцию, которая запрашивает у пользователя: возраст и ежемесячный доход

Если возраст больше 21 и меньше 60, а доход больше 30_000, то сообщает о том, что возможна выдача кредита

Возраст кошки

Напишите программу, которая переводит
возраст кошки в возраст человека:

Возраст кошки (в годах)	Возраст человека (в годах)
1	15
2	24
3	28
4	32
5	36
6	40
7	44
8	48

Множественные условия

```
func main() {  
    if <expr> {  
        // do something  
    } else if <expr> {  
        // do something  
    } else {  
        // do something  
    }  
}
```

switch

```
switch <expr> {  
    case <val>: <stmt>  
  
    case <val>: <stmt>  
  
    default: <stmt>  
  
}
```

switch

```
switch {  
    case <expr>: <stmt>  
    case <expr>: <stmt>  
    default: <stmt>  
}
```

switch

fallthrough - "проваливаемся" вниз через
следующие **case**

Важно: может быть только последним
statement'ом в case

ЦИКЛЫ

Цикл

Многократно повторяющийся набор инструкций

Цикл for

```
for {
```

```
    ... // здесь инструкции
```

```
}
```

Бесконечный цикл

Цикл for

```
for <check> {  
    ... // здесь инструкции  
}
```

check – условие продолжения

Цикл for

```
for <init>; <check>; <post> {
```

```
    ... // здесь инструкции
```

```
}
```

- init – выражение инициализации
- check – условие продолжения
- post – подготовка к следующей итерации

Цикл for + range

```
for index, value := range data {
```

```
    ... // здесь инструкции
```

массив



```
}
```

Сам следит за перебором

BREAK & CONTINUE

Break & Continue

break – прерывает текущую итерацию цикла и
ВЫХОДИТ ИЗ ЦИКЛА

continue – прерывает текущую итерацию цикла и
переходит к следующей итерации

Break & Continue*

`break` & `continue` по умолчанию осуществляют выход только из одного уровня цикла*

Возможно использование Labels для выхода из нескольких уровней вложенности

ПЗ: break

TODO: подсчитать сумму значений, если все значения положительные

```
scores := []int{1, 200, -50, 10, -20}
```

ПЗ: continue

TODO: подсчитать сумму значений,
пропуская все отрицательные

```
scores := []int{1, 200, -50, 10, -20}
```

ПЗ: break & continue

Решить, нужно ли начислять бонус сотруднику
(бонус начисляется, если все оценки больше 6)

[8, 7, 9, 5, 8, 7, 9] – 0 (no early exit)

[8, 7, 9, 8, 7, 9] – 6

Используйте цикл **for** и **break** или **continue**

за каждую > 6 делаем +1 балл (вернётся кол-во баллов)

Многомерные массивы/слайсы

```
data := [][]int{  
    {1, 8, 5, 12},  
    {10, 7, 7, 18},  
};
```

Доступ к элементу: `data[i][j]`

label, break & continue*

Label:

break Label

continue Label

Label – помеченная точка в функции, к которой можно перейти через break или continue (минуя ограничение на выход только из самого вложенного цикла)

```
data := [][][]int{
    {{5, 5, 2, 8}, {4, 3, 5, 6}}, {{5, 5, 4}, {5, 4, 4}, {4, 4, 3}},
};
```

Outer:

```
for _, days := range data {
    bonus := 0
    for _, day := range days {
        for _, score := range day {
            if score < 3 { continue Outer }
        }
        bonus++
    }
    fmt.Println(bonus)
}
```

Важно

Использовать label'ы не рекомендуется!

goto

В Go есть `goto`, который позволяет внутри функции перейти по `Label`, но делать так не нужно!

Выручка в магазинах

У вас есть ежедневные данные выручки в рублях из трёх магазинов

Вам нужно найти:

1. Лучший магазин (по суммарной выручке за неделю)
2. Лучший магазин (по средней ежедневной выручке)
3. Магазин с лучшей за неделю ежедневной выручкой
4. Магазин с худшей за неделю ежедневной выручкой

FUNCTIONS

Функция

Именованный* блок кода – который может принимать какие-то параметры на вход и возвращать результат

Примечание*: могут быть и без имени

Объявление функции

```
func max(a int, b int) int {  
    // TODO:  
    return ...  
}
```

параметры функции

имя функции

тип результата

возврат результата

Вызов функции

```
func main() {  
    result := max(1, 20)  
}
```


аргументы функции

имя функции

The diagram consists of two blue lines. The first line starts from the text 'аргументы функции' (arguments of the function) and points to the arguments '1, 20' inside the 'max' function call. The second line starts from the text 'имя функции' (function name) and points to the 'max' function name itself. Both lines have short horizontal segments at their starting points near the code.

Объявление функции

Если аргументы одного типа, то допускается тип писать один раз для нескольких:



параметры функции

```
func max(a, b int) int {  
    return ...  
}
```

Функция

В Go функция может возвращать и несколько результатов:

```
func minmax(items []int) (int, int) { }
```

типы возвращаемых результатов

```
min, max := minmax(items)
```

Функция

Каждому результату можно дать имя
(равносильно тому, что создали локально
переменные):

```
func minmax(items []int) (min, max int) {  
    return имена возвращаемых результатов  
}
```

Early Exit

В функции не обязательно должен быть только один `return`.

Вы можете возвращать значение сразу и не создавать вложенности кода

Early Exit

```
func creditCheck(age, salary int) boolean {
```

```
    const minAge = 21
```

```
    if age < minAge {
```

```
        return false
```

```
    }
```

дальше проверять нет смысла, поэтому сразу выходим

```
    ...
```

```
}
```

Функции как значения

Сигнатура функции определяет тип –
функцию можно класть в переменную и
передавать как значение (functions are first
class citizens)

Спасибо за внимание

Ильназ Гильязов

2020г.