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

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

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

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

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

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

16. Change lambda expression to arrow (→) 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 $f : x \to 5x$ and $g : x \to 2x$. 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 $f : x \to x^2$ and $g : x \to x + 1$. Write $f(g(5))$ as one expression using two lambdas. Don't use `define` or `compose`.

35. Let $f : x \to 2x + 1$ and $g : x \to 3x + 2$. 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 $x \to 1/\sqrt{x}$ 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) $\neg\neg\neg\neg\neg$F. (d) $\neg\neg\neg\neg\neg\neg\neg$T.

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

(a) $(\neg \mathsf{F} \wedge \mathsf{T}) \vee (\mathsf{F} \wedge \neg \mathsf{F})$. (b) $(\mathsf{F} \rightarrow \mathsf{T}) \rightarrow (\neg \mathsf{T} \vee \mathsf{F})$. (c) $\neg(\mathsf{T} \rightarrow \mathsf{F}) \wedge (\mathsf{F} \rightarrow \neg \mathsf{T})$.

51. Figure out the truth values.

(a) $(\neg \mathsf{T} \oplus \mathsf{F}) \Leftrightarrow (\mathsf{T} \oplus \mathsf{T})$. (b) $(\mathsf{T} \Leftrightarrow \mathsf{F}) \oplus \neg(\neg \mathsf{T} \Leftrightarrow \mathsf{F})$. (c) $((\mathsf{F} \Leftrightarrow \mathsf{T}) \Leftrightarrow (\neg \mathsf{T} \Leftrightarrow \mathsf{T})$.

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

56. Make a truth table for the expression

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

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 with have 8 rows. Is the expression a tautology?

58. Make a truth table for

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

Is this a tautology?

59. Make a truth table for

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

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) $(\neg \mathsf{T} \wedge \neg \mathsf{F}) \vee \neg(\mathsf{T} \vee \neg \mathsf{F})$.
(b) $\neg(\mathsf{T} \vee \mathsf{F}) \wedge ((\mathsf{T} \vee \mathsf{F}) \wedge (\mathsf{F} \vee \mathsf{F}))$.
(c) $(\neg(\mathsf{T} \wedge \mathsf{F}) \vee (\mathsf{F} \wedge \mathsf{F})) \vee \neg(\mathsf{T} \wedge \mathsf{T})$.

61. Make a truth table and show that $p \to q$ 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 $F = p \to q$ and $G = \neg p \vee q$.

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

64. Translate these logic propositions into Racket and evaluate them.
(a) $(\mathsf{T} \to (\mathsf{T} \wedge \mathsf{F})) \vee \neg(\mathsf{F} \to \mathsf{T})$.
(b) $((\mathsf{F} \wedge \mathsf{F}) \to (\mathsf{T} \vee \mathsf{F})) \to (\mathsf{F} \vee \mathsf{T})$.
(c) $\neg(\mathsf{T} \to \mathsf{F}) \to \neg(\mathsf{F} \to \mathsf{T})$.

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

66. Let $F = p \Leftrightarrow q$ and $G = p \to q \wedge q \to p$. 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 $(p \to q) \wedge (q \to p)$.

68. Translate these logic propositions into Racket and evaluate them.
(a) $(\mathsf{T} \Leftrightarrow (\mathsf{F} \to \mathsf{T})) \vee \neg(\mathsf{F} \Leftrightarrow (\mathsf{T} \to \mathsf{F}))$.
(b) $((\mathsf{T} \Leftrightarrow \mathsf{F}) \to (\mathsf{T} \wedge \mathsf{T})) \Leftrightarrow (\mathsf{T} \to \mathsf{F})$.
(c) $((\mathsf{T} \vee \mathsf{F}) \Leftrightarrow (\mathsf{F} \wedge \mathsf{T})) \Leftrightarrow (\mathsf{F} \to \mathsf{F})$.

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.
(a) $(\mathsf{T} \oplus \mathsf{F}) \Leftrightarrow (\mathsf{T} \oplus \mathsf{T})$.
(b) $(\mathsf{F} \Leftrightarrow (\mathsf{T} \oplus \mathsf{F})) \vee (\mathsf{F} \oplus (\mathsf{F} \Leftrightarrow \mathsf{T}))$.
(c) $((\mathsf{T} \Leftrightarrow \mathsf{F}) \oplus (\mathsf{T} \to \mathsf{F})) \oplus (\mathsf{T} \to \mathsf{F})$.

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 $F : p, q \longrightarrow ((p \to q) \Leftrightarrow (q \oplus p))$ and use it to evaluate $F(\mathsf{T}, \mathsf{T})$, $F(\mathsf{F}, \mathsf{F})$.

77. Define a Racket function $F : p, q \longrightarrow ((p \oplus (p \vee q)) \wedge (\,|q \to p)$ And use it to build a truth table for F by calculating $F(\mathsf{F}, \mathsf{F})$, $F(\mathsf{F}, \mathsf{T})$, $F(\mathsf{T}, \mathsf{F})$ and $F(\mathsf{T}, \mathsf{T})$.

78. Define a Racket function $F : p, q \longrightarrow (p \oplus (\neg q \wedge (q \to p)))$. Use it to build a truth table for F.

79. Define a Racket function $G : p, q, r \longrightarrow ((p \to q) \Leftrightarrow r) \oplus ((r \to p) \Leftrightarrow q)$ 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)?
(a) `(+ 1 2 3 4 5)`
(b) `(+ (list 1 2 3 4 5))`
(c) `(apply + (list 1 2 3 4 5))`
(d) `(apply sqrt 16)`
(e) `(apply sqrt (list 16))`
(f) `(sqrt (list 16))`
(g) `(remainder 33 10)`
(h) `(remainder (list 33 10))`
(i) `(apply remainder (list 33 10))`
(j) `(* (list 1 2 3 4))`
(k) `(apply * (list 1 2 3 4))`
(l) `(* 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 $F(p, q, r) = ((r \to q) \Leftrightarrow (p \oplus r)) \vee \neg(p \to r)$

(a) Define F in Racket.

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