

# Parsing

```
{defun {f x}
  {- 20 {+ {+ x x}
           {+ x x}}}}
```

vs

```
(fundef 'f 'x
  (sub (num 20) (add (add (id 'x)
                          (id 'x))
                     (add (id 'x)
                          (id 'x))))))
```

# Parsing

```
{def fun {f x}
  {- 20 {+ {+ x x}
           {+ x x}}}}
```

What we want to write

# Parsing

```
` { def fun { f x }  
    { - 20 { + { + x x }  
            { + x x } } } }
```

What we have to write

# The rules of Quasiquote

``{<exp> ...}`  $\Rightarrow_{qq}$  `(list `<exp> ...)`

``<id>`  $\Rightarrow_{qq}$  ``<id>`

``<number>`  $\Rightarrow_{qq}$  `<number>`

# The rules of Quasiquote

``{<exp> ...}`  $\Rightarrow_{qq}$  `(list `<exp> ...)`

``<id>`  $\Rightarrow_{qq}$  ``<id>`

``<number>`  $\Rightarrow_{qq}$  `<number>`

For example:

```
`{+ 3 {- x y}}  
 $\Rightarrow_{qq}$  (list `+ `3 `{- x y})  
 $\Rightarrow_{qq}$  (list + 3 {- x y})  
 $\Rightarrow_{qq}$  (list + 3 (list - x y))
```

# Writing a parser

```
(test (parse 1) (num 1))  
(test (parse 'z) (id 'z))  
(test (parse '{+ 1 2}') (add (num 1) (num 2)))  
...
```

# Writing a parser

```
(define (parse exp)
  (cond
    [(number? exp) (num exp)]
    [(symbol? exp) (id exp)]
    [(pair? exp)
     (case (car exp)
       [(+)
        (check-pieces exp 3 "add")
        (add (parse (list-ref exp 1))
              (parse (list-ref exp 2)))]
       ; middle bit shown on next slide
       )]
    [else
     (parse-error "an expression" exp)]))
```

# Writing a parser

```
[ (-)
  (check-pieces exp 3 "sub")
  (sub (parse (list-ref exp 1))
       (parse (list-ref exp 2))))]

[ (with)
  (check-pieces exp 3 "with")
  (check-pieces (list-ref exp 1) 2 "with binder")
  (with (list-ref (list-ref exp 1) 0)
        (parse (list-ref (list-ref exp 1) 1))
        (parse (list-ref exp 2))))]

[else
  (unless (symbol? (car exp))
    (parse-error "an expression" exp))
  (check-pieces exp 2 "app")
  (app (list-ref exp 0)
       (parse (list-ref exp 1))))]
```



# Writing a parser

```
(define (check-pieces expression size what)
  (unless (and (list? expression)
               (= (length expression) size))
    (parse-error what expression)))
```

```
(define (parse-error what expression)
  (error 'parser
         "expected: ~a, found:\n~a"
         what
         (pretty-format expression 30)))
```

# The actual rules of Quasiquote

``{<exp> ...}`  $\Rightarrow_{qq}$  `(list `<exp> ...)`

``<id>`  $\Rightarrow_{qq}$  ``<id>`

``<number>`  $\Rightarrow_{qq}$  `<number>`

``,<exp>`  $\Rightarrow_{qq}$  `<exp>`

# Abstracting over programs

```
(define (n-additions n)
  (cond
    [(zero? n)
     `3]
    [else
     `{+ 1 , (n-additions (sub1 n)) }]))
```

# Abstracting over programs

```
(define (n-additions n)
  (cond
    [(zero? n)
     `3]
    [else
     `{+ 1 , (n-additions (sub1 n))}])))
```

```
(n-additions 2)
```

```
=> `{+ 1 , (n-additions 1)}
```

```
=>qq (list `+ 1 (n-additions 1))
```

```
=> (list `+ 1 `{+ 1 , (n-additions 0)})
```

```
=>qq (list `+ 1 (list `+ 1 (n-additions 0)))
```

```
=> (list `+ 1 (list `+ 1 3))
```

```
aka `{+ 1 {+ 1 3}}
```

# Quote vs. Quasiquote

' {<exp> ...} =><sub>q</sub> (list ' <exp> ...)

' <id> =><sub>q</sub> ' <id>

' <number> =><sub>q</sub> <number>

~~' , <exp> =><sub>q</sub> <exp>~~

# Abbreviations

``<exp> = {quasiquote <exp>}`

`,<exp> = {unquote <exp>}`

`'<exp> = {quote <exp>}`

# Quoting mistakes

```
'{a 'b}
```

```
=>q (list 'a ''b)
```

```
= (list 'a '{quote b})
```

```
=>q (list 'a (list 'quote 'b))
```

# Quoting mistakes

```
'{a , (n-additions 2)}
```

```
=>q (list 'a ' , (n-additions 2))
```

```
= (list 'a '{unquote (n-additions 2)})
```

```
=>q (list 'a  
        (list 'unquote  
              '(n-additions 2)))
```

```
=>q (list 'a  
        (list 'unquote  
              (list 'n-additions 2)))
```