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`

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

4 Data and lists

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

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

30. Why do we need lists and data structures?

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

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