# Macros by example

Matthew Might
**University of Utah**
matt.might.net
www.ucombinator.org

# Objectives

- What are macros?
- Unhygienic macros
- Hygienic macros
- Next-time: Mixed hygiene

# Compiler topics

- Implementing record & structs
- Implementing pattern-matching

# What is a macro?

A **macro** expands code into code.

# What is a macro?

- A set of re-write rules for syntax; or
- A function from ASTs to ASTs; or
- A function from ASTs and environments

# The Lisp Philosophy

- Domain-specific languages offer better reasoning
- Define domain-specific language for problem
- Solve problem with the domain-specific language

# Code = Data = Code

- S-Expressions are abstract syntax tree
- S-Expressions are primary data type
- In Scheme/Lisp, all functions are macros
- In Scheme/Lisp, all macros are functions

# Notation

```
'e = (quote e)

`e = (quasiquote e)

,e = (unquote e)

,@e = (unquote-splicing e)
```

# Example

```
‘(lambda (v) v)
=
(cons ‘lambda
 (cons (list ‘v)
  (cons ‘v ‘())))
```

# In general

‘() = ‘()

‘(e1 e2 …) = (cons ‘e1 ‘(e2 …))

‘(e1 . e2) = (cons ‘e1 ‘e2)

# Quasiquoting

`() = `()

`(e1 e2 …) = (cons `e1 `(e2 …))

`(e1 . e2) = (cons `e1 `e2)

`,e = e

`(,@e1 e2 …) = (append e1 `(e2 …))

# Classic macros

# define-macro (Gambit)

```
(define-macro (name arg ...) exp)
```

# Example

```
(define-macro (magnitude x y)
 `(sqrt (+ (* ,x ,x) (* ,y ,y))))
```

# Example

```
(define-macro (magnitude x y)
 `(sqrt (+ (* ,x ,x) (* ,y ,y))))
```

```
(magnitude 3 4)
=>
(sqrt (+ (* 3 3) (* 4 4)))
```

# Example

```
(define-macro (magnitude x y)
 `(sqrt (+ (* ,x ,x) (* ,y ,y))))
```

```
(magnitude (read) (read))
=>
(sqrt (+ (* (read) (read)) (* (read) (read))))
```

# Example

```
(define-macro (magnitude x y)
 `(let* ((x2 ,x)
         (y2 ,y))
    (sqrt (+ (* x2 x2) (* y2 y2)))))
```

# Example

```
(define-macro (magnitude x y)
 `(let* ((x2 ,x)
         (y2 ,y))
    (sqrt (+ (* x2 x2) (* y2 y2)))))
```

```
(magnitude 3 x2)
```

=>

```
(let* ((x2 3)
       (y2 x2))
  (sqrt (+ (* x2 x2) (* y2 y2)))))
```

# Example

```
(define-macro (magnitude x y)
 (let (($x (gensym 'x))
       ($y (gensym 'y)))
`(let* ((,$x ,x)
        (,$y ,y))
  (sqrt (+ (* ,$x ,$x) (* ,$y ,$y))))))
```

Surely, it’s correct now.

# Counter-example

```
(let ((sqrt log))
 (magnitude 10 20)))
```

# The “fix”

```
(define-macro (magnitude x y)
 (let (($x (gensym ‘x))
       ($y (gensym ‘y)))
`(let* ((,$x ,x)
        (,$y ,y))
  (‘,sqrt (‘,+ (‘,* ,$x ,$x) (‘,* ,$y ,$y))))))
```

# Problems

- Works for the interpreter, not compiler
- Breaks on special forms, e.g. , `lambda`

# Concepts

- `gensym`
- `string->symbol`
- `symbol->string`

# Extended examples

# Hygienic macros with syntax-rules

# Example

```
(define-syntax magnitude
  (syntax-rules ()
    ((magnitude x y)
      ; =>
     (let* ((x2 x)
            (y2 y))
       (sqrt (+ (* x2 x2) (* y2 y2)))))))
```

# Syntax

```
(define-syntax name
  (syntax-rules (keyword ...)
   ((_ pat ...)  expansion)
   ...))
```

# Examples