# Ex1

- Entrega 19 fev em 23:59
- Pontos 2
- Perguntas 1
- Disponível 19 fev em 12:30 - 19 fev em 23:59 11 horas e 29 minutos
- Limite de tempo Nenhum
- Tentativas permitidas Sem limite

# Instruções

Para realizar esta tarefa, você:

- pode consultar a Bibliografia;

- não pode consultar colegas;

- deve observar o prazo máximo para responder, normalmente durante a aula.

- dê uma resposta objetiva, embora não haja limite mínimo ou máximo de conteúdo para sua resposta.

Este teste foi travado 19 fev em 23:59.

## Histórico de tentativas

| | Tentativa | Tempo | Pontuação |
|---|---|---|---|
| **MAIS RECENTE** | Tentativa 1 | 6 minutos | 2 de 2 |

Pontuação desta tentativa: 2 de 2
Enviado 19 fev em 16:24
Esta tentativa levou 6 minutos.

Pergunta 1
2 / 2 pts

Escolha uma Linguagem de Programação, criada entre 1950 e 2020, apontando sua característica marcante, preferencialmente que seja inovadora à época de seu "lançamento".

Sua resposta deve conter 2 componentes:

1) A Linguagem de Programação;

2) Característica [inovadora] marcante.

Sua Resposta:

1) Linguagem de programação Go;

2)

O Go, ao contrário do Java, não utiliza blocos try/catch/finally para lidar com exceções. Em vez disso, ele adota um modelo de tratamento de erros mais rigoroso, empregando funções panic e recover, juntamente com a instrução defer. Uma aplicação típica do panic é abortar a execução quando uma função retorna um erro que não pode ser manipulado. Por exemplo, ao executar o programa abaixo, ele entra em estado de pânico, exibe uma mensagem de erro, rastreia a goroutine e termina com um status diferente de zero:

```go
package main

import "os"

func main() {
    panic("um problema")

    _, err := os.Create("/tmp/arquivo")
    if err != nil {
        panic(err)
    }
}
```

A função recover é usada para recuperar o controle de uma goroutine em pânico, sendo útil apenas dentro de funções diferidas. Enquanto uma chamada para recover durante a execução normal retornará nulo, se a goroutine estiver em pânico, recuperará o valor associado ao pânico e retomará a execução normal. A instrução defer adia a execução de uma função até o final da função circundante. Isso é comumente utilizado para simplificar ações de limpeza em funções. Veja o exemplo a seguir:

```go
package main

import (
    "fmt"
```

```
    "os"
)

func main() {
    f := criarArquivo("/tmp/defer.txt")
    defer fecharArquivo(f)
    escreverArquivo(f)
}

func criarArquivo(p string) *os.File {
    fmt.Println("criando")
    f, err := os.Create(p)
    if err != nil {
        panic(err)
    }
    return f
}

func escreverArquivo(f *os.File) {
    fmt.Println("escrevendo")
    fmt.Fprintln(f, "dados")
}

func fecharArquivo(f *os.File) {
    fmt.Println("fechando")
    f.Close()
}
```

Pontuação do teste: 2 de 2