

Simple Pattern-Based Macros

```
(define-syntax-rule   
)
```

- `define-syntax-rule` indicates a simple-pattern macro definition

Simple Pattern-Based Macros

```
(define-syntax-rule pattern  
  template)
```

- A *pattern* to match
- Produce result from *template*

Simple Pattern-Based Macros

```
(define-syntax-rule (swap a b)  
  )
```

- Pattern for this macro: `(swap a b)`
- Each identifier matches anything in use

```
(swap x y) ⇒ a is x  
           b is y
```

```
(swap 9 (+ 1 7)) ⇒ a is 9  
                  b is (+ 1 7)
```

Simple Pattern-Based Macros

```
(define-syntax-rule (swap a b)
  (let ([tmp b])
    (set! b a)
    (set! a tmp)))
```

Bindings substituted into template to generate the result

```
(swap x y)           ⇒ (let ([tmp y])
                        (set! y x)
                        (set! x tmp))
```

```
(swap 9 (+ 1 7))    ⇒ (let ([tmp (+ 1 7)])
                        (set! (+ 1 7) 9)
                        (set! 9 tmp))
```

Pattern-Based Macros

```
(define-syntax flip  
  )
```

```
(let ([x 0]  
      [y 1])  
  (flip x y))
```

```
(let ([xb (box 0)]  
      [yb (box 1)])  
  (f xb yb))
```

```
(define (f xb yb)  
  (flip in xb yb))
```

Pattern-Based Macros

```
(define-syntax flip  
  )
```

- `define-syntax` indicates a macro definition

Pattern-Based Macros

```
(define-syntax flip  
  (syntax-rules (in)  
    ))
```


- `syntax-rules` means a pattern-matching macro
- `(in)` means that `in` is literal in patterns

Pattern-Based Macros

```
(define-syntax flip
  (syntax-rules (in)
    [pattern template]
    ...
    [pattern template]))
```

- Any number of *patterns* to match
- Produce result from *template* of first match

Pattern-Based Macros

```
(define-syntax flip
  (syntax-rules (in)
    [(flip in a b) ....]
    [(flip a b) ]))
```

Two patterns for this macro

- `(flip in xb yb)` matches first pattern
- `(flip x y)` falls through to second pattern

Pattern-Based Macros

```
(define-syntax flip
  (syntax-rules (in)
    [(flip in a b) (let ([tmp (unbox b)])
                     (set-box! b (unbox a))
                     (set-box! a tmp))]
    [(flip a b) (swap a b)]))
```

```
(flip in xb yb) ⇒ (let ([tmp (unbox yb)])
                    (set-box! yb (unbox xb))
                    (set-box! xb tmp))
```

```
(flip x y) ⇒ (swap x y)
```

Matching Sequences

Some macros need to match sequences

```
(rotate x y)
```

```
(rotate red green blue)
```

```
(rotate front-left  
rear-right  
front-right  
rear-left)
```

Matching Sequences

```
(define-syntax rotate
  (syntax-rules ()
    [(rotate a) (void)]
    [(rotate a b c ...) (begin
                          (swap a b)
                          (rotate b c ...))]))
```

- ... in a pattern: multiple of previous sub-pattern

`(rotate x y z w) ⇒ c is z w`

- ... in a template: multiple instances of previous sub-template

`(rotate x y z w) ⇒ (begin
 (swap x y)
 (rotate y z w))`

Matching Sequences

```
(define-syntax rotate
  (syntax-rules ()
    [(rotate a c ...)
     (shift-to (c ... a) (a c ...))]))
```

```
(define-syntax shift-to
  (syntax-rules ()
    [(shift-to (from0 from ...) (to0 to ...))
     (let ([tmp from0])
       (set! to from) ...
       (set! to0 tmp)) ]))
```

- ... maps over same-sized sequences
- ... duplicates constants paired with sequences

Macro Scope

```
(define-syntax-rule (swap a b)
  (let ([tmp b])
    (set! b a)
    (set! a tmp)))
```

What if we `swap` a variable named `tmp`?

```
(let ([tmp 5]
      [other 6])
  (swap tmp other))      ?  (let ([tmp 5]
                               [other 6])
  (let ([tmp other])
    (set! other tmp)
    (set! tmp tmp)))
```

Macro Scope

```
(define-syntax-rule (swap a b)
  (let ([tmp b])
    (set! b a)
    (set! a tmp)))
```

What if we `swap` a variable named `tmp`?

```
(let ([tmp 5]
      [other 6])
  (swap tmp other))      ?  (let ([tmp 5]
                               [other 6])
  (let ([tmp other])
    (set! other tmp)
    (set! tmp tmp)))
```

This expansion would break scope

Macro Scope

```
(define-syntax-rule (swap a b)
  (let ([tmp b])
    (set! b a)
    (set! a tmp)))
```

What if we `swap` a variable named `tmp`?

```
(let ([tmp 5]
      [other 6])
  (swap tmp other))      ⇒      (let ([tmp 5]
      [other 6])
  (let ([tmp1 other])
    (set! other tmp)
    (set! tmp tmp1)))
```

Rename the introduced binding

Macro Scope: Local Bindings

Macro scope means that local macros work, too:

```
(define (f x)
  (define-syntax swap-with-arg
    (syntax-rules ()
      [(swap-with-arg y) (swap x y)])))
```

```
(let ([z 12]
      [x 10])
  ; Swaps z with original x:
  (swap-with-arg z))
```

)

Implicit Syntactic Forms

To change functions:

```
(define-syntax-rule (lambda ...) ...)
```

To change function calls?

```
(define-syntax-rule (%app ...) ...)
```

```
(expr1 ... exprN)
```

is implicitly

```
(%app expr1 ... exprN)
```

Implicit Syntactic Forms

```
#lang s-exp path  
form1  
...  
formN
```

is implicitly

```
#lang s-exp path  
(#%module-begin  
  form1  
  ...  
  formN)
```

Transformer Definitions

In general, `define-syntax` binds a transformer procedure:

```
(define-syntax swap  
  (syntax-rules . . . . .))
```

⇒

```
(define-syntax swap  
  (lambda (stx)  
    use syntax-object primitives to  
    match stx and generate result  
  ))
```

Matching Syntax and Having It, Too

`syntax-case` and `#'` combine patterns and computation

```
(syntax-case stx-expr ()  
  [pattern result-expr]  
  ...  
  [pattern result-expr])  
  
#' template
```

Matching Syntax and Having It, Too

```
(define-syntax-rule (swap a b)
  (let ([tmp b])
    (set! b a)
    (set! a tmp)))
```

⇒

```
(define-syntax swap
  (lambda (stx)
    (syntax-case stx ()
      [(swap1 a b) #'(let ([tmp b])
                        (set! b a)
                        (set! a tmp))]))))
```

Matching Syntax and Having It, Too

Check for identifiers before expanding:

```
(define-syntax swap
  (lambda (stx)
    (syntax-case stx ()
      [(swap a b)
       (if (and (identifier? #'a)
                 (identifier? #'b))
           #'(let ([tmp b])
                (set! b a)
                (set! a tmp))
           (raise-syntax-error
            'swap "needs identifiers"
            stx))]))))
```