# Side Effects in Racket

When To Use Side Effects in Racket?

- for input/output
- when the program structure is clearer, e.g. updating a small part of a large structure representing information about the current state of the world
- when needed for efficiency (but don't get carried away with this!)

## Assignment and List Surgery

Racket includes a special form for assignment:

```
(set! var expr)
```

Example:

```
(define x 10)
x
(set! x (+ x 1))
x
```

You can only use `set!` on a variable after it's been defined.

Another special form with side effects is `for` for iteration (which has lots of bells and whistles, although the basic form is straightforward).

There is also a mutable version of cons cells in Racket to build mutable lists. The function `mcons` (instead of `cons`) builds a mutable cell. The functions `set-mcar!` and `set-mcdr!` change the car and cdr of the cell. See Mutable Pairs and Lists in the Racket Guide for more details. (In standard Scheme, you can change any cons cell, and so do surgery on any list, using `set-car!` and `set-cdr!`. This was changed in recent versions of Racket.)

Example:

```
(define x (mcons 3 (mcons 2 null)))
x     ; prints (mcons 3 (mcons 2 '()))
(set-mcar! x 100)
x     ; prints (mcons 100 (mcons 2 '()))
(set-mcdr! x '())
x     ; prints (mcons 100 '())
```

Racket (but not R5RS Scheme) includes structs for defining simple records. See structs.rkt for examples.

Suggestion: if you need a mutable data structure, use structs rather than mutable lists.

When you pass a list or struct to a function in Scheme, a reference to the list or struct is passed, rather than a copy. Thus, if you do any mutations on the list or struct inside the function, the actual parameter will be affected.

```
(define (change-it x)
    (set-mcar! x 'frog))

(define animals (mcons 'squid (mcons 'octopus null)))
(change-it animals)
```

However, consider:

```
(define (test x)
   (write x)
   (set! x 100)
   (write x))

(define y '(1 2 3))
(test y)  ; prints (1 2 3) then 100
```

```
  (write y) ; prints (1 2 3) ... y is unaltered
```