0º ATIVIDADE de CES-27 / 2018

CTA - ITA - IEC

Prof Juliana e Prof Vitor

Aluno: Gustavo Nahum Alvarez Ferreira

Loops and Functions

```
package main

import (
	"fmt"
	"math"
)

func Sqrt1(x float64) float64 {
	z := 1.0
	for v:=0; v<10; v++ {
		z -= (z*z - x) / (2*z)
	}
	return z
}

func Sqrt2(x float64) (float64, int) {
	var z float64 = 1.0
	var z1, z2 float64
	var delta float64 = math.Abs(z - x)
	var cnt int = 0

	for delta > 0.00001 {
		z1 = z
		z -= (z*z - x) / (2*z)
		z2 = z
		delta = math.Abs(z1 - z2)
		cnt++
	}
	return z, cnt
}

func Sqrt3(x float64) (float64, int) {
	var z float64 = x
	var z1, z2 float64
	var delta float64 = 0.1
	var cnt int = 0
```

```
	for delta > 0.00001 {
		z1 = z
		z -= (z*z - x) / (2*z)
		z2 = z
		delta = math.Abs(z1 - z2)
		cnt++
	}
	return z, cnt
}

func Sqrt4(x float64) (float64, int) {
	var z float64 = x/2
	var z1, z2 float64
	var delta float64 = 0.1
	var cnt int = 0

	for delta > 0.00001 {
		z1 = z
		z -= (z*z - x) / (2*z)
		z2 = z
		delta = math.Abs(z1 - z2)
		cnt++
	}
	return z, cnt
}


func main() {
	z1 := Sqrt1(2)
	fmt.Printf("Método 1 (10 iterações): %v\n", z1)
	fmt.Printf("Módulo da diferença entre Sqrt1 e math.Sqrt: %v\n\n", math.Abs(z1 -
math.Sqrt(2)))

	z2, cnt2 := Sqrt2(2)
	fmt.Printf("Método 2 (%v iterações): %v\n", cnt2, z2)
	fmt.Printf("Módulo da diferença entre Sqrt2 e math.Sqrt: %v\n\n", math.Abs(z2 -
math.Sqrt(2)))

	z3, cnt3 := Sqrt3(2)
	fmt.Printf("Método 3 (%v iterações): %v\n", cnt3, z3)
	fmt.Printf("Módulo da diferença entre Sqrt3 e math.Sqrt: %v\n\n", math.Abs(z3 -
math.Sqrt(2)))

	z4, cnt4 := Sqrt4(2)
	fmt.Printf("Método 4 (%v iterações): %v\n", cnt4, z3)
	fmt.Printf("Módulo da diferença entre Sqrt4 e math.Sqrt: %v\n\n", math.Abs(z4 -
math.Sqrt(2)))
}
```

Maps

```
package main

import (
	"golang.org/x/tour/wc"
	"strings"
)

func WordCount(s string) map[string]int {
	allStrings := strings.Fields(s)
	m := make(map[string]int)
	for _, word := range allStrings {
		m[word]++
	}
	return m
}

func main() {
	wc.Test(WordCount)
}
```

Fibonacci Closure

```
package main

import "fmt"

// fibonacci is a function that returns
// a function that returns an int.
func fibonacci() func(int) int {
	var atual, ant, antant int

	return func(n int) int {
		if n == 0 {
			return 0
		}
		if n == 1 {
			return 1
		}
		antant = 0
		ant = 1
		atual = 1
		for i := 2; i < n; i++ {
			antant = ant
			ant = atual
			atual = ant + antant
		}
		return atual
```

```
	}
}

func main() {
	f := fibonacci()
	for i := 0; i < 10; i++ {
		fmt.Println(f(i))
	}
}
```

Stringers

```
package main

import "fmt"

type IPAddr [4]byte

func (ip IPAddr) String() string {
	return fmt.Sprintf("%v.%v.%v.%v.", ip[0], ip[1], ip[2], ip[3])
}

func main() {
	hosts := map[string]IPAddr{
		"loopback":  {127, 0, 0, 1},
		"googleDNS": {8, 8, 8, 8},
	}
	for name, ip := range hosts {
		fmt.Printf("%v: %v\n", name, ip)
	}
}
```

Errors

```
package main

import (
	"fmt"
)

type ErrNegativeSqrt float64

func (e ErrNegativeSqrt) Error() string {
	return fmt.Sprint("cannot Sqrt negative number: ", float64(e))
}
```

```
func Sqrt(x float64) (float64, error) {
	if x >= 0.0 {
		z := 1.0
		for v:=0; v<10; v++ {
			z -= (z*z - x) / (2*z)
		}
		return z, nil
	}
	return 0, ErrNegativeSqrt(x)
}

func main() {
	var x float64
	x = -2
	if v, err := Sqrt(x); err == nil {
		fmt.Println(v)
	} else {
		fmt.Println(err)
	}
}
```

Equivalent Binary Trees

```
package main

import (
	"golang.org/x/tour/tree"
	"fmt"
)

func Walk(t *tree.Tree, ch chan int) {
	if t.Left != nil {
		Walk(t.Left, ch)
	}
	ch <- t.Value
	if t.Right != nil {
		Walk(t.Right, ch)
	}
}

func Same(t1, t2 *tree.Tree) bool {
	ch1 := make(chan int, 10)
	ch2 := make(chan int, 10)
	go Walk(t1, ch1)
	go Walk(t2, ch2)
	for i := 0; i < 10; i++ {
		if <-ch1 != <-ch2 {
			return false
		}
	}
```

```
	return true
}

func main() {
	fmt.Println(Same(tree.New(1), tree.New(1)))
	fmt.Println(Same(tree.New(1), tree.New(2)))
}
```

Web Crawler

```
package main

import (
	"fmt"
	"sync"
)

type Fetcher interface {
	Fetch(url string) (body string, urls []string, err error)
}

type Cache struct {
	visited map[string]bool
	mux sync.Mutex
}

func Crawl(url string, depth int, fetcher Fetcher, cache Cache) {
	if depth <= 0 {
		return
	}
	cache.mux.Lock()
	if cache.visited[url] {
		cache.mux.Unlock()
		fmt.Printf("already fetched: %s\n", url)
		return
	}
	cache.visited[url] = true
	cache.mux.Unlock()

	body, urls, err := fetcher.Fetch(url)
	if err != nil {
		fmt.Println(err)
		return
	}
	fmt.Printf("found: %s %q\n", url, body)
	for _, u := range urls {
		Crawl(u, depth-1, fetcher, cache)
	}
	return
}
```

```
func main() {
	Crawl("https://golang.org/", 4, fetcher, Cache{visited: make(map[string] bool)})
}

type fakeFetcher map[string]*fakeResult

type fakeResult struct {
	body string
	urls []string
}

func (f fakeFetcher) Fetch(url string) (string, []string, error) {
	if res, ok := f[url]; ok {
		return res.body, res.urls, nil
	}
	return "", nil, fmt.Errorf("not found: %s", url)
}

var fetcher = fakeFetcher{
	"https://golang.org/": &fakeResult{
		"The Go Programming Language",
		[]string{
			"https://golang.org/pkg/",
			"https://golang.org/cmd/",
		},
	},
	"https://golang.org/pkg/": &fakeResult{
		"Packages",
		[]string{
			"https://golang.org/",
			"https://golang.org/cmd/",
			"https://golang.org/pkg/fmt/",
			"https://golang.org/pkg/os/",
		},
	},
	"https://golang.org/pkg/fmt/": &fakeResult{
		"Package fmt",
		[]string{
			"https://golang.org/",
			"https://golang.org/pkg/",
		},
	},
	"https://golang.org/pkg/os/": &fakeResult{
		"Package os",
		[]string{
			"https://golang.org/",
			"https://golang.org/pkg/",
		},
	},
}
```