Outline

01 – Visão Geral do Curso

Bem-vindo!

- Bem vindo ao curso!
- Eu sou...
- Go foi criada por gente foda que criou o Unix, B, UTF-8...
- Em 2006 o Google queria...
- É uma linguagem que vem crescendo horrores...
- Nesse curso você vai aprender...
- O curriculum que vamos estudar...
- Para os novos na programação... Para os programadores experientes...
- Participe!

Por que Go?

- Antes de investir seu tempo em aprender a linguagem Go, é bom você entender por que isso é uma boa idéia.
- O que estava acontecendo no Google...
- Criada por Ken Thompson (Unix, B, C), Rob Pike (UTF-8), e Robert Griesemer.
- Em 2006, não tinha uma linguagem de compilação rápida, execução rápida, e fácil de programar. É uma linguagem criada para resolver as questões de performance e complexidade.
- https://golang.org/doc/faq#creating_a_new_language
- Eficiente
  - Standard library é déis
  - Multiplataforma.
  - Garbage collection (lightning fast!)
  - Cross-compile.
- Fácil de usar
  - É uma linguagem compilada, de tipagem forte e estática,
  - Tem pouquíssimas palavras reservadas, que vamos aprender todas no curso, ou seja, é muito de boas de aprender
  - só sobe nas listas de popularidade
- Killer feature: Em 2006, logo após o primeiro dual core. Thread: 1mb. Goroutine: 2kb.
- É massa!
- Quando usar Go?
  - Escala
  - Seviços web, redes, servers (machine learning, image processing, crypto, ...)
  - Quando precisar de uma lingaugem rápida, simples, fácil de aprender, e fácil de usar.
- Usa em: APIs, CLIs, microservices, libraries/framework, processamento de dados, ... É a base dos serviços de cloud e orquestração de containers.
- Quem usa? https://github.com/golang/go/wiki/GoUsers
- Go é OOP? https://golang.org/doc/faq#Is_Go_an_object-oriented_language
- Mais um: https://golang.org/doc/faq#principles
- $$$: https://insights.stackoverflow.com/survey/2017#technology-top-paying-technologies-by-region -> US

Sucesso

- Qual o motivo que algumas pessoa obtem sucesso e outras não?
- Eu quero que você obtenha sucesso com este curso, então vamos falar sobre o assunto.
- Perguntaram a Bill Gates e Warren Buffet, independentemente, qual seria sua principal característica responsável pelo seu sucesso. A resposta de ambos foi:
- Foco.

- (Fonte: https://www.linkedin.com/pulse/20140707144749-8353952-the-one-word-answer-to-why-bill-gates-and-warren-buffett-have-been-so-successful/)
- O que isso quer dizer?
- Escolha um objetivo e se concentre nele. Faça desse objetivo sua maior prioridade.
- Foco. Concentração. → "Vou conseguir chegar lá ou vou morrer tentando."
- Exemplo: minha carreira de violinista.
- Faça disso uma prioridade, não uma resolução de ano novo. Tem vezes que seus amigos vão te chamar pra fazer algo massa e você vai ter que dizer, "gente, não vai dar, estou estudando."
- O Todd costuma dizer: de gota em gota se enche um balde. A cada dia uma gota. Algumas semanas depois você olha o balde, e pô meu não tem quase nada ali, pra que esse esforço todo? Mas depois de alguns anos o balde vai estar transbordando. É só trabalhar, de gota em gota.
- Então:
  - Seja proativo.
  - Trabalhe. Invista as horas de esforço.
  - Faça exercício, coma direito, tenha uma atitude positiva.
  - Easy mode: pare de ver TV e use esse tempo pra estudar.
- E algumas dicas práticas:
  - Metade das pessoas que assiste esse curso diz que eu falo muito rápido. A outra metade diz que eu falo muito devagar. Então use o botãozinho ali no player e selecione a velocidade mais apropriada pra você.
  - Boa parte de qualquer aprendizado se resume a memória muscular. Então quando eu passar exercícios, digite o código. Não copie e cole. Digite. E quando eu estiver programando "ao vivo," não passe pra frente, assista, e preste atenção. Se acostume com o processo, com o ato de programar.

- Com paciência e com persistência você chega lá.

Recursos

- Leia as descrições dos vídeos!
  - Outline completo: https://github.com/ellenkorbes/aprendago/blob/master/OUTLINE.md
- Exercícios:
- Comunidade:
  - Slack: https://invite.slack.golangbridge.org/
  - #brazil
  - #brazilian-go-studies, toda quinta às 22h!
    - gravações em: https://www.youtube.com/cesargimenes
    - exercícios em: https://github.com/crgimenes/Go-Hands-On
    - FB: https://www.facebook.com/gophers.br/
- Livros:
  - A Linguagem de Programação Go (Alan Donovan, Brian Kernighan): https://www.amazon.com.br/dp/8575225464/
  - Go em Ação (William Kennedy, Brian Ketelsen, Erik St. Martin): https://www.amazon.com.br/dp/8575225065/
  - Introdução à Linguagem Go (Caleb Doxsey): https://www.amazon.com.br/dp/8575224891/
- Em inglês: gobyexample.com, forum.golangbridge.org, stackoverflow.com/questions/tagged/go
- Documentação oficial:
  - golang.org
  - godoc.org
  - golang.org/doc/effective_go.html
- Podcast Go Time: https://changelog.com/gotime
- Defective Go: https://www.youtube.com/channel/UC98qIvCCqd4fjOw1ks78SwA

Como esse curso funciona

- Velocidade de playback.
- Repetição.
- Erros.

- Português vs. inglês
  - Obviamente esse é um curso em português.
  - Mas a língua semi-oficial da programação e do mundo da tecnologia em geral é a língua inglesa.
  - Por isso vamos utilizar bastante termos em inglês ao longo do curso.
  - As explicações serão em português, claro, mas quero que todos fiquem bem confortáveis com os termos em inglês.
  - Desta maneira você será auto-suficiente e poderá procurar por documentação e ajuda por toda a internet, não somente nos sites em português.
  - (Alem de que eu não sei e nem quero saber o nome em português de boa parte dessas coisas :P)
- Constantemente "em progresso."
- Seu feedback é super importante!

Work in progress!

- Em construção!

02 – Variáveis, Valores & Tipos

Go Playground

- É online, funciona sem instalar nem configurar nada.
- Assim você pode começar a programar o mais rápido possível.
- Mais pra frente no curso vou explicar direitinho como configurar tudo no seu computador.
- [Go Playground](https://play.golang.org/).
  - Função share. Use para compartilhar código, por exemplo pra fazer uma pergunta em um fórum.
  - Função imports.
  - Função format.
    - Maneira idiomática: a gente fala da mesma maneira que a comunidade onde estamos.
  - Função run.

Hello world!

- Estrutura básica:
  - package main.
  - func main: é aqui que tudo começa, é aqui que tudo acaba.
  - import.
- Packages:
  - Pacotes são coleções de funções pré-prontas (ou não) que você pode utilizar.
  - Notação: pacote.Identificador. Exemplo: fmt.Println()
  - Documentação: fmt.Println.
- Variáveis: "uma variável é um objeto (uma posição na memória) capaz de reter e representar um valor ou expressão."
- Variáveis não utilizadas? Não pode: _ nelas.
- ...funções variádicas.
- Lição principal: package main, func main, pacote.Identificador.

Operador curto de declaração

- := parece uma marmota (gopher) ou o punisher.
- Uso:
  - Tipagem automática
  - Só pode repetir se houverem variáveis novas
  - != do assignment operator (operador de atribuição)
  - Só funciona dentro de codeblocks
- Terminologia:
  - keywords (palavras-chave) são termos reservados
  - operadores, operandos

  - statement (declaração, afirmação) → uma linha de código, uma instrução que forma uma ação, formada de expressões
  - expressão -> qualquer coisa que "produz um resultado"
  - scope (abrangência)
    - package-level scope
- Lição principal:
  - := utilizado pra criar novas variáveis, dentro de code blocks
  - = para atribuir valores a variáveis já existentes

A palavra-chave var

- Variável declarada em um code block é undefined em outro
- Para variáveis com uma abrangência maior, package level scope, utilizamos `var`
- Funciona em qualquer lugar
- Prestar atenção: chaves, colchetes, parênteses

Explorando tipos

- Tipos em Go são extremamente importantes. (Veremos mais quando chegarmos em métodos e interfaces.)
- Tipos em Go são estáticos.
- Ao declarar uma variável para conter valores de um certo tipo, essa variável só poderá conter valores desse tipo.
- O tipo pode ser deduzido pelo compilador:
  - x := 10
  - var y = "a tia do batima"
- Ou pode ser declarado especificamente:
  - var w string = "isso é uma string"
  - var z int = 15
  - na declaração var z int com package scope, atribuição z = 15 no codeblock (somente)
- Tipos de dados primitivos: disponíveis na linguagem nativamente como blocos básicos de construção
  - int, string, bool
- Tipos de dados compostos: são tipos compostos de tipos primitivos, e criados pelo usuário
  - slice, array, struct, map
- O ato de definir, criar, estruturar tipos compostos chama-se composição. Veremos muito disso futuramente.

Valor zero

- Declaração vs. inicialização vs. atribuição de valor. Variáveis: caixas postais.
- O que é valor zero?
- Os zeros:
  - ints: 0
  - floats: 0.0
  - booleans: false
  - strings: ""
  - pointers, functions, interfaces, slices, channels, maps: nil
- Use := sempre que possível.
- Use var para package-level scope.

O pacote fmt

- Setup: strings, ints, bools.
- Strings: interpreted string literals vs. raw string literals.
  - Rune literals.
  - Em ciência da computação, um literal é uma notação para representar um valor fixo no código fonte.

- Format printing: documentação.
  - Grupo #1: Print -> standard out
    - func Print(a ...interface{}) (n int, err error)
    - func Println(a ...interface{}) (n int, err error)
    - func Printf(format string, a ...interface{}) (n int, err error)
      - Format verbs. (%v %T)
  - Grupo #2: Print -> string, pode ser usado como variável
    - func Sprint(a ...interface{}) string
    - func Sprintf(format string, a ...interface{}) string
    - func Sprintln(a ...interface{}) string
  - Grupo #3: Print -> file, writer interface, e.g. arquivo ou resposta de servidor
    - func Fprint(w io.Writer, a ...interface{}) (n int, err error)
    - func Fprintf(w io.Writer, format string, a ...interface{}) (n int, err error)
    - func Fprintln(w io.Writer, a ...interface{}) (n int, err error)

Criando seu próprio tipo

- Revisando: tipos em Go são extremamente importantes. (Veremos mais quando chegarmos em métodos e interfaces.)
- Tem uma história que Bill Kennedy dizia que se um dia fizesse uma tattoo, ela diria "type is life."
- Grande parte dos aspectos mais avançados de Go dependem quase que exclusivamente de tipos.
- Como fundação para estas ferramentas, vamos aprender a declarar nossos próprios tipos.
- Revisando: tipos são fixos. Uma vez declarada uma variável como de um certo tipo, isso é imutável.
- type hotdog int → var b hotdog (main hotdog)
- Uma variável de tipo hotdog não pode ser atribuida com o valor de uma variável tipo int, mesmo que este seja o tipo subjacente de hotdog.

Conversão, não coerção

- Conversão de tipos é o que soa.
- Em Go não se diz casting, se diz conversion.
- a = int(b)
- ref/spec#Conversions
- Fim da sessão. Parabéns! Dicas, motivação e exercícios.

03 – Exercícios: Ninja Nível 1

Contribua seu código

- Nesse curso a gente vai fazer um monte de exercícios.
  - Talvez você queira contribuir suas próprias soluções.
  - Talvez você tenha exemplos melhores que os que estamos mostrando aqui.
- Para compartilhar me manda o link no twitter do seu código no Go Playground, twitter.com/ellenkorbes!
- "@ellenkorbes Olha essa solução pro exercício Ninja nível 5, exercício 2: <link> O que vc acha?"

Na prática: exercício #1

- Esses são seus primeiros exercícios, e seus primeiros passos.
- Completando os exercícios dessa seção, você será um ninja nível 1.
- É o seu primeiro passo pra se tornar um developer ninja.
- Esses exercícios servem pra reforçar seu aprendizado. Só se aprende a programar programando. Ninguem aprende a andar de bicicleta assistindo vídeos de pessoas andando de bicicleta. É necessário botar a mão na massa.
- Eu vou começar explicando qual é o exercício. Aí vou pedir pra você dar pausa. Esse é o momento de por os miolos pra trabalhar, encontrar sua solução, tec-tec-tec, e rodar pra ver se funciona. Depois é só dar play novamente, ver a minha abordagem para a mesma questão, e comparar nossas soluções.

- Vamos lá:

- Utilizando o operador curto de declaração, atribua estes valores às variáveis com os identificadores "x", "y", e "z".
   1. 42
   2. "James Bond"
   3. true
- Agora demonstre os valores nestas variáveis, com:
   1. Uma única declaração print
   2. Múltiplas declarações print
- Solução: https://play.golang.org/p/yYXnWXIQNa

Na prática: exercício #2

- Use var para declarar três variáveis. Elas devem ter package-level scope. Não atribua valores a estas variáveis. Utilize os seguintes identificadores e tipos para estas variáveis:
   1. Identificador "x" deverá ter tipo int
   2. Identificador "y" deverá ter tipo string
   3. Identificador "z" deverá ter tipo bool
- Na função main:
   1. Demonstre os valores de cada identificador
   2. O compilador atribuiu valores para essas variáveis. Como esses valores se chamam?
- Solução: https://play.golang.org/p/pAFd-F7uGZ

Na prática: exercício #3

- Utilizando a solução do exercício anterior:
   1. Em package-level scope, atribua os seguintes valores às variáveis:
      1. para "x" atribua 42
      2. para "y" atribua "James Bond"
      3. para "z" atribua true
   2. Na função main:
      1. Use fmt.Sprintf para atribuir todos esses valores a uma única variável. Faça essa atribuição de tipo string a uma variável de nome "s" utilizando o operador curto de declaração.
      2. Demonstre a variável "s".
- Solução: https://play.golang.org/p/QFctSQB_h3

Na prática: exercício #4

- Crie um tipo. O tipo subjacente deve ser int.
- Crie uma variável para este tipo, com o identificador "x", utilizando a palavra-chave var.
- Na função main:
   1. Demonstre o valor da variável "x"
   2. Demonstre o tipo da variável "x"
   3. Atribua 42 à variável "x" utilizando o operador "="
   4. Demonstre o valor da variável "x"
- Para os aventureiros: https://golang.org/ref/spec#Types
- Agora já somos quase ninjas nível 1!
- Solução: https://play.golang.org/p/snm4WuuYmG

Na prática: exercício #5

- Utilizando a solução do exercício anterior:
   1. Em package-level scope, utilizando a palavra-chave var, crie uma variável com o identificador "y". O tipo desta variável deve ser o tipo subjacente do tipo que você criou no exercício anterior.
   2. Na função main:
      1. Isto já deve estar feito:

    1. Demonstre o valor da variável "x"
    2. Demonstre o tipo da variável "x"
    3. Atribua 42 à variável "x" utilizando o operador "="
    4. Demonstre o valor da variável "x"
  2. Agora faça tambem:
    1. Utilize conversão para transformar o tipo do valor da variável "x" em seu tipo subjacente e, utilizando o operador "=", atribua o valor de "x" a "y"
    2. Demonstre o valor de "y"
    3. Demonstre o tipo de "y"
- Solução: https://play.golang.org/p/uq81T_fasP

Na prática: exercício #6

- Prova!
- Link: https://goo.gl/forms/s9y91iVSPvA4iahj1
- Se você deu pausa e fez todos os exercícios anteriores você mesmo, e só viu a resposta depois... e se você der pausa agora e fizer a prova inteira por conta própria, e só assistir as respostas depois... sabe o que isso quer dizer? Que você é ninja. Ninja nível 1. Tá no caminho certo pra ser um developer ninja mestre.

04 – Fundamentos da Programação

Tipo booleano

- Agora vamos explorar os tipos de maneira mais detalhada. golang.org/ref/spec. A começar pelo bool.
- O tipo bool é um tipo binário, que só pode conter um dos dois valores: true e false. (Verdadeiro ou falso, sim ou não, zero ou um, etc.)
- Sempre que você ver operadores relacionais (==, <=, >=, !=, <, >), o resultado da expressão será um valor booleano.
- Booleans são fundamentais nas tomadas de decisões em lógica condicional, declarações switch, declarações if, fluxo de controle, etc.
- Na prática:
  - Zero value
  - Atribuindo um valor
  - Bool como resultado de operadores relacionais
- Go Playground: https://play.golang.org/p/7joj615nZw

Como os computadores funcionam

- Isso é importante pois daqui pra frente vamos falar de ints, bytes, e etc.
- Não é necessário um conhecimento a fundo mas é importante ter uma idéia de como as coisas funcionam por trás dos panos.
- https://docs.google.com/presentation/d/1aVytiGOBVDMISFW-ZARJ5iFY1osU2XJIw0hQpNICXm8/
- ASCII: https://en.wikipedia.org/wiki/ASCII
- Filme: Alan Turing, The Immitation Game.

Tipos numéricos

- int vs. float: Números inteiros vs. números com frações.
- golang.org/ref/spec → numeric types
- Integers:
  - Números inteiros
  - int & uint → “implementation-specific sizes”
  - Todos os tipos numéricos são distintos, exceto:
    - byte = uint8
    - rune = int32 (UTF8)
    (O código fonte da linguagem Go é sempre em UTF-8).

    - Tipos são únicos
        - Go é uma linguagem estática
        - int e int32 não são a mesma coisa
        - Para "misturá-los" é necessário conversão
    - Regra geral: use somente int
- Floating point:
    - Números racionais ou reais
    - Regra geral: use somente float64
- Na prática:
    - Defaults com :=
    - Tipagem com var
    - Dá pra colocar número com vírgula em tipo int?
    - Overflow
    - Go Playground: https://play.golang.org/p/dt2x1ies5b
- “implementation-specific sizes”? Runtime package. Word.
    - GOOS
    - GORUNTIME
    - https://play.golang.org/p/1vp5DImIMM

Overflow

- Um uint16, por exemplo, vai de 0 a 65535.
- Que acontece se a gente tentar usar 65536?
- Ou se a gente estiver em 65535 e tentar adicionar mais 1?
- Playground: https://play.golang.org/p/t7Z4m127F2t

Tipo string (cadeias de caracteres)

- Strings são sequencias de bytes.
- Imutáveis.
- Uma string é um "slice of bytes" (ou, em português, uma fatia de bytes).
- Na prática:
    - %v %T
    - Raw string literals
    - Conversão para slice of bytes: []byte(x)
    - %#U, %#x
    - Go Playground: https://play.golang.org/p/dt2x1ies5b & https://play.golang.org/p/PpDnspiyA_7
- https://blog.golang.org/strings

Sistemas numéricos

- Base-10: decimal, 0–9
- Base-2: binário, 0–1
- Base-16: hexadecimal, 0–f
- https://docs.google.com/document/d/1GqXpubhMMIr4Sy5xwgiPIDh5PGVmVpF2u0c9vDrvykE/
- Demonstração em Go.

Constantes

- São valores imutáveis.
- Podem ser tipadas ou não:
    - const oi = "Bom dia"
    - const oi string = "Bom dia"
- As não tipadas só terão um tipo atribuido a elas quando forem usadas.
    - Ex. qual o tipo de 42? int? uint? float64?
    - Ou seja, é uma flexibilidade conveniente.
- Na prática: int, float, string.

  - const x = y
  - const (x = y)

Iota

- golang.org/ref/spec
- Numa declaração de constantes, o identificador iota representa números sequenciais.
- Na prática.
  - iota, iota + 1, a = iota b c, reinicia em cada const, _
- Go Playground: https://play.golang.org/p/eSrwoQjuYR

Deslocamento de bits

- Deslocamento de bits é quando deslocamos digitos binários para a esquerda ou direita.
- Na prática:
  - %d %b
  - x << y
  - iota * 10 << 10 = kb, mb, gb

- https://play.golang.org/p/7MOnbhx4R4
- https://splice.com/blog/iota-elegant-constants-golang/
- https://medium.com/learning-the-go-programming-language/bit-hacking-with-go-e0acee258827

- Fim da sessão. Massa!

05 – Exercícios: Ninja Nível 2

Na prática: exercício #1

- Escreva um programa que mostre um número em decimal, binário, e hexadecimal.
- Solução: https://play.golang.org/p/X7qm3aWSa6

Na prática: exercício #2

- Escreva expressões utilizando os seguintes operadores, e atribua seus valores a variáveis.
  - ==
  - !=
  - <=
  - <
  - >=
  - >
- Demonstre estes valores.
- Solução: https://play.golang.org/p/BMYEch6_s8

Na prática: exercício #3

- Crie constantes tipadas e não-tipadas.
- Demonstre seus valores.
- Solução: https://play.golang.org/p/eWnKI59ual

Na prática: exercício #4

- Crie um programa que:
  - Atribua um valor int a uma variável
  - Demonstre este valor em decimal, binário e hexadecimal
  - Desloque os bits dessa variável 1 para a esquerda, e atribua o resultado a outra variável
  - Demonstre esta outra variável em decimal, binário e hexadecimal

- Solução: https://play.golang.org/p/IiwgT0v3Mp

Na prática: exercício #5

- Crie uma variável de tipo string utilizando uma raw string literal.
- Demonstre-a.
- Solução: https://play.golang.org/p/RkpqPpRWuo

Na prática: exercício #6

- Utilizando iota, crie 4 constantes cujos valores sejam os próximos 4 anos.
- Demonstre estes valores.
- Solução: https://play.golang.org/p/zRBEooRvo4

Na prática: exercício #7

- Prova!
- Link: https://goo.gl/forms/fnPXMmxvKAEUD8xP2
- Motivação. Ninja nível 2!

06 – Fluxo de Controle

Entendendo fluxo de controle

- Computadores lêem programas de uma certa maneira, do mesmo jeito que nós lemos livros, por exemplo, de uma certa maneira.
- Quando nós ocidentais lemos livros, lemos da frente pra trás, da esquerda pra direito, de cima pra baixo.
- Computadores lêem de cima pra baixo.
- Ou seja, sua leitura é sequencial. Isso chama-se fluxo de controle sequencial.
- Alem do fluxo de controle sequencial, há duas declarações que podem afetar como o computador lê o código:
  - Uma delas é o fluxo de controle de repetição (loop). Nesse caso, o computador vai repetir a leitura de um mesmo código de uma maneira específica. O fluxo de controle de repetição tambem é conhecido como fluxo de controle iterativo.
  - E o outro é o fluxo de controle condicional, ou fluxo de controle de seleção. Nesse caso o computador encontra uma condição e, através de uma declaração if ou switch, toma um curso ou outro dependendo dessa condição.
- Ou seja, há três tipos de fluxo de controle: sequencial, de repetição e condicional.

- Nesse capítulo:
  - Sequencial
  - Iterativo (loop)
    - for: inicialização, condição, pós
    - for: hierarquicamente
    - for: condição ("while")
    - for: ...ever?
    - for: break
    - for: continue
  - Condicional
    - declarações switch/case/default
      - não há fall-through por padrão
      - criando fall-through
      - default
      - múltiplos casos
      - casos podem ser expressões
        - se resultarem em true, rodam

    - tipo
  - if
    - bool
    - o operador "!"
    - declaração de inicialização
    - if, else
    - if, else if, else
    - if, else if, else if, ..., else

Loops: inicialização, condição, pós

- For
  - Inicialização, condição, pós
  - Ponto e vírgula?
  - gobyexample.com
  - Não existe while!

Loops: nested loop (repetição hierárquica)

- For
  - Repetição hierárquica
  - Exemplos: relógio, calendário

Loops: a declaração for

- For: inicialização, condição, pós
- For: condição ("while")
- For: ...ever? (http servers)
- For: break
- golang.org/ref/spec#For_statements, Effective Go
- (Range vem mais pra frente.)

Loops: break & continue

- Operação módulo: %
- For: break
- For: continue
- Go Playground: https://play.golang.org/p/gpKMP1wAEM & https://play.golang.org/p/8erMGEbZQix

Loops: utilizando ascii

- Desafio surpresa!
- Format printing:
  - Decimal %d
  - Hexadecimal %#x
  - Unicode %#U
  - Tab \t
  - Linha nova \n
- Faça um loop dos números 33 a 122, e utilize format printing para demonstrá-los como texto/string.
- Solução: https://play.golang.org/p/REm2WHyzzz

Condicionais: a declaração if

- If: bool
- If: o operador não → "!"
- If: declaração de inicialização
- Go Playground: https://play.golang.org/p/6nq2Tjb07i

Condicionais: if, else if, else

- If, else.
- If, else if, else.
- If, else if, else if, ..., else.
- Go Playground: https://play.golang.org/p/18VrRX2pec

Condicionais: a declaração switch

- Switch:
  - pode avaliar uma expressão
    - switch statement == case (value)
    - default switch statement == true (bool)
  - não há fall-through por padrão
  - criando fall-through
  - default
  - cases compostos

Condicionais: a declaração switch pt. 2 & documentação

- Pode avaliar tipos
- Pode haver uma expressão de inicialização

Operadores lógicos condicionais

- &&
- ||
- !
- Go Playground: https://play.golang.org/p/MFwrt93xlc
- Qual o resultado de fmt.Println...
  - true && true
  - true && false
  - true || true
  - true || false
  - !true

07 – Exercícios: Ninja Nível 3

Na prática: exercício #1

- Põe na tela: todos os números de 1 a 10000.
- Solução: https://play.golang.org/p/MkdZiDW8SQ

Na prática: exercício #2

- Põe na tela: O unicode code point de todas as letras maiúsculas do alfabeto, três vezes cada.
- Por exemplo:
  65
    U+0041 'A'
    U+0041 'A'
    U+0041 'A'
  66
    U+0042 'B'
    U+0042 'B'
    U+0042 'B'
  ...e por aí vai.

- Solução: https://play.golang.org/p/QlP6nVchq-

Na prática: exercício #3

- Crie um loop utilizando a sintaxe: for condition {}
- Utilize-o para demonstrar os anos desde que você nasceu.
- Solução: https://play.golang.org/p/qnFjiDJzLor

Na prática: exercício #4

- Crie um loop utilizando a sintaxe: for {}
- Utilize-o para demonstrar os anos desde que você nasceu.
- Solução: https://play.golang.org/p/dIbfdiuw0ms

Na prática: exercício #5

- Demonstre o resto da divisão por 4 de todos os números entre 10 e 100
- Solução: https://play.golang.org/p/zcEsXqnBr8

Na prática: exercício #6

- Crie um programa que demonstre o funcionamento da declaração if.
- Solução: https://play.golang.org/p/OGPgTJZbiy

Na prática: exercício #7

- Utilizando a solução anterior, adicione as opções else if e else.
- Solução: https://play.golang.org/p/_cO7E-yL0o

Na prática: exercício #8

- Crie um programa que utilize a declaração switch, sem switch statement especificado.
- Solução: https://play.golang.org/p/TyIGp4Hi8B

Na prática: exercício #9

- Crie um programa que utilize a declaração switch, onde o switch statement seja uma variável do tipo string com identificador "esporteFavorito".
- Solução: https://play.golang.org/p/4-iTPZwfEz

Na prática: exercício #10

- Anote (à mão) o resultado das expressões:
  - fmt.Println(true && true)
  - fmt.Println(true && false)
  - fmt.Println(true || true)
  - fmt.Println(true || false)
  - fmt.Println(!true)
- Ninja nível 3! Parabéns!

08 – Agrupamentos de Dados

Array

- Estruturas de dados, ou agrupamentos de dados, nos permitem agrupar valores diferentes. Estes valores podem ser ou não do mesmo tipo.
- As estruturas que veremos são: arrays, slices, structs e maps.

- Vamos começar com arrays. Arrays em Go são uma fundação, e não algo que utilizamos todo dia.
- Seu tamanho deve estar presente na declaração: var x [n]int
- Atribui-se valores a suas posições com: x[i] = y (0-based)
- Para ver o tamanho usa-se: len(x)
- ref/spec: "The length is part of the array's type" → [5]int != [6]int
- Effective Go: Arrays são úteis para [umas coisas que a gente não vai fazer nunca] e servem de fundação para slices. Use slices ao invés de arrays.
- Go Playground: https://play.golang.org/p/Fv-sDF-ryZ

Slice: literal composta

- O que são tipos de dados compostos?
  - Wikipedia: Composite_data_type
  - Effective Go: Composite literals
  - ref/spec: Composite literals
- Uma slice agrupa valores de um único tipo.
- Criando uma slice: literal composta → x := []type{values}
- Go Playground: https://play.golang.org/p/W7Cxm8NPZC

Slice: for range

- Slices:
  - Tamanho: len(x)
  - Índice específico: x[i] (0-based)
- Para ver todos os itens de uma slice utilizamos o loop for com range.
- Range significa alcance, faixa, extensão.
- For range: for i, v := range x {}
- Go Playground:
  - https://play.golang.org/p/h5-RFJn-Fh
  - https://play.golang.org/p/2wj02m3-eM

Slice: fatiando ou deletando de uma fatia

- x[:], x[a:], x[:b], x[a:b]
- "a" é incluso;
- "b" não é.
- Exemplo: cabeça magnética de um disco rígido (relógio, fita).
  - Off-by-one error.
- Go Playground: https://play.golang.org/p/i5ZOLKb3Fi
- É fatiando que se deleta um item de uma slice. Na prática:
  - x := append(x[:i], x[:i]...)
  - Go Playground: https://play.golang.org/p/xK2HwCqvwd
- Exercício: tente acessar todos os itens de uma slice *sem* utilizar range.
- Solução: https://play.golang.org/p/aUC9qVCobH

Slice: anexando a uma slice

- Effective Go: append (package builtin)
- x = append(slice, ...values)
- x = append(slice, slice...)
- Todd: unfurl → desdobrar, desenrolar
- Nome oficial: enumeration
- Go Playground: https://play.golang.org/p/RpkDCTumpT

Slice: make

- Slices são feitas de arrays.

- Elas são dinâmicas, podem mudar de tamanho.
- Sempre que isso acontece, um novo array é criado e os dados são copiados.
- É conveniente, mas tem um custo computacional.
- Para otimizar as coisas, podemos utilizar make.
- make([]T, len, cap)
- "The length of a slice may be changed as long as it still fits within the limits of the underlying array; just assign it to a slice of itself. The capacity of a slice, accessible by the built-in function cap, reports the maximum length the slice may assume."
- len(x), cap(x)
- x[n] onde n > len é out of range. Use append.
- Append > cap modifica o array subjacente.
- pkg/builtin/#append: "If it has sufficient capacity, the destination is resliced to accommodate the new elements. If it does not, a new underlying array will be allocated."
- Effective Go.
- Go Playground: https://play.golang.org/p/e8GWzyEEL8

Slice: slice multi-dimensional

- Slices multi-dimensionais são slices que contem slices.
- São como planilhas.
- [][]type
- Go Playground: https://play.golang.org/p/vKyHiG1GtM
- Só pra sacanear: https://play.golang.org/p/ZSU_8eJ9Yp

Slice: a surpresa do array subjacente

- Isso tudo aqui a gente já viu:
- Toda slice tem um array subjacente.
- Um slice é: um ponteiro/endereço para um array, mais len e cap (que é o len to array).
- Exemplo:
  - x := []int{...números}
  - y := append(x[:i], x[:i]...)
  - pkg/builtin/#append: "If it has sufficient capacity, the destination is resliced to accommodate the new elements. If it does not, a new underlying array will be allocated."
  - Ou seja, y utiliza o mesmo array subjacente que x.
  - O que nos dá um resultado inesperado.
- Ou seja, bom saber de antemão pra não ter que aprender na marra.
- Go Playground: https://play.golang.org/p/BBJLuIjU_i

Maps: introdução

- Utiliza o formato key:value.
- E.g. nome e telefone
- Performance excelente para lookups.
- map[key]value{ key: value }
- Acesso: m[key]
- Key sem value retorna zero. Isso pode trazer problemas.
- Para verificar: comma ok idiom.
  - v, ok := m[key]
  - ok é um boolean, true/false
- Na prática: if v, ok := m[key]; ok { }
- Para adicionar um item: m[v] = value
- Maps *não tem ordem.*
- Go Playground: https://play.golang.org/p/JXDdJan8Ev

Maps: range & deletando

- Range: for k, v := range map { }
- Reiterando: maps *não tem ordem* e um range usará uma ordem aleatória.
- Go Playground: https://play.golang.org/p/6zEMfIP-AE
- delete(map, key)
- Deletar uma key não-existente não retorna erros!
- Go Playground: https://play.golang.org/p/0uuIicU3Zz

09 – Exercícios: Ninja Nível 4

Na prática: exercício #1

- Usando uma literal composta:
  - Crie um array que suporte 5 valores to tipo int
  - Atribua valores aos seus índices
- Utilize range e demonstre os valores do array.
- Utilizando format printing, demonstre o tipo do array.
- Solução: https://play.golang.org/p/tpWDzzlO2l

Na prática: exercício #2

- Usando uma literal composta:
  - Crie uma slice de tipo int
  - Atribua 10 valores a ela
- Utilize range para demonstrar todos estes valores.
- E utilize format printing para demonstrar seu tipo.
- Solução: https://play.golang.org/p/ST3TKusuOd

Na prática: exercício #3

- Utilizando como base o exercício anterior, utilize slicing para demonstrar os valores:
  - Do primeiro ao terceiro item do slice (incluindo o terceiro item!)
  - Do quinto ao último item do slice (incluindo o último item!)
  - Do segundo ao sétimo item do slice (incluindo o sétimo item!)
  - Do terceiro ao penúltimo item do slice (incluindo o penúltimo item!)
  - Desafio: obtenha o mesmo resultado acima utilizando a função len() para determinar o penúltimo item
- Solução: https://play.golang.org/p/1aPXVeR1mf

Na prática: exercício #4

- Começando com a seguinte slice:
  - x := []int{42, 43, 44, 45, 46, 47, 48, 49, 50, 51}
- Anexe a ela o valor 52;
- Anexe a ela os valores 53, 54 e 55 utilizando uma única declaração;
- Demonstre a slice;
- Anexe a ela a seguinte slice:
  - y := []int{56, 57, 58, 59, 60}
- Demonstre a slice x.
- Solução: https://play.golang.org/p/6WNJ0Otpy0

Na prática: exercício #5

- Comece com essa slice:
  - x := []int{42, 43, 44, 45, 46, 47, 48, 49, 50, 51}
- Utilizando slicing e append, crie uma slice y que contenha os valores:
  - [42, 43, 44, 48, 49, 50, 51]
- Solução: https://play.golang.org/p/26bT-UKmJH

Na prática: exercício #6

- Crie uma slice usando make que possa conter todos os estados do Brasil.
  - Os estados: "Acre", "Alagoas", "Amapá", "Amazonas", "Bahia", "Ceará", "Espírito Santo", "Goiás", "Maranhão", "Mato Grosso", "Mato Grosso do Sul", "Minas Gerais", "Pará", "Paraíba", "Paraná", "Pernambuco", "Piauí", "Rio de Janeiro", "Rio Grande do Norte", "Rio Grande do Sul", "Rondônia", "Roraima", "Santa Catarina", "São Paulo", "Sergipe", "Tocantins"
- Demonstre o len e cap da slice.
- Demonstre todos os valores da slice *sem utilizar range.*
- Solução: https://play.golang.org/p/cGYBphlyCE

Na prática: exercício #7

- Crie uma slice contendo slices de strings ([][]string). Atribua valores a este slice multi-dimensional da seguinte maneira:
  - "Nome", "Sobrenome", "Hobby favorito"
- Inclua dados para 3 pessoas, e utilize range para demonstrar estes dados.
- Solução: https://play.golang.org/p/Gh81-d5tMi

Na prática: exercício #8

- Crie um map com key tipo string e value tipo []string.
  - Key deve conter nomes no formato sobrenome_nome
  - Value deve conter os hobbies favoritos da pessoa
- Demonstre todos esses valores e seus índices.
- Solução: https://play.golang.org/p/nD3TW8VQmH

Na prática: exercício #9

- Utilizando o exercício anterior, adicione uma entrada ao map e demonstre o map inteiro utilizando range.
- Solução: https://play.golang.org/p/3fcvHlt8Lm

Na prática: exercício #10

- Utilizando o exercício anterior, remova uma entrada do map e demonstre o map inteiro utilizando range.
- Solução:

10 – Structs

Struct

- Struct é um tipo de dados composto que nos permite armazenar valores de tipos *diferentes.*
- Seu nome vem de "structure," ou estrutura.
- Declaração: type x struct { y: z }
- Acesso: x.y
- Exemplo: nome, idade, fumante.
- Go Playground: https://play.golang.org/p/5i0DqxuBp1

Structs embutidos

- Structs dentro de structs dentro de structs.
- Exemplo: um corredor de fórmula 1 é uma pessoa (nome, sobrenome, idade) *e tambem* um competidor (nome, equipe, pontos).

- Go Playground:

Lendo a documentação

- É importante se familiarizar com a documentação da linguagem Go.
- Neste vídeo vamos ver um pouco sobre o que a documentação diz sobre structs.
- Veremos:
  - ref/spec
    - Já vimos mais da metade dos tipos em Go!
    - Struct types.
      - x, y int
      - anonymous fields
      - promoted fields
- Go Playground: https://play.golang.org/p/z9UQej4IQT

Structs anônimos

- São structs sem identificadores.
- x := struct { name type }{ name: value }
- Go Playground: https://play.golang.org/p/xyhNnSCu1f

Colocando ordem na casa

- [???]
- Go Playground:

11 – Exercícios: Ninja Nível 5

Na prática: exercício #1

- Crie um tipo "pessoa" com tipo subjacente struct, que possa conter os seguintes campos:
  - Nome
  - Sobrenome
  - Sabores favoritos de sorvete
- Crie dois valores do tipo "pessoa" e demonstre estes valores, utilizando range na slice que contem os sabores de sorvete.
- Solução: https://play.golang.org/p/Pyp7vmTJfY

Na prática: exercício #2

- Utilizando a solução anterior, coloque os valores do tipo "pessoa" num map, utilizando os sobrenomes como key.
- Demonstre os valores do map utilizando range.
- Os diferentes sabores devem ser demonstrados utilizando outro range, dentro do range anterior.
- Solução: https://play.golang.org/p/GLK11Q1_x8y

Na prática: exercício #3

- Crie um novo tipo: veículo
  - O tipo subjacente deve ser struct
  - Deve conter os campos: portas, cor
- Crie dois novos tipos: caminhonete e sedan
  - Os tipos subjacentes devem ser struct
  - Ambos devem conter "veículo" como struct embutido
  - O tipo caminhonete deve conter um campo bool chamado "traçãoNasQuatro"
  - O tipo sedan deve conter um campo bool chamado "modeloLuxo"
- Usando os structs veículo, caminhonete e sedan:

  - Usando composite literal, crie um valor de tipo caminhonete e dê valores a seus campos
  - Usando composite literal, crie um valor de tipo sedan e dê valores a seus campos
- Demonstre estes valores.
- Demonstre um único campo de cada um dos dois.
- Solução: https://play.golang.org/p/3eoGb0kxzT

Na prática: exercício #4

- Crie e use um struct anônimo.
- Desafio: dentro do struct tenha um valor de tipo map e outro do tipo slice.
- Solução: https://play.golang.org/p/iTGLyH0Ijc & https://play.golang.org/p/h247Kid5adG

12 – Funções

Sintaxe

- Qual a utilidade de funções?
  - Abstrair funcionalidade
  - Reutilização de código
- func (receiver) identifier(parameters) (returns) { code }
- A diferença entre parâmetros e argumentos:
  - Funções são definidas com parâmetros
  - Funções são chamadas com argumentos
- Tudo em Go é *pass by value.*
  - Pass by reference, pass by copy, ... não.
- Parâmetro pode ser ...variádico.
- Exemplos:
  - Função básica.
    - Go Playground: https://play.golang.org/p/FebJblBenP
  - Função que aceita um argumento.
    - Go Playground:
    https://play.golang.org/p/CE6Ij3U4QB
  - Função com retorno.
    - Go Playground: https://play.golang.org/p/gKxwYe6btP
  - Função com múltiplos retornos e parâmetro variádico.
    - Go Playground: https://play.golang.org/p/OcQ1wXwM2c
  - Mais um: https://play.golang.org/p/8wc2TA9xH_

Desenrolando (enumerando) uma slice

- Quando temos uma slice, podemos passar os elementos individuais através "deste..." operador.
- Exemplos:
  - Desenrolando uma slice de ints com como argumento para a função "soma" anterior
    - Go Playground: https://play.golang.org/p/k8O3__8UDa
  - Pode-se passar *zero* ou mais valores
    - Go Playground: https://play.golang.org/p/C238I9n7Vs
  - O parâmetro variádico deve ser o parâmetro final →
ref/spec#Passing_arguments_to_..._parameters
    - Go Playground: https://play.golang.org/p/8wc2TA9xH_
    - Não roda: https://play.golang.org/p/2qTAnLWfgB

Defer

- Funções são ótimas pois tornam nosso código modular. Podemos alterar partes do nosso programa sem afetar o resto!

- Uma declaração defer chama uma função cuja execução ocorrerá no momento em que a função da qual ela faz parte finalizar.
- Essa finalização pode ocorrer devido a um return, ao fim do code block da função, ou no caso de pânico em uma goroutine correspondente.
- "Deixa pra última hora!"
- ref/spec
- Sempre usamos para fechar um arquivo após abri-lo.
- Go Playground: https://play.golang.org/p/sFj8arw0E_

Métodos

- Um método é uma função anexada a um tipo.
- Quando se anexa uma função a um tipo, ela se torna um método desse tipo.
- Pode-se anexar uma função a um tipo utilizando seu receiver.
- Utilização: valor.método()
- Exemplo: o tipo "pessoa" pode ter um método oibomdia()
- Go Playground: https://play.golang.org/p/tQtoqUBpY5

Interfaces & polimorfismo

- Em Go, valores podem ter mais que um tipo.
- Uma interface permite que um valor tenha mais que um tipo.
- Declaração: keyword identifier type → type x interface
- Após declarar a interface, deve-se definir os métodos necessários para implementar essa interface.
- Se um tipo possuir todos os métodos necessários (que, no caso da interface{}, pode ser nenhum) então esse tipo implicitamente implementa a interface.
- Esse tipo será o seu tipo *e também* o tipo da interface.

- Exemplos:
  - Os tipos *profissão1* e *profissão2* contem o tipo *pessoa*
  - Cada um tem seu método *oibomdia()*, e podem dar oi utilizando *pessoa.oibomdia()*
  - Implementam a interface *gente*
  - Ambos podem acessar o função *serhumano()* que chama o método *oibomdia()* de cada *gente*
  - Tambem podemos no método *serhumano()* tomar ações diferentes dependendo do tipo: switch pessoa.(type) { case profissão1: fmt.Println(h.(profissão1).valorquesóexisteemprofissão1) [...] }*
  - Go Playground pré-pronto: https://play.golang.org/p/VLbo_1uE-U https://play.golang.org/p/zGKr7cvTPF
  - Go Playground ao vivo: https://play.golang.org/p/njiKbTT20Cr
- Onde se utiliza?
  - Área de formas geométricas (gobyexample.com)
  - Sort
  - DB
  - Writer interface: arquivos locais, http request/response
- Se isso estiver complicado, não se desespere. É foda mesmo. Com tempo e prática a fluência vem.

Funções anônimas

- Anonymous self-executing functions → Funções anônimas auto-executáveis.
- func(p params) { ... }()
- Vamos ver bastante quando falarmos de goroutines.
- Go Playground: https://play.golang.org/p/Rnqmo6X6jh

Func como expressão

- f := func(p params){ ... }
- f()
- Go Playground: https://play.golang.org/p/cPxhPUbfLy

Retornando uma função

- Pode-se usar uma função como retorno de uma função
- Declaração: func f() return
- Exemplo: func f() func() int { [...]; return func() int{ return [int] } }
  - ????: fmt.Println(f()())
- Go Playground: https://play.golang.org/p/zPjoWNrCJF

Callback

- Primeiro veja se você entende isso: https://play.golang.org/p/QkAtwMZU-z
- Callback é passar uma função como argumento.
- Exemplo:
  - Criando uma função que toma uma função e um []int, e usa somente os números pares como argumentos para a função.
  - Go Playground:
- Desafio: Crie uma função no programa acima que utilize somente os números *ímpares.*
- Solução:

Closure

- Closure é cercar ou capturar um scope para que possamos utilizá-lo em outro contexto. Já vimos:
  - Package-level scope
  - Function-level scope
  - Code-block-in-code-block scope
- Exemplo de closure:
  - func i() func() int { x := 0; return func() int { x++; return x } }
  - Quando fizermos a := i() teremos um scope, um valor para x.
  - Quando fizermos b := i() teremos outro scope, e x terá um valor independente do x acima.
- Closures nos permitem salvar dados entre function calls e ao mesmo tempo isolar estes dados do resto do código.
- Go Playground: https://play.golang.org/p/AdFciYwI2Z

Recursividade

- WP: "The most common application of recursion is in mathematics and computer science, where a function being defined is applied within its own definition."
- Exemplos de recursividade: Fractais, matrioscas, efeito Droste (o efeito produzido por uma imagem que aparece dentro dela própria), GNU (“GNU is Not Unix”), etc.
- No estudo de funções: é uma função que chama a ela própria.
- Exemplo: fatoriais.
  - 4! = 4 * 3 * 2 * 1 (e no zero, deu.)
  - Com recursividade. Go Playground: https://play.golang.org/p/ujsLnUhRp_
  - Com loops. Go Playground: https://play.golang.org/p/F2VsUjYVhc

13 – Exercícios: Ninja Nível 6

Na prática: exercício #1

- Exercício:
  - Crie uma função que retorne um int
  - Crie outra função que retorne um int e uma string
  - Chame as duas funções

  - Demonstre seus resultados
- Solução: https://play.golang.org/p/rxJM5fgI-9

- Revisão:
  - Funções!
    - Servem para abstrair código
    - E para reutilizar código
  - A ordem das coisas é:
    - func (receiver) identifier (parameters) (returns) { code }
  - Parâmetros vs. argumentos
  - Funções variádicas
    - Múltiplos parâmetros
    - Múltiplos argumentos
  - Métodos
  - Interfaces & polimorfismo
  - Defer
    - "Deixa pra depois!"
  - Returns
    - Múltiplos returns
    - Returns com nome (blé!)
  - Funcs como expressões
    - Atribuindo uma função a uma variável
  - Callbacks
    - Passando uma função como argumento para outra função
  - Closure
    - Capturando um scope
    - Variáveis declaradas em scopes externos são visíveis em scopes internos
  - Recursividade
    - Uma função que chama a ela mesma
    - Fatoriais

Na prática: exercício #2

- Crie uma função que receba um parâmetro variádico do tipo int e retorne a soma de todos os ints recebidos;
- Passe um valor do tipo slice of int como argumento para a função;
- Crie outra função, esta deve receber um valor do tipo slice of int e retornar a soma de todos os elementos da slice;
- Passe um valor do tipo slice of int como argumento para a função.
- Solução: https://play.golang.org/p/Tgv3wwxKV-

Na prática: exercício #3

- Utilize a declaração defer de maneira que demonstre que sua execução só ocorre ao final do contexto ao qual ela pertence.
- Solução: https://play.golang.org/p/b5Ua2kNN8a

Na prática: exercício #4

- Crie um tipo struct "pessoa" que contenha os campos:
  - nome
  - sobrenome
  - idade
- Crie um método para "pessoa" que demonstre o nome completo e a idade;
- Crie um valor de tipo "pessoa";
- Utilize o método criado para demonstrar esse valor.
- Solução: https://play.golang.org/p/GBZcnu0Ajp

Na prática: exercício #5

- Crie um tipo "quadrado"
- Crie um tipo "círculo"
- Crie um método "área" para cada tipo que calcule e retorne a área da figura
  - Área do círculo: 2 * π * raio
  - Área do quadrado: lado * lado
- Crie um tipo "figura" que defina como interface qualquer tipo que tiver o método "área"
- Crie uma função "info" que receba um tipo "figura" e retorne a área da figura
- Crie um valor de tipo "quadrado"
- Crie um valor de tipo "círculo"
- Use a função "info" para demonstrar a área do "quadrado"
- Use a função "info" para demonstrar a área do "círculo"
- Solução: https://play.golang.org/p/qLY-q3vffQ

Na prática: exercício #6

- Crie e utilize uma função anônima.
- Solução: https://play.golang.org/p/Kgo6hVr5G5

Na prática: exercício #7

- Atribua uma função a uma variável.
- Chame essa função.
- Solução: https://play.golang.org/p/RMHLL3N5Ww

Na prática: exercício #8

- Crie uma função que retorna uma função.
- Atribua a função retornada a uma variável.
- Chame a função retornada.
- Solução: https://play.golang.org/p/A74rufv6Rs

Na prática: exercício #9

- Callback: passe uma função como argumento a outra função.
- Solução: https://play.golang.org/p/2epLD_Yyap

Na prática: exercício #10

- Demonstre o funcionamento de um closure.
- Ou seja: crie uma função que retorna outra função, onde esta outra função faz uso de uma variável alem de seu scope.
- Solução: https://play.golang.org/p/sA7NHpkCCg

Na prática: exercício #11

- Uma das melhores maneiras de aprender é ensinando.
- Para este exercício escolha o seu código favorito dentre os que vimos estudando funções. Pode ser das aulas ou da seção de exercícios. Então:
  - Faça download e instale isso aqui: https://obsproject.com/
  - Grave um vídeo onde *você* ensina o tópico em questão
  - Faça upload do vídeo no YouTube
  - Compartilhe o vídeo no Twitter e me marque no tweet (@ellenkorbes)

14 – Ponteiros

O que são ponteiros?

- Todos os valores ficam armazenados na memória.
- Toda localização na memória possui um endereço.
- Um pointeiro se refere a esse endereço.
- Notações:
  - &variável mostra o endereço de uma variável
    - %T: variável vs. &variável
  - *variável faz de-reference, mostra o valor que consta nesse endereço
  - ????: *&var funciona!
  - *type é um tipo que contem o endereço de um valor do tipo type, nesse caso * não é um operador
- Exemplo: a := 0; b := &a; *b++
- Go Playground: https://play.golang.org/p/gC1qGFUYrV

Quando usar ponteiros

- Ponteiros permitem compartilhar endereços de memória. Isso é útil quando:
  - Não queremos passar grandes volumes de dados pra lá e pra cá
  - Queremos mudar um valor em sua localização original (tudo em Go é pass by value!)
- Exemplos:
  - x := 0; funçãoquemudaovalordoargumentopra1(x); Print(x)
  - x := 0; funçãoquemudaovalordo*argumentopra1(&x); Print(x)
- Go Playground: https://play.golang.org/p/VZmfWfw76s

15 – Exercícios: Ninja Nível 7

Na prática: exercício #1

- Crie um valor e atribua-o a uma variável.
- Demonstre o endereço deste valor na memória.
- Solução: https://play.golang.org/p/0jVt1yaoFL

Na prática: exercício #2

- Crie um struct "pessoa"
- Crie uma função chamada mudeMe que tenha *pessoa como parâmetro. Essa função deve mudar um valor armazenado no endereço *pessoa.
  - Dica: a maneira "correta" para fazer dereference de um valor em um struct seria (*valor).campo
  - Mas consta uma exceção na documentação. Link: https://golang.org/ref/spec#Selectors
  - "As an exception, if the type of x is a named pointer type and (*x).f is a valid selector expression denoting a field (but not a method),
  - → x.f is shorthand for (*x).f." ←
  - Ou seja, podemos usar tanto o atalho p1.nome quanto o tradicional (*p1).nome
- Crie um valor do tipo pessoa;
- Use a função mudeMe e demonstre o resultado.
- Solução: https://play.golang.org/p/qiYp9leJcn

16 – Aplicações

Documentação JSON

- Já entendemos ponteiros, já entendemos métodos. Já temos o conhecimento necessário para começar a utilizar a standard library.
- Nesse vídeo faremos uma orientação sobre como abordar a documentação.

- Essa aula não foi preparada. Vai ser tudo ao vivo no improviso pra vocês verem como funciona o processo.
  - golang.org → Documents → Package Documentation
  - godoc.org → encoding/json
    - files
    - examples
    - funcs
    - types
    - methods

JSON marshal (ordenação)

- Exemplo: transformando structs em Go em código JSON.
- No improviso tambem.
- Go Playground: https://play.golang.org/p/_JvCOlK-H9

JSON unmarshal (desordenação)

- E agora o contrário.
- Link: https://cdn.rawgit.com/GoesToEleven/golang-web-dev/17e3852d/040_json/README.html
- JSON-to-Go
- Tags
- Marshal/unmarshal vs. encoder/decoder
  - Marshal vai pra uma variável
  - Encoder "vai direto"
- Go Playground: https://play.golang.org/p/l6wbuLu1NS
- Com Encoder: https://play.golang.org/p/Pgwr0O07aL

A interface Writer

- A interface writer do pacote io.
- type Writer interface { Write(p []byte) (n int, err error) }
  - pkg os: func (f *File) Write(b []byte) (n int, err error)
  - pkg json: func NewEncoder(w io.Writer) *Encoder
- "Println [...] writes to standard output."
  - func Println [...] return Fprintln(os.Stdout, a...)
  - func Fprintln(w io.Writer, a ...interface{}) (n int, err error)
  - Stdout: NewFile(uintptr(syscall.Stdout), "/dev/stdout") (Google: Standard streams)
  - func NewFile(fd uintptr, name string) *File
  - func (f *File) Write(b []byte) (n int, err error)
- Exemplo:
  - Println
  - Fprintln os.Stdout
  - io.WriteString os.Stdout
  - Ou:
    - func Dial(network, address string) (Conn, error)
    - type Conn interface { [...] Write(b []byte) (n int, err error) [...] }
- Go Playground:

O pacote sort

- Sort serve para ordenar slices.
  - Como faz?
  - golang.org/pkg/ → sort
  - godoc.org/sort → examples
  - Sort altera o valor original!
- Exemplo: Ints, Strings.

- Go Playground:
  - sort.Strings: https://play.golang.org/p/Rs1NVwmg7h
  - sort.Ints: https://play.golang.org/p/I2_vsHujZa

Customizando o sort

- O sort que eu quero não existe. Quero fazer o meu.
- Para isso podemos usar o func Sort do package sort. Vamos precisar de um sort.Interface.
  - type Interface interface { Len() int; Less(i, j int) bool; Swap(i, j int) }
- Ou seja, se tivermos um tipo que tenha esses métodos, ao executar sort.Sort(x) as funções que vão rodar são as minhas, não as funções pré-prontas como no exercício anterior.
- E aí posso fazer do jeito que eu quiser.
- Exemplo:
  - struct carros: nome, consumo, potencia
  - slice []carros{carro1, carro2, carro3} (Sort ordena *slices!*)
  - tipo ordenarPorPotencia
  - tipo ordenarPorConsumo
  - tipo ordenarPorLucroProDonoDoPosto
- Go Playground: https://play.golang.org/p/KOIhAsE3OK

bcrypt

- É uma maneira de encriptar senhas utilizando hashes.
- x/crypto/bcrypt
  - GenerateFromPassword
  - CompareHashAndPassword
- Sem Go Playground!
  - go get golang.org/x/crypto/bcrypt
- Arquivo:
https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/16_aplicacao/bcrypt/main.go

17 – Exercícios: Ninja Nível 8

Na prática: exercício #1

- Partindo do código abaixo, utilize marshal para transformar []user em JSON.
  - https://play.golang.org/p/U0jea43X55
- Atenção! Tem pegadinha aqui.
- Solução: https://play.golang.org/p/gUOHXruFGs

Na prática: exercício #2

- Partindo do código abaixo, utilize unmarshal e demonstre os valores.
  - https://play.golang.org/p/b_UuCcZag9
- Dica: JSON-to-Go.
- Solução: https://play.golang.org/p/UyL4MCGs_u

Na prática: exercício #3

- Partindo do código abaixo, utilize NewEncoder() e Encode() para enviar as informações como JSON para Stdout.
  - https://play.golang.org/p/BVRZTdlUZ_
- Desafio: descubra o que é, e utilize, method chaining para conectar os dois métodos acima.
- Solução: https://play.golang.org/p/U4xSnZD_3r

Na prática: exercício #4

- Partindo do código abaixo, ordene a []int e a []string.
  - https://play.golang.org/p/H_q75mpmHW
- Solução: https://play.golang.org/p/3J3wcfVQBZ

Na prática: exercício #5

- Partindo do código abaixo, ordene os []user por idade e sobrenome.
  - https://play.golang.org/p/BVRZTdlUZ_
- Os valores no campo Sayings devem ser ordenados tambem, e demonstrados de maneira esteticamente harmoniosa.
- Solução: https://play.golang.org/p/3wgW4BDasu

18 – Concorrência

Concorrência vs. paralelismo

- Concorrência é quando abre uma padaria do lado da outra e as duas quebram :)
- Fun facts:
  - O primeiro CPU dual core "popular" veio em 2006
  - Em 2007 o Google começou a criar a linguagem Go para utilizar essa vantagem
  - Go foi a primeira linguagem criada com multi-cores em mente
  - C, C++, C#, Java, JavaScript, Python, etc., foram todas criadas antes de 2006
  - Ou seja, Go tem uma abordagem única (fácil!) para este tópico
- E qual a diferença entre concorrência e paralelismo?

Goroutines & WaitGroups

- O código abaixo é linear. Como fazer as duas funções rodarem concorrentemente?
  - https://play.golang.org/p/XP-ZMeHUk4
- Goroutines!
- O que são goroutines? São "threads."
- O que são threads?
[WP](https://pt.wikipedia.org/wiki/Thread_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o))
- Na prática: go func.
- Exemplo: código termina antes da go func executar.
- Ou seja, precisamos de uma maneira pra "sincronizar" isso.
- Ah, mas então... não.
- Qualé então? sync.WaitGroup:
- Um WaitGroup serve para esperar que uma coleção de goroutines termine sua execução.
  - func Add: "Quantas goroutines?"
  - func Done: "Deu!"
  - func Wait: "Espera todo mundo terminar."
- Ah, mas então... sim!
- Só pra ver: runtime.NumCPU() & runtime.NumGoroutine()
- Go Playground: https://play.golang.org/p/8iiqLX4sWc

Discussão: Condição de corrida

- Agora vamos dar um mergulho na documentação:
  - https://golang.org/doc/effective_go.html#concurrency
  - https://pt.wikipedia.org/wiki/Multiplexador
  - O que é yield? runtime.Gosched()
- Race condition:

*Função 1	var	Função 2*
Lendo: 0	→ 0	
Yield	0	→ Lendo: 0
var++: 1		Yield

Grava: 1 → 1 var++: 1
1 ← Grava: 1
Lendo: 1 ← 1
Yield 1 → Lendo: 1
var++: 2 Yield
Grava: 2 → 2 var++: 2
2 ← Grava: 2

- E é por isso que vamos ver mutex, atomic e, por fim, channels.

Condição de corrida

- Aqui vamos replicar a race condition mencionada no artigo anterior.
  - time.Sleep(time.Second) vs. runtime.Gosched()
- go help → go help build → go run -race main.go
- Como resolver? Mutex.
- Código:
https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/18_concorrencia/05_race_condition/main.go

Mutex

- Agora vamos resolver a race condition do programa anterior utilizando mutex.
- Mutex é mutual exclusion, exclusão mútua.
- Utilizando mutex somente uma thread poderá utilizar a variável contador de cada vez, e as outras deve aguardar sua vez "na fila."
- Na prática:
  - type Mutex
    - func (m *Mutex) Lock()
    - func (m *Mutex) Unlock()
- RWMutex
- Código:
https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/18_concorrencia/06_mutex/main.go

Atomic

- Agora vamos fazer a mesma coisa, mas com atomic ao invés de mutex.
  - atomic.AddInt64
  - atomic.LoadInt64
- Código:
https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/18_concorrencia/07_atomic/main.go

19 – Seu Ambiente de Desenvolvimento

O terminal

- Terminologia:
  - GUI: Graphical User Interface
  - CLI: Command Line Interface
    - Terminal, console, etc
  - Unix, Linux, Mac:
    - Shell, bash
  - Windows:
    Command prompt, cmd, dos prompt, powershell
- Shell/bash commands:
  - pwd

  - ls
    - ls -la
    - Permissions: owner, group, world
    - r, w, x → 4, 2, 1 (d = directory)
    - rwxrwxrwx = owner, group, world
  - touch
  - clear
  - chmod
    - chmod options permissions filename
    - chmod 777 arquivo.ext
  - cd
    - cd ../
    - cd qualquer/pasta/
  - env
  - rm <file or folder name>
    - rm -rf <file or folder name>
  - .bash_profile & .bashrc
    - .bash_profile is executed for login shells, while .bashrc is executed for interactive non-login shells.
    - When you login (type username and password) via console, either sitting at the machine, or remotely via ssh: .bash_profile is executed to configure your shell before the initial command prompt.
  - nano <file name>
  - cat <file name>
  - grep
    - cat temp2.txt | grep enter
    - ls | grep -i documents

Go workspace & environment variables

- $GOPATH/
  bin/
  pkg/
  src/
    github.com/
      <Nome do usuário (github.com)>/
        <Nome do projeto ou repo>/
        <Nome do projeto ou repo>/
        <Nome do projeto ou repo>/
        <Nome do projeto ou repo>/
        ...
        <Nome do projeto ou repo>/
- GOROOT: onde os binários da instalação do Go foram instalados
  - GOROOT="/usr/lib/go"
- GOPATH: onde seus arquivos de trabalho, seu workspace, fica localizado
  - GOPATH="/home/ellen/go"
  - export GOPATH=$HOME/go (.bashrc)
- Package management? go get.
  - Na prática → e.g. gouuid

IDE's

- Integrated development environment. WP: "[...] é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo."
- IDEs:
  - Visual Studio Code → https://code.visualstudio.com/
    - go get -v github.com/nsf/gocode

  - Goland → https://www.jetbrains.com/go/ ($?)
  - Atom → https://atom.io/
  - Sublime → https://www.sublimetext.com/ ($)
- Fontes:
  - https://github.com/tonsky/FiraCode
  - https://www.fsd.it/shop/fonts/pragmatapro/ ($)

Comandos Go

- go version
- go env
- go help
- go fmt
  - ./…
- go run
  - go run <file name>
  - go run *.go
- go build
  - para um executável:
    - gera o arquivo binário
    - informa caso tenham havido erros
    - caso não hajam erros, cria um executável e salva-o no diretório atual
  - para um pacote:
    - gera o arquivo
    - informa caso tenham havido erros
    - descarta o executável
- go install
  - para um executável:
    - faz o build
    - nomeia o arquivo com o nome do diretório atual
    - salva o arquivo binário em $GOPATH/bin
  - para um pacote:
    - faz o build
    - salva o arquivo binário em $GOPATH/pkg
    - cria archive files (arquivo.a), os arquivos pré-compilados utilizados pelos imports
- flags
  - "-race"

Repositórios no GitHub

- Git foi feito pelo Linus Torvalds. O cara que criou o Linux.
- GitHub, GitLab.
- Como funciona?
  - Vá em github.com e crie um repo
  - Crie uma pasta com o mesmo nome no seu $GOPATH
    - $GOPATH/src/github.com/<username>/<repo>
  - Rode “git init” nesta pasta
  - Adicione arquivos, e.g. README.md e .gitignore
  - git add -A
  - git commit -m “here’s some commit message”
  - git remote add origin git@github.com:username/repo.git
  - git push -u origin master
- Comandos:
  - git status
  - git add --all
  - git commit -m "mensagem"
  - git push

Explorando o GitHub

- Clonando um repo
  git clone <repo>
- SSH
  - Mac/Linux: ssh-keygen -t rsa
    - id_rsa: Esta é sua chave privada, que fica no diretório ~/.ssh, e serve para verificar sua chave pública.
    - id_rsa.pub: Esta é sua chave pública, que você pode compartilhar.
  - Windows: Google :)
- git remote
  - git remote get-url origin
  - git remote blablabla ← help
- Truque sujo: apaga tudo e clona denovo. (Não recomendo se o repo tiver 4 GB...)

Compilação cruzada

- GOOS
- GOARCH
- `GOOS=darwin GOARCH=amd64 go build test.go`
- https://godoc.org/runtime#pkg-constants
- git push
- git clone
- go get
- Arquivos: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/19_seu-ambiente-de-desenvolvimento/compilacaocruzada

Pacotes

- Opção 1: uma pasta, vários arquivos.
  - package declaration em todos os arquivos
  - package scope: um elemento de um arquivo é acessível de todos os arquivos
  - imports tem file scope
- Opção 2: separando por packages.
  - pastas diferentes
  - requer imports
  - para usar: package.Função()
- Exportado vs. não-exportado, ou seja, visível vs. não-visível
  - Em Go não utilizamos os termos "público" e "privado" como em outras linguagens
  - É somente questão de capitalização
    - Com maiúscula: exportado, visível fora do package
    - Com minúscula: não exportado, não utilizável fora do package
- Artigo: https://rakyll.org/style-packages/
- Exemplo: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/19_seu-ambiente-de-desenvolvimento/pacotes

20 – Exercícios: Ninja Nível 9

Na prática: exercício #1

- Alem da goroutine principal, crie duas outras goroutines.
- Cada goroutine adicional devem fazer um print separado.
- Utilize waitgroups para fazer com que suas goroutines finalizem antes de o programa terminar.
- Solução:
  - https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/20_exercicios-ninja-9/01_foda/main_foda.go

  - https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/20_exercicios-ninja-9/01_moleza/main_moleza.go

Na prática: exercício #2

- Esse exercício vai reforçar seus conhecimentos sobre conjuntos de métodos.
  - Crie um tipo para um struct chamado "pessoa"
  - Crie um método "falar" para este tipo que tenha um receiver ponteiro (*pessoa)
  - Crie uma interface, "humanos", que seja implementada por tipos com o método "falar"
  - Crie uma função "dizerAlgumaCoisa" cujo parâmetro seja do tipo "humanos" e que chame o método "falar"
  - Demonstre no seu código:
    - Que você pode utilizar um valor do tipo "*pessoa" na função "dizerAlgumaCoisa"
    - Que você não pode utilizar um valor do tipo "pessoa" na função "dizerAlgumaCoisa"
- Se precisar de dicas, veja: https://gobyexample.com/interfaces
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/20_exercicios-ninja-9/02/main.go

Na prática: exercício #3

- Utilizando goroutines, crie um programa incrementador:
  - Tenha uma variável com o valor da contagem
  - Crie um monte de goroutines, onde cada uma deve:
    - Ler o valor do contador
    - Salvar este valor
    - Fazer yield da thread com runtime.Gosched()
    - Incrementar o valor salvo
    - Copiar o novo valor para a variável inicial
  - Utilize WaitGroups para que seu programa não finalize antes de suas goroutines.
  - Demonstre que há uma condição de corrida utilizando a flag -race
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/20_exercicios-ninja-9/03/main.go

Na prática: exercício #4

- Utilize mutex para consertar a condição de corrida do exercício anterior.
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/20_exercicios-ninja-9/04/main.go

Na prática: exercício #5

- Utilize atomic para consertar a condição de corrida do exercício #3.
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/20_exercicios-ninja-9/05/main.go

Na prática: exercício #6

- Crie um programa que demonstra seu OS e ARCH.
- Rode-o com os seguintes comandos:
  - go run
  - go build
  - go install
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/20_exercicios-ninja-9/06/main.go

Na prática: exercício #7

- "If you do not leave your comfort zone, you do not remember the trip" — Brian Chesky
- Faça download e instale: https://obsproject.com/
- Escolha um tópico dos que vimos até agora. Sugestões:
  - Motivos para utilizar Go
  - Instalando Go
  - Configurando as environment variables (e.g. GOPATH)
  - Hello world
  - go commands e.g. go help
  - Variáveis
  - O operador curto de declaração
  - Constantes
  - Loops
    - init, cond, post
    - break
    - continue
  - Funções
  - func (receiver) identifier(params) (returns) { code }
  - Métodos
  - Interfaces
  - Conjuntos de métodos
  - Tipos
    - Conversão?
  - Concorrência vs. paralelismo
  - Goroutines
  - WaitGroups
  - Mutex
- Grave um vídeo onde *você* ensina o tópico em questão.
- Faça upload do vídeo no YouTube.
- Compartilhe o vídeo no Twitter e me marque no tweet (@ellenkorbes).

21 – Canais

Entendendo canais

- Canais são o Jeito Certo® de fazer sincronização e código concorrente.
- Eles nos permitem trasmitir valores entre goroutines.
- Servem pra coordenar, sincronizar, orquestrar, e buffering.
- Na prática:
  - make(chan type, b)
  - <- 42
    - <-c
- Canais bloqueiam:
  - Eles são como corredores em uma corrida de revezamento
  - Eles tem que "passar o bastão" de maneira sincronizada
  - Se um corredor tentar passar o bastão pro próximo, mas o próximo corredor não estiver lá...
  - Ou se um corredor ficar esperando receber o bastão, mas ninguem entregar...
  - ...não dá certo.
- Exemplos:
  - Poe um valor num canal e faz um print. Block.
    - Código acima com goroutine.
    - Ou com buffer. Via de regra: má idéia; é legal em certas situações, mas em geral é melhor sempre passar o bastão de maneira sincronizada.
- Interessante: ref/spec → types
- Código:
  - Block: https://play.golang.org/p/dClS7vQlYE (não roda!)
  - Go routine: https://play.golang.org/p/ZbNCwUuiPi
  - Buffer: https://play.golang.org/p/32vYvCR7qn

  - Buffer block: https://play.golang.org/p/smeW6vigAT
  - Mais buffer: https://play.golang.org/p/Pe2pcboGiA

Canais direcionais & utilizando canais

- Canais podem ser direcionais.
- E isso serve pra...?
- Um send channel e um receive channel são tipos diferentes. Isso permite que os type-checking mechanisms do compilador façam com que não seja possível, por exemplo, escrever num canal de leitura.
- Aos aventureitos: https://stackoverflow.com/questions/13596186/whats-the-point-of-one-way-channels-in-go
- Canais bidirecionals (send & receive)
  - send chan<-
    - error: "invalid operation: <-cs (receive from send-only type chan<- int)"
  - receive <-chan
    - error: "invalid operation: cr <- 42 (send to receive-only type <-chan int)"
- Exemplo: https://play.golang.org/p/TlcSm8bHkW
- A seta sempre aponta para a esquerda.
- Assignment/conversion:
  - de geral para específico
  - de específico para geral não
  - Exemplos:
    - geral pra específico: https://play.golang.org/p/H1uk4YGMBB
    - específico pra específico: https://play.golang.org/p/8JkOnEi7-a
    - específico pra geral: https://play.golang.org/p/4sOKuQRHq7
    - atribuição tipos !=: https://play.golang.org/p/bG7H6l03VQ
- Em funcs podemos especificar:
  - receive channel
    - Parâmetro receive channel: (c <-chan int)
    - No scope dessa função, esse canal só recebe
    - Não podemos fechar um receive channel
  - send channel
    - Parâmetro send channel: (c chan<- int)
    - No scope dessa função, esse canal só envia
    - Podemos fechar um send channel
- Exemplo: passando informação de uma função para outra.
- Código: https://play.golang.org/p/TlcSm8bHkW (replay)

Range e close

- Range:
  - gofunc com for loop com send e close(chan)
  - recebe com range chan
- Código: https://play.golang.org/p/_g5IEjSkh1

Select

- Select é como switch, só que pra canais, e não é sequencial.
- "A select blocks until one of its cases can run, then it executes that case. It chooses one at random if multiple are ready." — https://tour.golang.org/concurrency/5
- Na prática:
  - Exemplo 1:
    - Duas go funcs enviando X/2 numeros cada uma pra um canal
    - For loop X valores, select case <-x
  - Exemplo 2:
    - Func 1 recebe X valores de canal, depois manda qualquer coisa pra chan quit

    - Func 2 for infinito, select: case envia pra canal, case recebe de quit
  - Exemplo 3:
    - Chans par, ímpar, quit
    - Func send manda números pares pra um canal, ímpares pra outro, e fecha/quit
    - Func receive é um select entre os três canais, encerra no quit
    - Problema!
- Go Playground:
  - 1. https://play.golang.org/p/xC3e1wBxgv
  - 2. https://play.golang.org/p/_NZqhBXN-v
  - 3. https://play.golang.org/p/rK8QwsBo0H

A expressão comma ok

- v, ok := <-chan
- Se receber valor: v, true
- Canal fechado, nada, etc.: zero v, false
- Agora vamos resolver o problema do exercício anterior usando comma ok.
- Código:
https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/21_canais/06_exerc%C3%ADcio_anterior/main.go

Convergência

- Observamos convergência quando informação de vários canais é enviada a um número menor de canais.
- Interessante: <- <-
- Na prática, exemplos:
  - 1. Todd:
    - Canais par, ímpar, e converge.
    - Func send manda pares pra um, ímpares pro outro, depois fecha.
    - Func receive cria duas go funcs, cada uma com um for range, enviando dados dos canais par e ímpar pro canal converge. Não esquecer de WGs!
    - Por fim um range retira todas as informações do canal converge.
  - 2. Rob Pike (palestra Go Concurrency Patterns):
    - Func trabalho cria um canal, cria uma go func que manda dados pra esse canal, e retorna o canal. Interessante: time.Duration(rand.Intn(1e3))
    - Func converge toma dois canais, cria um canal novo, e cria duas go funcs com for infinito que passa tudo para o canal novo. Retorna o canal novo.
    - Por fim chamamos canal := converge(trabalho(nome1), trabalho(nome2)) e usamos um for para receber dados do canal var.
- Código: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/21_canais/07

Divergência

- Divergência é o contrário de convergência :)
- Na prática, exemplos:
  - 1. Um stream vira centenas de go funcs que depois convergem.
    - Dois canais.
    - Uma func manda X números ao primeiro canal.
    - Outra func faz um range deste canal, e para cada ítem lança uma go func que poe o retorno de trabalho() no canal dois.
    - Trabalho() é um timer aleatório pra simular workload.
    - Por fim, range canal dois demonstra os valores.
  - 2. Com throttling! Ou seja, com um número máximo de go funcs.
    - Ídem acima, mas a func que lança go funcs é assim:
    - Cria X go funcs, cada uma com um range no primeiro canal que, para cada item, poe o retorno de trabalho() no canal dois.

- Código: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/21_canais/08

Context

- Só pra ter uma idéia geral:
- Se a gente lança 500 goroutines pra fazer uma tarefa, e cancelamos a tarefa no meio do caminho, como fazemos pra matar as goroutines?
- Documentação: https://golang.org/pkg/context/
- Aos aventureiros: https://blog.golang.org/context
- Destaques:
  - ctx := context.Background
  - ctx, cancel = context.WithCancel(context.Background)
  - goroutine: select case <-ctx.Done(): return; default: continua trabalhando.
  - check ctx.Err();
  - Tambem tem WithDeadline/Timeout
- Exemplos (Todd):
  - Analisando:
    - Background: https://play.golang.org/p/cByXyrxXUf
    - WithCancel: https://play.golang.org/p/XOknf0aSpx
    - Função Cancel: https://play.golang.org/p/UzQxxhn_fm
  - Exemplos práticos:
    - func WithCancel: https://play.golang.org/p/Lmbyn7bO7e
    - func WithCancel: https://play.golang.org/p/wvGmvMzIMW
    - func WithDeadline: https://play.golang.org/p/Q6mVdQqYTt
    - func WithTimeout: https://play.golang.org/p/OuES9sP_yX
    - func WithValue: https://play.golang.org/p/8JDCGk1K4P

22 – Exercícios: Ninja Nível 10

Na prática: exercício #1

- Nível 10?! Êita! Parabéns!
- Faça esse código funcionar: https://play.golang.org/p/j-EA6003P0
  - Usando uma função anônima auto-executável
  - Usando buffer
- Solução:
  - 1. https://play.golang.org/p/MNqpJ29FZJ
  - 2. https://play.golang.org/p/Y0Hx6lZc3U

Na prática: exercício #2

- Faça esse código funcionar: https://play.golang.org/p/oB-p3KMiH6
- Solução: https://play.golang.org/p/isnJ8hMMKg

Na prática: exercício #3

- Utilizando este código: https://play.golang.org/p/sfyu4Is3FG
- ...use um for range loop para demonstrar os valores do canal.
- Solução: https://play.golang.org/p/N2N6oN3f0b

Na prática: exercício #4

- Utilizando este código: https://play.golang.org/p/MvL6uamrJP
- ...use um select statement para demonstrar os valores do canal.
- Solução: https://play.golang.org/p/UeJweL3Ola

Na prática: exercício #5

- Utilizando este código: https://play.golang.org/p/YHOMV9NYKK
- ...demonstre o comma ok idiom.
- Solução: https://play.golang.org/p/qh2ywLB5OG

Na prática: exercício #6

- Escreva um programa que coloque 100 números em um canal, retire os números do canal, e demonstre-os.
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/22_exercicios-ninja-10/06/main.go

Na prática: exercício #7

- Crie um programa que lance 10 goroutines onde cada uma envia 10 números a um canal;
- Tire estes números do canal e demonstre-os.
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/22_exercicios-ninja-10/07/main.go

23 – Tratamento de Erros

Entendendo erros

- Para quem já programa em outras linguagens:
  - Em Go não temos exceções. → https://golang.org/doc/faq#exceptions
  - "We believe that coupling exceptions to a control structure, as in the try-catch-finally idiom, results in convoluted code."
  - "Go's multi-value returns make it easy to report an error without overloading the return value. A canonical error type, coupled with Go's other features, makes error handling pleasant but quite different from that in other languages."
  - Aventureiros: https://blog.golang.org/error-handling-and-go
- É interessante criar o hábito de lidar com erros imediatamente, similar a e.g. defer close.
- package builtin, type error interface
- package errors

Verificando erros

- Na minha religião, underscore é pecado.
- Verifique seus erros!
- (Exceção: fmt.Println)
- Na prática:
  - Exemplo 0: fmt.Println
  - Exemplo 1: fmt.Scan(&var)
  - Exemplo 2: os.Create → strings.NewReader → io.Copy
  - Exemplo 3: os.Open → io.ReadAll
- Código: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/23_tratamento-de-erros/02_verificando-erros

Print & log

- Opções:
  - fmt.Println() → stdout
  - log.Println() → timestamp + pode-se determinar onde o erro ficará logado
  - log.Fatalln() → os.Exit(1) sem defer
  - log.Panicln() → println + panic → funcões em defer rodam; dá pra usar recover
  - panic()
- Recomendação: use log.

- Código:
  - 1. fmt.Println
  - 2. log.Println
  - 3. log.SetOutput
  - 4. log.Fatalln
  - 5. log.Panicln
  - 6. panic
- panic: http://godoc.org/builtin#panic
- Código: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/23_tratamento-de-erros/03_print-e-log

Recover

- https://blog.golang.org/defer-panic-and-recover
- https://golang.org/pkg/builtin/#recover
- Exemplo: https://play.golang.org/p/ZocncqtwaK
- Código: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/23_tratamento-de-erros/04_recover/main.go

Erros com informações adicionais

- Para que nossas funções retornem erros customizados, podemos utilizar:
  - return errors.New()
  - return fmt.Errorf() ← tem um errors.New() embutido, olha na fonte!
  - https://golang.org/pkg/builtin/#error
- “Error values in Go aren’t special, they are just values like any other, and so you have the entire language at your disposal.” - Rob Pike
- Código:
  - 1. errors.New
  - 2. var errors.New
  - 3. fmt.Errorf
  - 4. var fmt.Errorf
  - 5. type + method = error interface
- Código: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/23_tratamento-de-erros/05_erros-com-informa%C3%A7%C3%B5es-adicionais

24 – Exercícios: Ninja Nível 11

Na prática: exercício #1

- Utilizando este código: https://play.golang.org/p/3W69TH4nON
- ...remova o underscore e verifique e lide com o erro de maneira apropriada.
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/24_exercicios-ninja-11/01/main.go

Na prática: exercício #2

- Utilizando este código: https://play.golang.org/p/9a1IAWy5E6
- ...crie uma mensagem de erro customizada utilizando fmt.Errorf().
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/24_exercicios-ninja-11/02/main.go

Na prática: exercício #3

- Crie um struct "erroEspecial" que implemente a interface builtin.error.
- Crie uma função que tenha um valor do tipo error como parâmetro.
- Crie um valor do tipo "erroEspecial" e passe-o para a função da instrução anterior.

- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/24_exercicios-ninja-11/03/main.go

Na prática: exercício #4

- Utilizando este código: https://play.golang.org/p/wlEM1tgfQD
- ...use o struct sqrt.Error como valor do tipo erro.
- Solução: https://github.com/ellenkorbes/aprendago/blob/master/c%C3%B3digo/24_exercicios-ninja-11/04/main.go

Na prática: exercício #5

- Nos capítulos seguintes, uma das coisas que veremos é testes.
- Para testar sua habilidade de se virar por conta própria... desafio:
  - Utilizando as seguintes fontes: https://godoc.org/testing & http://www.golang-book.com/books/intro/12
  - Tente descobrir por conta própria como funcionam testes em Go.
  - Pode usar tradutor automático, pode rodar código na sua máquina, pode procurar no Google. Vale tudo.
  - O exercício é: crie um teste simples de uma função ou método ou pedaço qualquer de código.

25 – Documentação

Introdução

- Antes de escrever documentação, vamos ver como lê-la. Temos algumas possibilidades:
  - golang.org → documentação da standard library
  - godoc.org → documentação da standard library e outros
  - go doc → comando para ler documentação na linha de comando
  - godoc → idem acima, para pode-se servir a documentação local via http

go doc

- go help doc
- go doc demonstra a documentação de um package, const, func, type, var, método, etc.
- go doc aceita zero, um, ou dois argumentos:
  - zero: demonstra a documentação do package do diretório atual
  - um: toma argumentos nos padrões abaixo
    - go doc <pkg>
    - go doc <sym>[.<method>]
    - go doc [<pkg>.]<sym>[.<method>]
    - go doc [<pkg>.][<sym>.]<method>
  - dois: o primeiro argumento deve ser o nome do package
    - go doc <pkg> <sym>[.<method>]

godoc
- godoc extrai e gera documentação de programas em Go. Funciona de duas maneiras:
  - Sem o flag http é um comando normal, mostra a documentação no stdout e é isso aí. Pode conter o flag src, que mostra o código fonte.
  - Com o flag http roda um servidor web local e mostra a documentação como página web.
- Exemplo: godoc -http=:8080 → http://localhost:8080/

godoc.org

- Documentação da standard library e outros
- Como colocar a documentação do seu package no godoc.org
- refresh, delete

Escrevendo documentação

- Documentação é uma parte extremamente importante de fazer com que software seja acessível e sustentável.
- Documentação deve ser bem escrita e correta, mas tambem fácil de escrever e manter.
- Deve ser acoplada com o código e evoluir junto com este. Quanto mais fácil for para os programadores criarem boa documentação... melhor fica pra todos os envolvidos.
- godoc:
  - Analisa código fonte em Go, incluindo comentários, e gera documentação em HTML ou texto
  - O resultado é uma documentação firmemente atrelada ao código que documenta.
  - Por exemplo, na interface web de godoc pode-se navegar da documentação à implementação de um código com apenas um clique.
  - https://blog.golang.org/godoc-documenting-go-code
- Na prática:
  - Para documentar um tipo, uma variável, uma constante, ou um pacote, escreva um comentário imediatamente antes de sua declaração, sem linhas em branco
  - Comece a frase com o nome do elemento. No caso de pacotes, a primeira linha aparece no "package list."
  - Caso esteja escrevendo bastante documentação, utilize um arquivo doc.go. Exemplo: package fmt.
- A melhor parte dessa abordagem minimalista é que é super fácil de usar. Como resultado, muita coisa em Go, incluindo toda a standard library, já segue estas convenções.
- Outro exemplo: errors package.
- Código: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/25_escrevendo-documentacao

26 – Exercícios: Ninja Nível 12

Na prática: exercício #1

- Crie um package "cachorro".
  - Este package deverá exportar uma função Idade, que toma como parâmetro um número de anos e retorna a idade equivalente em anos caninos. (1 ano humano → 7 anos caninos)
  - Documente seu código com comentários, e utilize a função Idade na sua função main.
- Rode seu programa para verificar se ele funciona.
- Rode um local server com godoc e leia sua documentação.
- Solução: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/26_exercicios-ninja-12/01_cachorro

Na prática: exercício #2

- Coloque seu código no GitHub.
- Faça sua documentação aparecer em godoc.org, e tire um screenshot.
- Delete seu código do GitHub.
- Faça um refresh em godoc.org e veja se seu código sumiu.
- Compartilhe seu exercício aqui: https://github.com/ellenkorbes/aprendago/issues/79

Na prática: exercício #3

- Use godoc na linha de comando para ver a documentação sobre:
  - fmt
  - fmt Print
  - strings
  - strconv

27 – Testes & Benchmarking

Introdução

- Testes devem:
  - ficar num arquivo cuja terminação seja _test.go
  - ficar na mesma package que o código a ser testado
  - ficar em funções com nome "func TestNome(*testing.T)"
- Para rodar os testes:
  - go test
  - go test -v
- Para falhas, utilizamos t.Error(), onde a maneira idiomática é algo do tipo "expected: x. got: y."
- Código: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/27_testes-e-benchmarking/01_introducao

Testes em tabela

- Podemos escrever testes em série para testar variedades de situações.
- Exemplo:
  - struct test, fields: data []int, answer int
  - tests := []test{[]int{}, int}
  - range tests

Testes como exemplos

- Outra maneira é fazer testes como exemplos.
- Estes exemplos são os mesmos que aparecem na documentação.
- Para exemplos o formato é "func ExampleFuncao()"
- Deve haver um comentário "// Output: resultado", que é o que será testado
- Para visualizar seu exemplo na documentação, fazemos o de sempre:
  - godoc -http :8080
- Tanto para testes quanto para exemplos podemos utilizar: go test ./...
- Mais: https://blog.golang.org/examples

go fmt, govet e golint

- gofmt: formata o código
- go vet: correctness → procura constructs suspeitos
- golint: suggestions → procura coding style ruim

Benchmark

- Benchmarks nos permitem testar a velocidade ou performance do nosso código.
- Na prática:
  - Arquivo: _test.go
  - BET: Testes, Exemplos e...
  - func BenchmarkFunc (b *testing.B) { for i := 0; i < b.N; i++ { ... } }
  - go test -bench . ← todos
  - go test -bench Func ← somente Func
- go help testflag
- Código: https://github.com/ellenkorbes/aprendago/tree/master/c%C3%B3digo/27_testes-e-benchmarking/benchtest

Cobertura

- "Cobertura" em se tratando de testes se refere a quanto do seu código, percentualmente, está sendo testado. (E antes que alguem fique neurótico querendo 100%: em geral, 70-80% tá ótimo.)
- A flag -cover faz a análise de cobertura do nosso código.
- Podemos utilizar a flag -coverprofile <arquivo> para salvar a análise de cobertura em um arquivo.

- Na prática:
  - go test -cover
  - go test -coverprofile c.out
  - go tool cover -html=c.out ← abre no browser
  - go tool cover -h ← para mais detalhes