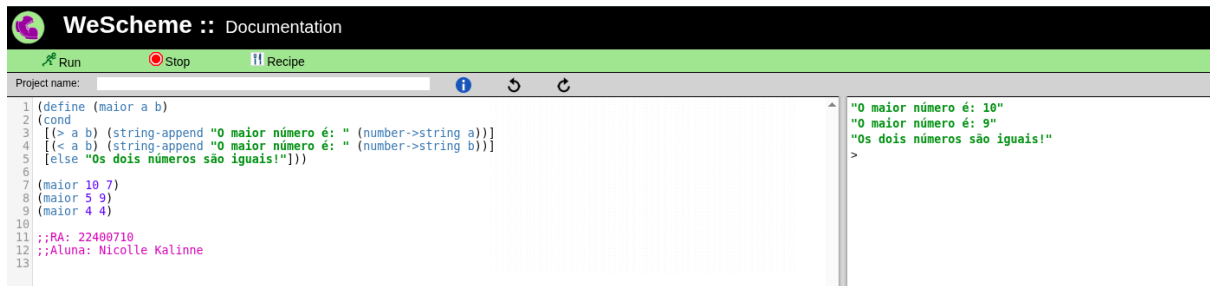


Aluno(a): Nicolle Kalinne Sousa Lima RA: 22400710

Livro Base

1)



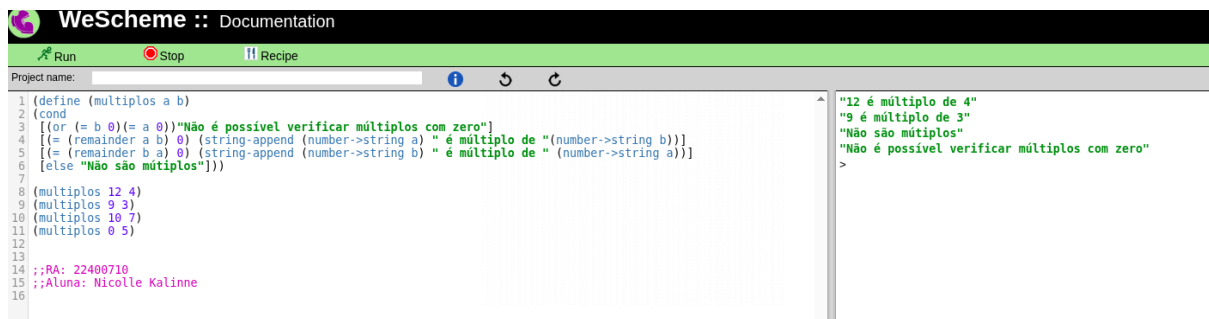
The screenshot shows the WeScheme Documentation interface. The top bar has a green background with a 'Run' button (a green play icon), a 'Stop' button (a red circle with a white dot), and a 'Recipe' button (a blue book icon). Below the bar, the 'Project name:' field is empty. The main editor area contains Scheme code for a function named 'maior'. The code defines a function that takes two arguments, 'a' and 'b', and returns a string indicating the larger number or if they are equal. The code is as follows:

```
1 (define (maior a b)
2   (cond
3     [(> a b) (string-append "O maior número é: " (number->string a))]
4     [(< a b) (string-append "O maior número é: " (number->string b))]
5     [else "Os dois números são iguais!"]])
6
7 (maior 10 7)
8 (maior 5 9)
9 (maior 4 4)
10
11 ;;RA: 22400710
12 ;;Aluna: Nicolle Kalinne
13
```

The output area on the right shows the results of the function calls:

```
"O maior número é: 10"
"O maior número é: 9"
"Os dois números são iguais!"
>
```

2)



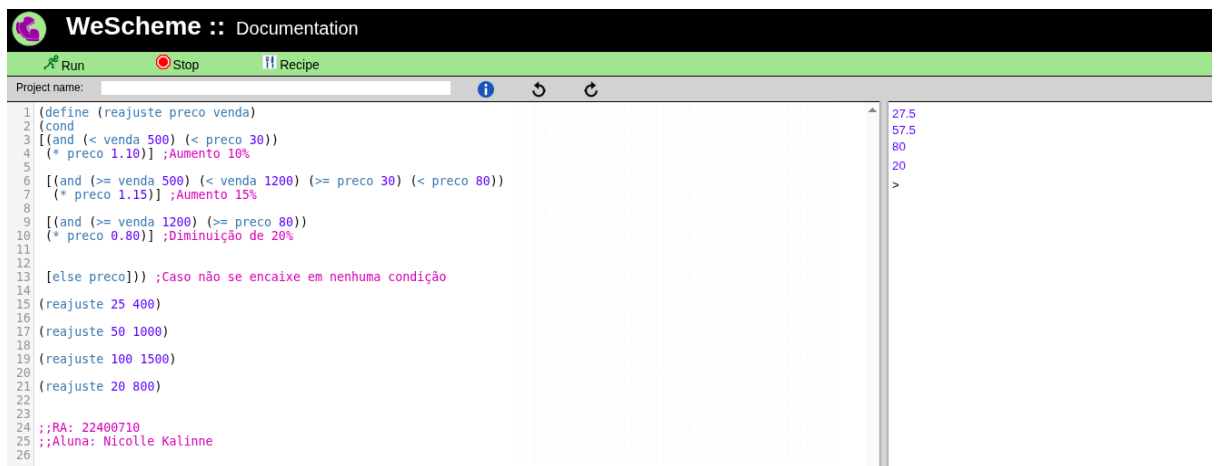
The screenshot shows the WeScheme Documentation interface. The top bar has a green background with a 'Run' button (a green play icon), a 'Stop' button (a red circle with a white dot), and a 'Recipe' button (a blue book icon). Below the bar, the 'Project name:' field is empty. The main editor area contains Scheme code for a function named 'multiplos'. The code defines a function that takes two arguments, 'a' and 'b', and returns a string indicating if 'a' is a multiple of 'b' or if it's not possible to verify. The code is as follows:

```
1 (define (multiplos a b)
2   (cond
3     [(or (= b 0) (= a 0)) "Não é possível verificar múltiplos com zero"]
4     [(= (remainder a b) 0) (string-append (number->string a) " é múltiplo de " (number->string b))]
5     [(= (remainder b a) 0) (string-append (number->string b) " é múltiplo de " (number->string a))]
6     [else "Não são múltiplos"]])
7
8 (multiplos 12 4)
9 (multiplos 9 3)
10 (multiplos 10 7)
11 (multiplos 0 5)
12
13
14 ;;RA: 22400710
15 ;;Aluna: Nicolle Kalinne
16
```

The output area on the right shows the results of the function calls:

```
"12 é múltiplo de 4"
"9 é múltiplo de 3"
"Não são múltiplos"
"Não é possível verificar múltiplos com zero"
>
```

3)



The screenshot shows the WeScheme Documentation interface. The top bar has a green background with a 'Run' button (a green play icon), a 'Stop' button (a red circle with a white dot), and a 'Recipe' button (a blue book icon). Below the bar, the 'Project name:' field is empty. The main editor area contains Scheme code for a function named 'reajuste'. The code defines a function that takes two arguments, 'preco' and 'venda', and returns a string indicating the adjusted price based on certain conditions. The code is as follows:

```
1 (define (reajuste preco venda)
2   (cond
3     [(and (< venda 500) (< preco 30))
4      (* preco 1.10)] ;Aumento 10%
5
6     [(and (>= venda 500) (< venda 1200) (>= preco 30) (< preco 80))
7      (* preco 1.15)] ;Aumento 15%
8
9     [(and (>= venda 1200) (>= preco 80))
10      (* preco 0.80)] ;Diminuição de 20%
11
12     [else preco]] ;Caso não se encaixe em nenhuma condição
13
14 (reajuste 25 400)
15
16 (reajuste 50 1000)
17
18 (reajuste 100 1500)
19
20 (reajuste 20 800)
21
22
23
24 ;;RA: 22400710
25 ;;Aluna: Nicolle Kalinne
26
```

The output area on the right shows the results of the function calls:

```
27.5
57.5
80
20
>
```

4)

The screenshot shows the WeScheme web interface. The editor contains a Scheme function `converter-tempo` that takes seconds and returns a list of hours, minutes, and seconds. The code is as follows:

```
1 (define (converter-tempo segundos)
2   (let* ([horas (quotient segundos 3600)]
3         [resto-horas (remainder segundos 3600)]
4         [minutos (quotient resto-horas 60)]
5         [segundos-finais (remainder resto-horas 60)])
6     (list horas minutos segundos-finais)))
7
8 (converter-tempo 3670)
9
10 (converter-tempo 7322)
11
12 (converter-tempo 59)
13 ;;RA: 22400710
14 ;;Aluna: Nicolle Kalinne
15
```

The right-hand output pane shows the results of the function calls:

```
(list 1 1 10)
(list 2 2 2)
(list 0 0 59)
>
```

5)

The screenshot shows the WeScheme web interface. The editor contains a Scheme function `contar` that counts down from a given number `n` to 0, displaying each number on a new line. The code is as follows:

```
1 (define (contar n)
2   (cond
3     [(= n 0) (display "Encerrado")]
4     [else
5      (begin
6        (display n) (newline)
7        (contar (- n 1)))]))
8
9 (contar 10)
10
11 ;;RA: 22400710
12 ;;Aluna: Nicolle Kalinne
13
```

The right-hand output pane shows the output of the function call:

```
10
9
8
7
6
5
4
3
2
1
Encerrado
>
```

6)

The screenshot shows the WeScheme web interface. The editor contains a Scheme function `idade-em-anos` that calculates the age in years, months, and days from a given number of days. The code is as follows:

```
1 (define (idade-em-anos idade-dias)
2   (let* ([anos (quotient idade-dias 365)]
3         [resto-anos (remainder idade-dias 365)]
4         [meses (quotient resto-anos 30)]
5         [dias (remainder resto-anos 30)])
6     (list anos meses dias)))
7
8 (idade-em-anos 400)
9
10 (idade-em-anos 800)
11
12 (idade-em-anos 59)
13
14 ;;RA: 22400710
15 ;;Aluna: Nicolle Kalinne
16
```

The right-hand output pane shows the results of the function calls:

```
(list 1 1 5)
(list 2 2 10)
(list 0 1 29)
>
```

7)

```
WeScheme :: Documentation
Run Stop Recipe
Project name:
1 (define (operacao a b op)
2   (cond
3     ((eq? op '+) (+ a b))
4     ((eq? op '-') (- a b))
5     ((eq? op '*') (* a b))
6     ((eq? op '/') (if (= b 0) "Erro: divisão por zero" (/ a b)))
7     (else "Operador inválido")))
8
9 (operacao 10 5 '+)
10
11 (operacao 2 4 '-')
12
13 (operacao 5 7 '*')
14
15 (operacao 4 2 '/')
16
17 ;;RA: 22400710
18 ;;Aluna: Nicolle Kalinne
19
```

8)

```
WeScheme :: Documentation
Run Stop Recipe
Project name:
1 (define (inss salario)
2   (cond
3     ((<= salario 1200) (list (* salario 0.02) (- salario (* salario 0.02))))
4     ((<= salario 2500) (list (* salario 0.05) (- salario (* salario 0.05))))
5     (else (list (* salario 0.08) (- salario (* salario 0.08)))))
6
7 (inss 1200)
8
9 (inss 800)
10
11
12 ;;RA: 22400710
13 ;;Aluna: Nicolle Kalinne
14
```

9)

```
WeScheme :: Documentation
Run Stop Recipe
Project name:
1 (define (fatorial n)
2   (if (= n 0)
3     1
4     (* n (fatorial (- n 1)))))
5
6 (fatorial 5)
7
8 (fatorial 8)
9
10 (fatorial 2)
11 ;;RA: 22400710
12 ;;Aluna: Nicolle Kalinne
13
```

2.2.1:

- a) `(+ (* 1.2 (- 2 (/ 1 3))) -8.7)`
- b) `(/ (+ (/ 2 3) (/ 4 9)) (- (/ 5 11) (/ 4 3)))`
- c) `(+ 1 (/ 1 (+ 2 (/ 1 (+ 1 (/ 1 2))))))`
- d) `(* 1 -2 3 -4 5 -6 7)`

2.2.3:

- a) `(car . cdr)`
- b) `(this (is silly))`
- c) `(is this silly?)`
- d) `(+ 2 3)`
- e) `(+ 2 3)`
- f) `+`
- g) `(2 3)`
- h) `#<procedure:cons>`
- i) `cons`
- j) `(quote cons)`
- k) `quote`
- l) `5`
- m) erro
- n) `5`
- o) `5`

2.2.4:

Ele retorna: a,b,c,d nesta ordem

2.3.1:

Ele dá como resultado 12.

Cria uma lista com as quatro funções aritméticas:

`(list + - * /)`

Pega tudo menos o primeiro elemento:

`(- * /)`

Agora pega o primeiro elemento:

`(car ...)`

resultado:

`(- 17 5)`

2.4.1:

- a) `(let ((triple-a (* 3 a)))
 (+ (- triple-a b)
 (+ triple-a b)))`
- b) `(let ((my-list (list a b c)))
 (cons (car my-list)
 (cdr my-list)))`

2.4.2:

x externo = 9

x interno = (\div x externo 3) = 3

(+ x interno x interno) = 6

multiplicar pelo x externo: $9 * 6 = 54$

2.5.1:

a) 'a