Синтаксис условных конструкций и циклов

Время изучения: 12 мин

Автор: Матвей Ефимов, Разработчик учебных программ и эксперт в Go, Python, SQL

Вы узнаете:

- Зачем нужны условия и циклы
- Особенности Go
- Условные конструкции
- Конструкция switch
- Цикл for

Зачем нужны условия и циклы

Когда вы пишете программу, важно уметь **принимать решения** и **повторять действия**. Это делают с помощью **условных конструкций** и **циклов**.

- **`if, else, switch`** — помогают программе выбирать, что делать.
  Например: «Если пользователь ввёл верный пароль — впустить, иначе — отказать».
- **`for, range`** — позволяют повторить действие несколько раз.
  Например: «Показать все товары в корзине» или «Посчитать сумму от 1 до 100».

Без этих конструкций код был бы однообразным и длинным — пришлось бы всё делать вручную.

Особенности Go

Go — строго типизированный язык с простым и строгим синтаксисом. Вот что важно знать:

- В Go **только один тип цикла** — **`for`**.
  Он заменяет и **`while`**, и **`do-while`**, которые есть в других языках.
- Скобки **`()`** вокруг условий **не нужны**, а фигурные **`{}`** — **обязательны**, даже для одной строки.
- Нельзя создавать переменные «впустую» — **если не используете переменную, будет ошибка**.
- Цикл **`for`** умеет работать в трёх режимах:
  - как обычный счётчик: **`for i := 0; i < 10; i++`**
  - как **`while: for i < 10`**
  - как бесконечный цикл: **`for { }`**
- **`range`** — удобный способ перебрать элементы в срезе, массиве, строке, мапе или канале.

В этом конспекте мы разберём, как работают **`if`**, **`else`**, **`switch`**, **`for`** и **`range`** — с простыми примерами и пошаговыми пояснениями. Это поможет вам **читать и писать код на Go уверенно и без страха**.

Условные конструкции: `if, else, else if`

В Go конструкция **`if`** выглядит компактно: скобки `()` вокруг условия не требуются, а фигурные **`{}`** — обязательны, даже если внутри только одна строка.

Пример базового синтаксиса:

```
if условие {
    // действия, если условие истинно
} else if другое_условие {
    // действия, если второе условие истинно
} else {
    // действия, если ни одно из условий не выполнено
}
```

Особенности:

- фигурные скобки **`{}`** обязательны;
- можно использовать логические операторы **`&&`**, **`||`**;
- допускается объявление переменных прямо в **`if`** — они работают только внутри блока.

Пример 1. Простое условие:

```go
package main

import "fmt"

func main() {
    age := 18

    if age >= 18 {
        fmt.Println("Доступ разрешён")
    }
}
```

Объяснение:

- `age := 18` — задаём переменную возраста;
- `if age >= 18` — проверяем: возраст не меньше 18?;
- `fmt.Println(...)` — если условие `true`, выводим сообщение.

Пример 2. if + else:

```go
package main

import "fmt"

func main() {
    temperature := 12

    if temperature < 10 {
        fmt.Println("Холодно")
    } else {
        fmt.Println("Тепло")
    }
}
```

Объяснение:

- **`temperature := 12`** — текущая температура;
- **`if temperature < 10`** — условие: меньше 10 градусов?;
- **`else`** — выполняется, если условие выше не сработало.

Пример 3. if + else if + else:

```go
package main

import "fmt"

func main() {
    score := 75

    if score >= 90 {
            fmt.Println("Оценка: A")
    } else if score >= 75 {
        fmt.Println("Оценка: B")
    } else if score >= 60 {
        fmt.Println("Оценка: C")
    } else {
        fmt.Println("Оценка: F")
    }
```

Объяснение:

- **`score := 75`** — значение баллов;
- **`if score >= 90`** — проверка на A;
- **`else if score >= 75`** — вторая проверка, если первая не прошла;
- **`else`** — если ни одно условие не подошло.

Пример 4. Объявление переменной в if:

```go
package main

import "fmt"

func main() {
    if x := 5; x%2 == 0 {
        fmt.Println("Чётное число")
    } else {
        fmt.Println("Нечётное число")
    }
}
```

Объяснение:

- **`if x := 5; x%2 == 0`** — объявляем переменную x прямо в if и проверяем, делится ли на 2;
- **`x`** доступна только внутри блока **`if`** и **`else`**;

- **`if`** инициализация через; перед условием — особенность Go.

Кратко об if, else if, else

- **`if`** проверяет условие и выполняет код, если оно истинно.
- **`else if`** добавляет дополнительные условия.
- **`else`** выполняется, если ни одно из условий выше не подошло.
- В **`if`** можно объявить переменные — они будут видны только внутри блока. Это помогает локализовать логику и не «засорять» внешний код.

Синтаксис `switch`

`switch` позволяет сравнивать значение с разными вариантами. В Go не нужно писать **`break`** — переход к следующему блоку происходит автоматически.

Особенности:

- **`break`** не нужен — Go сам завершает **`switch`** после совпадения;
- можно использовать **`switch`** без выражения — он работает как **`switch true`**;
- допускается несколько значений в одном **`case`** через запятую.

Пример 1. Строки и выходные дни

```
package main

import "fmt"

func main() {
    day := "суббота"

    switch day {
    case "суббота", "воскресенье":
        fmt.Println("Выходной") // если день — суббота или
воскресенье
    default:
        fmt.Println("Будний день") // для всех остальных дней
    }
}
```

Пояснение:

- В переменной **`day`** хранится день недели — как текст (строка).

- Команда **`switch`** сравнивает это значение с каждым вариантом, который указан в **`case`**.
- Если день — **«суббота»** или **«воскресенье»**, выполнится первый блок: **`fmt.Println("Выходной").`**
- Они объединены в одном **`case`**, потому что обрабатываются одинаково.
- Если день не совпадает ни с одним из значений, сработает **`default`** — как запасной вариант на все остальные случаи.

Пример 2. Оценки по числам

```
package main

import "fmt"

func main() {
    grade := 3

    switch grade {
    case 5:
        fmt.Println("Отлично")
    case 4:
        fmt.Println("Хорошо")
    case 3:
        fmt.Println("Удовлетворительно")
    default:
        fmt.Println("Неудовлетворительно")
    }
}
```

Пояснение:

- В переменной **`grade`** хранится целое число — оценка.
- **`switch`** сравнивает это число с вариантами в **`case`**.
- Если значение — 5, 4 или 3, выводится соответствующая строка: **`«Отлично»,`** **`«Хорошо»`** или **`«Удовлетворительно».`**
- Если ни один из **`case`** не совпал, сработает **`default`** — в нём указано, что оценка «**Неудовлетворительно**».

Пример 3. `switch true` (как альтернатива `if`)

```
package main

import "fmt"

func main() {
    score := 82

    switch {
    case score >= 90:
        fmt.Println("Оценка: A")
    case score >= 75:
        fmt.Println("Оценка: B")
    case score >= 60:
        fmt.Println("Оценка: C")
    default:
        fmt.Println("Оценка: F")
    }
}
```

Пояснение:

- Это пример **`switch`** без выражения — его ещё называют **`switch true`**.
- В таком случае каждая строка **`case`** — это логическое условие, как в **`if`** или **`else if`.**
- Go проверяет условия по порядку сверху вниз и выполняет **только первый** подходящий вариант.
- В нашем примере переменная **`score`** — это балл, и в зависимости от его значения выводится оценка по шкале: A, B, C или F.

Пример 4. Условия по температуре

```
package main

import "fmt"

func main() {
    temp := -5

    switch {
    case temp < 0:
```

```
        fmt.Println("Мороз") // температура ниже нуля
    case temp >= 0 && temp < 20:
        fmt.Println("Прохладно") // от 0 до 19
    default:
        fmt.Println("Тепло") // всё остальное (20 и выше)
    }
}
```

Пояснение:

- Это ещё один пример **`switch`** без выражения — Go будет просто по очереди проверять условия.
- В переменной **`temp`** — температура.
- Если значение меньше 0, программа выведет «Мороз».
- Если от 0 до 19 — «Прохладно».
- Если больше или равно 20 — сработает **`default`** и выведется «Тепло».

Такой подход работает как цепочка **`if / else if`**, но выглядит аккуратнее и читается проще.

Кратко о `switch`

- **`switch`** помогает выбрать нужный вариант из нескольких — работает как альтернатива цепочке **`if/else if`**, но читается проще.
- Не требует **`break`** — Go сам завершает **`switch`** после первого совпадения.
- В одном **`case`** можно указать сразу несколько значений через запятую.
- Поддерживает логические условия — через **`switch`** без выражения (**`switch true`**).

Цикл `for`

В Go только одна ключевая конструкция цикла — **`for`**. Она заменяет **`for, while, do-while`** из других языков.

Варианты использования:

1. **`for условие { ... }`** — аналог **`while`**;
2. **`for инициализация; условие; шаг { ... }`** — классический счётчик;
3. **`for { ... }`** — бесконечный цикл с **`break`** для выхода.

Пример 1. `for` как `while`

```
package main

import "fmt"

func main() {
    x := 0

    for x < 5 { // пока x меньше 5
        fmt.Println(x) // выводим значение
        x++            // увеличиваем x на 1
    }
}
```

Пояснение:

- Это цикл **`for`** с одним условием — он работает так же, как **`while`** в других языках.
- Переменная **`x`** сначала равна **`0`**.
- Пока **`x < 5`**, программа выполняет тело цикла: печатает значение **`x`** и увеличивает его на **`1`**.
- Когда **`x`** становится **`5`**, условие перестаёт быть истинным — цикл завершает работу.

Такой формат удобен, когда заранее известна граница, но не нужен счётчик прямо в заголовке цикла.

Пример 2. `for` с тройным выражением

```
package main

import "fmt"

func main() {
    for i := 0; i < 5; i++ { // начинаем с i = 0, пока i < 5,
увеличиваем i на 1
        fmt.Println(i) // выводим i на каждой итерации
    }
```

```
}
```

Пояснение:

- Это цикл с тремя элементами в заголовке: инициализация **`(i := 0)`**, условие **`(i < 5)`** и шаг **`(i++)`**.
- Цикл выполняется, пока условие истинно — в данном случае, пока **`i`** меньше 5.
- На каждой итерации выводится текущее значение переменной **`i`**.
- Такой формат удобен, когда нужно пройтись по диапазону чисел — например, от **`0`** до **`N`**. Его часто используют в Go и других языках.

Пример 3. Бесконечный `for` и выход через `break`

```
package main

import "fmt"

func main() {
    counter := 0

    for { // бесконечный цикл
        fmt.Println("Цикл номер", counter)
        counter++

        if counter == 3 { // условие выхода
            break // выход из цикла
        }
    }
}
```

Пояснение:

- Это бесконечный цикл — в заголовке **`for`** нет условий.
  Он выполняется до тех пор, пока явно не будет прерван командой **`break`**.
- В примере переменная **`counter`** увеличивается на каждой итерации.
  Когда она достигает 3, срабатывает условие **`if`**, и цикл завершается.
- Такая конструкция полезна, когда заранее неизвестно, сколько раз нужно повторить действия.

Часто используется при обработке пользовательского ввода или сетевых событий — когда цикл должен работать до наступления нужного события.

Кратко о `for`

В Go только одна конструкция цикла — **`for`**, и она закрывает все основные сценарии:

1. **`for условие { ... }`** — работает как **`while`**: повторяет действия, пока условие истинно.
2. **`for инициализация; условие; шаг { ... }`** — удобно, когда нужен счётчик, например от 0 до 10.
3. **`for { ... }`** — бесконечный цикл. Его можно остановить с помощью **`break`**, когда наступит нужное условие.

Что стоит запомнить

- **`if, else`** и **`switch`** помогают управлять логикой — выбирать, что делать программе в разных ситуациях.
- В **`switch`** не нужен **`break`** — Go сам завершает конструкцию после первого совпадения.
- **`switch`** можно использовать с логическими условиями — через **`switch true`**.
- В Go только один цикл — **`for`**, и его возможностей хватает для всех задач повторения.
- Фигурные скобки **`{}`** обязательны и в условиях, и в циклах — даже если внутри одна строка.
- Если вы объявляете переменную в **`if`**, она будет доступна только внутри этого блока.