Everything outside of parens is a comment.
Your source code can double as a markdown document.
I’ve done some pretend syntax highlighting to make it easier to read.

This is a quicksort example which partitions in the middle of the array.

```
(quicksort = ((list : Vector) ->
 “I have to assert that quicksort only works on Vector types”,
 ? ((# list) > 1)
 then (swap = (pv i j ->
       “I’m chosing to do a recursive implementation (swap) instead of a loop”.
       ??
         (((list _ i) < (list _ pv)) and (i < pv))
         then (swap pv (i + 1) j)
         (((list _ j) > (list _ pv)) and (j > pv))
         then (swap pv i (j - 1))
         (((list _ j) < (list _ pv)) and ((list _ i) < (list _ pv)))
         then (tmp = (list _ i),
               “You can swap an index with a new value with (Arr _ index new)”,
               list _ i (list _ j),
               list _ j tmp,
               swap pv i j),
         (quicksort ((@ (@ (list _ 1)))
                     : (Vector | (c -> (? ((# c) = ((# list) - pv)) True))))),
         (quicksort ((@ (@ (list _ (pivot _ 1))))
                     : (Vector | (c -> (? ((# c) = ((# list) - pv)) True))))))))
```

More comments here...
In the recursive quicksort calls, I create an array slice similar to how you can increment an array pointer in C, except you can’t do pointer math in my language. Instead, I take the address of an index in the array, then dereference it as an array that starts with that index.
In C it would be something like

```
      quicksort(((int[])((void*)(&list[pv]))));
```

going from an array index, to an untyped pointer, to another array.
You can technically dereference anything as anything in my language because at the very least, each pointer is guaranteed to point to a live object. But in order to call quicksort on the slice, you have to assert that the thing you’re dereferencing is actually an array with the right amount of indicies. That’s what I’m doing with those two lines in the bottom that start with “:”. Dereferencing gives a Vector “Vector | ...” where its length “# c” is equivalent to the length of the list minus the top or bottom portion “... = # list - pv”