

Golang

ОСНОВЫ ЯЗЫКА



2021



Информация

Данная публикация стала возможной благодаря помощи американского народа, оказанной через Агентство США по международному развитию (USAID). Алиф Академия несёт ответственность за содержание публикации, которое не обязательно отражает позицию USAID или Правительства США.



ПРЕДИСЛОВИЕ



Переменные

Сегодня мы в первую очередь будем говорить о тех вещах, которые многие новички пропускают, считая их не интересными.

Но эти пробелы в знаниях приводят к тому, что в программах появляются ошибки (баги), которые обходятся очень дорого. Особенно это касается работы с деньгами.



ПЕРЕМЕННЫЕ



Переменные

На прошлой лекции мы познакомились с переменными - специальными именами для наших данных.

Мы рассмотрели следующий сценарий: внутри функции `main` (пока для нас - это просто кусок кода, который выполняется при запуске нашей программы) мы можем объявлять переменные следующим образом:

```
main.go > ...  
1  package main  
2  
3  func main() {  
4      amount := 100.0  
5      buyRate := 10.3150  
6      result := amount * buyRate  
7      println(result)  
8  }
```

Примечание: приложение с предыдущей лекции.



Переменные

Общий синтаксис (правила написания) вот такой:

имя := значение

Этот синтаксис называется сокращённым (short variable declaration)



Переменные

Полный синтаксис выглядит следующим образом:

ключевое слово

↓
`var имя тип` // объявление (declaration)

`имя = значение` // присваивание

↑
комментарий

Либо если мы сразу знаем, какое значение хотим положить в переменную, то:

`var имя тип = значение`



Переменные

Т.е. наша программа будет выглядеть вот так:

```
GO main.go > ...  
1  package main  
2  
3  func main() {  
4      var amount float64 = 100.0  
5      var buyRate float64 = 10.3150  
6      var result float64 = amount * buyRate  
7      println(result)  
8  }
```

Несмотря на то, что так делать можно, старайтесь так не делать. В Go предпочитают сокращённый вариант инициализации переменных (через `:=`).



Переменные

Итак, у каждой переменной есть:

1. Имя
2. Значение
3. Тип
4. Время жизни



Имя переменной

1. Должно быть осмысленным
2. Начинается с буквы (_, \$ - не рекомендуется)
3. Содержит буквы (цифры, _, \$ - не рекомендуется)
4. Если состоит из двух и более слов пишется в нотации camelCase (**buyRate**, а не buy_rate)
5. Имя на английском языке (без транслита – никаких summaOperacii и подобных!)

Замечание про английский язык очень важное: можно называть переменные на русском (или таджикском), но тогда вы не сможете работать в международной команде. Поэтому привыкайте использовать переводчик и называть переменные по-английски.



Имя переменной

Важно: имена переменных должны быть понятными!

GO main.go > ...

```
1  package main
2
3  func main() {
4      a := 100.0
5      b := 10.3150
6      c := a * b
7      println(c)
8  }
```

Вот в этой программе вообще непонятно, что значит **a**, **b** и **c**. И для чего они нужны.



Ключевые слова

Ключевые слова - это специальные зарезервированные слова, которые используются в программе для определённых конструкций и не могут быть именами переменных.

В Go их не так уж и много, поэтому нужно запомнить их:

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



КОММЕНТАРИИ



Комментарии

Комментарии - это пояснения к тексту кода, не влияющие на итоговый результат. В

Go есть два типа комментариев:

1. Строчные (начинаются с символов `//` и до конца строки) ← используются в основном
2. Блочные (начинаются с символов `/*` и заканчиваются `*/`)

```
go main.go > ...
1  package main
2
3  func main() {
4      var amount float64 = 100.0
5      var buyRate float64 = 10.3150
6      // Конвертация USD в TJS
7      var result float64 = amount * buyRate
8      println(result)
9  }
```

Если мы запустим эту программу, ничего не изменится, потому что Go просто проигнорирует эту строку (целиком).



Комментарии

Комментарии нужно использовать для документирования своего кода:

1. Участков кода, которые содержат не очень простую логику
2. Функций, типов данных (о них поговорим чуть позже)

Но это надо делать аккуратно! Не нужно комментировать каждую строку:

```
go main.go > ...
1  package main
2
3  // Функция main
4  func main() {
5      // Объявление переменной amount
6      var amount float64 = 100.0
7      // Объявление переменной buyRate
8      var buyRate float64 = 10.3150
9      // Конвертация USD в TJS
10     var result float64 = amount * buyRate
11     // Вывод результата
12     println(result)
13 }
```

пример плохого кода



Комментарии

Вместо того, чтобы писать комментарии к каждой строке программы, старайтесь писать саму программу так, чтобы комментарии не потребовались:

```
main.go > ...  
1  package main  
2  
3  func main() {  
4      count := 10  
5      price := 200  
6      sum := count * price  
7      discount := 30  
8      total := sum * (100 - discount) / 100  
9      println(total)  
10 }
```

Если взять словарь и перевести имена переменных, получится:

- count - количество
- price - цена
- sum - сумма
- discount - скидка
- total - итого



Комментарии

Иногда комментарии используют, чтобы временно "отключить" какую-то часть кода (для комментирования куска кода используйте **Ctrl + /** и это же сочетание для раскомментирования):

```
go main.go > ...  
1  package main  
2  
3  func main() {  
4      // amount := 100.0  
5      // buyRate := 10.3150  
6      // result := amount * buyRate  
7      // println(result)  
8  }
```

Теперь, если запустить, то ничего не выведется, поскольку функция **main** пуста.

Так можно сделать на время, пока вы редактируете код, но использовать как постоянный механизм - не нужно. Для этого существуют специальные системы контроля версий (мы будем проходить Git).



ТИПЫ ДАННЫХ



Типы данных

У каждой переменной есть свой тип данных. Тип определяет, какие значения может хранить переменная (представляйте тип данных как тип автомобиля - легковая, грузовая и т.д.).

В Go есть следующие типы:

- целые числа
- вещественные числа
- строки
- логический тип



Хранение данных

Всё в компьютере хранится в виде бит и байт, где бит - это 0 или 1. А байт - группа из 8 бит (так удобнее считать):

0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

И всё зависит от того, как мы договоримся понимать эти байты. Например, мы можем сказать, что это целое число. Тогда пронумеруем справа налево с нуля и каждое значение будем умножать на 2 в соответствующей степени:

7	6	5	4	3	2	1	0	← номер
0	1	0	0	0	0	0	1	

$$0 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 65$$



Хранение данных

7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1

$$0 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 65$$

На самом деле, ничего здесь сложного нет и мы уже делаем это в повседневной жизни, только с десятками.

Например, число 65 - это $6 * 10^1 + 5 * 10^0$. То, на степень какого числа умножается, обычно записывается в виде нижнего индекса:

$$0100\ 0001_2 = 65_{10}$$



Хранение данных

0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

А можем договориться, что это не числа, а буквы. Тогда нам придётся придумать табличку вроде такой (написать, какой букве что соответствует):

ASCII - Binary Character Table

Letter	Binary
A	01000001
B	01000010
C	01000011
D	01000100



Литералы

Литерал - это значение, которое записано в коде (зелёного цвета или красного цвета):

GO main.go > ...

```
1 package main
2
3 func main() {
4     amount := 100.0
5     buyRate := 10.3150
6     result := amount * buyRate
7     println(result)
8 }
```

GO main.go > ...

```
1 package main
2
3 func main() {
4     count := 10
5     price := 200
6     sum := count * price
7     discount := 30
8     total := sum * (100 - discount) / 100
9     println(total)
10 }
```

GO main.go > ...

```
1 package main
2
3 func main() {
4     println("Hello world")
5 }
```



Литералы

На основании этих литералов Go сам определяет тип данных для переменной (нам необязательно его указывать):

```
go main.go > ...  
1  package main  
2  
3  func amount float64  
4      amount := 100.0  
5      buyRate := 10.3150  
6      result := amount * buyRate  
7      println(result)  
8  }
```

Для целых чисел: `int`

Для вещественных: `float64`

Для данных в кавычках: `string`

Давайте разбираться с тем, что же это за типы



ЦЕЛЫЕ ЧИСЛА



Целые числа

- `int` - 4 или 8 байт
- `int8` – 1 байт (-128 до 127)
- `int16` – 2 байта (-32768 до 32767)
- `int32` – 4 байта (-2147483648 до 2147483647)
- `int64` – 8 байт (-9223372036854775808 до 9223372036854775807)

`int` - по умолчанию для целых чисел будет `int32` для 32 битных систем и `int64` для 64 битных систем.

Мы почти везде будем использовать `int`, поэтому тип писать не потребуется (если мы используем `:=`)



Целые числа

- `uint` - 4 или 8 байт
- `uint8` – 1 байт (0 до 255)
- `uint16` – 2 байта (0 до 65535)
- `uint32` – 4 байта (0 до 4294967295)
- `uint64` – 8 байт (0 до 18446744073709551615)



Целочисленный тип

Что же это за границы такие? Давайте посмотрим на следующий пример:

main.go > ...

```
1 package main
2
3 func main() {
4     var balance int32 = 1_500_000_000
5     var income int32 = 1_500_000_000
6     var total int32 = balance + income
7     println(total)
8 }
```

поскольку в короткой форме будет просто `int` (на x64 будет `int64`), то мы специально указали `int32`

Как вы думаете какой будет результат?

Результат будет `-1294967296`. Почему так? Давайте вспомним, что всё хранится в байтах (а тут - 32 байта). Когда значение превышает допустимые пределы, оно просто переходит на другую сторону (в случае чисел со знаком - в отрицательную, в случае чисел без знака - начинает заново с нуля).



Как вы видели, мы использовали `_`, чтобы было удобнее читать, какое же это число.

Это одна из ключевых вещей - удобочитаемость кода. Для Go никакой разницы нет, пишете вы `_` или нет. Но когда вы и ваши коллеги будут читать, вам гораздо легче будет понять, что это за число, чем считать количество `0`.



Отрицательные числа

Мы с вами видели, как можно хранить положительные числа:

7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1

$$0 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 65$$

А как же хранят отрицательные? Договорились, что самый старший бит (слева) будут выделять под хранение знака (если там 0 - то число положительное, если 1 - отрицательное).

Именно поэтому диапазон значений для чисел без знака (`uint8`) от 0 до 255, а для чисел со знаком (`int8`) от -128 до 127.



Отрицательные числа

Но с этим есть проблема:

Если $1_{10} = 0000\ 0001_2$

Тогда $-1_{10} = 1000\ 0001_2$

Проблема (складываем столбиком, как в школе):

$$\begin{array}{r} 0000\ 0001_2 \\ + \\ 1000\ 0001_2 \\ \hline 1000\ 0010_2 \end{array}$$

Получилось -2, а должно было получиться 0!



Отрицательные числа

Поэтому придумали делать вот так:

1. Берём модуль числа (значение без знака) $-1_{10} \rightarrow 0000\ 0001_2$
2. Инвертируем все биты (меняем 0 на 1, а 1 на 0) $\rightarrow 1111\ 1110_2$
3. Прибавляем 1 $\rightarrow 1111\ 1111_2$

Тогда проблемы нет:

$$\begin{array}{r} 0000\ 0001_2 \\ + \\ 1111\ 1111_2 \\ = \\ 0000\ 0000_2 \end{array}$$



Отрицательные числа

Но если числа "не умещаются", то происходит переполнение:

$$\begin{array}{r} 0111\ 1111_2\ (127_{10}) \\ + \\ 0000\ 0001_2\ (1_{10}) \\ = \\ 1111\ 1110_2\ (-128_{10}) \end{array}$$

В случае, если у нас не один байт, а 4, всё будет так же, только числа будут больше.

Об этом важно знать, поскольку будет достаточно забавно, когда у клиента на счету была положительная сумма, ему пришёл перевод и сумма на счету вдруг стала отрицательной.



Отрицательные числа

GO main.go > ...

```
1 package main
2
3 func main() {
4     var num int8 = 127
5     var result int8 = num + 1
6     println(result)
7 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:18009
-128
Process exiting with code: 0

GO main.go > ...

```
1 package main
2
3 func main() {
4     var num int8 = 0b0111_1111
5     var result int8 = num + 0b0000_0001
6     println(result)
7 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:41356
-128
Process exiting with code: 0

0b позволяет вам записывать числа сразу в двоичной системе.

0x в шестнадцатеричной

0 или 0o в восьмеричной (поэтому никогда не пишите 0 спереди):

```
3 func main() {
4     var num int8 = 022
5     println(num)
6 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:26651
18

← не пишите 0 спереди, это будет не 22!

ОПЕРАТОРЫ



Операторы

Операторы - это специальные инструкции, которые позволяют выполнять различные операции. Самые простые: $+$, $-$, $*$, $/$ и т.д.

Набор операторов фиксирован:

- нельзя создать новый оператор
- нельзя переопределить поведение существующего



Арифметические операторы

Именно тип определяет, какие операторы можно использовать:

- **+** (сумма или "склеивание строк") - целые и вещественные числа, строки
- **-** (разность) - целые и вещественные числа
- ***** (умножение) - целые и вещественные числа
- **/** (деление) - целые и вещественные числа
- **%** (остаток от деления) - целые числа



Важно

В Go в одном выражении (комбинации операторов и имён) могут участвовать только переменные одного типа (как это обойти, мы поговорим на следующей лекции).

Т.е. нельзя написать вот так:

```
main.go > ...  
1  package main  
2  
3  func main() {  
4      amount := 100  
5      buyRate := 10.315  
6      result := amount * buyRate  
7      println(result)  
8  }
```

invalid operation: amount * buyRate (mismatched types int and float64) go
Peek Problem (Alt+F8) No quick fixes available

Когда видите красное подчёркивание - всегда наводите указатель мыши, чтобы понять, что именно неправильно.



Важно

Кроме того, тип выражения равен типу входящих в него переменных. Что это значит, это значит, что если мы будем делить целые числа, то получим целое число:

```
main.go > ...  
1  package main  
2  
3  func main() {  
4      amount := 100  
5      count := 3  
6      result := amount / count  
7      println(result)  
8  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

33



Операторы сравнения

Операторы сравнения позволяют сравнивать два значения и возвращать результат сравнения - верно (**true**) или ложно (**false**).

true и **false** - это два возможных значения для типа **bool**.

go main.go > ...

```
1 package main
2
3 func main() {
4     balance := 100
5     limit := 150
6     println(balance > limit)
7 }
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

API server listening at: 127.0.0.1:11858

false



ВЕЩЕСТВЕННЫЕ ЧИСЛА



Вещественные числа

Вещественные числа - это числа с плавающей точкой. В Go представлены типами `float32` и `float64`. Для литералов тип по умолчанию - `float64`.

Что за плавающая точка? Дело в том, что мы рассмотрели, как хранятся целый и отрицательные числа. Очень упрощённо можно сказать, что*: любое вещественное число, например, 10.315 можно представить как $0.10315 * 10^2$. Таким образом, мы можем отдельно хранить:

знак (1 бит)	степень (8 бит)	дробная часть (23 бит)
0	2	10315

Примечание*: на самом деле всё немного сложнее, но для общего понимания нам будет достаточно этой идеи.



Вещественные числа

знак (1 бит)	степень (8 бит)	дробная часть (23 бит)
0	2	10315

Такая схема позволяет хранить как очень большие, так и очень маленькие числа:

$0.10315 * 10^{100}$ - большое число

$0.10315 * 10^{-100}$ - маленькое число



Вещественные числа

Но есть большая проблема:

```
go main.go > ...  
1  package main  
2  
3  func main() {  
4      before := 1_000_000_000_000_000_000_000.0  
5      after := before + 10_000  
6      difference := after - before  
7      println(difference)  
8      println(difference == 0)  
9  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
API server listening at: 127.0.0.1:11792  
+0.000000e+000  
true
```

Что произошло? Фактически, 10_000 улетело в никуда. Это особенность вещественных чисел почти во всех языках (не только в Go).



СТРОКИ



Строки

Строки - это набор символов, заключённых в кавычки.

"Кавычки" могут быть:

- `""` (двойные) - значение должно целиком помещаться в одну строку программы

```
~GO main.go > ...  
1  package main  
2  
3  func main() {  
4      message := "Добро пожаловать на курс Go!"  
5      println(message)  
6  }
```

- ``` (бэктики) - значение может располагаться на нескольких строках программы

```
~GO main.go > ...  
1  package main  
2  
3  func main() {  
4      message := `Добро пожаловать на курс Go!  
5      Вам предстоит 33 занятия и 66 домашних работ.  
6  
7      Нужно сдать ДЗ каждой недели до дедлайна, иначе вы не получите допуск к следующей неделе.  
8      `  
9      println(message)  
10 }
```



НУЛЕВЫЕ ЗНАЧЕНИЯ



Нулевые значения

Когда вы объявляете переменную с помощью `var` (например, `var count int`), но не присваиваете ей значение, то исходя из типа данных переменная принимает нулевое значение:

- целые числа - `0`
- вещественные - `0.0`
- `bool` - `false`
- строки - пустая строка (строка, в которой нет символов)



ИТОГИ



Итоги

В этой лекции мы обсудили

1. Ключевые типы, с которыми вы будете сталкиваться чаще всего
2. Проблемы, связанные с целыми и вещественными числами
3. Ключевые операторы

Вам важно чётко в этом разобраться, чтобы в дальнейшем не наткнуться на те ошибки, которые мы озвучили.



ДОМАШНЕЕ ЗАДАНИЕ



ДЗ №1: Ожидаемый доход

Alif [предлагает вклады](#), для которых можно рассчитать ожидаемый доход по следующей логике:

Рассчитайте свой доход

Пример расчёта процентов по вкладу носит информационный характер, не является публичной офертой и не включает расходы по налогам

Ожидаемый доход в год, TJS

Ожидаемый доход в год, TJS

Сумма вклада	500 000	сомони / TJS
Корти милли БЕССРОЧНО	Полугодовой 6 МЕС.	Годовой 12 – 36 МЕС.
от 20 000 до 30 000	от 50 000 до 60 000	от 70 000 до 80 000
4 – 6%	10 – 12%	14 – 16%
Открыть вклад		



ДЗ №1: Ожидаемый доход

Что нужно сделать? Напишите программу, которая рассчитывает и выводит минимальный (от) и максимальный (до) ожидаемый доход в год TJS для Корти Милли.

Ваша программа должна выглядеть вот так:

```
go main.go > ...  
1  package main  
2  
3  func main() {  
4      amount := 999_999_00  
5      minPercent := 0  
6      maxPercent := 0  
7      minIncome := 0  
8      maxIncome := 0  
9      println(minIncome)  
10     println(maxIncome)  
11 }
```

Все суммы должны считаться в дирамах и выводиться в дирамах (обратите внимание, денег ниже 1 дирама в обороте не существует).

Вам нужно подставить соответствующие значения и выражения вместо 0.

Важно: бот будет проверять, что переменные называются именно так, как указано и имеют те же значения, как указано.



ДЗ №1: Ожидаемый доход

Каталог и модуль (`go mod init`) должны называться deposit.



ДЗ №2: Мегафон Спасибо

Мегафон Таджикистан предлагает бонусную программу [Спасибо](#):

Подключаясь к сети «МегаФон Таджикистан» вы автоматически становитесь участником бонусной программы «Спасибо» и **получаете 5 пригласительных баллов**. А в дальнейшем просто пользуясь мобильной связью, получаете за это бонусные баллы, которые можете обменять на приятные вознаграждения: дополнительные минуты общения, пакеты SMS, Интернет-пакеты, услуги, брендированные предметы и мобильное оборудование.

Как накопить баллы?

- Бонусные баллы начисляются автоматически: **за каждые 5 сомони (с учетом НДС и акциза), потраченные на услуги связи, Вам начисляется 1 балл.**
- Баллы начисляются исходя из суммы, потраченной за предыдущий календарный месяц.
- Проверить текущий бонусный счет можно набрав USSD команду *100#



ДЗ №2: Мегафон Спасибо

Что нужно сделать? Напишите программу, которая рассчитывает и выводит бонус в баллах для нового пользователя, который потратил на услуги связи 666 сомони.

Ваша программа должна выглядеть следующим образом:

```
main.go > {} main > main
```

```
1 package main
2
3 func main() {
4     amount := 666
5     welcomeBonus := 0
6     // другие переменные на ваше усмотрение
7     bonus := 0
8     println(bonus)
9 }
```

Все суммы должны считаться в сомони и выводиться в сомони.

Вам нужно подставить соответствующие значения и выражения вместо 0.



ДЗ №2: Мегафон Спасибо

Каталог и модуль (`go mod init`) должны называться `megafon`.



Спасибо за внимание

alif academy

2021

