

Computer Programming

Training problems for M3 2018 term 2

Ted Szylowiec
tedszy@gmail.com

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

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

Download Racket here:

<https://racket-lang.org/>

Use Racket online at Tio:

<https://tio.run/#racket>

Have a look at Racket code:

<https://github.com/tedszy/Racketry>

1 Lambda

1. Use `define` to define a symbol having an integer value.
2. Use `define` to define a symbol having a string value.
3. Use `define` to define a symbol having a boolean value.
4. Define a symbol to have a rational value.
5. Define a symbol to have a float value.
6. Use `define` and `lambda` to define a symbol having a function value.
7. Explain why these give you errors.
 - (a) `(define "x" 10)`
 - (b) `(define 10 5)`
 - (c) `(define #f a)`
 - (d) `("string-append" "good" "night")`
 - (e) `(define (f "x") (* x x))`
 - (f) `(define ("f" x) (* x x))`
8. What is a lambda? Who discovered it? Why is it so interesting in computer science?
9. Give some examples of computer programming languages that have lambda and support lambda-style programming.

10. Practice arrow notation. What is the result?

- (a) $(x \rightarrow x^2 + 1)(3)$
- (b) $(x, y \rightarrow 2x + 5y)(3, 7)$
- (c) $(x, y, z \rightarrow \sqrt{xy} + \sqrt{xz} + \sqrt{yz})(2, 3, 5)$
- (d) $(x, y, z \rightarrow |xy| + |xz| + |yz|)(-1, 2, -3)$
- (e) $(x, y \rightarrow x^2 + y^2)((x \rightarrow x + 1)(2), (x \rightarrow x - 2)(7))$

11. Write this as a lambda expression: $x \rightarrow x^2 + 3x + 1$.

12. Write this as a lambda expression: $x \rightarrow x^2$ if x is odd, else x^3 . Use Racket's `if` and `odd?` function.

13. Write this as a lambda expression: $x, y \rightarrow \sqrt{xy}$. Use Racket's `sqrt` function.

14. Write using lambda: $x, y, z \rightarrow \frac{x^2 + y^2 + z^2}{2}$.

15. The identity function takes x and returns x without any changes: $x \rightarrow x$. Write the identity function using lambda.

16. Change lambda expression to arrow (\rightarrow) notation:

```
(lambda (x y) (+ (* 2 x) (* 3 y)))
```

17. Change lambda expression to arrow notation:

```
(lambda (x y z) (+ (/ (sqrt x)
                      (/ (sqrt y)
                          (/ (sqrt z))))))
```

18. What does Racket return?

- (a) `> (lambda (x) (* x x))`
- (b) `> ((lambda (x) (* x x)) 5)`
- (c) `> ((lambda (x y) (+ 1 (* x y))) 6 7)`
- (d) `> ((lambda (x) (string-append "happy " x)) "halloween")`
- (e) `> ((lambda (x) (string-append x "happy ")) "halloween")`

19. What does Racket return?

- (a) `> ((lambda (x y z) (+ x y z)) 10 21 32)`
- (b) `> ((lambda (x y z) (+ (/ x) (/ y) (/ z))) 2 3 5)`
- (c) `> ((lambda (x y) (* (+ x y) (- x y))) 7 5)`

20. What does this expression return?

```
((lambda (x)
  (* ((lambda (y) (+ (* 2 y) 1)) x)
     ((lambda (y) (- y 1)) x)))
10)
```

21. Write a lambda-expression that adds the square roots of 3 and 5.

22. Write a lambda expression that finds the harmonic mean of 2, 5 and 7.

23. Write a lambda expression that finds the average of the lengths of these two lists: `(list 'a 'b 'c)` and `(list 1 2 3 4 5)`. Use the `length` function to get the length of a list.

24. Let $f : x \rightarrow 5x$ and $g : x \rightarrow 2x$. Write a one-line lambda expression that does $f(3) + g(6)$.

25. Change this to lambda-style function definition.

```
(define (f x)
  (+ (* x x) 5))
```

26. Change to lambda-style function definition.

```
(define (f x)
  (if (even? x) (/ x 2) (* x 2)))
```

27. Change to lambda-style definition.

```
(define (g x y)
  (/ (+ x y) 2))
```

28. Change to lambda-style definition.

```
(define (h x y z)
  (expt (* x y z) 1/3))
```

29. Do this computation with a one-shot expression using a lambda and no definitions.

```
(define (f x)
  (+ (* 2 x) 1))
(f 10)
```

30. Do this as a one-line expression using lambda, without definitions.

```
(define (greetings s)
  (string-append "hello there " s))
(greetings "Jim")
```

31. Rewrite this as one expression using lambda and no definitions.

```
(define a 10)
(define b 25)
(define (f x y) (- (* x y) 5))
(f a b)
```

32. Rewrite all this as a one-line expression using lambda.

```
(define s1 "greetings ")
(define s2 "earthman")
(define (F a b)
  (string-append a b ", take me to your leader"))
(F s1 s2)
```

33. Get rid of all symbol definitions and rewrite this program as a one-line expression using lambda.

```
(define a 30)
(define b 40)
(define c 60)
(define (average x y z)
  (/ (+ x y z) 3))
(average a b c)
```

34. Let $f : x \rightarrow x^2$ and $g : x \rightarrow x + 1$. Write $f(g(5))$ as one expression using two lambdas. Don't use define or compose.

35. Let $f : x \rightarrow 2x + 1$ and $g : x \rightarrow 3x + 2$. Write $f(g(10))$ in Racket using only lambdas.

2 Map and filter

36. What does this expression return?

```
(map (lambda (x) (* x x))
     (list 1 2 3 4 5 6 7))
```

37. What does this expression return?

```
(map (lambda (x y) (* (+ x 3) (- y 2)))
     (list 1 2 3 4 5 6 7)
     (list 7 6 5 4 3 2 1))
```

38. Write a one-shot expression that takes the numbers from 0 to 99, squares them if they are odd, and cubes them if they are even. Use map, lambda, if, odd? and range.

39. What do these expressions do?

- (a) (map even? (range 10))
- (b) (filter even? (range 10))
- (c) (map odd? (list 1 2 3 4 5 6 7))
- (d) (filter odd? (list 1 2 3 4 5 6 7))
- (e) (filter even? (list 1 2 3 4 5 6 7))
- (f) (filter (lambda (x) (= (remainder x 3) 0)) (list 1 2 3 4 5 6 7))

40. What does this expression do?

```
(filter (lambda (x) (> x 2))
       (list -2 5 -8 3 2 1 9 8 -1 0))
```

41. How many numbers from 0 to 999 are divisible by 7? Write a Racket expression to calculate this. Use length, filter, lambda, range, = and remainder.

42. Write a Racket expression that takes (list 0 -3 6 -8 7 9 -4 2) keeps only the elements > 1 , and then squares them. Use filter, map and lambda.

43. Write Racket expression that calculates how many numbers from 0 to 999 are divisible by 2, 3 and 7. Use length, filter, lambda, if, and, remainder, = and range.

44. Map the function $x \rightarrow 1/\sqrt{x}$ onto the list of numbers 1,2,... 10. Then filter the result to keep all the ones that are bigger than $1/3$. Use map, filter, > and lambda.

3 Logic

45. The crystal ball says “tomorrow you will *not* eat an apple”. If we let p be “you will eat an apple”, then we can write what the crystal ball predicts as $\neg p$.

Draw some cartoons for what can happen tomorrow. When is the crystal ball right? When is it wrong? When is $\neg p$ true and when is it false?

46. The crystal ball says “tomorrow you will either eat an apple or see an alien *but not both*.” If we let p be “you will eat an apple” and q be “you will see an alien” then we can write the crystal ball prediction as $p \oplus q$. This is called *xor* or *exclusive or*. Either p can be true or q can be true but not both.

Draw cartoons for what can happen tomorrow. When is the crystal ball right and when is it wrong? Use this to figure out when $p \oplus q$ is false and when it is true.

47. Fill in these logic tables.

\wedge	T	F
T		
F		

\vee	T	F
T		
F		

\rightarrow	T	F
T		
F		

48. Fill in these logic tables.

\Leftrightarrow	T	F
T		
F		

\oplus	T	F
T		
F		

49. Figure out the truth values.

- (a) $\neg F$. (b) $\neg\neg T$. (c) $\neg\neg\neg\neg F$. (d) $\neg\neg\neg\neg\neg T$.

50. Figure out the truth values. Work from the inside out, like the way you evaluate Racket expressions.

- (a) $(\neg F \wedge T) \vee (F \wedge \neg F)$. (b) $(F \rightarrow T) \rightarrow (\neg T \vee F)$. (c) $\neg(T \rightarrow F) \wedge (F \rightarrow \neg T)$.

51. Figure out the truth values.

- (a) $(\neg T \oplus F) \Leftrightarrow (T \oplus T)$. (b) $(T \Leftrightarrow F) \oplus \neg(\neg T \Leftrightarrow F)$. (c) $((F \Leftrightarrow T) \Leftrightarrow (\neg T \Leftrightarrow T))$.

52. Make truth tables.

- (a) Make a truth table for $p \rightarrow q$.
 (b) Make a truth table for $\neg p \vee q$. Is it the same as in (a)?
 (c) Make a truth table for $(p \rightarrow q) \Leftrightarrow (\neg p \vee q)$. Is it a tautology?

53. Make truth tables.

- (a) Make a truth table for $\neg(p \wedge q)$.
 (b) Make a truth table for $\neg p \vee \neg q$. Is it the same as in (a)?
 (c) Make a truth table for $\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$. Is it a tautology?

54. Make truth tables.

- (a) Make a truth table for $\neg p \oplus \neg q$.

- (b) Make a truth table for $\neg(p \Leftrightarrow q)$. Is it the same as in (a)?
 (c) Make a truth table for $(\neg p \oplus \neg q) \Leftrightarrow \neg(p \Leftrightarrow q)$. Is it a tautology?

55. Make truth tables.

- (a) Make a truth table for $p \Leftrightarrow q$.
 (b) Make a truth table for $(p \rightarrow q) \wedge (q \rightarrow p)$. Is it the same as in (a)?
 (c) Make a truth table for $(p \Leftrightarrow q) \Leftrightarrow ((p \rightarrow q) \wedge (q \rightarrow p))$. Is it a tautology?

56. Make a truth table for the expression

$$((\neg p \rightarrow q) \wedge (\neg p \rightarrow \neg q)) \rightarrow p.$$

Is this expression a tautology?

57. Make a truth table for the expression

$$\neg((p \wedge q) \wedge \neg r).$$

It has three variables, so the table will have 8 rows. Is the expression a tautology?

58. Make a truth table for

$$((p \rightarrow q) \wedge (p \rightarrow r)) \rightarrow r.$$

Is this a tautology?

59. Make a truth table for

$$((p \wedge q) \wedge (p \rightarrow r)) \vee (\neg(p \vee q) \vee \neg(p \rightarrow r))$$

Is it a tautology?

60. Racket has not, and, or, #t and #f built into the language. Translate the following logic propositions into Racket and evaluate them.

- (a) $(\neg T \wedge \neg F) \vee \neg(T \vee \neg F)$.
 (b) $\neg(T \vee F) \wedge ((T \vee F) \wedge (F \vee F))$.
 (c) $(\neg(T \wedge F) \vee (F \wedge F)) \vee \neg(T \wedge T)$.

61. Make a truth table and show that $p \rightarrow q$ is the same as $\neg p \vee q$.

62. Let F and G be two logic expressions. Another way we can show that F is the same as G is to make a truth table for $F \Leftrightarrow G$ and show that it is a tautology. Do this with $F = p \rightarrow q$ and $G = \neg p \vee q$.

63. Define a Racket function called `implies` that does $p, q \rightarrow p \rightarrow q$. Use the idea that $p \rightarrow q$ is the same as $\neg p \vee q$.

64. Translate these logic propositions into Racket and evaluate them.

- (a) $(T \rightarrow (T \wedge F)) \vee \neg(F \rightarrow T)$.
 (b) $((F \wedge F) \rightarrow (T \vee F)) \rightarrow (F \vee T)$.
 (c) $\neg(T \rightarrow F) \rightarrow \neg(F \rightarrow T)$.

65. Make a truth table and show that $p \Leftrightarrow q$ can be expressed as $(p \rightarrow q) \wedge (q \rightarrow p)$.

66. Let $F = p \Leftrightarrow q$ and $G = p \rightarrow q \wedge q \rightarrow p$. Show that F and G are the same by making a truth table and showing that $F \Leftrightarrow G$ is a tautology.

67. Define a Racket function called `iff` (if and only if) that does $p, q \longrightarrow p \Leftrightarrow q$. Use the idea that $p \Leftrightarrow q$ is the same as $(p \rightarrow q) \wedge (q \rightarrow p)$.
68. Translate these logic propositions into Racket and evaluate them.
- $(T \Leftrightarrow (F \rightarrow T)) \vee \neg(F \Leftrightarrow (T \rightarrow F))$.
 - $((T \Leftrightarrow F) \rightarrow (T \wedge T)) \Leftrightarrow (T \rightarrow F)$.
 - $((T \vee F) \Leftrightarrow (F \wedge T)) \Leftrightarrow (F \rightarrow F)$.
69. Make a truth table and show that $p \oplus q$ is the same as $\neg(p \Leftrightarrow q)$.
70. Let $F = p \oplus q$ and $G = \neg(p \Leftrightarrow q)$. Make a truth table and show that $F \Leftrightarrow G$ is a tautology.
71. Define a Racket function called `xor` (exclusive or) that does $p, q \longrightarrow p \oplus q$. Use the idea that $p \oplus q$ is the same as $\neg(p \Leftrightarrow q)$.
72. Translate these logic propositions into Racket and evaluate them.
- $(T \oplus F) \Leftrightarrow (T \oplus T)$.
 - $(F \Leftrightarrow (T \oplus F)) \vee (F \oplus (F \Leftrightarrow T))$.
 - $((T \Leftrightarrow F) \oplus (T \rightarrow F)) \oplus (T \rightarrow F)$.
73. Find a way to express $p \Leftrightarrow q$ using only \wedge , \vee and \neg .
74. Find a way to express $p \oplus q$ using only \wedge , \vee and \neg .
75. Save your definitions for `implies`, `iff` and `xor` in a file called `logic.rkt`. Make sure it works by loading it in Racket.
76. Define a Racket function $F : p, q \longrightarrow ((p \rightarrow q) \Leftrightarrow (q \oplus p))$ and use it to evaluate $F(T, T)$, $F(F, F)$.
77. Define a Racket function $F : p, q \longrightarrow ((p \oplus (p \vee q)) \wedge (q \rightarrow p))$ And use it to build a truth table for F by calculating $F(F, F)$, $F(F, T)$, $F(T, F)$ and $F(T, T)$.
78. Define a Racket function $F : p, q \longrightarrow (p \oplus (\neg q \wedge (q \rightarrow p)))$. Use it to build a truth table for F .
79. Define a Racket function $G : p, q, r \longrightarrow ((p \rightarrow q) \Leftrightarrow r) \oplus ((r \rightarrow p) \Leftrightarrow q)$ and use it (and Racket) to help you quickly build a truth table for G .
80. What do these expressions evaluate into and which ones give errors (and why)?
- `(+ 1 2 3 4 5)`
 - `(+ (list 1 2 3 4 5))`
 - `(apply + (list 1 2 3 4 5))`
 - `(apply sqrt 16)`
 - `(apply sqrt (list 16))`
 - `(sqrt (list 16))`
 - `(remainder 33 10)`
 - `(remainder (list 33 10))`
 - `(apply remainder (list 33 10))`
 - `(* (list 1 2 3 4))`
 - `(apply * (list 1 2 3 4))`
 - `(* 1 2 3 4)`

81. Given the code:

```
(define mydata (list 10 20 50))
(define (average x y z) (/ (+ x y z) 3))
```

- (a) Use `car` and `cdr` and `average` to find the average of `mydata`
- (b) Use `apply` and `average` to do the same job much more easily.

82. Suppose you did six experiments and you have your experimental results packed into a list of lists.

```
(define mydata (list (list 4 12 17)
                    (list 8 19 14)
                    (list 12 16 18)
                    (list 20 10 15)
                    (list 7 17 19)
                    (list 9 10 13)))
```

- (a) Write a Racket function that finds the geometric mean of three arguments. Use `expt`.
- (b) Use `map`, `apply` and `lambda` to write one expression that calculates the geometric mean of each one of your experimental results and returns the answers in a list.

83. You did four experiments. Each experiment gave you a list of five numbers as a result. All this data is packed into a list of lists.

```
(define mydata (list (list 4 1 11 7 21)
                    (list 8 19 21 7 5)
                    (list 3 12 22 9 15)
                    (list 21 5 16 8 9)))
```

We want to divide the product of the numbers of one experiment by their sum. And we want to do that for each experimental result. Write a one-shot expression in Racket that does the job. Use `map`, `lambda`, `apply`, etc.

84. Add these to your `logic.rkt` library and save it.

```
(define pvalues (list #t #f))
(define pqvalues (list (list #f #f) (list #f #t)
                      (list #t #f) (list #t #t)))
(define pqrvalues (list (list #f #f #f) (list #f #f #t)
                        (list #f #t #f) (list #f #t #t)
                        (list #t #f #f) (list #t #f #t)
                        (list #t #t #f) (list #t #t #t)))
```

We will use them to quickly build truth tables.

85. Use Racket to quickly build a truth table for $F(p) = ((p \oplus \neg p) \rightarrow p) \vee p$.

- (a) Define F in Racket.
- (b) Use `map`, F and `pvalues` to build a truth table.

86. Let $F(p, q) = (p \vee \neg q) \rightarrow (p \oplus (p \wedge \neg q))$.

- (a) Define F in Racket.

- (b) Use `F`, `map`, `lambda`, `apply` and `pqvalues` to quickly build a truth table with a one-shot expression.

87. Let $F(p, q, r) = ((r \rightarrow q) \Leftrightarrow (p \oplus r)) \vee \neg(p \rightarrow r)$

- (a) Define F in Racket.
 (b) Use `F`, `map`, `lambda`, `apply` and `pqrvalues` to quickly build a truth table.

4 Structs

88. Create a name data type using lists. The first element of the list is the symbol `'name`, then comes the first-name, middle-name and last-name. Write these functions: `make-name`, `get-first-name`, `get-middle-name`, `get-last-name`. Also write a predicate `name?` that checks the symbol at the beginning to see if a list is actually someone's name.

89. Make a name and test your functions to see if they work as expected.

```
(define n (make-name "Alvin" "Cortez" "Mendoza"))
(name? n)
(get-first-name n)
(get-middle-name n)
(get-last-name n)
```

90. Create a point data type using lists. The first element of the list is the symbol `'point`, then comes the x component and the y component. Write these functions: `make-point`, `get-point-x`, `get-point-y`. Also write a predicate `point?` that checks the symbol at the beginning to see if the list really is a point.

91. Make a point and test your functions to see if they work as expected.

```
(define P (make-point 42 -11))
(point? P)
(get-point-x P)
(get-point-y P)
```

92. A line can be represented by three numbers. Create a line data type using lists. The first element of the list will be the symbol `'line`, then comes the l component, the m component and the n component. Write these functions: `make-line`, `get-line-l`, `get-line-m` and `get-line-n`. Also write a predicate `line?` that checks the symbol at the beginning to see if the list is really a line.

93. Make a line and test your functions to see if everything works as expected.

```
(define r (make-line 5 7 -8))
(line? r)
(get-line-l r)
(get-line-m r)
(get-line-n r)
```

94. What are the differences between structs and lists?

95. When is it best to use a struct? When is it best to use a list? Give examples.

96. What is the difference between a normal struct and a transparent struct?

97. Create a transparent point struct for names. Like in problem 88, names have three components: first, middle and last. Test your code by trying this in Racket.

```
(define n (name "Alvin" "Cortez" "Mendoza"))
(name? n)
(name-first n)
(name-middle n)
(name-last n)
```

98. As in problem 90, a point (x, y) has two components: x and y . Create a transparent point struct in Racket and test your code like this:

```
(define P (point 42 -11))
(point? P)
(point-x P)
(point-y P)
```

99. As in problem 92, a typical line $lx + my + n = 0$ has three components: l , m and n . Create a transparent line struct in Racket and test that everything works as expected:

```
(define r (line 5 7 -8))
(line? r)
(line-l r)
(line-m r)
(line-n r)
```

100. What is the difference between positive and negative area? How can area be negative? Draw examples of positive and negative area.

101. The area-determinant formula for three vertices (triangle) is:

$$A = \frac{1}{2} \left(\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \begin{vmatrix} x_3 & y_3 \\ x_1 & y_1 \end{vmatrix} \right)$$

Write down the area-determinant formula for a square.

102. Draw a pentagon and a hexagon and label the vertices. Write down area-determinant formulas for the pentagon and hexagon.

103. Draw these points and find the area of the triangle: $A(-1, -4)$, $B(3, -1)$, $C(2, 5)$.

104. Plot these points and find the area by area-determinant formula: $A(-5, -2)$, $B(0, -4)$, $C(4, -3)$, $D(1, 6)$.

105. Given two points (x_1, y_1) and (x_2, y_3) find the equation of the line going through them $lx + my + n = 0$ by area-determinant method.

106. Write a Racket function `join-points` that takes two points and returns the line going through them.

107. Use your Racket function to find the line going through the points $(-3, 8)$ and $(7, 2)$.

108. Cramer's method. Consider the following equations:

$$ax + by = e \quad (1)$$

$$cx + dy = f. \quad (2)$$

Let

$$D = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \quad D_x = \begin{vmatrix} e & b \\ f & d \end{vmatrix} \quad D_y = \begin{vmatrix} a & e \\ c & f \end{vmatrix}$$

Substitute $x = D_x/D$ and $y = D_y/D$ into equations (1) and (2). Show that they are both true.

109. Use Cramer's method to solve the system:

$$5x - 3y = 7$$

$$2x + 9y = -1.$$

110. What does it mean to "join" a point and a line? There are actually *two* natural ways to do this (we saw one way in class). Both ways give something unique. One way gives you a unique line. The other way gives a unique point. We can say that joining a line to a point gives a special line, while joining a point to a line gives a special point:

$$\text{join-line-point} : r, P \longrightarrow \text{special line}$$

$$\text{join-point-line} : P, r \longrightarrow \text{special point}$$

Draw examples of both ways and explain what is the special line and the special point.

111. Given two lines $[l_1, m_1, n_1]$ and $[l_2, m_2, n_2]$, use Cramer's method to find the point (x, y) that is the join of these two lines.

112. Write a Racket function `join-lines` that takes two lines and returns the intersection point (the join of two lines).

113. Use your Racket function to find the intersection point of the lines $[5, 2, -1]$ and $[2, 9, 5]$.

114. If you did problem **110** then you know that joining a line to a point gives the perpendicular line through the point. Given the components $[l_0, m_0, n_0]$ of the line and (x_0, y_0) of the point, figure out the components $[l, m, n]$ of this line. Use the idea that if line r has slope k , then a line perpendicular to r has slope $-1/k$.

115. If you did problem **110** then you already know that joining a point (x_0, y_0) to a line $[l_0, m_0, n_0]$ gives the intersection point of the given line and the perpendicular line through the given point. Figure out the components (x, y) of this point. Use the idea that the join of two lines is a point (you already know how to do that.) We call this special point the *perpendicular projection* of (x_0, y_0) onto the line $[l_0, m_0, n_0]$.

116. Write a Racket function `join-line-point` that joins a line with a point giving a special line.

117. Use your Racket function to find the line perpendicular to $[1, 1, -5]$ going through the point $(4, 3)$.

118. Write a Racket function `join-point-line` that joins a point with a line, giving a special point.

119. Use your Racket function to find the perpendicular projection of $(4, 3)$ onto the line $[1, 1, -5]$.

120. This is an exercise in building cool and useful software. Combine all the different ways to join points and lines into one all-purpose `join` function:

```
join : point, point → line
join : line, line → point
join : point, line → point
join : line, point → line
```

You will need to test the arguments of `join` to see what type of structs they are. Use the predicates `point?` and `line?` for that. To check the different possibilities, use `cond`, which is like a super-version of the `if` conditional. Read the Racket documentation (help system) on how to use `cond`. If you want to make your software project even better, have `cond` call the error function when the arguments to `join` are not one of those four cases. You can read about the error function in the Racket documentation.