Go

Начальный курс по Go

Anonymous functions

АНОНИМНЫЕ ФУНКЦИИ

Анонимные функции

Функции, у которых нет имени

Можем объявлять как внутри других функций, так и на уровне файла

Self Invoked

```
func main() {

  func () { fmt.Println("invoked") }()

}
```

автоматически вызовется

Closures

ЗАМЫКАНИЯ

Замыкания

Возможность использовать внутри функции переменные, объявленные вне тела: см. пример с sort.Slice

УКАЗАТЕЛИ

Указатели

Указатели (pointers) хранят адрес переменной в памяти (т.е. по какому адресу хранятся значения)

Через указатель можно читать и записывать значение по этому адресу

Указатели

```
count := 10

countPtr := &count

// полная запись

// var countPtr *int = &count

fmt.Println(*countPtr)

*countPtr = 11 // в count теперь тоже 11
```

Указатели

& - взятие адреса

* - dereferencing (разыменование указателя)

nil

Нулевое значение для указателя любого типа

– nil (константа)

ptr != nil – значит ptr указывает на данные

nil

ptr1 == ptr2 – оба указывают на одну и ту же переменную или оба nil

nil

Важно: nil можно присвоить любой переменной типа интерфейса или reference type (ссылочного типа)

Reference Types

- slices
- maps
- channels

Важно: массивы и структуры не являются reference types (т.е. копируются при передаче)

Reference Types

На самом деле и все остальные также копируются, но они представляют из себя сложные структуры, содержащие указатели

NEW

new

Функция new создаёт переменную нужного типа, кладёт туда нулевое значение и возвращает адрес

```
var ptr *int = new(int)
```

CONSTRUCTORS

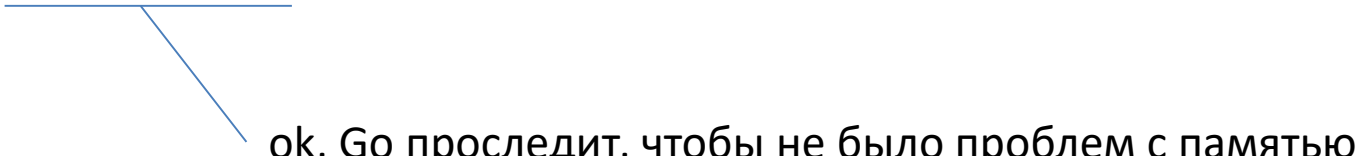
Constructors

Иногда необходимо выполнить некоторую логику перед созданием "объекта" нужного типа (т.е. простое нулевое значение или значение, созданное через литерал, недостаточно)

Constructor

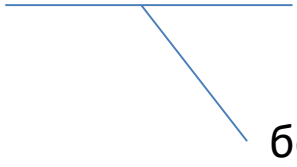
Специальная функция, которая создаёт нужный объект:

```
func NewStuff(args ...) *Stuff {
  // TODO: some checks
  return &Stuff{...}
}
```



Composite literals & new

&Stuff{} эквивалентно new(Stuff)



MAKE

make

make в отличие от new создаёт инициализированные объекты и возвращает значение (не указатель)

Используется только для создания slice, map и channel

INTERFACES

Interfaces

Интерфейс – контракт на реализацию поведения

Должны быть методы определённого типа (с нужной сигнатурой)

Interfaces

В Go интерфейсы реализуются неявно: т.е. если нужные методы реализованы, то тип уже соответствует интерфейсу

Важно: интерфейс ничего не говорит о внутренней реализации методов

Interfaces

```
type printer interface {
  print()
}
```

Рассмотрение пакета sort

Практика на базе пакета sort

GoLand: Alt + Enter -> Implement Interface

interface{}

Возможность принимать любой тип данных

РАБОТА С ОШИБКАМИ

error

Интерфейс, объявленный в builtin.go:

```
type error interface {
   Error() string
}
```

errors

errors.New(message) создаёт stringError (см. errors.go)

errors

Ошибки возвращаются функцией как обычное значение (по соглашению – последним значением)

Общий формат (для main)

```
result, err := functionCall()

if err != nil {

   log.Fatal(err)

}
```

Печать ошибки + os.Exit(1)

fmt.Errorf

Форматированная ошибка, позволяющая "форматировать" текст, так же, как Printf

Начиная с Go 1.13 появился модификатор %w, который позволяет заворачивать ошибку для дальнейших тестов

Общий формат (для не main)

```
result, err := doStuff(arg)

if err != nil {

   return fmt.Errorf("can't do stuff %v, %w", arg, err)

}
```

вернёт ошибку вызывающей функции
и завернёт исходную ошибку

Общий формат (для не main)

```
result, err := doStuff(arg)

if err != nil {

   return fmt.Errorf("can't do stuff %v, %v", arg, err)

}
```

вернёт ошибку вызывающей функции
и не завернёт исходную ошибку

Но чаще всего делают

```
result, err := doStuff(arg)
if err != nil {
    return
}
```

не делайте так

Надо обработать ошибку, а не просто "молчаливо"

возвращать управление вызывающей функции

Format Strings

- %d, %x, %o, %b – integer
- %f, %g, %e - floating point number
- %t – bool
- %c – rune
- %s – string
- %v - any value in natural format
- %T - type of value
- %w - error wrapping (only for Errorf)

Format Strings

fmt.Printf, log.Printf и fmt.Errorf используют одни и те же (за исключением %w) format strings

LOGGING

log

Достаточно простой логгер, по умолчанию всё логгирует в стандартный поток вывода

Используйте log.Printf с флагами форматирования

log

Что нужно логгировать:

- события
- ошибки
- контекст (но не перс.данные, платёжные данные и другие sensitive)

за это будут большие штрафы

PACKAGE

Package

Единица инкапсуляции: один или несколько файлов с расширением .go (чаще всего определяется каталогом)

Package

Для использования имён (переменных, констант или функций) из другого пакета необходимо импортировать пакет (см. fmt)

Для использования доступны только имена, начинающиеся с большой буквы (exported), например, fmt.Println

Спасибо за внимание

Ильназ Гильязов

2020г.