Writing Down Large Lists

What does the list containing 0 to 10 look like?

```
(cons 0 (cons 1 (cons 2 (cons 3 (cons 4 (cons 5 (cons 6 (cons 7 (cons 8 (cons 9 (cons 10 empty)))))))))))
```

Here's a shorthand:

```
(list 0 1 2 3 4 5 6 7 8 9 10)
```

The `list` operator takes any number of arguments and constructs a list

Still, DrRacket prints 11 `cons`es

Printing Large Lists

If you change DrRacket's language level to
Beginning Student with List Abbreviations
then DrRacket prints using the shorthand

```
> (list 0 1 2 3 4 5 6 7 8 9 10)
(list 0 1 2 3 4 5 6 7 8 9 10)

> (cons 1 (cons 2 (cons 3 empty)))
(list 1 2 3)
```

When to Change Language Levels

1. You're not tempted to write examples like this:

```
(check-expect (feed-fish (cons 1 (cons 2 empty)))
              2 3)
```

2. Your eyes hurt when you see

```
(cons 1 (cons 2))
```

   because it isn't a `list-of-num`

3. When you see

```
(list 1 2 3)
(cons 1 (cons 2 (cons 3 empty)))
```

   you recognize instantly that they're the same

Don't switch until you understand how `list-of-...` functions match the shape of the data definition

Even Shorter

When you're ready, there's an even shorter shorthand!

```
'(1 2 3)
```

is the same as

```
(list 1 2 3)
```

The apostrophe above doesn't make a symbol—it makes a list because it precedes a parenthesis

Furthermore, the apostrophe gets distributed to everything inside:

```
'(apple banana)
```

is the same as

```
(list 'apple 'banana)
```

For consistency, `'1` is the same as `1`

Even Shorter

Here's a `list-of-lon` using the shorthand:

```
'((1 2 3) (2 4 6 8) (3 9 27))
```

which is the same as

```
(list (list 1 2 3) (list 2 4 6 8) (list 3 9 27))
```

which is the same as

```
(cons (cons 1 (cons 2 (cons 3 empty)))
      (cons (cons 2 (cons 4 (cons 6 (cons 8 empty))))
            (cons (cons 3 (cons 9 (cons 27 empty)))
                  empty)))
```