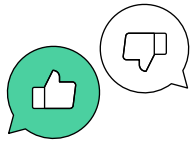
Условные конструкции и циклы

Матвей Ефимов
Эксперт в Go, Python, SQL

Проверка связи

Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти



Поставьте в чат:

\+ если меня видно и слышно

\- если нет

Рекомендации

Если смотрите с компьютера

- Используйте браузеры **Google Chrome** или **Microsoft Edge**
- Если есть проблемы с изображением или звуком, обновите страницу — **F5**

Если смотрите с мобильного телефона или планшета

- Перейдите с мобильного интернет-соединения на **Wi-Fi**
- Если есть проблемы с изображением или звуком, перезапустите приложение **МТС Линк** на телефоне
- Предварительно проверьте, подходит ли ваше устройство для подключения к вебинару, по ссылке

Правила участия

1. Приготовьте блокнот и ручку, чтобы записывать важные мысли и идеи
2. Продолжительность вебинара — 1 час
3. Вы можете писать свои вопросы в чате
4. Запись вебинара будет доступна в личном кабинете

Матвей Ефимов

О спикере:

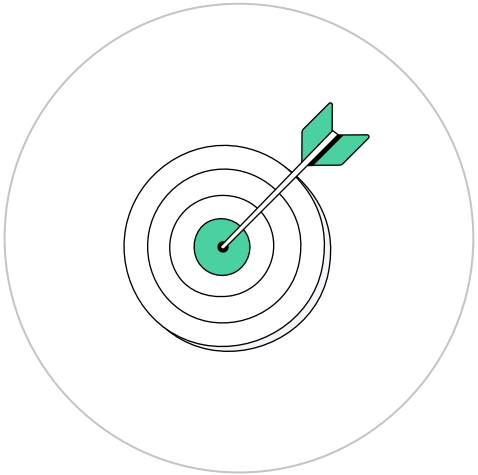
- Преподаватель с более чем 5-летним опытом в IT-образовании
- Разработчик учебных программ и эксперт в Go, Python, SQL
- Фриланс-программист и наставник студентов от школ до вузов

Какие у вас ожидания от вебинара?

Ваши вопросы?

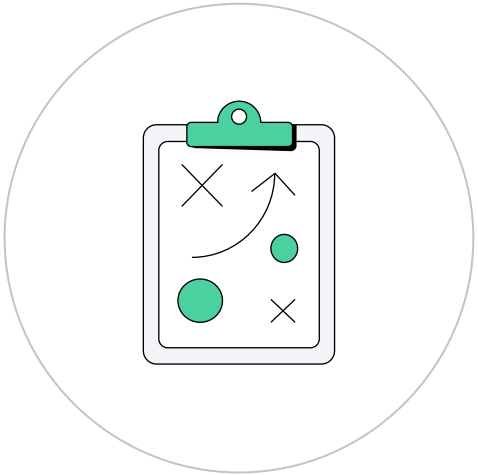
Цели занятия

- Научиться использовать if, else, switch
- Разобрать цикл for во всех его формах
- Узнать частые ошибки и лучшие практики
- Решить практические задачи



План занятия

1. Условные конструкции в Go
2. Циклы в Go
3. Примеры задач и кода
4. Итоги



*Нажмите на нужный раздел для перехода

Условные конструкции в Go

1

Что такое условные конструкции?

Условные конструкции — это способ задать в программе логику: выполнить одни действия, если условие выполнено, и другие — если нет

Они позволяют:

- Управлять логикой программы
- Реагировать на разные ситуации
- Делать код гибким и читаемым
- В Go используются конструкции `if`, `else if`, `else`

Синтаксис if / else if / else в Go

- Порядок условий важен (сверху вниз)
- Фигурные скобки **{}** обязательны, даже для одной строки
- Вопрос: Что будет, если убрать **else if**?

```
if условие {
    // блок кода
} else if другое_условие {
    // блок кода
} else {
    // блок кода
}
```

```
age := 20
if age < 18 {
    fmt.Println("Несовершеннолетний")
} else if age < 65 {
    fmt.Println("Взрослый")
} else {
    fmt.Println("Пожилой")
}
```

Особенности Go — синтаксис условий

Круглые скобки не нужны

- В языке Go условие пишется без круглых скобок
- Это отличает его от C-подобных языков — например, Java, JavaScript, C++

Правильно

```
if x > 0 {
    fmt.Println("Положительное число")
}
```

Ошибка

```
if (x > 0) {
```

Особенности Go — синтаксис условий

Фигурные скобки обязательны

- В Go фигурные скобки — обязательны для любого блока **if**, даже если внутри только одна строка

```
if condition {
    // тело условия
}
```

- Это не просто соглашение о стиле, а жёсткое требование компилятора
- Если вы пропустите скобки — код не скомпилируется
- В отличие от Python, где блоки кода определяются отступами, в Go всё должно быть явно обёрнуто в **{}**

Особенности Go — синтаксис условий

Объявление переменной в условии

- В Go можно объявить переменную прямо внутри условия **if**

```
if x := getValue(); x > 10 {

    fmt.Println("Значение больше 10")
}
```

- В первой части условия **x := getValue()** вызывается функция и сохраняется результат
- Во второй части **x > 10** сразу проверяется значение

Переменная x доступна только внутри блока **if**

Это удобно, если значение больше нигде не нужно — код становится чище и короче

Ваши вопросы?

Когда использовать switch в Go

switch — удобная конструкция, когда нужно проверить одну переменную на несколько значений

- Читабельнее, чем **if / else if**
- Не требует **break** — выполнение останавливается автоматически
- Поддерживает **default** — как в других языках

Пример: проверка значения переменной

- Переменная **day** объявляется прямо внутри **switch**
- Выполняется только тот **case**, который совпал
- **break** не нужен — Go завершает **switch** автоматически

```
switch day := 3; day {
case 1:
   fmt.Println("Понедельник")
case 2:
   fmt.Println("Вторник")
case 3:
   fmt.Println("Среда")
default:
   fmt.Println("Другой день")
}
```

Можно использовать без значения

- Такой **switch** работает как цепочка **if / else if**
- Условия проверяются сверху вниз
- Выполняется только первый совпавший блок

```
switch {
case x > 0:
    fmt.Println("Положительное")
case x < 0:
    fmt.Println("Отрицательное")
default:
    fmt.Println("Ноль")
}
```

Когда выбирать switch, а когда if

1

Используйте **switch**, если:

- Проверяете **одно значение на несколько вариантов**
- Хотите, чтобы код был **читаемым и компактным**

2

Используйте **if**, если:

- Условия **зависят от логики или нескольких переменных**
- Требуется **гибкость в проверках**

Особенность: fallthrough

- **fallthrough** вручную переносит выполнение на следующий **case**
- Используется редко и только осознанно
- По умолчанию в Go переход не происходит

```
switch x := 1; x {
case 1:
    fmt.Println("Один")
    fallthrouagh
case 2:
    fmt.Println("Два")
}
```

Вопрос на подумать

Вы получаете код от коллеги:

Что вы скажете про этот код?

- Всё ли в нём работает как задумано?
- Почему в **switch** нет значения?
- Как вы думаете, какой результат будет, если **x == 8?**
- А если **x == 3?**

```
switch x := getValue(); {
case x > 10:
    fmt.Println("Больше десяти")
case x > 5:
    fmt.Println("Больше пяти")
default:
    fmt.Println("Пять или меньше")
}
```

Ваши вопросы?

Частые ошибки при работе с if

Ошибка №1: круглые скобки в условии

```
if (x > 0) {
```

В Go нельзя использовать круглые скобки вокруг условия — это не C/Java

Правильно:

```
if x > 0 {
```

Ошибки со скобками и переменными

Ошибка №2: отсутствие фигурных скобок

- Go требует фигурные скобки всегда, даже если одна строка

```
if x > 0
    fmt.Println("Да")
```

Ошибка №3: область видимости переменной в if

- Переменная x объявлена внутри **if** и недоступна за его пределами

```
if x := getValue(); x > 10 {
    fmt.Println(x)
}

// Здесь x больше не существует
```

Ошибки при использовании switch

Ошибка №4: лишний break

- В Go **switch** завершает выполнение автоматически — **break** не нужен

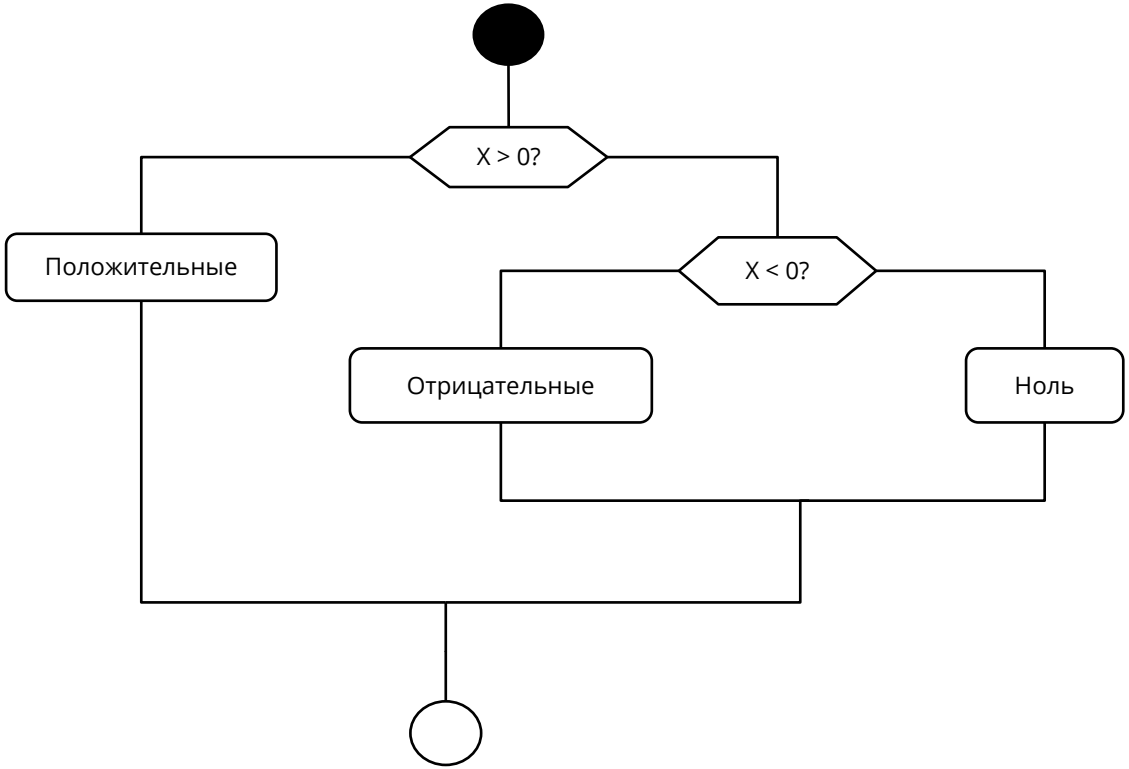
Ошибка №5: неправильный fallthrough

```
switch x {
case 1:
    fmt.Println("Один")
    fallthrough
case 2:
    fmt.Println("Два")
}
```

fallthrough пропускает проверку следующего **case**, а просто продолжает выполнение — это может быть неожиданно и опасно

Визуальная блок-схема принятия решений

Совет: Рисуйте схемы для сложных условий (например, в draw.io)



Ваши вопросы?

Циклы в Go

2

Циклы в Go — основы

Циклы в Go — только for

- В Go всего один тип цикла — **for**
- Он заменяет и **for**, и **while**, и **do while** из других языков

Почему это удобно?

- Один универсальный синтаксис — меньше путаницы
- Вы не думаете *что выбрать*, вы просто пишете **for**
- И он работает в трёх формах

Ваши вопросы?

Цикл for как счётчик

Пример: Вывод чисел от 1 до 10

```
for i := 1; i <= 10; i++ {
    fmt.Println(i)
}
```

Что происходит:

- **i := 1** — создаётся счётчик
- **i <= 10** — условие: цикл работает, пока истина
- **i++** — шаг: увеличиваем значение на 1

Особенности и частые ошибки

```
for i := 1; i <= 10; i++ {
    fmt.Println(i)
}
```

- **i** существует только внутри цикла
- Можно изменить шаг: **i += 2, i**-- и т.д.
- Если забыть шаг **(i++),** цикл будет бесконечным
- Форма **i--** запускает обратный счёт

Вопрос: Что произойдёт, если вместо **i++** написать **i--**?

Пример и польза

```
i := 1
for i <= 10 {
    fmt.Println(i)
    i++
}
```

Подходит, если:

- Переменная объявляется заранее
- Шаг зависит от логики в теле
- Нужно больше гибкости, чем в цикле-счётчике

for с условием — аналог while

Форма цикла, в которой указывается только условие

```
for x != 0 {
    // выполняем действия
}
```

- Если условие никогда не станет ложным — цикл бесконечный
- Без инициализации и шага в заголовке
- Выполняется, пока условие истинно
- В Go заменяет привычный **while**

Важно управлять условием вручную

Бесконечный for

Цикл без условия — работает вечно

- Такой цикл не завершится сам
- Используется, когда условие выхода неизвестно заранее
- Чтобы остановить цикл — используйте **break**

```
for {
    // бесконечный цикл
}
```

Пример с break

Здесь цикл продолжается, пока пользователь явно не введёт «стоп»
Это удобно при работе с:

- Пользовательским вводом
- Ожиданием ответа от сервера
- Обработкой событий

```
for {
    var input string
    fmt.Print("Введите 'стоп' для выхода: ")
    fmt.Scanln(&input)

    if input == "стоп" {
        break
    }

    fmt.Println("Вы ввели:", input)
}
```

continue и зачем нужен бесконечный цикл

continue — пропускает текущую итерацию

```
for i := 1; i <= 10; i++ {
    if i%2 == 0 {
        continue
    }
    fmt.Println(i) // напечатает только нечётные числа
}
```

continue и зачем нужен бесконечный цикл

Вывод: только нечётные числа

- Зачем использовать бесконечный **for**?
- Если условие выхода зависит от внешнего события
- Когда нужно слушать «вечно», но завершить по команде
- Вы контролируете выход вручную (через **break, return, panic**)

Частые ошибки: бесконечный цикл

Ошибка: забыли шаг

```
for i := 1; i <= 10; {
    fmt.Println(i) // i не увеличивается
}
```

- Переменная **i** не изменяется — условие остаётся истинным
- Результат — бесконечный цикл

Частые ошибки: бесконечный цикл

Ошибка: условие всегда ложное

```
for i := 1; i > 10; i++ {
    fmt.Println(i)
}
```

Условие изначально ложное — цикл не запустится ни разу

break и continue

- **break** — завершает весь цикл
- **continue** — пропускает текущую итерацию

Используйте только там, где точно понимаете их эффект

Избегайте «вложенной путаницы» с **continue** без комментариев — особенно на ранних этапах обучения

Ошибки в области видимости переменных

```
for i := 0; i < 3; i++ {
    fmt.Println(i)
}

fmt.Println(i) // ошибка — i не существует вне цикла
```

- Переменная **i**, объявленная в **for**, видна только внутри цикла
- Вне цикла вы не сможете её использовать

Советы и задание

Полезные советы:

- Начинайте с цикла-счётчика — он самый наглядный
- Осваивайте **for** с условием и бесконечный **for** поэтапно
- Проверяйте, есть ли у цикла точка выхода
- Не бойтесь отладки — бесконечный цикл проще поймать в раннем тесте

Задание: Напишите цикл, который выводит числа от 10 до 1 в обратном порядке

Ваши вопросы?

Примеры задач и кода

3

Задача с условием — проверка возраста

Напишите программу, которая определяет статус пользователя по возрасту

Возрастные категории:

- Меньше 18 → Вы несовершеннолетний
- От 18 до 64 → Вы совершеннолетний
- От 65 и выше → Вы пенсионер

Используйте:

```
package main // Объявляем основной пакет программы

import (
    "fmt" // Импортируем пакет fmt для ввода/вывода
)

func main() {
    var age int
    fmt.Scan(&age)
```

Подсказка: как подойти к решению

- Объявите переменную **age**
- Считайте возраст с клавиатуры
- Проверьте возраст через **if, else if, else**
- Выведите соответствующее сообщение

Примерная структура:

```
package main // Объявляем основной пакет программы

import (
    "fmt" // Импортируем пакет fmt для ввода/вывода
)

func main() {
    var age int
    fmt.Scan(&age)
```

Разбор решения задачи с условием

- **if age** < 18 — проверяется первым
- **else if age** < 65 — выполняется, если первое условие ложно
- **else** — охватывает все оставшиеся случаи (возраст 65 и выше)

```
package main // Объявляем основной пакет программы
import (
   "fmt" // Импортируем пакет fmt для ввода/вывода
)
func main() {
   var age int // Объявляем переменную age типа int (целое число)
   fmt.Print("Введите ваш возраст: ") // Выводим приглашение ко вводу
   fmt.Scan(&age) // Считываем введённое пользователем число и сохраняем в
переменную age
   // Начинаем конструкцию if-else для проверки возраста
   if age < 18 { // Если возраст меньше 18
      fmt.Println("Вы несовершеннолетний.") // Выводим соответствующее
сообщение
   } else if age < 65 { // Иначе, если возраст меньше 65
      fmt.Println("Вы совершеннолетний.") // Выводим сообщение о
совершеннолетии
   } else { // Во всех остальных случаях (то есть от 65 и выше)
      fmt.Println("Вы пенсионер.") // Сообщаем, что человек пенсионер
   }
}
```

Задача с циклом — суммирование чисел

Напишите программу, которая считает сумму чисел от 1 до n

Значение n пользователь вводит с клавиатуры

- **sum := 0** — переменная для накопления суммы
- Цикл **for** перебирает числа от 1 до n
- На каждой итерации текущее число прибавляется к **sum**

После завершения цикла в **sum** будет храниться итоговая сумма

Разбор решения задачи с циклом

```
package main // Основной пакет программы
import (
    "fmt" // Импортируем пакет fmt для ввода и вывода
)
func main() {
    var n int // Объявляем переменную n типа int
    fmt.Print("Введите положительное целое число: ") // Просим пользователя ввести число
    fmt.Scan(&n) // Считываем значение n с клавиатуры
    sum := 0 // Объявляем переменную sum и инициализируем её нулём — в ней будет храниться сумма
    // Цикл от 1 до n включительно
    for i := 1; i <= n; i++ {
        sum += i // На каждой итерации добавляем i к текущему значению sum
        // Например, при i = 1 → sum = 0 + 1
        // затем i = 2 → sum = 1 + 2 и т.д.
    }
    // После завершения цикла выводим итоговую сумму
    fmt.Printf("Сумма чисел от 1 до %d равна %d\n", n, sum)
}
```

Итоги

4

Итоги занятия

Ветвление в Go: как работает логика

- Используем **if, else if, else** — для проверки условий
- **switch** — удобен, когда условий много и они зависят от одного значения
- Порядок условий важен: от более частного к более общему
- Без **()** в условиях, но **{}** обязательны

В Go только один цикл — for, но он заменяет и while, и do while

- Цикл-счётчик: **for i := 0; i < n; i++**
- Цикл с условием: **for i < n {}**
- Бесконечный цикл**: for {}** — нужен **break** внутри
- Выбор формы зависит от задачи

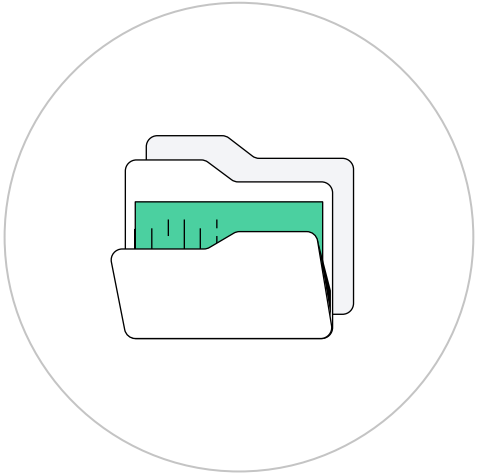
Итоги занятия

Как управлять циклом

- **break** — прерывает выполнение цикла полностью
- **continue** — пропускает текущую итерацию и переходит к следующей
- Используйте их осознанно — они влияют на поток выполнения

Дополнительные материалы

- Официальная документация Go
- Go Playground (онлайн-редактор)
- Go by Example



Условные конструкции и циклы

Матвей Ефимов
Эксперт в Go, Python, SQL