Logic, Discrete Mathematics and Computer Programming

Training problems for M3 2018 term 1

Ted Szylowiec
tedszy@gmail.com

You can find SICP (*Structure and Interpretation of Computer Programs*) online here:

https://sarabander.github.io/sicp/

1 Try out Racket code

1. What is an *S-expression* (or simply *expression* for short)?

2. Who got the idea that it is possible to develop a computer programming language based on S-expressions?

3. Try these expressions at the Racket prompt >.

(a) `(+ 2 3)`
(b) `(* 2 3)`
(c) `(- 2 3)`
(d) `(/ 2 3)`
(e) `(+ 2)`
(f) `(* 2)`
(g) `(- 2)`
(h) `(/ 2)`

4. Try these in DrRacket. What do you get? What do these expressions evaluate to?

(a) `(/ 3 5)`
(b) `(/ 35 5)`
(c) `(/ 99 55)`
(d) `(/ 3)`
(e) `(/ 1)`
(f) `(/ 1 2)`
(g) `(/ 2 1)`
(h) `(/ 2)`
(i) `(/ -1)`
(j) `(/ (- 1))`

5. Use DrRacket to multiply these big numbers.

```
(* 29888118
   7777444499991111
   325325325325)
```

6. Use Racket to compute 2^{200}.

7. Use Racket to compute 19^{500}.

8. Figure out what these things evaluate to, and then try them in DrRacket.

(a) `(expt (/ 3) (- 3))`
(b) `(expt (/ (/ 3)) 4`
(c) `(expt 2 -1)`
(d) `(expt 1/3 -3)`
(e) `(expt 3 -2)`
(f) `(expt 8 1/3)`
(g)
```
(/ (expt (/ 2)
         (- (abs (- 4)))))
```

9. Given this code:

```
(define a 2)
(define b 3)
(define c 5)
```

Figure out the return values for these expression and try them at the Racket prompt >.

(a) `a`
(b) `b`
(c) `c`
(d) `(+ a b c)`
(e) `(* a b c)`
(f) `(+ (expt a b)`
`(expt b c))`
(g) `(+ (- a b)`
`(- b c)`
`(- a c))`

10. Do exercise 1.1 from SICP.

11. Do exercise 1.2 from SICP.

12. Translate this into a Racket expression. Test it in DrRacket.

$$2 \times (8 + 12) + 4 \times (9 - 5).$$

13. Translate into a Racket expression. Test your code in DrRacket.

$$\frac{4 \times 3) - 15/5}{(7 + 9) \times (3 - 8)}.$$

2 The onion model of evaluation

14. Draw the onion and explain how Racket evaluates this expression. Show how the steps start in the center of the onion and proceed outward to surrounding layers. Show each step.

```
(+ (* 1 2 3)
   (* 3 4 5)
   (* 6 7 8))
```

15. Draw the onion. Show the steps Racket goes through to evaluate this expression.

```
(+ (* 2 (+ (* 3 4) (* 2 9)))
   (- 10 (+ (* 1 2) (* 2 3))))
```

16. Draw the onion and show the steps by which Racket evaluates this expression.

```
(/ (+ (* 1 3 7)
      (- 7 4))
   (+ (* 1 2 5)
      (- 5 9)))
```

17. Draw the evaluation onion and show the steps by which Racket evaluates this expression.

```
(+ (abs (- (abs (- 3 11))))
   (abs (- (abs -5))))
```

18. Draw the onion and show the evaluation steps.

```
(define a 1)
(define b 2)
(define c 3)
(/ (+ (* a b)
      (* c a))
   (- (* b c)
      (* a b)))
```

3 Functions

19. In a typical Racket expression, where is the function located and where are the arguments located?

20. Suppose `f` is a function. What is the difference between `f` and `(f)`?

21. Type these into the Racket prompt >. Which expression gives an error, and why?

(a) `(- 5)` (b) `-5` (c) `(- -5)` (d) `(-5)`

22. Define a Racket function `square` that squares its argument:

$$\texttt{square} : x \longrightarrow x^2$$

Test it in DrRacket: `(square 2)`, `(square 5)`, etc.

23. Use your `square` function to define `power4` and `power8` functions. Here you are building new functions from functions you made before in Racket.

$$\texttt{power4} : x \longrightarrow x^4$$
$$\texttt{power8} : x \longrightarrow x^8$$

Test these functions and make sure they work properly.

24. Define a function `cube` that cubes its argument:

$$\texttt{cube} : x \longrightarrow x^3$$

Test it in DrRacket.

25. Define `power9` and `power27` functions. Build these functions from `cube`, which you defined before. Test them in Racket. Make sure they work as expected.

26. Starting with Racket's `sqrt` function, define `fourthroot`:

$$\texttt{fourthroot} : x \longrightarrow \sqrt[4]{x}$$

27. Starting with Racket's `expt` function, make a `cuberoot`

$$\texttt{cuberoot} : x \longrightarrow \sqrt[3]{x}$$

28. Explain the steps that Racket will take to evaluate this code. Draw the onion.

```
(define (f x)
  (+ (* x x) 1))

(+ (f (+ 2 3))
   (f 4))
```

29. Draw the onion and show the steps that Racket will take to evaluate this.

```
(define (f x) (- x 10))

(f (+ (abs (f (- 5 7)))
      (abs (+ 8 (f 9)))))
```

30. Show the steps of evaluation. Draw the onion.

```
(define (f n)
  (* n n))

(f (f (f (f 2))))
```

Try it in DrRacket and make sure you got the same answer as Racket does.

31. Given the function

```
(define (f x)
  (+ (* x x) 1))
```

Show the evaluation steps for the expression

```
(+ (- (f 2)
      (f 3))
   (+ (- (f 5)
         (f 10))))
```

Try it in DrRacket.

32. Define these funtions:

```
(define (f x)
  (- (* 3 x) 1))

(define (g x)
  (+ (* 5 x) 1))
```

Show the evaluation steps for this expression:

```
(f (g (f 1)))
```

And this expression:

```
(g (f (g 1)))
```

Test them in DrRacket.

33. Given functions `f` and `g`:

```
(define (f x)
  (+ (* 2 x) y))
(define (g x)
  (- x (* 3 y))
```

Show the evaluation steps for the expression:

```
(f (g 1 2)
   (f 1 2))
```

And this expression:

```
(f (f (g 2 1)
      (f 1 2))
   (g (f 2 1)
      (g 1 2)))
```

Try all this out in DrRacket.

34. Write a function that computes the arithmetic mean of two numbers.

$$\texttt{am} : x, y \longrightarrow \frac{x+y}{2}.$$

Use Racket and try it on $x = 15$ and $y = 20$. Your answer should be in between x and y.

35. Write a function that computes the geometric mean of two numbers.

$$\texttt{gm} : x, y \longrightarrow \sqrt{xy}$$

Test it in DrRacket with the numbers 15 and 20. The geometric mean should be in between these two numbers.

36. We know from mathematics that the geometric mean is always less than or equal to the arithmetic mean:

$$\text{gm}(x, y) \leqslant \text{am}(x, y).$$

Choose pairs of positive numbers and try this in DrRacket. For example, choose $x = 18$, $y = 45$ and see if it is true.

37. Define harmonic mean as a Racket function.

$$\texttt{hm} : x, y \longrightarrow \frac{2}{\frac{1}{x} + \frac{1}{y}}.$$

Harmonic mean of x and y is always somewhere in between x and y, like all the other means. Try this in DrRacket for pairs of x and y. For example, $x = 15$, $y = 21$.

38. In functional programming, we try to build new functions from old functions. You have already defined `am` and you already have `/` (division function). Find a way to define `hm` from `am` and `/`.

39. Choose positive values for x and y and find verify that

$$\text{hm}(x,y) \leqslant \text{gm}(x,y) \leqslant \text{am}(x,y)$$

for your values. Do it in DrRacket.

40. Define the root-mean-square function of two numbers:

$$\texttt{rms} : x, y \longrightarrow \sqrt{\frac{x^2+y^2}{2}}.$$

Choose positive numbers x, y and use Racket to compute `hm`, `gm`, `am`, `rms`. Is this inequality true for your choices of x and y?

$$\text{hm}(x,y) \leqslant \text{gm}(x,y) \leqslant \text{am}(x,y) \leqslant \text{rms}(x,y)$$

41. Write a function `square` that does $x \longrightarrow x^2$. Use this function and `am` that you wrote before to build `rms`. Again, the idea is to build new functions (`rms`) from smaller functions (`sqrt`, `am`, `square`).

42. Write a 3-argument version of arithmetic mean in Racket:

$$\texttt{am3} : x, y, z \longrightarrow \frac{x+y+z}{3}.$$

Test it in DrRacket.

43. Use the `cuberoot` function that you created before to write a 3-argument version of geometric mean:

$$\texttt{gm3} : x, y, z \longrightarrow \sqrt[3]{xyz}.$$

Test it on some numbers in DrRacket.

44. Write a 3-argument version of harmonic mean:

$$\texttt{hm3} : x, y, z \longrightarrow \frac{3}{\frac{1}{x}+\frac{1}{y}+\frac{1}{z}}.$$

Test it in DrRacket.

45. Use the functions `square` and `am3` that you defined before, and use them to build a 3-argument version of root-mean-square:

$$\texttt{rms3} : x, y, z \longrightarrow \sqrt{\frac{x^2+y^2+z^2}{3}}.$$

Test it on some numbers in DrRacket.

4 Data and lists

46. Give four examples of basic data types in Racket.

47. Give some examples of complex data that you see in the real world.

48. Why do we need lists and data structures?

49. What is a *proper list*? What makes it proper?

50. We have seen two ways of creating lists from basic data types. What are those two ways?

51. Draw the cons-cell diagram for this code:

```
(cons 'red 'green)
```

52. Given this code:

```
(define x (cons 'red 'green))
```

what are `(car x)` and `(cdr x)`?

53. Draw the cons-cell diagram for this code:

```
(cons (cons 'red 'green)
      (cons 'blue 'yellow))
```

54. Given the code

```
(define x (cons (cons 'red 'green)
          (cons 'blue 'yellow)))
```

what are `(car x)` and `(cdr x)`? What are `(car (car x))` and `(car (cdr x))`?

55. Given the code

```
(define g (cons 'red
                (cons 'blue 'green)))
```

(a) Draw the cons-cell diagram for `g`.
(b) What is `(car g)` and `(cdr g)`?
(c) Use `car` and `cdr` to get `'blue`.

56. Draw a cons-cell diagram for this data structure:

```
(cons 'red
      (cons (cons 'blue 'green)
            (cons 'yellow
                  (cons 'black 'white))))
```

57. Given the definition

```
(define z (cons (cons (cons 'red 'green)
                      (cons 'blue 'yellow))
                (cons (cons 1 2)
                      (cons 3 4))))
```

(a) Draw the cons-cell diagram for `z`.
(b) Is `z` a proper list?
(c) What are `(car z)` and `(cdr z)`? `'blue`.

58. Draw the cons-cell diagram and write code to construct these lists using `cons` and `empty`.
(a) `(list)`
(b) `(list 1)`
(c) `(list 1 2)`
(d) `(list 1 2 3)`
(e) `(list 1 2 3 4)`

59. Given the definition

```
(define x (list 'a 'b 'c 'd 'e))
```

(a) Draw the cons-cell diagram for `x`.
(b) Use `car` and `cdr` to get `'c`.

60. Which ones can Racket evaluate and which ones give you an error?

(a) `(+ 1234 4678)`
(b) `(+ "1234" "4567")`
(c) `(string-append "1234" "4567")`
(d) `(string-append 1234 4567)`

61. Which ones can Racket evaluate and which ones give an error?

(a) `(+ true false)`
(b) `(string-append #t #t)`
(c) `(and true true)`
(d) `(or false true)`

62. Use the onion model of evaluation. Explain the steps by which Racket will evaluate the expression.

```
(define a "Hello ")
(define b "there. ")
(define c "What time ")
(define d "is it?")
(string-append (string-append a b)
               (string-append c d))
```

63. Use `string-append` to define a function that takes a name, say `"Jim"` and returns a string `"Hello, Jim, how are you today?"`.

64. Starting from the inside of the onion and working outward, show the steps by which Racket evaluates this:

```
(define a 12)
(define b 23)
(define c 34)
(reverse (list (- a b)
               (- b c)
               (- a c)))
```

65. What are the lengths of these lists? Figure them out for yourself and then try the expressions in Racket.

(a)
```
(length (append (list 1 2 3 4)
                (list 'a 'b 'c)))
```
(b)
```
(length (list (append (list 1 2 3) (list 'a 'b 'c))
              (append (list 'a 'b 'c) (list 1 2 3))))
```

66. Given the following code

```
(define x (list 1 2 3 4))
(define y (list 'a 'b 'c 'd))
```

What are the return values of these expressions? Draw the onion and show the evaluation steps.

(a)
```
(reverse (append x y))
```
(b)
```
(append (reverse y)
        (reverse x))
```
(c)
```
(reverse (append (reverse x)
                 (reverse y)))
```

67. What is the length of the empty list? You can try it in Racket:

```
> (length empty)
```

68. Does `(list)` evaluate to the same thing as `empty`? Explain why.

69. Does `empty` have a `car` and a `cdr`? Look at the way we draw it and guess. Now try it in DrRacket and see what happens.

(a) `(car empty)` (b) `(cdr empty)`

70. Figure out what the return values will be and then test these expressions in DrRacket.

(a)
```
(length (list 1 2 3 4 5))
```
(b)
```
(string-length "abcde")
```
(c)
```
(map length (list (list 1 2 3 4)
                  (list 1 2 3)
                  (list 1 2)
                  (list 1)))
```
(d)
```
(map string-length (list "abcd" "abc" "ab" "a"))
```

71. Given the definitions

```
(define (f y)
  (+ (* 3 y)
     1))
(define (g y)
  (- (* 2 y)
     2))
```

Evaluate the expression

```
(map g (map f (list 1 2 3 4)))
```

72. Write a function that cubes a number n and use it with `map` to cube every number in the list `'(1 2 3 4 5)`.

73. What does this evaluate to?

```
(define (square x) (* x x))
(map square (reverse (list 1 2 3 4 5 6)))
```

74. What does this evaluate to?

```
(map reverse (list (list 1 2 3)
                   (list 'a 'b 'c)
                   (list 'red 'green 'blue)))
```

75. Given the function

```
(define (f s)
  (string-append "Hello " s))
```

what does this do?

```
(map f (list "Bob" "Bill" "Roger" "Kendrick"))
```

76. Suppose we have a polynomial p: $p = ax^2 + bx + c$. Write a function `disc` that calculates the discriminant of p:

$$\texttt{disc} : a, b, c \longrightarrow b^2 - 4ac$$

77. Write a function `roots` that takes a and b and the discriminant d and returns a list of two roots of the polynomial p:

$$\texttt{roots} : a, b, d \longrightarrow \left(\frac{-b + \sqrt{d}}{2a}, \frac{-b - \sqrt{d}}{2a} \right)$$

78. Combine the functions `disc` and `roots` to make a function that returns a list of the two roots of a quadratic polynomial $ax^2 + bx + c$:

$$\texttt{quadratic} : a, b, c \longrightarrow (\texttt{root1}, \texttt{root2}) .$$

5 More on functions

79. Let $f(x) = 2x + 3$ and $g(x) = (x - 3)/2$. Figure out $f(g(x))$ and $g(f(x))$.

80. Again, let $f(x) = 2x + 3$ and $g(x) = (x - 3)/2$. Define these functions in Racket and then use `compose` to define the functions $F = f(g(x))$ and $G = g(f(x))$. Find:

(a) `(F 5)` (b) `(G 5)` (c) `(F 10)` (d) `(G 10)`

81. Given the code:

```
(define (f x)
  (+ (* x x) 1))

(define (g x)
  (- (* 2 x) 1))

(define F (compose f g))
(define G (compose g f))
```

what will Racket do?

(a) `(F 1)` (b) `(G 1)` (c) `(F 2)` (d) `(G 2)` (e) `(F 0)`

82. Given the code:

```
(define (f n)
   (* n n))

(define F (compose f f f))
```

Figure out:

(a) `(F 0)` (b) `(F 1)` (c) `(F 2)`

83. What is a predicate?

84. The predicate = tests numbers. The predicate `eq?` tests symbols. The predicate `equal?` tests numbers, symbols, lists, strings and other things. What will Racket return?

(a) `(= 1 2)`
(b) `(= -5 -5)`
(c) `(eq? 'red 'blue)`
(d) `(eq? 'blue 'blue)`
(e) `(equal? "hello" "goodbye")`
(f) `(equal? "goodbye" "goodbye")`
(g) `(equal? (list 1 'red 3) (list 2 'red 1))`
(h) `(equal? (list 3 'red 5) (list 3 'red 5))`

85. The predicates >, <, >=, <= compare numbers in the way that you expect. What does Racket return?

(a) `(> 1 -3)` (c) `(< 1 1)` (e) `(<= 3 -1)` (g) `(>= 5 10)`
(b) `(< 5 2)` (d) `(<= 1 1)` (f) `(>= 5 5)` (h) `(>= -5 -4)`

86. The predicates `zero?`, `positive?`, `negative?`, `odd?` and `even?` tell you if a number has a particular property. What will Racket return?

(a) `(zero? 0)` (d) `(negative? -10)` (g) `(odd? 18)`
(b) `(zero? 1)` (e) `(positive? 0)` (h) `(even? 13)`
(c) `(positive? -3)` (f) `(even? 16)` (i) `(odd? 27)`

87. What does the `empty?` predicate do? Give some examples.

6 Decisions

88. Given the function

```
(define (F n)
  (if (positive? n)
      (- n 1)
      (+ n 1)))
```

What will Racket return?

(a) `(F 1)` (b) `(F 0)` (c) `(F -1)` (d) `(F 2)` (e) `(F -2)`

89. Given the function

```
(define (F n)
   (if (and (>= n 5) (odd? n))
       (* 2 n)
       (* 3 n)))
```

What will Racket return? You can guess what `and` does.

(a) `(F 1)` (b) `(F 0)` (c) `(F -1)` (d) `(F 2)` (e) `(F -2)`

90. Write a function $F(n)$ that returns n^2 if n is even and n^3 otherwise.

91. Write a function $F(n)$ that returns n if n is positive and odd, and $-n$ otherwise. You can use `and` or a combination of two `if` expressions if you like.

7 Recursion

92. Give a sequence solution to the recursion problem

$$f(1) = 1$$
$$f(n) = 2f(n-1).$$

Give at least 8 terms of the sequence. Try to find an algebraic solution if you can.

93. Give a sequence solution to the recursion problem

$$f(1) = 3$$
$$f(n) = 2f(n-1) - 1.$$

Give at least 8 terms of the sequence.

94. Give a sequence solution to the recursion problem

$$f(1) = 1$$
$$f(n) = 2f(n-1) + n - 1.$$

Give at least 8 terms of the sequence.

95. Fibonacci numbers can be generated by recursion. Give a sequence solution to the recursion problem

$$f(1) = 1$$
$$f(2) = 1$$
$$f(n) = f(n-1) + f(n-2)$$

Give at least 8 terms of the sequence.

96. Explain how a recursion problem is similar to money gaining interest in a bank.

97. Explain how a recursion problem is similar to a physics problem.

98. Explain the analogy between a recursion problem and the growth of crystals.

99. Give some pros and cons of the different styles of writing recursive functions: the direct style versus the state-variable style.

100. Let F be the factorial function:

$$F(n) = n(n-1)(n-2)\cdots(1).$$

Figure out the initial condition and recursion equation for F.

101. Write the Racket code for the factorial function in problem **100**. Use direct recursion style.

102. Write the Racket code for the factorial function in problem **100**. Use state-variable recursion style.

103. Draw the first 8 triangular numbers, from $T(1)$ to $T(8)$.

104. Figure out the initial condition and recursion equation for $T(n)$.

105. Use the initial condition and recursion equation for to write the Racket code for $T(n)$. Do it in direct recursion style and then in state-variable style.

106. Draw the first 6 square numbers, from $S(1)$ to $S(6)$. Look at the pattern and figure out the intial condition and recursion relation for $S(n)$. Write the Racket code for $S(n)$ using direct recursion style. Then write it in state-variable recursion style.

107. Draw the first 4 pentagonal numbers, from $P(1)$ to $P(4)$. Study the pattern, how each one is constructed from the one that came before. Then figure out the initial condition and the recursion relation for $P(n)$. Write the code for P using direct style and then using state variable style. Test your code with this: $P(10) = 145$.

108. Draw the first 4 hexagonal (6-sided) numbers, from $H(1)$ to $H(4)$. Study the pattern and figure out the initial condition and recursion equation for $H(n)$. Write Racket code for $H(n)$ in direct style and in state-variable style. Test your code with $H(10) = 190$.

109. Fill in this table, study the patterns, and guess the last three rows.

$k = 3$	triangular numbers	$T(n) = T(n-1) + n$
$k = 4$		
$k = 5$		
$k = 6$		
$k = 7$	heptagonal numbers	
$k = 8$	octogonal numbers	
any k	k-gonal numbers	

110. Challenge. Now that you have the initial condition and recursion equations for all kinds of polygonal numbers, use that to write the Racket code for $Y(k, n)$, the nth k-gonal number. Write it in both direct and state-variable styles.

111. Explain the analogy between taking the `cdr` of a list and subtracting 1 from an integer.

112. Let `L` be a function that counts the elements of a list. Give the initial condition and the recursion equation for `L`.

113. Write the Racket code for `L` in **112** using direct recursion style.

114. Write the Racket code for `L` in **112** using state-variable recursion style.

115. Let the function F be defined by the initial condition $F(\texttt{empty}) = 0$ and recursion equations

$$F(\texttt{mylist}) = F(\texttt{(cdr mylist)}) + 1$$

if `(car mylist)` is even, and

$$F(\texttt{mylist}) = F(\texttt{(cdr mylist)}) - 1$$

when `(car mylist)` is odd. What does function F do? What will Racket return if F is applied to these lists?

(a) `'(6 5 4 3 2 1 0)`
(b) `'(11 17 12 18 15 19)`
(c) `'()`
(d) `'(-3 -2 -1 0 1 2 3 4)`

116. Write the code for function F in problem **115** using direct recursion.

117. Write the code for F in problem **115** using state-variable recursion.

118. Let `G` a function that takes the product of all the elements of a list. Figure out the initial condition and recursion equation for `G`.

119. Write the Racket code for the list product function `G` from problem **118**...
(a) in direct recursion style.
(b) in state-variable recursion style.

120. Challenge. Let $B(n)$ be the *bifactorial function*. The bifactorial function is defined like this:

$$B(n) = (2)(4)(6)\cdots(n-2)(n) \quad \text{if } n \text{ is even}$$
$$B(n) = (1)(3)(5)\cdots(n-2)(n) \quad \text{if } n \text{ is odd.}$$

Write Racket code for the function $B(n)$. Use direct style and then use state-variable style.