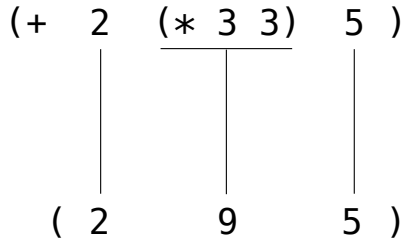
Interpreters

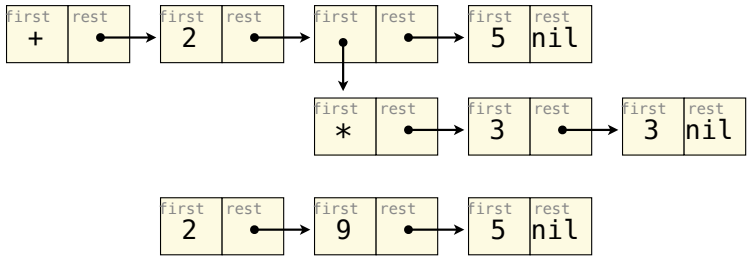
Announcements

Lab 10 Review

Evaluating Operands



Representation as Pairs

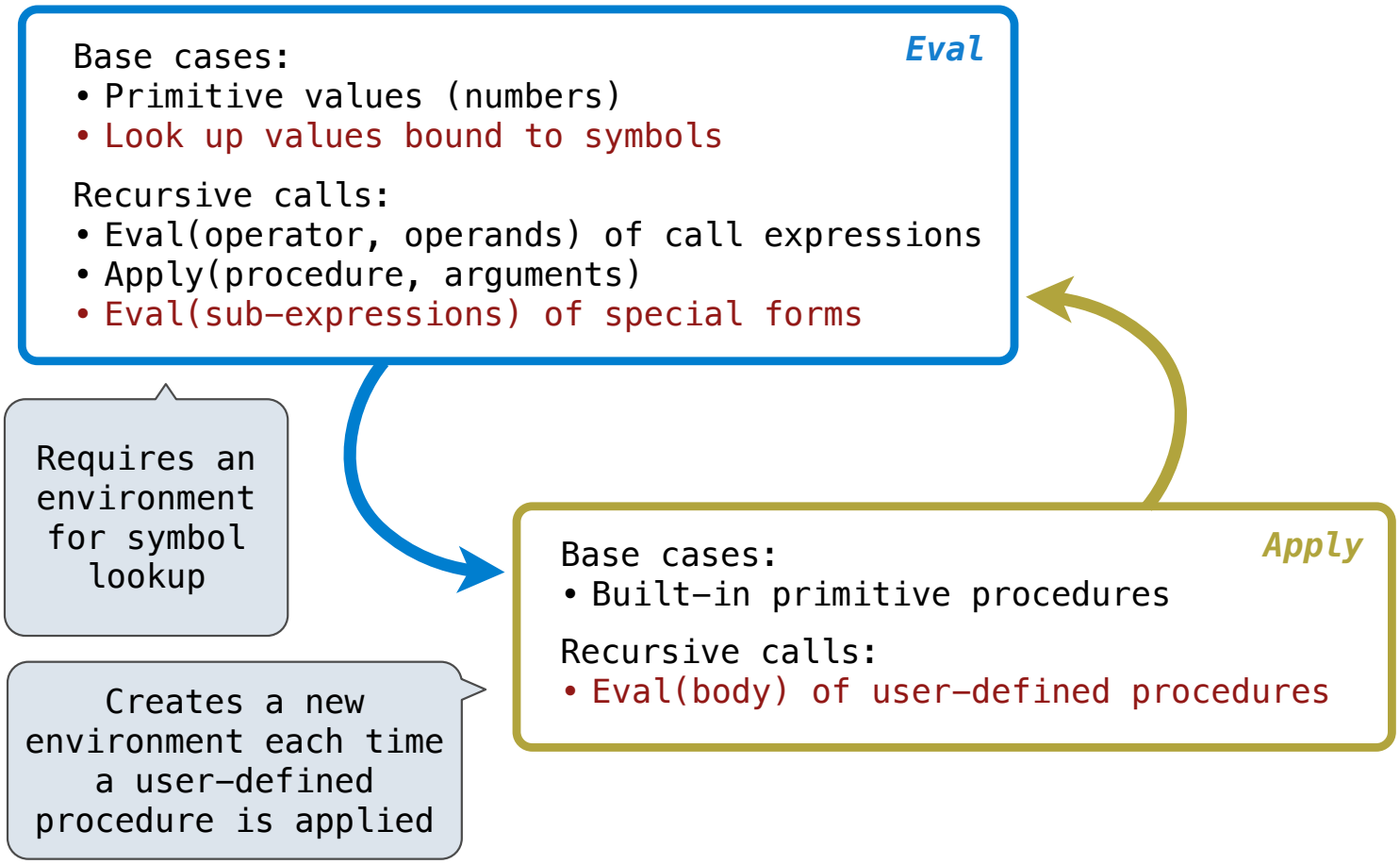


From lecture 2: ... evaluate the operands to get the arguments ...

(Demo)

Interpreting Scheme

The Structure of an Interpreter



Project 4

Pairs in Project 4: Scheme

https://cs61a.org/proj/scheme/

Tokenization/Parsing: Converts text into Python representation of Scheme expressions:

- Numbers are represented as numbers
- Symbols are represented as strings
- Lists are represented as instances of the Pair class (Demo)

Evaluation: Converts Scheme expressions to values while executing side effects:

- scheme_eval(expr, env) returns the value of an expression in an environment
- scheme_apply(procedure, args) applies a procedure to its arguments
- The Python function scheme_apply returns the return value of the procedure it applies

(Demo)

Discussion Question: The Symbol of a Define Expression

Return the symbol of a define expression. There are two formats for define expressions:
(define **x** (+ 2 3)) or (define (**f** x) (+ x 3))

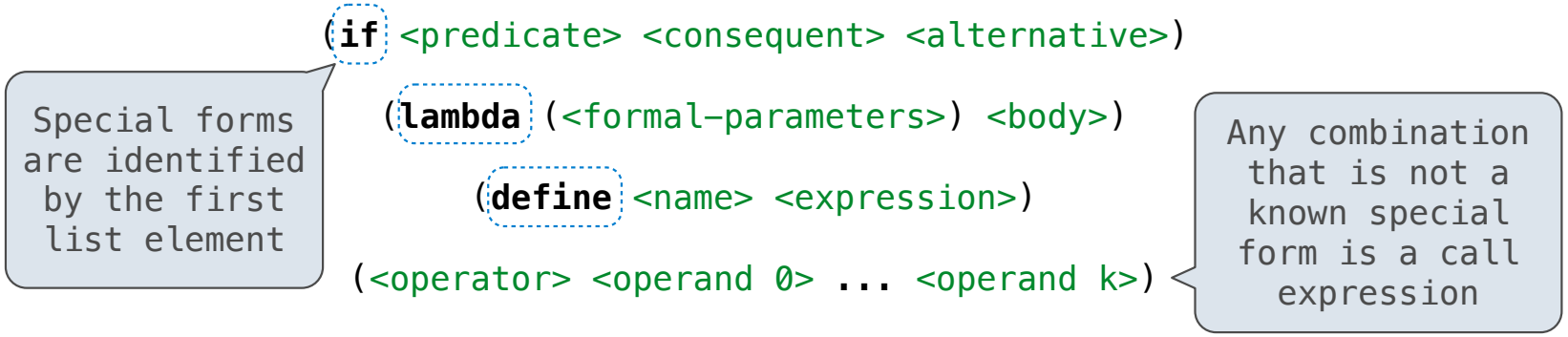
```
def symbol(exp):
    """Given a define expression exp, return the symbol defined.
    >>> def_x = read_line("(define x (+ 2 3))")
    >>> def_f = read_line("(define (f x) (+ x 3))")
    >>> symbol(def_x)
    'x'
    >>> symbol(def_f)
    'f'
    """
    assert exp.first == 'define' and exp.rest is not nil and exp.rest.rest is not nil
    signature = ____exp.rest.first____
    if scheme_symbolp(signature):
        return signature
    else:
        return ____signature.first____
```

Special Forms

Scheme Evaluation

The scheme_eval function choose behavior based on expression form:

- Symbols are looked up in the current environment
- Self-evaluating expressions are returned as values
- All other legal expressions are represented as Scheme lists, called combinations




```
(define (demo s) (if (null? s) '(3) (cons (car s) (demo (cdr s))) ))

(demo (list 1 2))
```

Lambda Expressions

Lambda Expressions

Lambda expressions evaluate to user-defined procedures

```
(lambda (<formal-parameters>) <body>)
```

```
(lambda (x) (* x x))
```

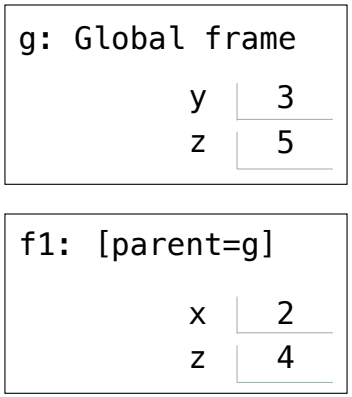
```
class LambdaProcedure:
    def __init__(self, formals, body, env):
        self.formals = formals ..... A scheme list of symbols
        self.body = body ..... A scheme list of expressions
        self.env = env ..... A Frame instance
```

Frames and Environments

A frame represents an environment by having a parent frame

Frames are Python instances with methods **lookup** and **define**

In Project 4, Frames do not hold return values



(Demo)