# Notes and Tips, week 5

Subscribe for email updates. PINNED

No tags yet. + Add Tag

Sort replies by: Oldest first Newest first Most popular

Charilaos Skiadas COMMUNITY TA · 15 days ago

## Notes on material

- Always use `#lang racket` at the top of the file.
- Need `(provide (all-defined-out))` near the top.
- Comments start with a semicolon, go to end of line.
- Almost everything is has the form `(e1 e2 e3 ..)` where first term is operator, the rest are the arguments.
- Use `define` to define new variables.
- `lambda` similar to ML's `fn`.
- Some arithmetic operations can take any number of arguments.
- Two ways to define functions: `(define f (lambda (x) body))` or `(define (f x) body)`. Most often use the latter.
- `if-then-else` is `(if eTest eThen eElse)`
- Parentheses ALWAYS matter in Racket! (Changes the meaning if you add "extra" parentheses)
- List operations named differently: `[] -> null`, `:: -> cons`, `hd -> car`, `tl -> cdr`, `null -> null?`.
- Can also use `'()` instead of `null`. And `list` "convenience": `(list 4 2 1 2 ..)`.
- Hyphens allowed in variable names, conventional to use them as separators (rather than `_`).
- `true -> #t`, `false -> #f`.
- Can use square brackets instead of parentheses.
- In conditionals, anything other than `#f` is "truthy" (so empty lists, empty strings, 0 etc are all "true").
- Let expressions: `(let ([x1 e1] [x2 e2] [x3 e3]) body)`.
- Variety of let expressions, with different semantics (`let`, `let*`, `letrec`, `define`).
- Top-level bindings can refer to later bindings, but only inside functions (so they don't actually get used until after they have been defined). But try to avoid it!
- Mutation allowed via `set!`. Very like assignment. (Should only be used when really appropriate)
- `cons` makes pairs. `car/cdr` are like `#1/#2`.
- `lists` are nested conses that end with a cdr of null.
- Use *thunks* to delay evaluation of expressions
- Be careful not to evaluate thunks too soon when defining streams.
- Memoization: Store previous results and reuse on same input to avoid recomputation.

- Documentation for pairs and list: http://docs.racket-lang.org/reference/pairs.html
- `begin` used to sequentially group expressions (useful in conjuction with `set!`).

## Notes on assignment

- Watch out for the numbering! It is hw4, in week 5. All homeworks will "lag behind" from now on.
- Some questions expect you to look up and use specific Racket library functions. Do so!
- Note: `funny-number-stream` needs to be the stream (i.e. the thunk), not a function that would return the stream. Same for `dan-then-dog`.
- In problems 9 and 10, when searching in the vector/list, you need to return the whole *pair*, not just the `cdr`.
- Test your functions! A lot.
- Extra parentheses are NEVER OK. Be careful to use only where needed.
- Remember that you cannot write arithmetic the "normal" way.

## Racket forms for common tasks:

Defining a function:

```
(define (f x y) body)

(define (f x y)
   body)

(define f (lambda (x y) body))

; no-argument function:
(define (f) body)
(define f (lambda () body))
```

Let expressions:

```
(let ([x1 e1]
      [x2 e2]
      [f1 (lambda (x) body)])
   letBody))
```

If expressions:

```
(if testExp
    thenExp
    elseExp)
```

Cond expressions:

```
(cond [test1 exp1]
      [test2 exp2]
```

```
[test3 exp3]
[#t expDefault])
```

7 · flag

 Chad Miller · 15 days ago

★ APPROVED

Some stuff I found while doing the assignment:

- If you see an error message of the form "expected procedure but got (some number)" then you probably accidentally used infix notation like `(i + 1)` or `(x > y)`
- `cond`'s syntax allows `[else expDefault]` for the last expression (I find it more natural)
- Racket has a `when` keyword that acts as an `if` with no else clause (generally useless if you're not using mutation, but I found it handy for problem 10)

9 · flag

 Pavel Lepin COMMUNITY TA · 15 days ago

Thanks for mentioning `when`, Chad. I was doing something unnecessarily byzantine last time instead of using it.

0 · flag

Charilaos Skiadas COMMUNITY TA · 15 days ago

I abused the behavior of `or` and `and` on non-boolean values instead, to avoid those awkward `if`s. Rather pleased with the result, but the peers will probably rip it to shreds.

2 · flag

Fatih Altınok · 11 days ago

Using or to avoid recomputation is great style IMO.

0 · flag

Charilaos Skiadas COMMUNITY TA · 11 days ago

Well it's not so much a question of recomputation, rather than of code repetition or creating an extra let block.

But my "problem" with it, and I know it's ironic since I'm the one who did it, is that it uses non-booleans in a boolean's place, essentially overloading the behavior of boolean operations to do more than they are supposed to. It is something that has been misused since the first time someone wrote `p && *p` in C. These are all shortcuts, perhaps elegant

ones at that, to avoid a lengthier check, but they make the code a tad harder to reason about. Now every time you encounter an "and" or "or", you have to think about whether it is used really as combinator of boolean values, or if they are really non-boolean values that cleverly use that to do their job.

Dynamic typing shouldn't be an excuse to arbitrarily mix types because you can. If you are writing code that would have failed a static typecheck, then you should have a very good reason for doing so. And I'm not sure "but it's much shorter" is necessarily a good reason.

In general, I find the idea that every value in most dynamic languages has a "truthiness" value, i.e. it can be interpreted as a boolean, is really bad, especially seeing how most languages cannot agree on which values should be "falsy". I'm sure each language's proponent can explain why it makes sense for the "truthy" values to be what they are, and perhaps if you never have to think of another language it may work for you, but in a world where we have to often switch between different languages, the truthiness of values is just one more cognitive hurdle we have to go through when trying to read and reason about someone's code, and reading code is already a hard enough process as it is.

end of rant

0 · flag

Fatih Altınok · 11 days ago

Well of course you can avoid recomputation with a let-expression, but the question is, would it be good style?

After seeing benefits of a strongly typed language as ML, it's hard to reason about dynamic languages. But if makers of the language didn't want it used this way, they would simply remove it for non-boolean expressions. It's widely used in Javascript and it's considered good style. I agree there's no such agreed upon behavior for them but as long as you know what you're doing, I think it's fine.

0 · flag

Peter Eriksen COMMUNITY TA · 11 days ago

> *...especially seeing how most languages cannot agree on which values should be "falsy".*

Relevant thread: Conflation between static vs dynamic and strong vs weak typing?.

0 · flag

+ Comment

 ryan davis · 14 days ago

Some more tidbits:

`(thunk ...)` is the same as `(lambda () ...)`.

Curried lambdas:

```
(define ((name creation-args) call-args) ...)
```

Is the same as:

```
(define (name creation-args)
(lambda (call-args) ...))
```

Personally, I like writing my tests inline. You can do that with `(module+ test ...tests...)` as many times as you want throughout your impl file. The run button will run your tests automatically. So will `raco test filename` or `C-c t` in racket-mode.

↑ 1 ↓ · flag

Joshua Snyder · 14 days ago

Thanks! Anything that makes this language more readable and less verbose is greatly appreciated!

↑ 0 ↓ · flag

+ Comment

VISHAL DEEPAK AWATHARE · 12 days ago

how to use the test file? I completed a few problems and with the help of the test file given i copied only those tests in the file and its runs, but how do I know if everything is working? I created an error on purpose but the file compiled any way

↑ 0 ↓ · flag

+ Comment

 ryan davis · 12 days ago

Your test file starts with `(require rackunit)` and defines your tests within a `test-suite` call. It then has `(require rackunit/text-ui)` and runs the tests with `(run-tests tests)`. If you're missing that last line, nothing will happen.

↑ 2 ↓ · flag

VISHAL DEEPAK AWATHARE · 12 days ago

ok forgot the last two lines , thaknx

0 · flag

+ Comment

VISHAL DEEPAK AWATHARE · 11 days ago

ok i have this syntax doubt, considee the case of

```
 (cond [(c1).....]
       ;Want to use a let expression only for the next 3 cond constructs, can give err
or if defined above since c1 must not be true for the let expression to be valid
       [.....]
       [.....]
       [.....]
```

If I create a let expression in between the '[ ]' i get syntax error and due to the use of braces I cant use let expression  at the start of the second 'cond' construct because il have to close it before the close of that construct.(I solved it without the let but it was a little bit more costly)

0 · flag

+ Comment


ryan davis · 11 days ago

The syntax for cond is: `(cond [c1 r1] [c2 r2] ... [cN rN])`.

You can use a `let` in the place of a condition (or result), but it will be harder to read:

```
(cond [#f 'bad] [(let ([x #t]) x) 42])
42
```

0 · flag

VISHAL DEEPAK AWATHARE · 11 days ago

not that, I actually what to define a variable there - to store the value of a vector for the current position that i want to reuse on next three conds(but not the first [c1 r1] since there i check if the length has exceeded or not)

0 · flag

Charilaos Skiadas COMMUNITY TA · 11 days ago

That's like defining something in the middle of a switch statement, or defining something for only some of the clauses in a case expression in SML. I don't think it can be done, and it would be really weird if you could do it (maybe you can do it for the switch statement, I don't know).

Sounds to me like you should be doing a separate if for the first case, then a cond for the others.

↑ 2 ↓ · flag

VISHAL DEEPAK AWATHARE · 11 days ago

now that I think about it like SML way i feel that yeah it would be kinda weird, we cant do the same thing in JAVA too

↑ 0 ↓ · flag

 ryan davis · 10 days ago

```
(if c1 v1 (let (...) (cond ...)))
```

↑ 0 ↓ · flag

+ Comment

 Diana Cook · 9 days ago

Here is a good article on streams from the computer book, Structure and Interpretation of Computer Programs by Abelson, Sussman and Sussman.
http://mitpress.mit.edu/sicp/full-text/book/book-Z-H-24.html#%_sec_3.5

This looks like another good article:
http://srfi.schemers.org/srfi-41/srfi-41.html

↑ 0 ↓ · flag

+ Comment

Sergio Pelin Voloc · 8 days ago

What is the difference between the streams that we create in the homework (the same way as Dan did in the lectures) and the `Streams` from The Racket Reference. Because if I apply `stream?` to any of my homework ones, I get `\#f`, therefore I can't use any `Streams` functions.

Am I doing something wrong?

0 · flag

Charilaos Skiadas COMMUNITY TA · 8 days ago

Yeah they are kind of a similar concept but probably somewhat differently implemented, they certainly are not like the assignment's streams. You're not supposed to use Racket's streams and their functions. You're supposed to simply create appropriate thunks and cons pairs.

0 · flag

Sergio Pelin Voloc · 8 days ago

Ok, thanks!

0 · flag

+ Comment

New post

To ensure a positive and productive discussion, please read our forum posting policies before posting.

B | I | Link | <code> | Pic | Math | Edit: Rich | Preview

Make this post anonymous to other students

Subscribe to this thread at the same time

Add post