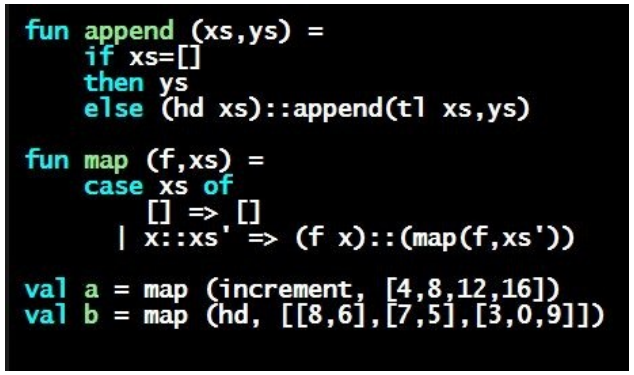

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

fun map (f,xs) =
    case xs of
        [] => []
      | x::xs' => (f x)::(map(f,xs'))

val a = map (increment, [4,8,12,16])
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman

2013

Optional: Variables, Macros, and Hygiene

Another bad macro

Any *function* that doubles its argument is fine for clients

```
(define (dbl x) (+ x x))
(define (dbl x) (* 2 x))
```

- These are equivalent to each other

So macros for doubling are bad style but instructive examples:

```
(define-syntax dbl (syntax-rules()[(dbl x)(+ x x)]))
(define-syntax dbl (syntax-rules()[(dbl x)(* 2 x)]))
```

- These

```
(dbl (begin (print "hi") 42))
```

er:

More examples

Sometimes a macro *should* re-evaluate an argument it is passed

- If not, as in **`dbl`**, then use a local binding as needed:

```
(define-syntax dbl
  (syntax-rules ()
     [(dbl x)
      (let ([y x]) (+ y y))]))
```

Also good style for macros not to have surprising evaluation order

- Good rule of thumb to preserve left-to-right
- Bad exam

```
(define-syntax take
  (syntax-rules (from)
     [(take e1 from e2)
      (- e2 e1)]))
```

Local variables in macros

In C/C++, defining local variables inside macros is unwise

- When needed done with hacks like **`__strange_name34`**

Here is why with a silly example:

- Macro:

```
(define-syntax dbl
  (syntax-rules ()
     [(dbl x) (let ([y 1])
                (* 2 x y))]))
```

- Use:

```
(let ([y 7]) (dbl y))
```

- Naïve expansion:

```
(let ([y 7]) (let ([y 1])
               (* 2 y y)))
```

- But instead Racket "gets it right," which is part of *hygiene*

The other side of hygiene

This also looks like it would do the “wrong” thing

- Macro:

```
(define-syntax dbl
  (syntax-rules ()
     [(dbl x) (* 2 x)]))
```

- Use:

```
(let ([* +]) (dbl 42))
```

- Naïve expan

```
(let ([* +]) (* 2 42))
```

- But again Racket’s *hygienic macros* get this right!

How hygienic macros work

A hygienic macro system:

1. Secretly renames local variables in macros with fresh names
2. Looks up variables used in macros where the macro is defined

Neither of these rules are followed by the “naïve expansion” most macro systems use

- Without hygiene, macros are much more brittle (non-modular)

On rare occasions, hygiene is not what you want

- Racket has somewhat complicated support for that