1 Predicates

A predicate is a procedure that tests something, returning a boolean.

1. empty? returns whether a list is empty.
2. cons? returns whether a list is cons.
3. > returns whether the first argument is greater than the second.

2 Style and Good Practices

Procedures first and rest have the same function as car and cdr, respectively.

2.1 Comments

Comments (indicated by ;;) are completely ignored by the Racket compiler, but provide helpful information to anyone reading your program. Before writing a procedure, you should comment the input and output specifications, and data definitions for data types that might require them.

2.2 Testing

Every procedure you write should be tested to make sure it behaves as expected. It’s best to write test cases before you write the procedure, to better tailor them to the procedure’s expected behavior rather than its actual behavior. In Racket, you can write a check-expect using the syntax
(check-expect expr val), where expr is the expression whose value you want to check, and val is the value you expect the expression to evaluate to.

When testing a recursive function, you should write test cases to specifically test both the base case(s) and the recursive case(s).

2.3 Errors

Errors are awesome! Although they can be a pain, errors provide helpful information about where and when a program failed its execution. It’s always better for a procedure to throw an error, which will terminate the program immediately, than for it to produce an inaccurate or invalid output, which could upset other parts of the program that use that procedure, and will be much more difficult to find.

3 Recursive Diagrams

A great way to begin writing a recursive procedure is to draw a recursive diagram. There are 5 parts of such a diagram:

1. Overall input: the input to the original function call.
2. Recursive input: the input to the recursive call within the original function call.
3. Recursive output: the output of the recursive call.
4. Ideation state: brainstorm what operations could be made on the recursive output to produce the overall output.
5. Overall output: the expected output of the original function call.

Often, it can be necessary to draw multiple nested recursive diagrams to pinpoint exactly what needs to happen in the ideation state. Look at the lecture slides from today for an example of how to solve a problem by drawing a recursive diagram.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: http://cs.brown.edu/courses/csci0170/feedback.