# Объявление ошибок

В этом уроке мы научимся называть наши "ошибочные" переменные и типы, а также поговорим о формате сообщений об ошибке.


## Нейминг переменных и типов ошибок

Здесь всё просто, есть несколько правил ~~бойцовского клуба~~:

- названия переменных начинаются с `err` или `Err`;
- названия кастомных типов заканчиваются на `Error`;
- старайтесь избегать дублирования `Err`/`Error` в названии типа/переменной.

```
import (
    "errors"
    "fmt"
)

var (
    // Sentinel errors.
```

```
    // Название начинается с приставки err или Err.
    errNotFound = errors.New("error not found") // Неэкспортируемая.
    ErrNotFound = errors.New("error not found") // Экспортируемая.
)

// Кастомный тип ошибки. Название заканчивается на Error.
type NotFoundError struct {
    page string
}

func (e *NotFoundError) Error() string {
    return fmt.Sprintf("page %q not found", e.page)
}

var (
    // Sentinel errors.
    // Точно так же название начинается с приставки err или Err.
    errNotFound2 = &NotFoundError{"https://www.golang-courses.ru/"}
// Неэкспортируемая.
    ErrNotFound2 = &NotFoundError{"https://www.golang-courses.ru/"}
// Экспортируемая.
)
```

---

```
// Плохо.
var ErrExecErr = errors.New("exec sql error")

// Хорошо.

var ErrExecSQL = errors.New("exec sql error")
```

---

Небольшой пример из компилятора:

```
// src/go/build/constraint/expr.go
package constraint

// A SyntaxError reports a syntax error in a parsed build expression.
type SyntaxError struct {
    Offset int    // byte offset in input where error was detected
    Err    string // description of error
```

```
}

func (e *SyntaxError) Error() string {
    return e.Err
}

var errNotConstraint = errors.New("not a build constraint")


var errComplex = errors.New("expression too complex for // +build
lines")
```

Мы ещё не раз увидим подтверждение правилу выше, когда будем касаться ошибок в стандартной библиотеке Go.

## Тест "Выберите корректные именования типов/переменных ошибок"

Выберите все подходящие ответы из списка

- [ ] type ErrOrderNotFound struct
- [ ] var errorTimeout error
- [ ] type WithTimeError struct
- [ ] errZeroDivision := errors.New("zero division")
- [ ] var timeoutError error
- [ ] var ErrTimeout = new(timeoutError)
- [ ] type OrderNotFoundErr struct
- [ ] type PipelineErr struct
- [ ] type OrderNotFoundError struct
- [ ] type PipelineError struct
- [ ] type ZeroDivisionError struct
- [ ] type ErrorWithTime struct

## Текст в ошибках

В соответствующем разделе методички ревьювера Go-кода есть пункт, посвященный правилам оформления текста ошибок (результата вызова метода `Error()`):

*Error strings should not be capitalized (unless beginning with proper nouns or acronyms) or end with punctuation, since they are usually printed following other context.*

*That is, use `fmt.Errorf("something bad")` not `fmt.Errorf("Something bad")`,*

*so that `log.Printf("Reading %s: %v", filename, err)` formats without a spurious capital letter mid-message.*

*This does not apply to logging, which is implicitly line-oriented and not combined inside other messages.*

Итак, текст ошибок не должен:

- начинаться с заглавной буквы (если не начинается с имен собственных или аббревиатур), чтобы далее в логах мы не получали "ложные" заглавные буквы в середине сообщения ~~(кто нибудь расскажите им про `grep -i`~~);
- оканчиваться знаком пунктуации.

## Тест "Выберите корректные с точки зрения Error() ошибки"

Выберите все подходящие ответы из списка

- [ ] err := errors.New("don't do that!")
- [ ] err := errors.New("EOF")
- [ ] return fmt.Errorf("HTML parsing error: %w", err)
- [ ] err := errors.New("Text should not be capitalized")
- [ ] err := errors.New("zero division")
- [ ] return fmt.Errorf("Cannot load file: %w", err)
- [ ] return fmt.Errorf("cannot do operation: %w", err)

## errors.New vs fmt.Errorf

Часто в коде можно встретить подход, когда разработчик создаёт все ошибки через `fmt.Errorf`, даже если их текст не требует динамических параметров, например:

```
return "", fmt.Errorf("invalid token")
```

У перфекциониста-ревьювера могут возникнуть следующие замечания:

- зачем использовать функцию не по её прямому назначению, когда есть специальная `errors.New`?
- `Errorf` к тому же несёт накладные расходы из-за своей непростой реализации (исходник примера):

```
package errorfbench

import (
    "errors"
    "fmt"
    "testing"
)

// ErrGlobal экспортируемая переменная уровня пакета,
// необходимая для предотвращений оптимизаций компилятора.
var ErrGlobal error

func BenchmarkErrorsNew(b *testing.B) {
```

```
    for i := 0; i < b.N; i++ {
        ErrGlobal = errors.New("invalid token")
    }
}

func BenchmarkErrorf(b *testing.B) {
    for i := 0; i < b.N; i++ {
        ErrGlobal = fmt.Errorf("invalid token")
    }
}
```

```
$ go test -benchmem -bench .
goos: darwin
goarch: amd64
BenchmarkErrorsNew-8    71727601    16.41 ns/op    16 B/op    1
allocs/op
BenchmarkErrorf-8       22454073    54.33 ns/op    32 B/op    2
allocs/op
PASS
ok      examples/05-errors-best-practices/errorf-bench      1.692s
```

Тем не менее недостатки, связанные с производительностью, мы отметаем по следующим причинам:

- если ошибка объявляется на уровне пакета (`var`), то всё это происходит единожды;
- возврат ошибки чаще всего тянет за собой завершение обработчика, остановку программы и пр., т.е. редка ситуация, когда мы в цикле, не прекращая, генерируем ошибки;
- использование `fmt.Errorf` вряд ли будет критическим местом вашего супер-хайлоад проекта.

Остаётся лишь чисто эстетическая составляющая (читай, вкусовщина) и как делать – решать вам.

---

Лично мы рекомендуем использовать `errors.New`, когда у вас нет аргументов для форматирования и вы просто создаете новую ошибку на основе константы/литерала. Так вы подчеркнёте "неформатируемость" ошибки и будете использовать инструменты по их прямому назначению:

```
// OK.
err := errors.New("bad connection")

// Not OK.
err := fmt.Errorf("bad connection")

// Not OK.
errWrapped := errors.New("bad connection: " + err.Error())

// OK
const badConnectionMsg = "bad connection"

func newBadConnError() error {
    return errors.New(badConnectionMsg)

}
```

---

Используйте `fmt.Errorf`, если нужно обернуть ошибку или добавить в создаваемую ошибку какую-то специфическую информацию:

```
err := fmt.Errorf("something went wrong on %d machine", i) // OK.


errWrapped := fmt.Errorf("something went wrong: %w", err) // OK.
```

## Тест "Выберите примеры желательного использования errors.New и fmt.Errorf"

Выберите все подходящие ответы из списка

- [ ] err := errors.New("bad connection")
- [ ] return errors.New("something went wrong on machine: " + strconv.Itoa(i))
- [ ] err := fmt.Errorf("user %q saving failed", userID)

- [ ] return fmt.Errorf("user saving failed: %w", err)
- [ ] return fmt.Errorf("do not do that")
- [ ] err := errors.New(fmt.Sprintf("insufficient privileges for user %q", userID))