Содержание

Функции

- передача параметров по ссылке/по значению
- именованные параметры
- замыкания
- функции высшего порядка
- defer
- системные функции

Структуры

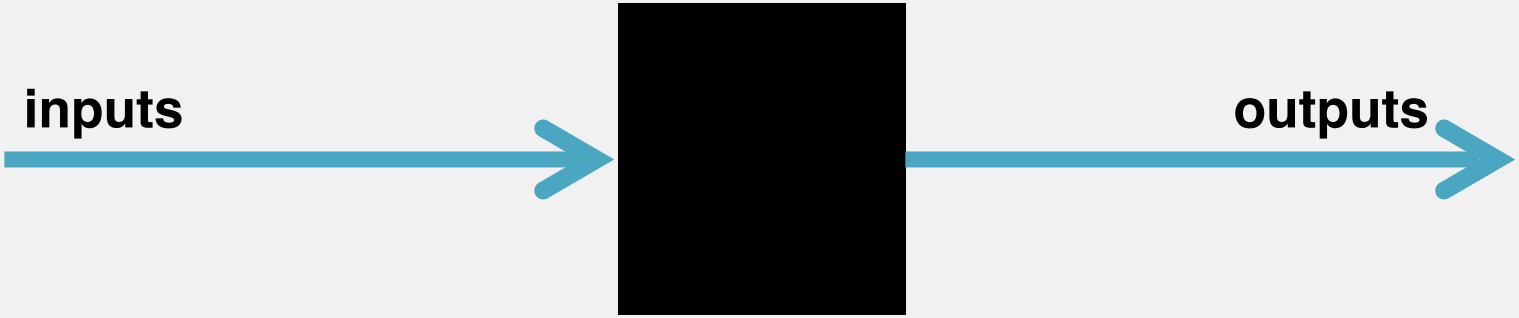
- struct, ооп
- методы
- embeded объекты

Интерфейс

- пустой интерфейс
- использование интерфейсов

Функции

- Функция — независимая часть кода, связывающая один или несколько входных параметров с одним или несколькими выходными параметрами.
- Функции (*процедуры*, *подпрограммы*) можно представить как черный ящик



Функции

Для объявления функции используется следующий **синтаксис**:

```
func имяФункции( [список параметров] ) [Возвращаемые
значения] {
        тело функции
}
```

Тип определяет возвращаемый тип функции.

Имя функции служит для ее вызова в программе и ее правило определения совпадает с правилом определения имен переменных.

Список параметров необходим для передачи функции каких-либо данных при ее вызове.

Тело функции — это набор операторов, которые выполняются при ее вызове.

Функции

Пример функции в Go

```
1.  Package main
2.
3.  func average(xs []float64) float64 {
4.      total := 0.0
5.      for _, v := range xs {
6.          total += v
7.      }
8.      return total / float64(len(xs))
9.  }
10.
11. func main() {
12.     xs := []float64{98,93,77,82,83}
13.     fmt.Println(average(xs))
14. }
```

Функции

Функции init() и main()

```
1.  Package main
2.
3.  var stuff = “not ready”
4.
5.  func init() {
6.    stuff = “ready”
7.  }
8.
9.  func main() {
10.   fmt.Println(“The stuff is ”,stuff)
11. }
```

Функции

- Возврат нескольких значений
- Именованные возвращаемые значения
- Переменное число аргументов функции

Функции

Передача по значению

используется по умолчанию

Создается копия объекта

Изменения не сохраняются при выходе

Кроме slice и map

Передача по ссылке

- В функцию передается указатель на значение
- Возможны изменения в незапланированных местах

Функции

Замыкания

- Замыкание — что это?
- Анонимные функции в Go
- Функции — «Обертки»
- Фабричные функции

Функции

Рекурсия, плюсы и проблемы

- Рекурсивная функция вызывает себя
- Взаимно рекурсивные функции вызывают друг друга
- Рекурсивные функции упрощают обработку данных с рекурсивной структурой
- НО могут оказаться весьма неэффективными

Функции

Выбор функции во время исполнения

во время исполнения

- можно выбирать, какую функцию вызвать
- можно создавать функции

Функции

Системная функция — Defer

- Defer — функция, которая будет вызвана сразу после выхода из текущей функции
- Defer захватывает переменные объявленные до ее регистрации
- Блоков Defer может быть более одного, порядок вызова filo (стэк)

Функции

Системные функции

Make

New

Len/cap

Close

Append

Copy

Delete

Panic

Recover

Функции

Make

make(T, n)	slice	slice of type T with length n and capacity n
make(T, n, m)	slice	slice of type T with length n and capacity m
make(T)	map	map of type T
make(T, n)	map	map of type T with initial space for n elements
make(T)	channel	unbuffered channel of type T
make(T, n)	channel	buffered channel of type T, buffer size n

Функции

Append

append(s S, x ...T) S // T is the element type of S

Функции

Паника

- Механизм panic
- Использование recover

Структуры

Структуры — основной способ создания своих типов данных

```
1.  type Circle struct {
2.      x float64
3.      y float64
4.      r float64
5.  }
```

*в Go используется правило регистра первой буквы имени

- если название начинается заглавной буквы — это public-доступ
- если со строчной — private

Структуры

Пользовательские типы

- type typeName existingType
- Методы можно навешивать на любой тип

Методы

Методы в Go — это функции, определенные для конкретного типа.

Два варианта объявления методов:

```
1.  func (s Shape) Render() {
2.    fmt.Println(“Shape width is ”, s.width,”, and height is ”,
    s.height)
3.  }
4.
5.  func (s *Shape) Rotate() {
6.    s.width, s.height = s.height, s.width
7.  }
```

Структуры

Встраивание

Структуры могут содержать в себе анонимные или «встроенные» поля.

- Инициализация
- Доступ
- Отличия от наследования

Go Interfaces
Crash Course

Интерфейсы

Интерфейсы определяют поведение

```
1. type Speaker interface {
2.     SayHello()
3. }
```

- В именовании принято постфикс -er (Sender, Reader, Closer, etc)
- Duck typing (неявная, латентная или утиная типизация)
- Интерфейс не может содержать данные, только методы.

Интерфейсы

Если это выглядит как утка, плавает как утка и крякает как утка, то это, вероятно, утка и есть.

- В Go структура с методами будет удовлетворять интерфейсу просто самим фактом объявления метода.

- Структура хранит данные, но не поведение. Интерфейс хранит поведение, но не данные.

Интерфейсы

Как жить с интерфейсами

- заверну-ка я все в interface{} — думает разработчик
- работайте всегда с конкретными типами
- используйте интерфейс ТОЛЬКО там где это необходимо
- пустой интерфейс — ТОЛЬКО когда иначе никак
- Для того, чтобы преобразовать массив в массив interface{} требуется линейное время

Интерфейсы

Особенности

- Когда использовать интерфейсы осмысленно
- Когда стоит привязываться к конкретным типам
- Какие есть еще альтернативы (например, замыкания и функции)