Macros
Syntax Objects

Based on sections 3.1, 8.0, 8.1 and 8.2 from TSPL

1

Macros

- Macros enable us to extend a language.
- Macros are Racket procedures that take in code and output Racket code
- All macro transformation occurs before any normal execution occurs

2

Keyword Bindings

- A macro needs a name.
- This name will be the name of the expanded syntax of our language.
- We will use **`define-syntax`** to do so.
- Example: **`(define-syntax when …)`**
- We can then use **`when`** in our language.

3

Syntax-Rules Transformers

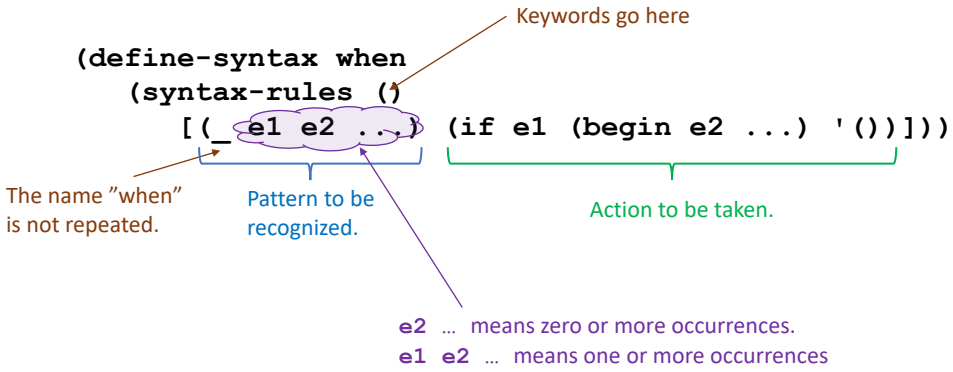
- We will use syntax-rules to extend Racket.
- There are other ways of doing so, however, this method is powerful.
- Syntax-rules takes patterns that are to be recognized and templates that should be returned.
- The specification is as follows:

```
(syntax-rules (literal-id ...)
    [(id . pattern)] template ...)
```

- Which of course is super helpful.
- Let's figure this out by example.

4

When



5

Let

- As we have seen, let is not necessary.
- It can be implemented in terms of a lambda-app expression.

```
(define-syntax let
  (syntax-rules ()
    [(_ ((s1 e1) ...) b1 b2 ...) ; must have at least one body
     ((lambda (s1 ...) b1 b2 ...) e1 ...)])) ; may not have
                                              parameters nor arguments.
```

6

Let*

- Recursion! Yippee!!!

```
(define-syntax let*
  (syntax-rules ()
    [(_ () b1 b2 ...)
     (let () b1 b2 ...)]
    [(_ ((s1 e1) (s2 e2) ...) b1 b2 ...)
     (let ((s1 e1)) (let* ((s2 e2) ...) b1 b2 ...))]))
```

7

Cond

- Use of keyword “else”
- Ordering of clauses is important

```
(define-syntax cond
  (syntax-rules (else)
    [(_ (else e1 e2 ...)) (begin e1 e2 ...)]
    [(_ (e0 e1 e2 ...))   (if e0 (begin e1 e2 ...))]
    [(_ (e0 e1 e2 ...) c1 c2 ...)
     (if e0 (begin e1 e2 ...) (cond c1 c2 ...))]))
```

8

Class exercise

- Define a macro for Boolean **`or`**
- Define a macro for Boolean **`and`**
- Define a macro for **`while`** loops

9