Como Projetar Funções

Prof. Helio H. L. C. Monte-Alto

”Cristo padeceu uma vez pelos pecados, o justo pelos injustos, para levar-nos a Deus”
”Porque Deus amou o mundo de tal maneira que deu o seu Filho unigênito, para que todo aquele que nele crê não pereça, mas tenha a vida eterna.”
”Crê no Senhor Jesus Cristo e serás salvo”
(1 Pedro 3:18, João 3:16, Atos 16:31)

Pra que tudo isso?

- Código legível
- Propósito bem definido
  - “Faça apenas uma coisa”
- Entradas e saídas bem definidas
- Desenvolvimento Dirigido a Testes
- Reuso de código
- Dados e informações definidos e relacionados

Mas é chato...

… para problemas simples é mesmo…

… mas quando a coisa complicar, vai ser bem útil...

Mas é chato...

- Imagine também que você, ou outra pessoa, queira modificar o código depois…
- … ou imagine que você tem que fazer manutenção em um código mal documentado e sem testes...

Mas é chato...

- A ideia do Projeto Sistemático de Programas aqui apresentada é utilizada em diversas universidades famosas, como MIT e University of British Columbia

- Muitos que experimentaram o curso afirmam que se tornaram programadores melhores após concluí-lo

Receita

1. Assinatura, propósito, cabeçalho (stub)
2. Exemplos (usando *check-expect*)
3. Template
4. Corpo do código (lógica da solução)
5. Teste e depuração

Receita

1. **Assinatura, propósito, cabeçalho (stub)**
2. Exemplos (usando *check-expect*)
3. Template
4. Corpo do código (lógica da solução)
5. Teste e depuração

Receita

- Assinatura:
  - O que a função consome (entrada) e o que produz (saída)
- Propósito
  - Em uma linha, o propósito/objetivo da função

- Exemplo: Encontrar dobro de um número

```
; Numero -> Numero
; Retorna o dobro do valor passado
```

Receita

- Cabeçalho (*stub*)
  - Um **protótipo bobo** da função
  - É a versão do programa que vai **falhar nos testes**!!

```
; Numero -> Numero
; Retorna o dobro do valor passado

(define (dobro n) 0)
```

Receita

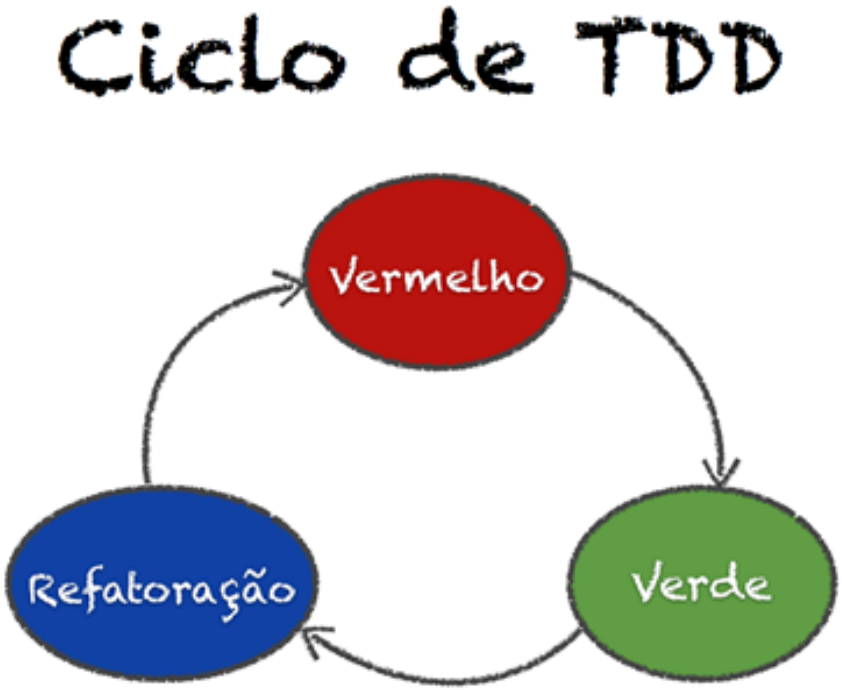
1. Assinatura, propósito, cabeçalho (stub)
2. **Exemplos (testes usando *check-expect*)**
3. Template
4. Corpo do código (lógica da solução)
5. Teste e depuração

Receita

- Exemplos / testes:
  - Baseados na ideia de *Test Driven Development* (TDD) - Desenvolvimento Dirigido a Testes
  - Criar **testes unitários** antes da lógica do programa
  - Os testes inicialmente devem **falhar**, pois irão executar o **cabeçalho (protótipo bobo)**

Receita

- Exemplos / testes:
  - TDD:



Receita

- Exemplos / testes:
  - No Racket, podemos usar a função

```
(check-expect expr expected-expr)
;;Checa se o valor da expressão expr
 é igual ao valor da expressão
 expected-expr
```

Receita

- Exemplos / testes:
  - No Racket, podemos usar a função

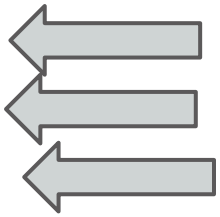
```
(check-expect expr expected-expr)
```

Atenção: Coloque a linguagem *Beginning Student!!*

```
; Numero -> Numero
; Retorna o dobro do valor passado

;(define (dobro n) 0)

; Exemplos
(check-expect (dobro 2) 4)
(check-expect (dobro 3) 6)
(check-expect (dobro -3) -6)
```



;; verifica se dobro de 2 é 4

;; verifica se dobro de 3 é 6

;; verifica se dobro de -3 é -6

Receita

- Exemplos / testes:
  - Testes falhando com apenas o cabeçalho (stub):

```
Ran 3 tests.
0 tests passed.
No signature violations.

Check failures:
                    Actual value 0 differs from 4, the expected value.
in exemplo_dobro.rkt, line 8, column 0
                    Actual value 0 differs from 6, the expected value.
in exemplo_dobro.rkt, line 9, column 0
                    Actual value 0 differs from -6, the expected value.
in exemplo_dobro.rkt, line 10, column 0
```

Receita

- Exemplos / testes:
  - Quantos exemplos são suficientes??
    - Pensar em todos os tipos de casos possíveis
    - Verificar Cobertura de Código
      - Mostrarei em um exemplo

Receita

1. Assinatura, propósito, cabeçalho (stub)
2. Exemplos (testes usando *check-expect*)
3. **Template**
4. Corpo do código (lógica da solução)
5. Teste e depuração

Receita

- Template
  - Esqueleto do código baseado nos tipos de dados (assinatura)
  - Permite reuso de código

- Cabeçalho (stub) x Template
  - O cabeçalho é a **versão da função que falha**
  - O template não faz nada: é apenas um **modelo**

Receita

- Template para dados atômicos:

```
; Numero -> Numero
; Retorna o dobro do valor passado

;(define (dobro n) 0)

; Exemplos
(check-expect (dobro 2) 4)
(check-expect (dobro 3) 6)
(check-expect (dobro -3) -6)

; Template
;(define (fn-for-number n)
;  (... n))
```

Receita

- Template para dados enumerados

```
;; EstadoSemaforo -> EstadoSemaforo
;; Recebe uma cor/estado do semáforo e retorna a próxima cor/estado

(define (proximo-estado-semaforo cor) "red")  ;esse é o cabeçalho (stub) / prototipo bobo

(check-expect (proximo-estado-semaforo "red") "green")
(check-expect (proximo-estado-semaforo "green") "yellow")
(check-expect (proximo-estado-semaforo "yellow") "red")
```

```
;Template usado
(define (fn-for-estado-semaforo ls)
  (cond [(string=? "red" ls) (...)]
        [(string=? "yellow" ls) (...)]
        [(string=? "green" ls) (...)]))
```

Receita

- Template
  - A ideia é você ter um arquivo com *templates* comuns e apenas copiá-los para começar a programar
  - Você vai definindo seus próprios *templates* e pode **reusá-los** mais tarde
  - Ver slides Como Projetar Dados
  - Ver guias Data Driven Templates e How To D Data

Receita

1. Assinatura, propósito, cabeçalho (stub)
2. Exemplos (testes usando *check-expect*)
3. Template
4. **Corpo do código (lógica da solução)**
5. Teste e depuração

Receita

- Corpo do código (lógica)
  - Até agora definimos:
    - Tipos de dados consumidos e produzidos (**assinatura**)
    - **Exemplos / testes** unitários
    - Um protótipo bobo (**cabeçalho**) que **falha nos testes**
  - Agora temos que escrever a lógica de uma função que **passa nos testes**
  - Utilizamos um **template** para termos ideia do corpo da função

Receita

- Corpo do código (lógica)

```
; Numero -> Numero
; Retorna o dobro do valor passado

;(define (dobro n) 0)

; Exemplos
(check-expect (dobro 2) 4)
(check-expect (dobro 3) 6)
(check-expect (dobro -3) -6)

; Template
;(define (fn-for-number n)
;  (... n))

; Corpo
(define (dobro n)
  (* n 2))
```

Receita

- Corpo do código (lógica)
  - Testes **passando**

```
Linguagem: Beginning Student [costumizado]; memory limit: 128 MB.
Teachpack: testing.rkt.
All 3 tests passed!
```

Sobre trabalhos e exercícios

- O uso adequado dos métodos de projeto apresentados será considerado na avaliação dos trabalhos
- Não esqueçam de consultar os slides e o material na página para tirar dúvidas
- Em caso de dúvidas, me procurem na minha sala no segundo andar do bloco 2