

## 4. Golang приложения

# Как понять, что код работает

- `fmt.Println()`
  - смотрим, что программа печатает во время своего выполнения
- ТЕСТЫ
  - ручной запуск с разными параметрами
  - автоматические тесты (как в leetcode)
- регулярно запускать:
  - `go build`
  - `go run`
  - тесты

# Печать работы приложения

- `fmt.Println`
  - для того, чтобы разобраться в ошибках приложения у себя
  - а если надо разобраться в поведении приложения у пользователя?
  - а если хочется смотреть сообщения не в консоли, а в файле
  - а если хочется более структурированный формат печати?
  - можно лучше!
- журналирование (logging)
  - важность сообщений задается уровнем логирования (severity): `debug`, `error`, `warning`
  - на Go есть множество библиотек для логирования
  - `zap`, `logrus`
  - стандартный пакет – `log`, сейчас – `log/slog`

# Пример log/slog

- <https://pkg.go.dev/log/slog#pkg-overview>
- ОСНОВНЫЕ ВОЗМОЖНОСТИ:
  - <https://pkg.go.dev/log/slog#Logger> – умеет логировать с определенным уровнем (debug, warn и т.д.)
  - <https://pkg.go.dev/log/slog#Handler> – отвечает за то, куда логировать (в файл, в stdout)
  - <https://pkg.go.dev/log/slog#Level> – уровни логирования (можно добавить свои)
- пример: <https://go.dev/play/p/GYakUAb5EDj>
- библиотеки для логирования могут быть устроены по-разному, но там всегда будет:
  - возможность логировать с определенным уровнем, который можно менять
  - настройки для формата логирования
  - настройки куда писать логи (в файл, в облако, и т.д.)

# Тестирование

- есть стандартный пакет <https://pkg.go.dev/testing>
- но есть лучше! <https://pkg.go.dev/github.com/stretchr/testify>
  - <https://pkg.go.dev/github.com/stretchr/testify/require> – проверка останавливает запуск тестов после падения
  - `assert` – не останавливает запуск даже, если тест упал
- `require.Equal(t, "42", myCounter)`
  - сравниваем актуальное значение (переменной/функции) с ожидаемым
- тесты на функцию, которая:
  - переворачивает строки?
  - конвертирует строку из lower case в UPPER CASE
  - определяет, палиндром строка или нет
  - удваивает слайс
  - вычисляет корень из числа?
  - считывает число, введенное пользователем в консоли, и возводит его в квадрат

# Тестирование

- <https://go.dev/wiki/TableDrivenTests>
- хорошая практика – писать табличные тесты
  - тесты, где наборы входных данных и ожидаемые для них результаты описываются таблицей
- VSCode, Go Extension
  - можно автоматически сгенерировать шаблон для тестов на функцию
  - но шаблон использует testing
- testing <https://go.dev/play/p/ECwmV7-1Ohx>
- testify <https://go.dev/play/p/LF6kbG1QLmx> , <https://gobyexample.com/testing>
- есть: `require.Equal()`, `require.True()/False`, `require.Empty()`, `require.Error()`, `require.NoError()`, `require.Contains()`, ...
- запуск через `go test ./...`



Go: Generate Unit Tests For Function

# CI/CD: Continuous integration / Continuous delivery

- Чтобы проверить, что все работает, надо запустить:
  - `go run`
  - `go test`
  - может добавиться еще `golangci-lint` / `go fmt` / ...
  - можно забыть!
- Команда разработчиков постоянно интегрирует свой код в master
  - кто-то забыл запустить `go test`
  - все сломалось
  - можно залить свой код в уже сломанный master
  - как разобраться, какой именно код все сломал?
- → Надо обеспечить возможность непрерывно интегрировать код

# Continuous integration

- Код должен быть готов к интеграции
  - т.е., не сломан
  - надо автоматически проверять, что он не сломан
- Когда это делать?
  - когда появился новый коммит в master (после интеграции кода PR в master)
    - но тогда master все равно может оказаться сломанным, и его придется чинить
  - на PR (перед интеграцией в master)
- Нужен инструмент для выполнения проверок качества кода на PR!
  - часть работы DevOps



# Continuous integration: Github Actions

- <https://github.com/katevi/golang-course/actions>

The screenshot displays the GitHub Actions interface for the repository `katevi/golang-course`. The top navigation bar includes links for Code, Issues, Pull requests, Actions (highlighted), Projects, Security, Insights, and Settings. The left sidebar shows the 'Actions' section with a 'New workflow' button and a list of workflows: 'All workflows' (selected), 'Go', and 'Lint'. Below the sidebar, there are links for 'Management', 'Caches', 'Attestations', 'Runners', 'Usage metrics', and 'Performance metrics'. The main content area is titled 'All workflows' and shows 'Showing runs from all workflows'. A search bar for 'Filter workflow runs' is present. A notification banner asks for feedback on GitHub Actions. Below this, a table lists '52 workflow runs' with columns for Event, Status, Branch, and Actor. The table shows three recent runs, all with a status of 'Success' (green checkmark). The first two runs are 'Merge pull request #7 from katevi/dev/eskazhenik/add-ci-lecture3' on the 'master' branch, triggered by a commit push. The third run is 'Add check go build stage passes for lecture 3 to ci' on the 'dev/eskazhenik/add-ci-lectu...' branch, triggered by a pull request synchronization. Each run entry includes a timestamp and a link to view the run details.

Actions

New workflow

All workflows

Go

Lint

Management

Caches

Attestations

Runners

Usage metrics

Performance metrics

All workflows

Showing runs from all workflows

Filter workflow runs

Help us improve GitHub Actions

Tell us how to make GitHub Actions work better for you with three quick questions.

Give feedback

52 workflow runs

Event	Status	Branch	Actor
✓ Merge pull request #7 from katevi/dev/eskazhenik/add-ci-lecture3	Success	master	...
Lint: #24: Commit b6c1621 pushed by katevi			
✓ Merge pull request #7 from katevi/dev/eskazhenik/add-ci-lecture3	Success	master	...
Go #28: Commit b6c1621 pushed by katevi			
✓ Add check go build stage passes for lecture 3 to ci	Success	dev/eskazhenik/add-ci-lectu...	...
Go #27: Pull request #7 synchronize by katevi			

# Continuous integration: Github Actions

- workflow описывает действия, которые надо выполнять
- workflow лежат в папке `.github/workflows/`
- <https://github.com/katevi/golang-course/tree/master/.github/workflows>
- должны запускаться автоматически на PR / мердж в мастер / ...

golang-course / .github / workflows / go.yml 

 katevi Add check go build stage passes for lecture 3 to ci 

Code Blame 40 lines (35 loc) · 956 Bytes  Code 55% faster with G




```
1  # This workflow will build a golang project
2  # For more information see: https://docs.github.com/en/actions/a
3
4  name: Go
5
6  on:
7    push:
8      branches: ["master"]
9    pull_request:
10     branches: ["master"]
11
12 jobs:
13   build:
14     runs-on: ubuntu-latest
15     steps:
16       - uses: actions/checkout@v4
17
18       - name: Set up Go
19         uses: actions/setup-go@v4
20         with:
21           go-version: "1.24"
22
23       - name: Build Lecture 1
24         run: |
25           for dir in ./maps ./slices ./phoneBook; do
26             echo "Building $dir"
27             cd $dir
28             go build -v main.go
29             cd -
```


# Continuous integration: Github Actions

- как понять, что workflow работает/прошел?
  - пример: <https://github.com/katevi/golang-course/pull/4/checks>
  - раздел checks в PR
  - галочки на коммите в master
  - иконка результата (галочка, крестик)
  - можно посмотреть логи

All checks have passed

3 successful checks

✓		Go / build (push)	Successful in 16s	<a href="#">Details</a>
✓		Linters / build (push)	Successful in 1m	<a href="#">Details</a>
✓		--> Linted: MARKDOWN - No errors were found in the linting process		<a href="#">Details</a>

 **golang-course** Public



Unwatch

1

 master

 9 Branches

 0 Tags

 Go to file

Add file

 Code




**katevi** Merge pull request #7 from katevi/dev/eskazhenik/add-ci-lecture3



b6c1621 · 2 hours ago

 20 Commits

 .github

Add check go build stage passes for lecture 3 to ci

success

2 hours ago

# Continuous delivery

- Программные продукты всегда имеют какую-то версию
  - Windows 10 / Android 11 / Linux kernel 6.5
  - выпуск новой версии – релиз
    - различную новую функциональность объединяют в версию
    - перед релизом ее тестируют
    - устанавливать новые версии продуктов может быть долго
    - пользователи любят получать новую функциональность побыстрее
- Как сделать процесс менее долгим?
  - ускорить процесс поставки новых версий
  - continuous delivery – непрерывное развертывание новых версий пользователям
    - любая версия программы может быть быстро развернута (установлена)