

3 ways to use local:

- eliminate identical recursive calls
- encapsulate 2 or more definitions
- improve code clarity by giving names to intermediate values

refactoring

- a fundamental technique for improving code
- take a correct running program, refactor to improve structure

3 ways to use local

- achieved through refactoring
- or achieved when initially writing code

	Eliminate...	Encapsulate 2...	Give names to vals
Via refactoring			
While originally writing code			


```
(define (render-bst t)
  (cond [(false? t) MTTREE]
        [else
         (above (render-key-val (node-key t) (node-val t))
                  (lines (image-width (render-bst (node-l t)))
                          (image-width (render-bst (node-r t))))
                  (beside (render-bst (node-l t))
                          (render-bst (node-r t)))))]))
```

1. identify duplicate recursive calls
2. find nearest enclosing conditional branch
3. add local there
4. move call to local definition
5. and replace original call with local constant name

```

(@htdf all-labels--region all-labels--lor)
(@signature Region -> ListOfString)
(@signature ListOfRegion -> ListOfString)
;; produce labels of all regions in region (including root)
(check-expect (all-labels--lor empty) '())
(check-expect (all-labels--region S1) (list "one"))
(check-expect (all-labels--lor LOR123) (list "one" "two" "three"))
(check-expect (all-labels--region G4)
  (list "one" "two" "three"
        "four" "five" "six"))

(@template-origin Region)

(define (all-labels--region r)
  (cond [(single? r) (list (single-label r))]
        [else
         (all-labels--lor (group-subs r))]))

(@template-origin ListOfRegion)

(define (all-labels--lor lor)
  (cond [(empty? lor) empty]
        [else
         (append (all-labels--region (first lor))
                  (all-labels--lor (rest lor)))]))

```

```

(htdf all-labels)
(signature Region -> ListOfString)
;; produce labels of all regions in region (including root)
(check-expect (all-labels L1) (list "one"))
(check-expect (all-labels I4) (list "one" "two" "three" "four" "five" "six"))

(template-origin encapsulated Region ListOfRegion)

(define (all-labels r)
  (local [(define (all-labels--region r)
              (cond [(leaf? r) (list (leaf-label r))]
                    [else      (all-labels--lor (inner-sub r))]))

          (define (all-labels--lor lor)
            (cond [(empty? lor) empty]
                  [else
                   (append (all-labels--region (first lor))
                           (all-labels--lor (rest lor)))])])

    (all-labels--region r)))

```

```

(@htdf all-with-color--region all-with-color--lor)
(@signature Color Region -> ListOfRegion)
(@signature Color ListOfRegion -> ListOfRegion)
;; produce all regions with given color
(check-expect (all-with-color--lor "red" empty) '())
(check-expect (all-with-color--region "red" S1) (list S1))
(check-expect (all-with-color--region "blue" S1) '())
(check-expect (all-with-color--lor "red" LOR123) (list S1))
(check-expect
  (all-with-color--region "red"
    (make-group "blue"
      (list G4
        (make-single "X" 90 "red")
        (list G1 S1 (make-single "X" 90 "red")))))
    (list G1 S1 (make-single "X" 90 "red"))))

(@template-origin Region)

(define (all-with-color--region c r)
  (cond [(single? r) (if (string=? (single-color r) c) (list r) '())
        [else
         (if (string=? (group-color r) c)
             (cons r (all-with-color--lor c (group-subst r)))
             (all-with-color--lor c (group-subst r))))])

(@template-origin ListOfRegion)

(define (all-with-color--lor c lor)
  (cond [(empty? lor) empty]
        [else
         (append (all-with-color--region c (first lor))
                   (all-with-color--lor c (rest lor))))])

```

```

#| Refactor using local to encapsulate. |#

(@htdf all-with-color)
(@signature Color Region -> ListOfRegion)
;; produce all regions with given color
(check-expect (all-with-color "red" S1) (list S1))
(check-expect (all-with-color "blue" S1) empty)
(check-expect (all-with-color "red"
                           (make-group "blue"
                                         (list G4
                                               (make-single "X" 90 "red"))))
              (list G1 S1 (make-single "X" 90 "red")))

(@template-origin Region ListOfRegion encapsulated)

(define (all-with-color c r)
  (local [(define (all-with-color--region c r)
                (cond [(single? r)
                       (if (string=? (single-color r) c) (list r) empty)]
                      [else
                       (if (string=? (group-color r) c)
                           (cons r (all-with-color--lor c (group-subst r)))
                           (all-with-color--lor c (group-subst r))))])

        (define (all-with-color--lor c lor)
          (cond [(empty? lor) empty]
                [else
                 (append (all-with-color--region c (first lor))
                         (all-with-color--lor c (rest lor))))])

        (all-with-color--region c r))

```