Design is Essentially Reverse Stepping

So it's crucial to practice stepping until it becomes second nature.

When you practice writing out steps, by hand, and check your work with 'step', you will soon master it. Practice on expressions from the course lectures, practice, projects, etc, until you have achieved mastery: reliably and automatically "seeing" the steps. Then when you encounter an expression that represents a common kind of step, it will look like that step.

Let's assume you're very experienced with mapping user-defined functions, so the following stepping summary looks and feels very familiar. We have a Full Design check-expect ...

```
(check-expect (f c) c)
```

... which we can change into a definition by replacing the literal (represented by 'c' here) with a parameter name (represented by 'v' here):

```
(define (f v) v)
```

; (the blue rectangles are all the same except for the highlighted part)

If we map that user-defined function, the steps are:

```
(map f (list a b c d))
```

; →

```
(list (f a) (f b) (f c) (f d))
```

; →

```
(list a b c d)
```

Because that looks and feels so familiar, you see it even if you see just that last expression on its own. In particular, when your goal is to produce a list that you've expressed as ...

```
(list a b c d)
```

... and you've looked for the similarities and differences to see that the elements vary in only one spot, then you automatically think: "map!"

So it can be produced by `(map f (list a b c d))`, for some function `f` described by:

```
(check-expect (f c) c)
```

Such a function might be in the language, or you defined it before. Otherwise, define it:

```
(define (f v) v)
```