61A Lecture 27

Wednesday, November 5

Announcements

Announcements

- Homework 7 due Wednesday 4/8 @ 11:59pm

Announcements

- Homework 7 due Wednesday 4/8 @ 11:59pm
  - Homework party Tuesday 4/7 5pm–6:30pm in 2050 VLSB

Announcements

- Homework 7 due Wednesday 4/8 @ 11:59pm
  - Homework party Tuesday 4/7 5pm–6:30pm in 2050 VLSB
- Project 1, 2, & 3 composition revisions due Monday 4/13 @ 11:59pm

Announcements

- Homework 7 due Wednesday 4/8 @ 11:59pm
  - Homework party Tuesday 4/7 5pm–6:30pm in 2050 VLSB
- Project 1, 2, & 3 composition revisions due Monday 4/13 @ 11:59pm
- Quiz 2 released Tuesday 4/7 & due Thursday 4/9 @ 11:59pm

Announcements

- Homework 7 due Wednesday 4/8 @ 11:59pm
  - Homework party Tuesday 4/7 5pm–6:30pm in 2050 VLSB
- Project 1, 2, & 3 composition revisions due Monday 4/13 @ 11:59pm
- Quiz 2 released Tuesday 4/7 & due Thursday 4/9 @ 11:59pm
  - Open note, open interpreter, closed classmates, closed Internet

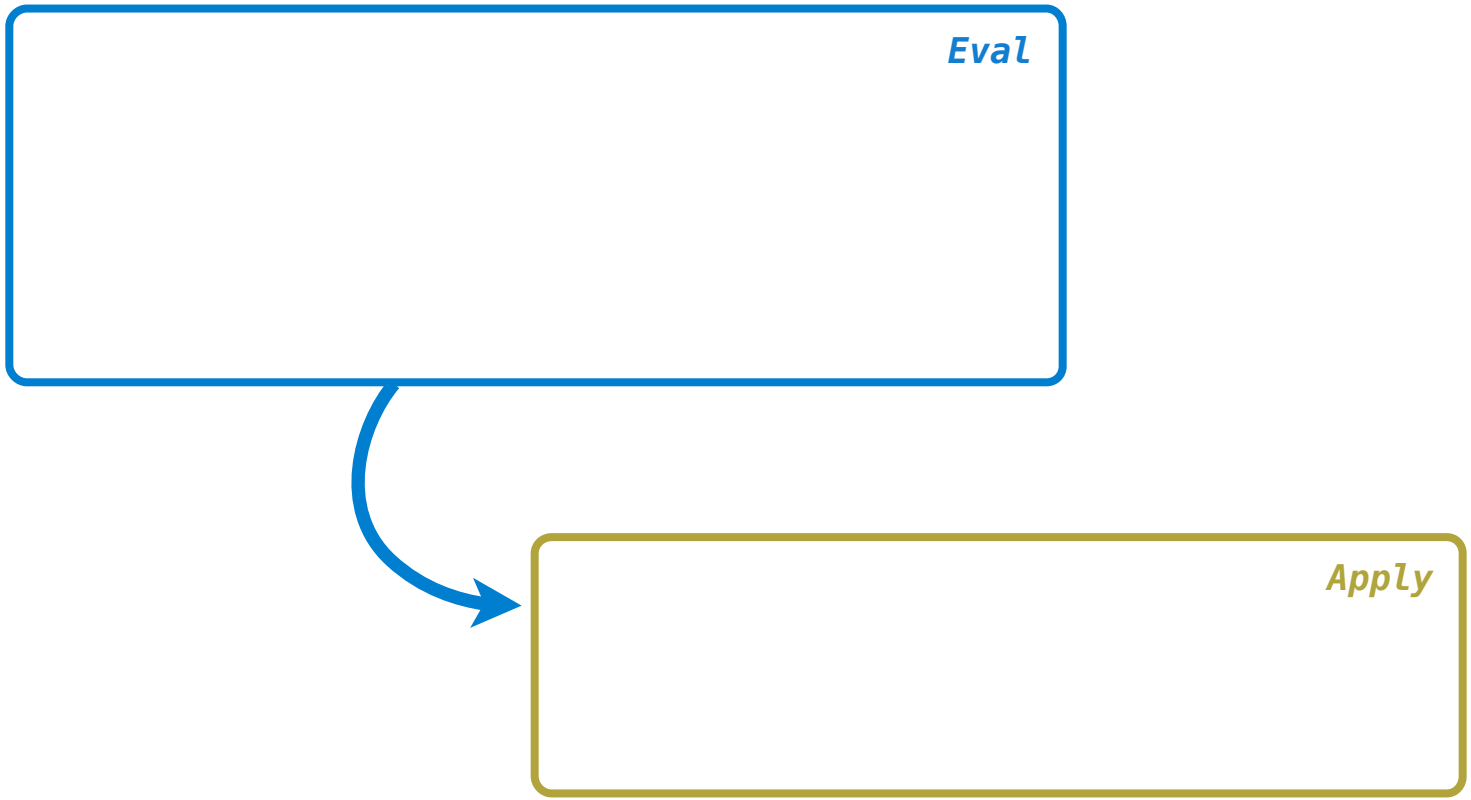
Announcements

- Homework 7 due Wednesday 4/8 @ 11:59pm
  - Homework party Tuesday 4/7 5pm–6:30pm in 2050 VLSB
- Project 1, 2, & 3 composition revisions due Monday 4/13 @ 11:59pm
- Quiz 2 released Tuesday 4/7 & due Thursday 4/9 @ 11:59pm
  - Open note, open interpreter, closed classmates, closed Internet
- Project 4 due Thursday 4/23 @ 11:59pm (Big!)

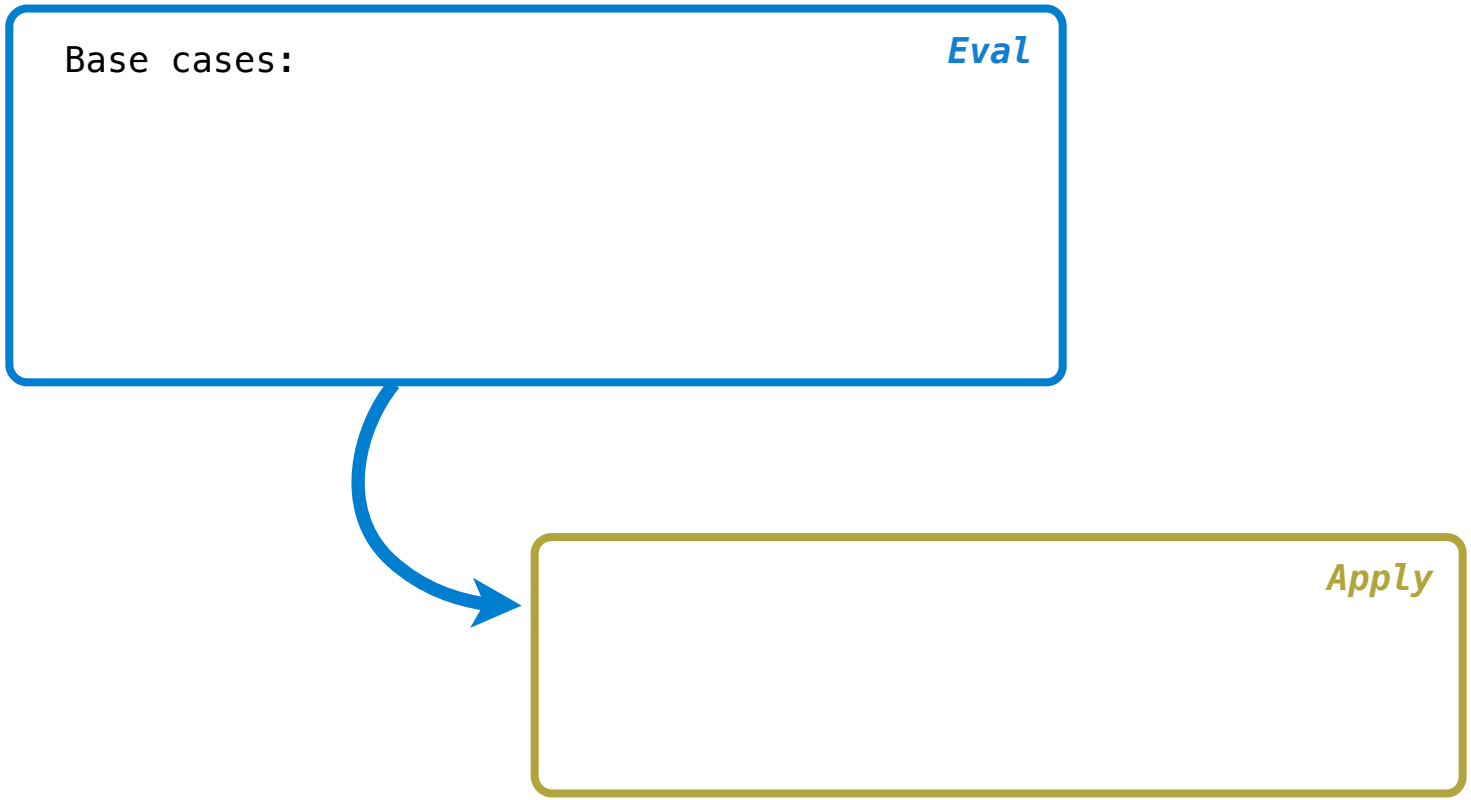
Interpreting Scheme

The Structure of an Interpreter

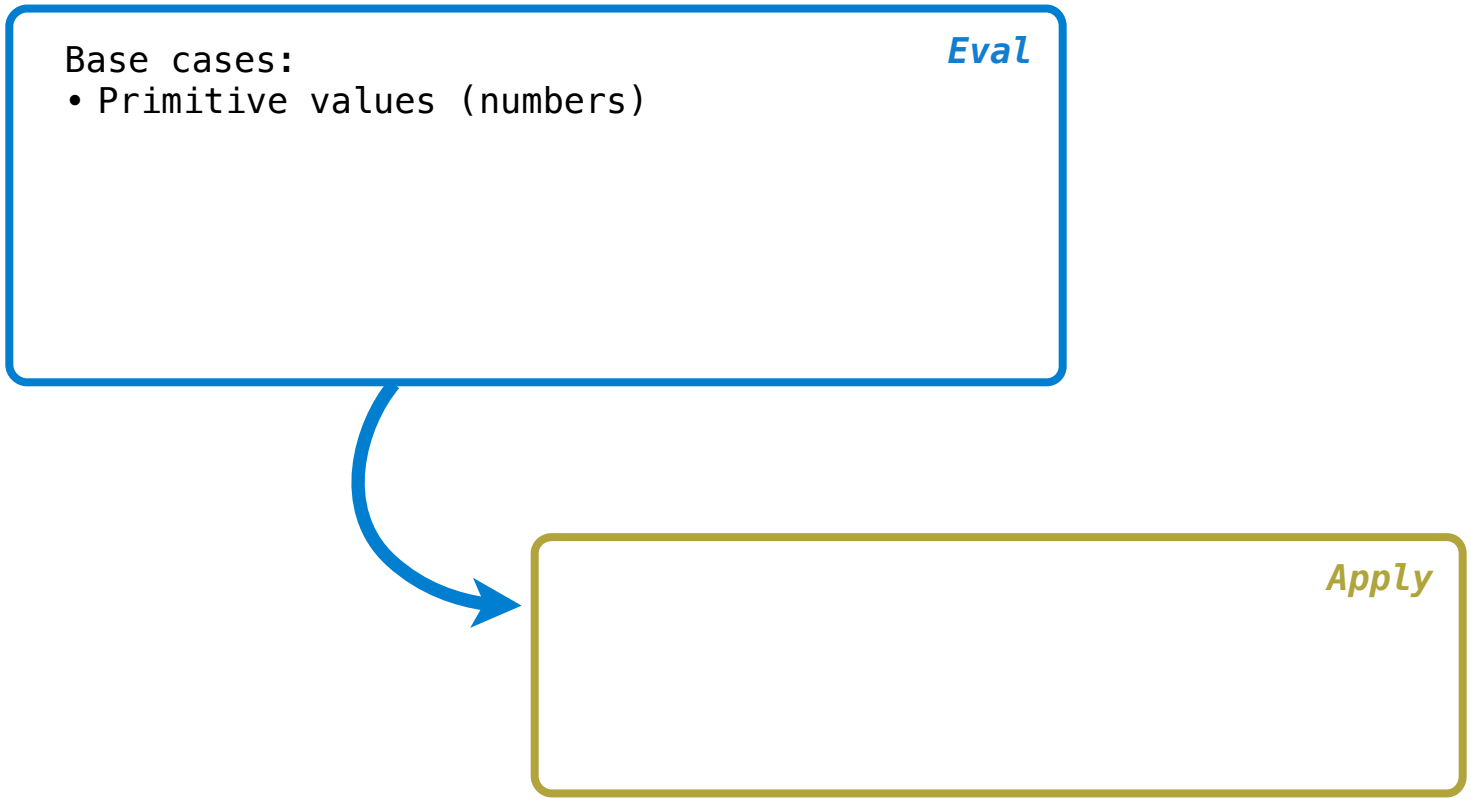
The Structure of an Interpreter



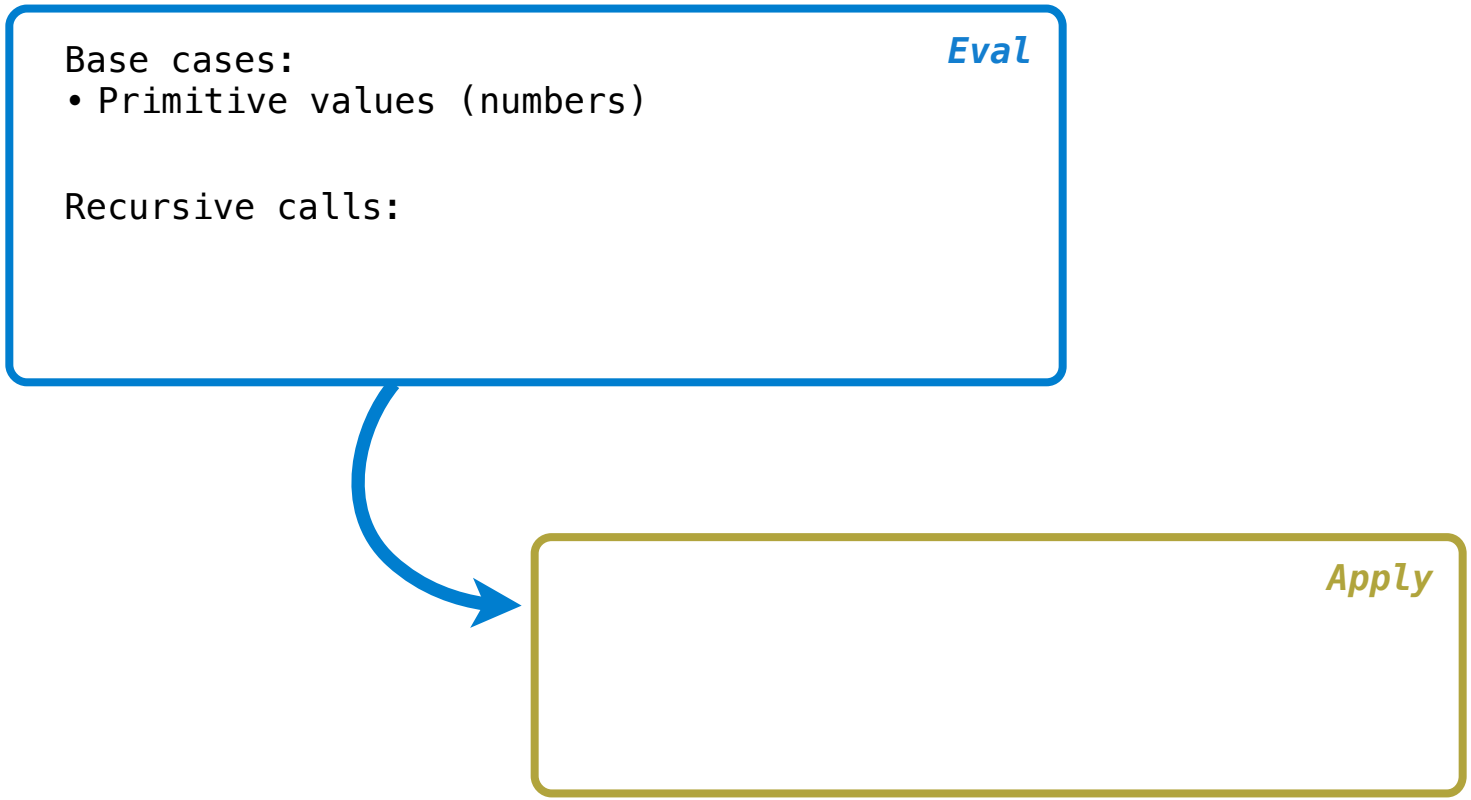
The Structure of an Interpreter



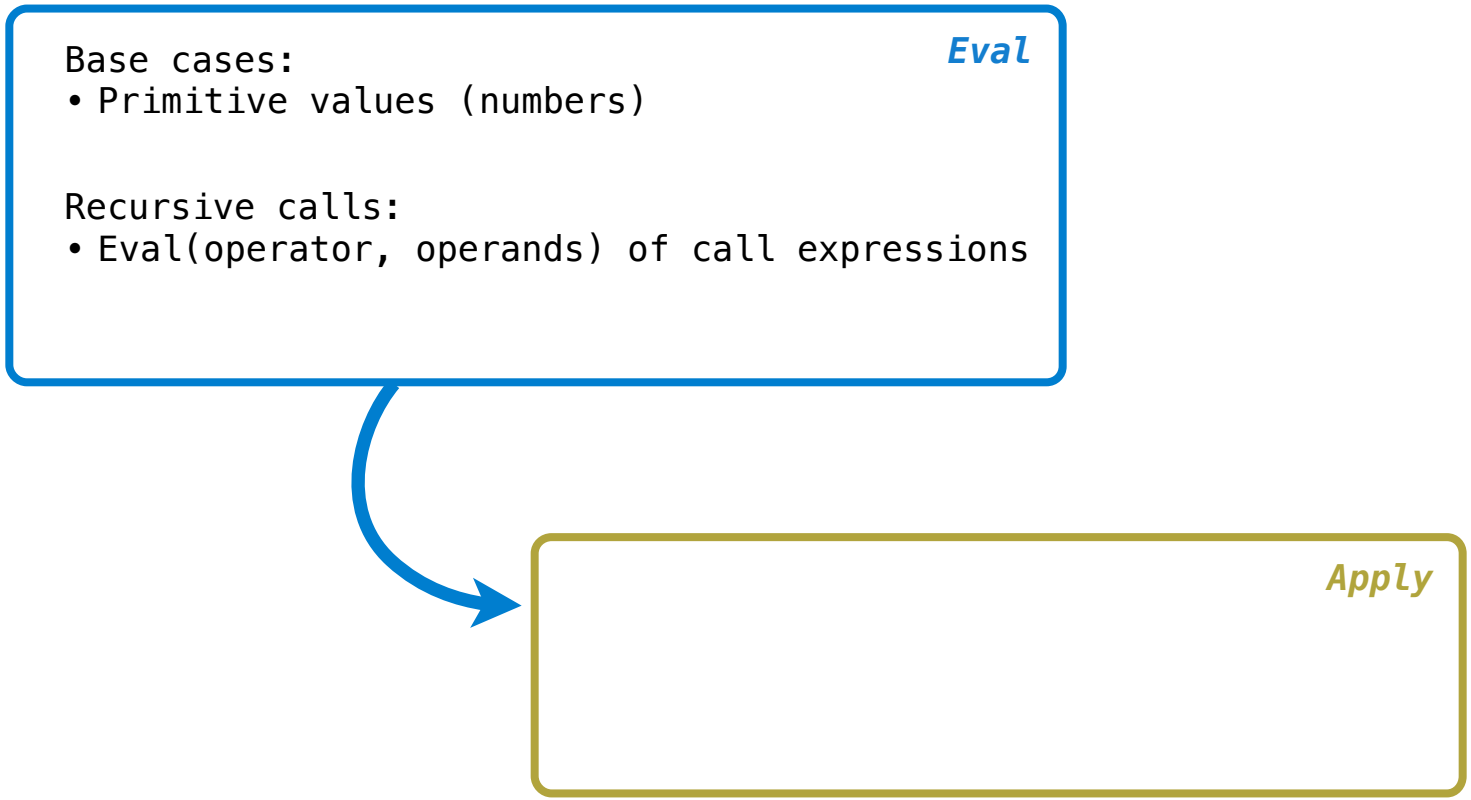
The Structure of an Interpreter



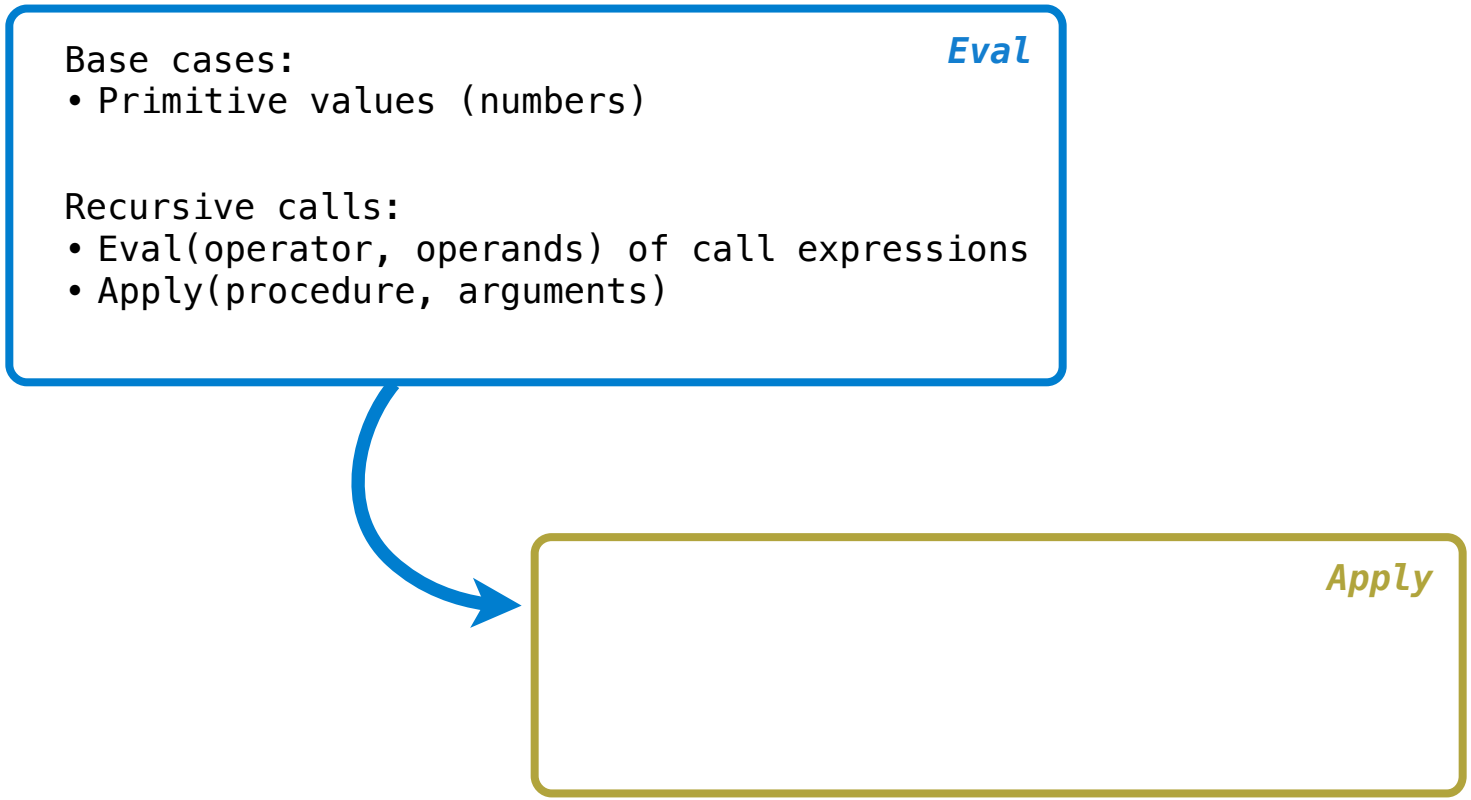
The Structure of an Interpreter



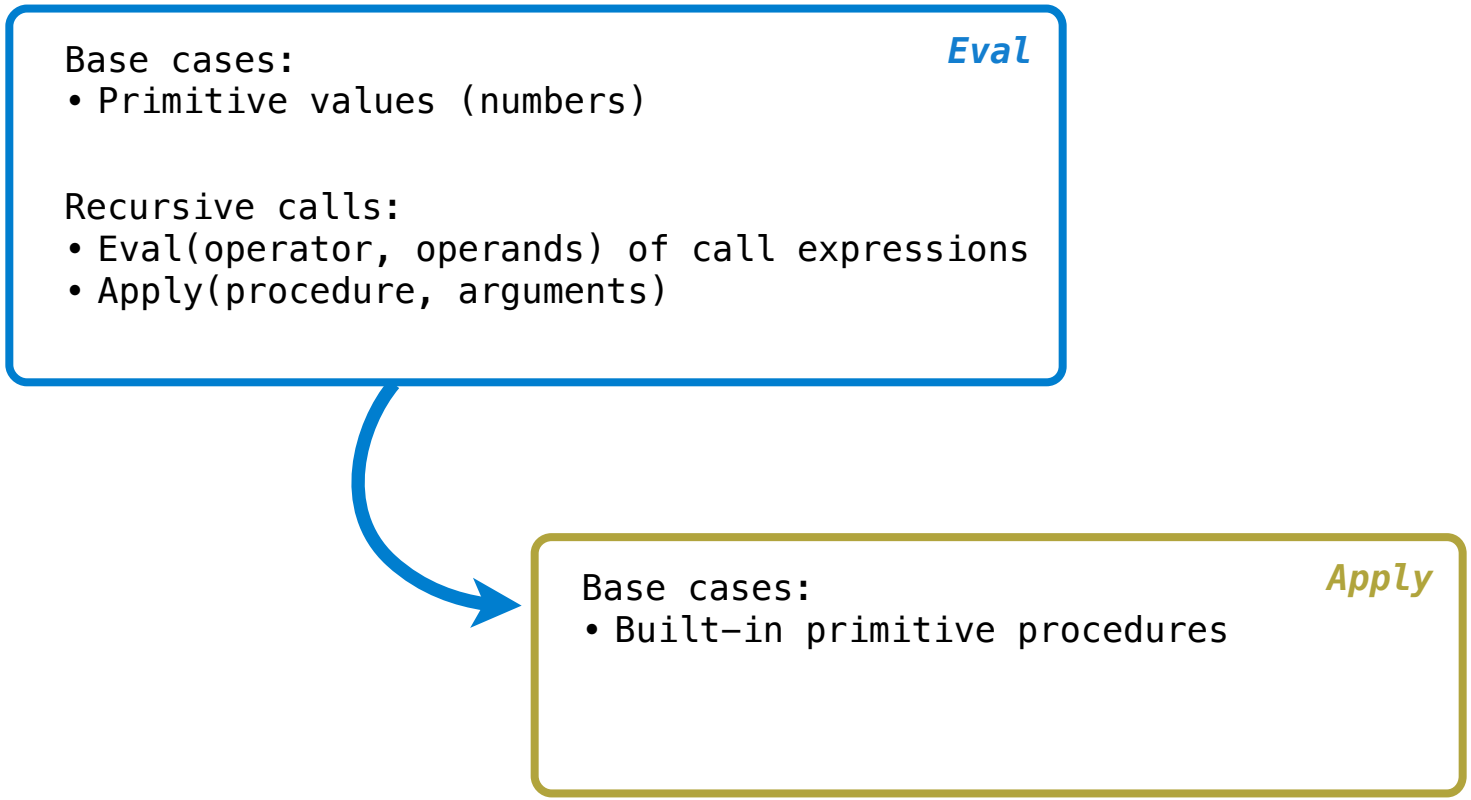
The Structure of an Interpreter



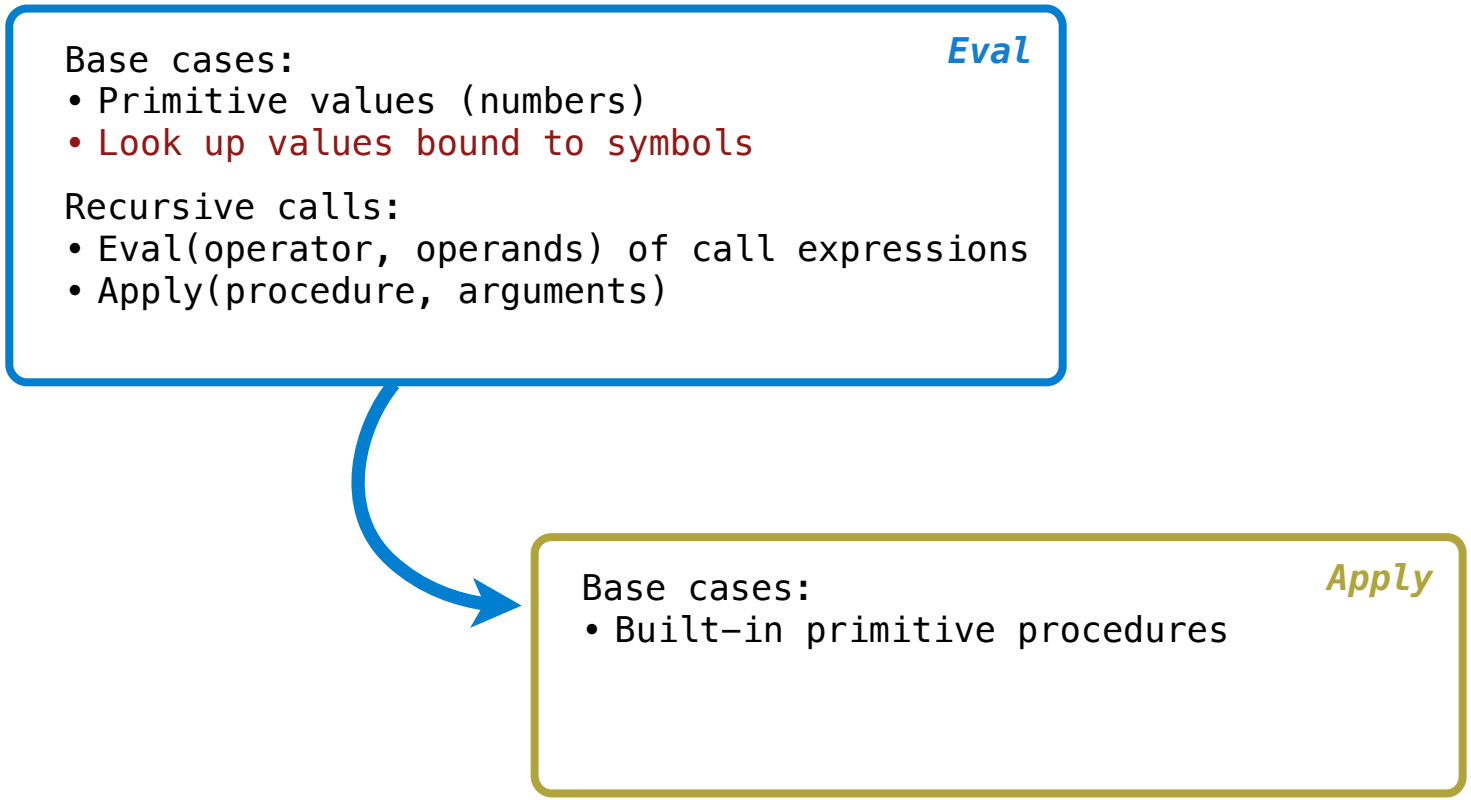
The Structure of an Interpreter



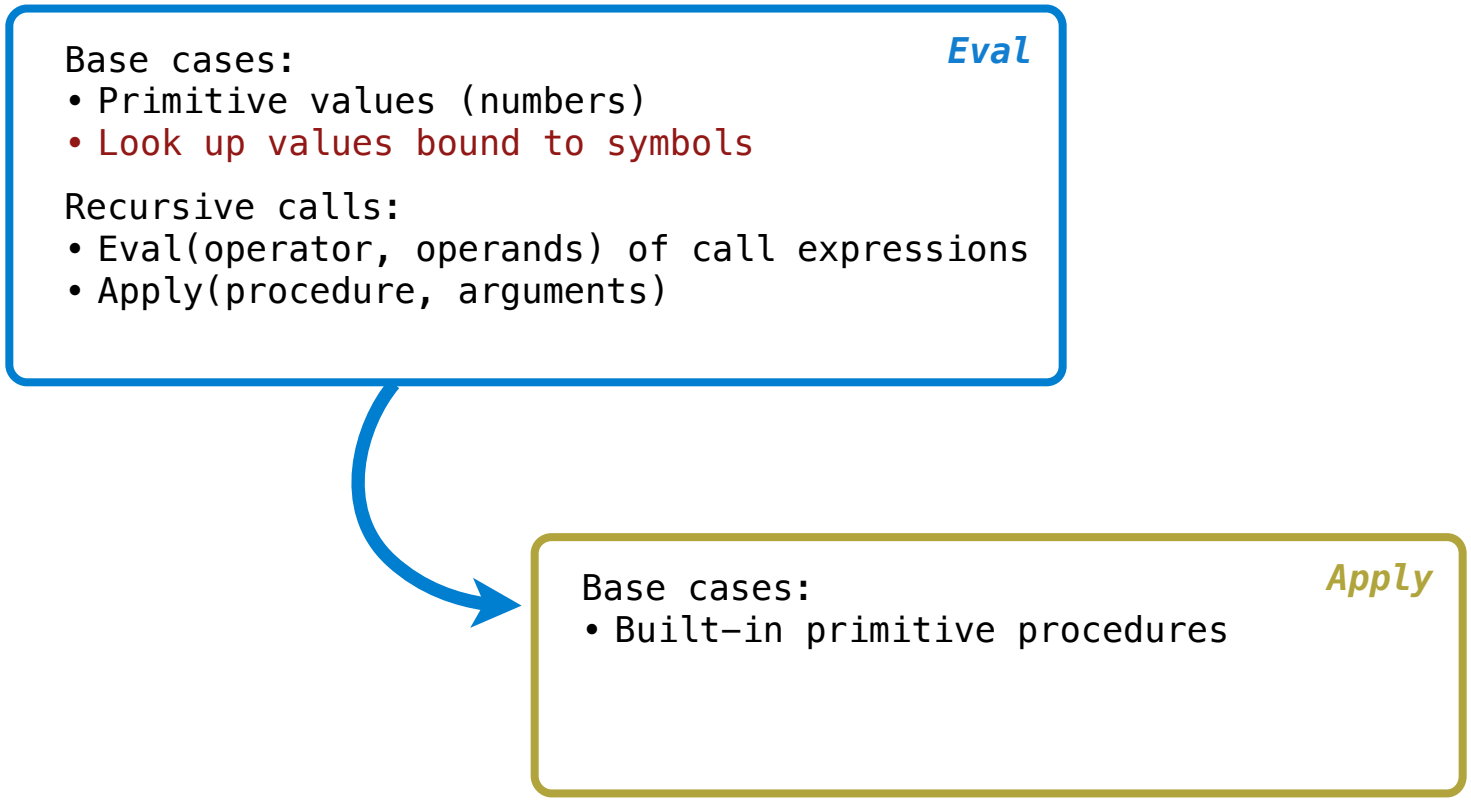
The Structure of an Interpreter



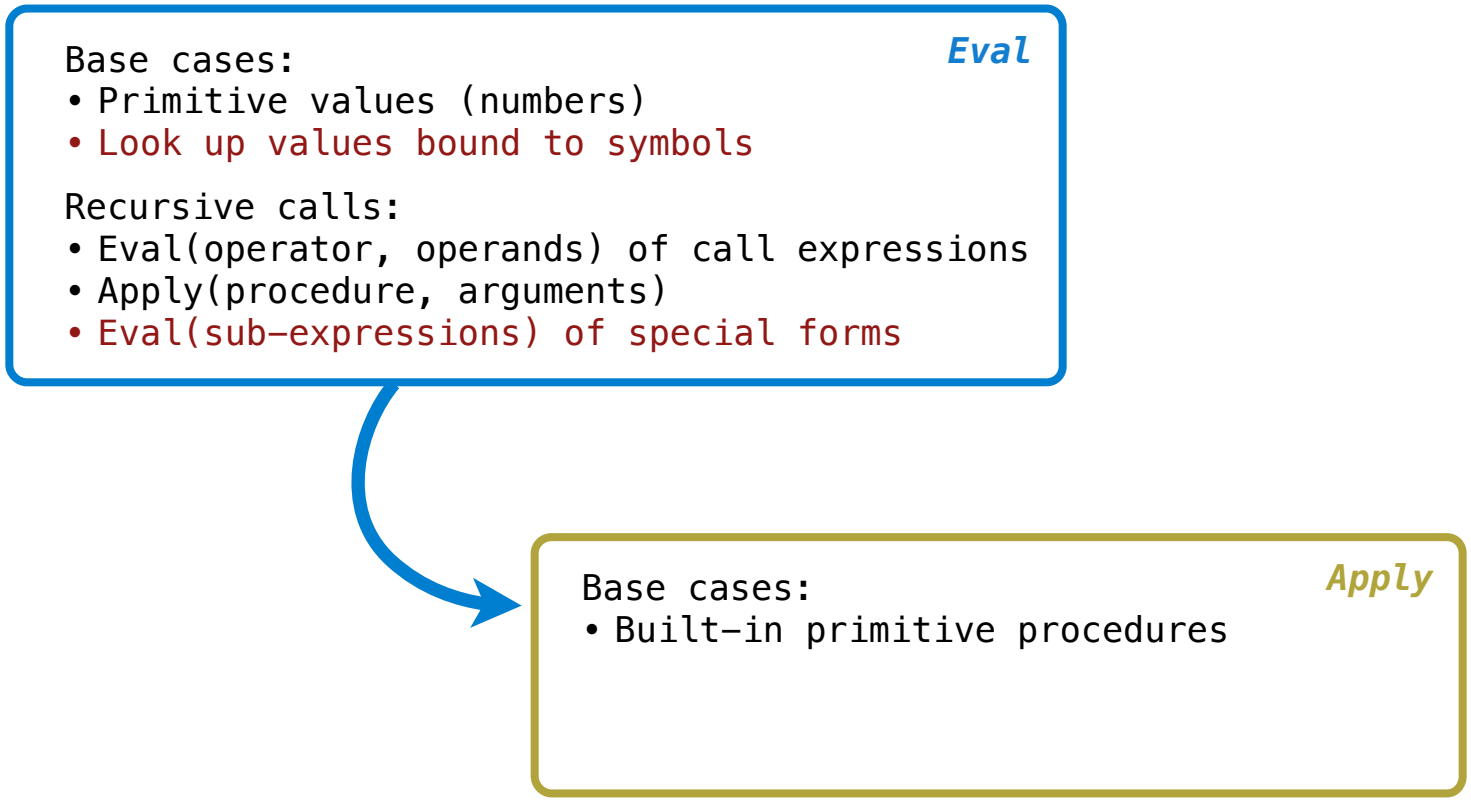
The Structure of an Interpreter



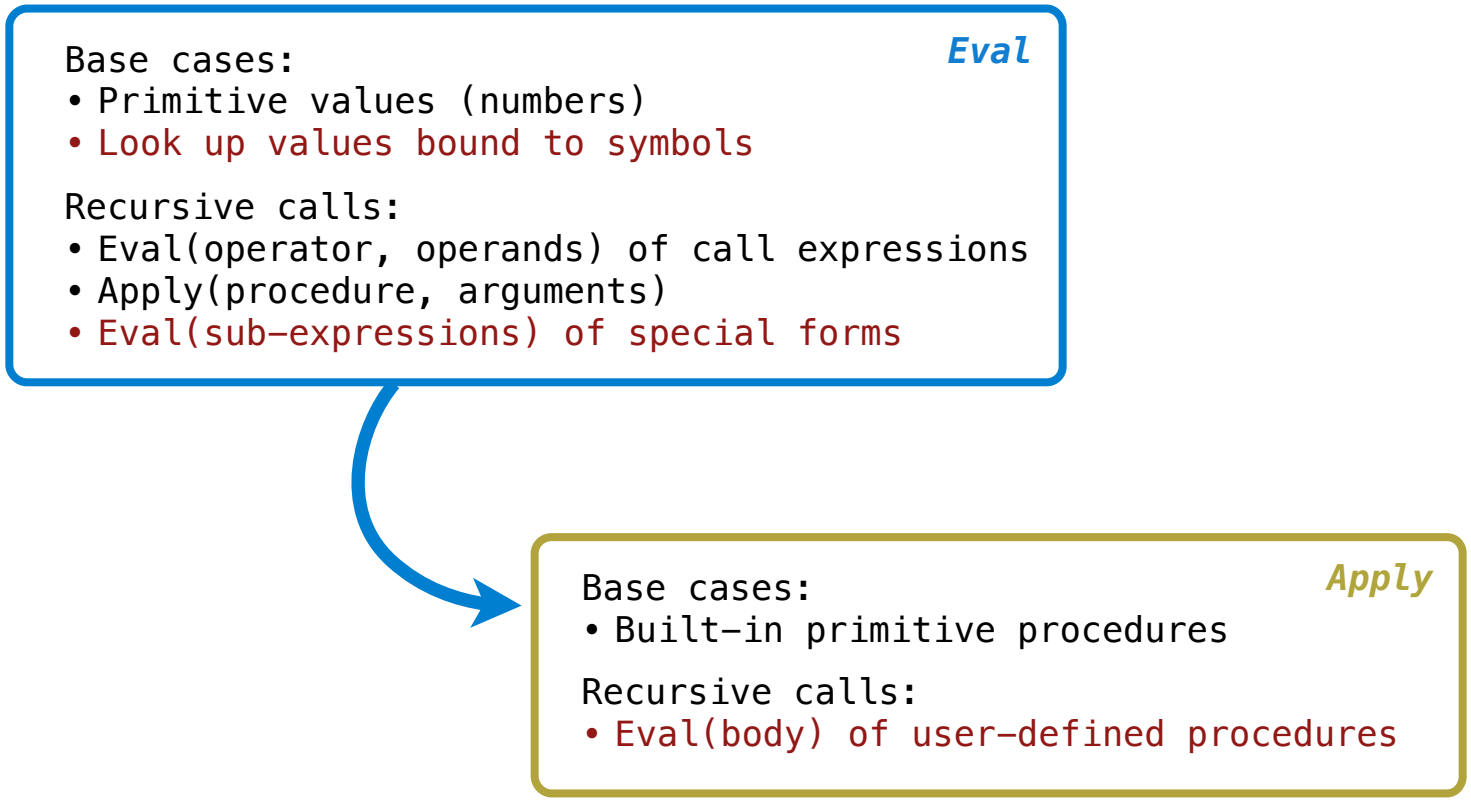
The Structure of an Interpreter



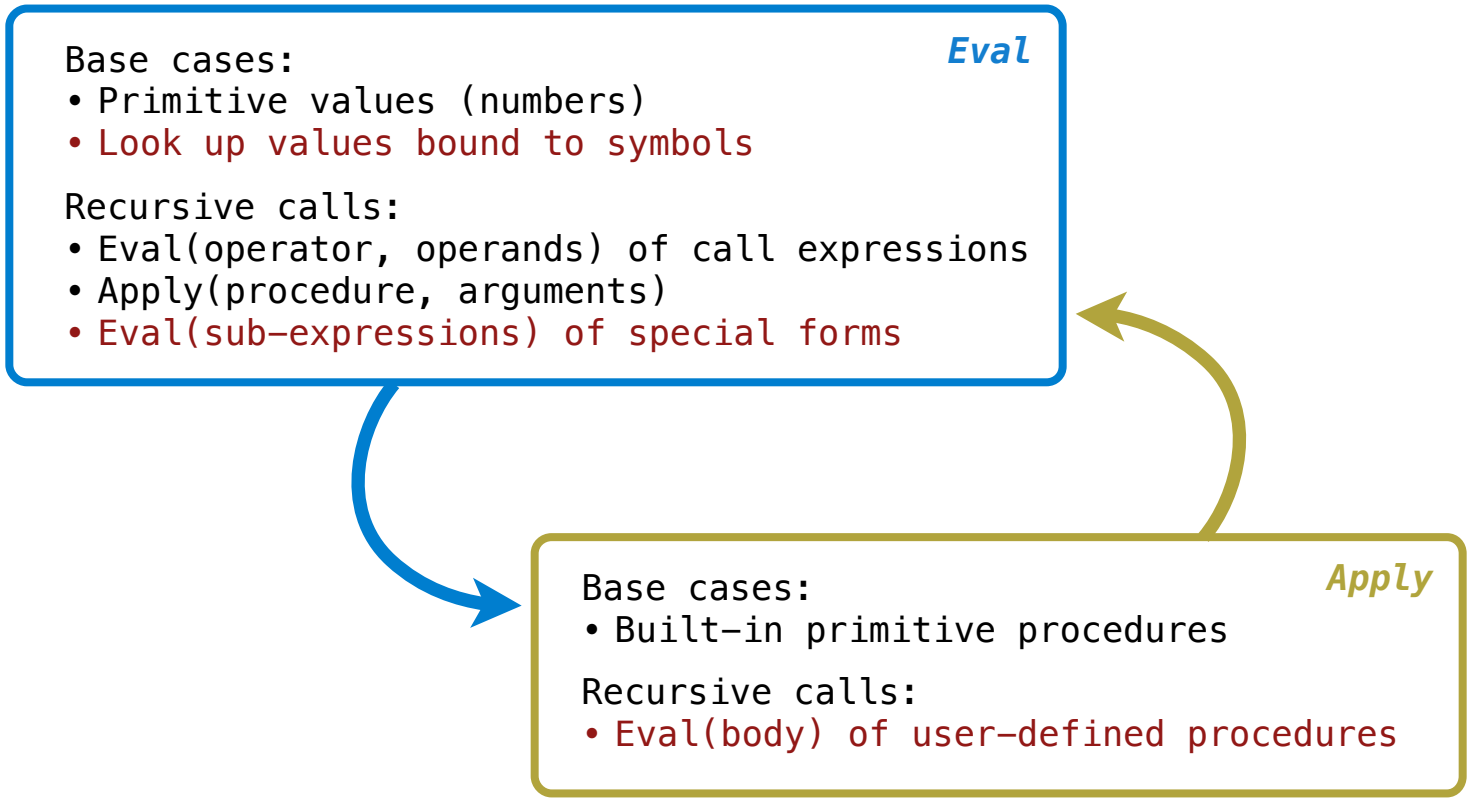
The Structure of an Interpreter



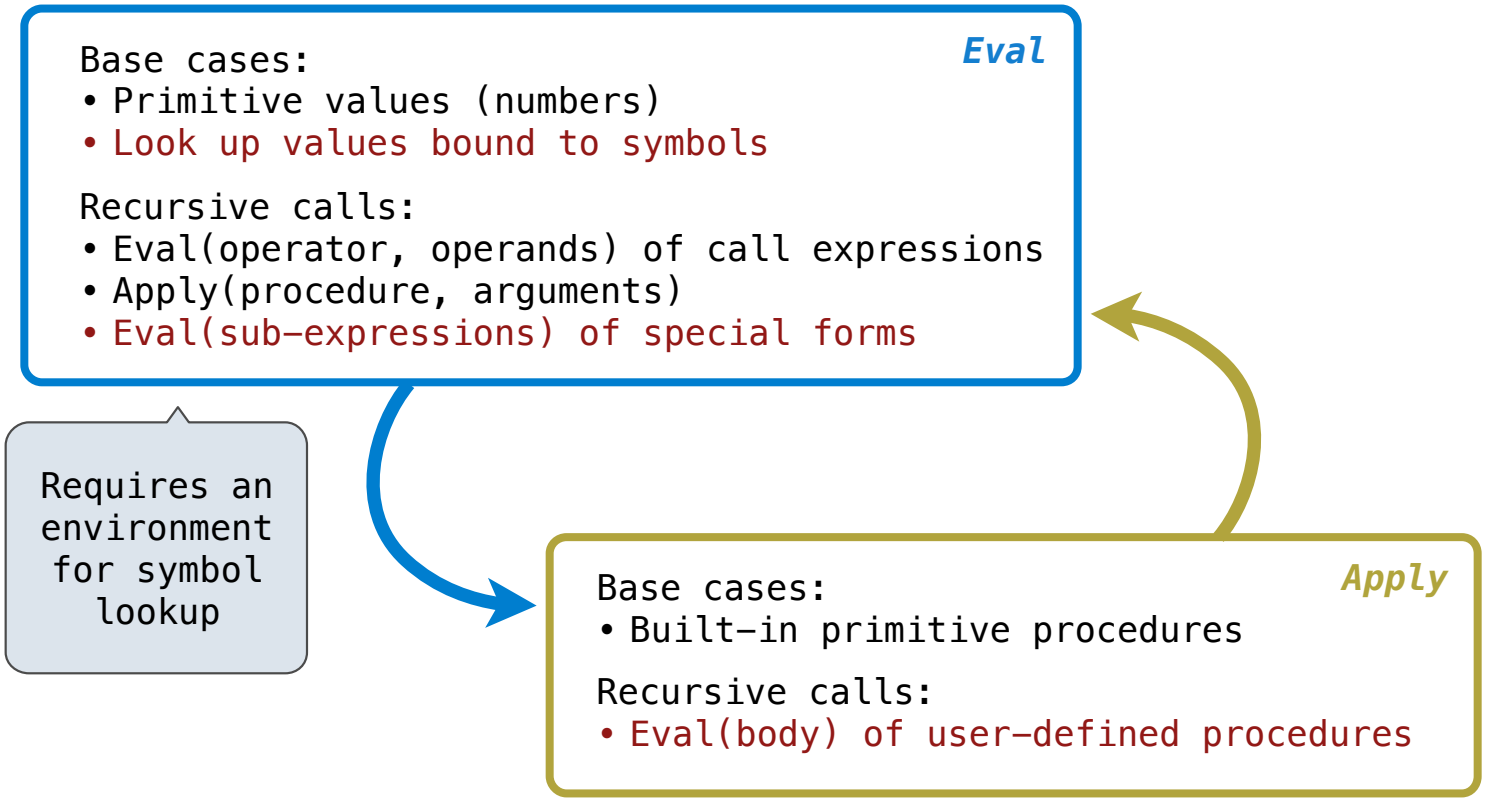
The Structure of an Interpreter



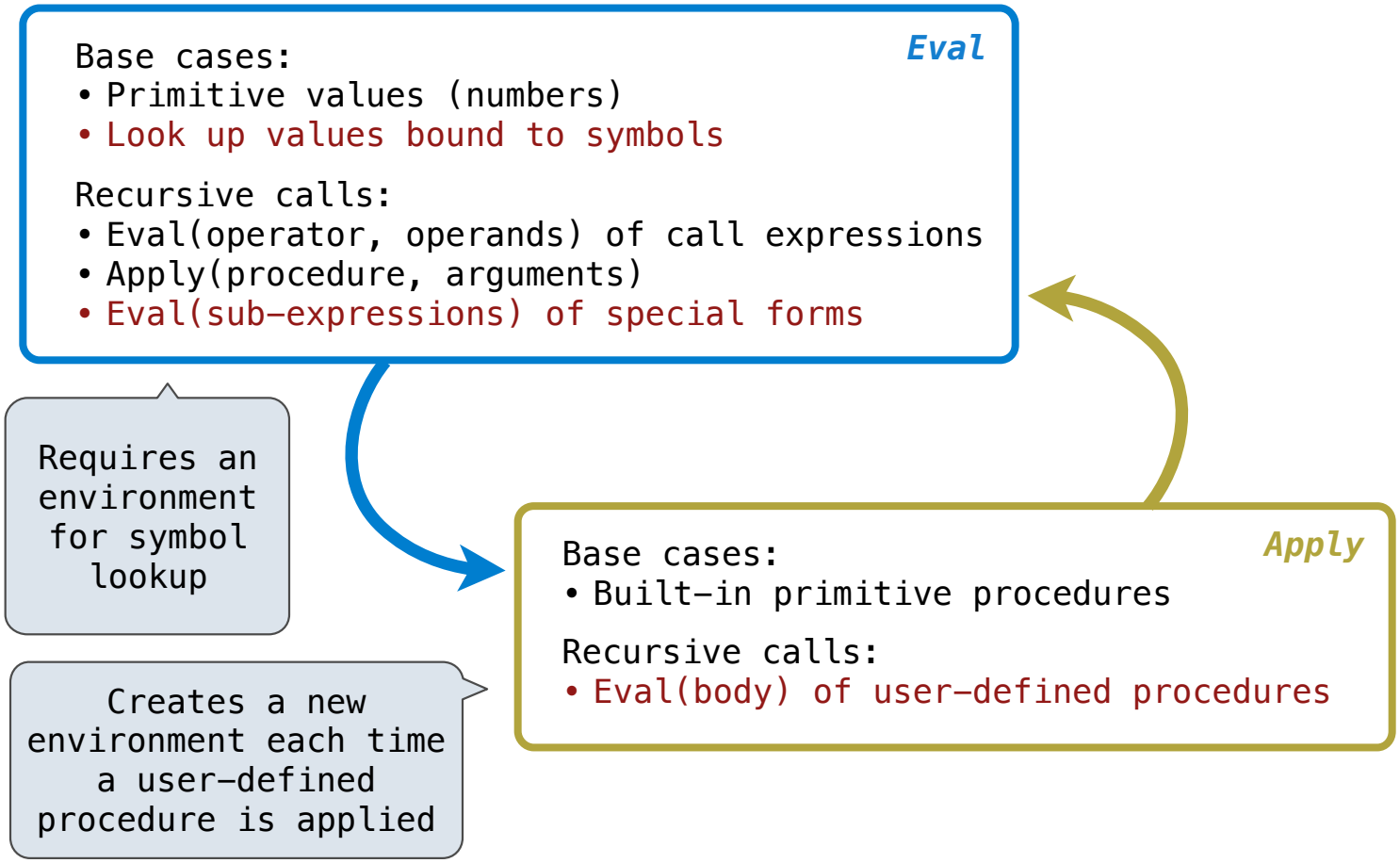
The Structure of an Interpreter



The Structure of an Interpreter



The Structure of an Interpreter



Special Forms

Scheme Evaluation

Scheme Evaluation

The scheme_eval function choose behavior based on expression form:

Scheme Evaluation

The scheme_eval function choose behavior based on expression form:

- Symbols are looked up in the current environment

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

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

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

(**if** <predicate> <consequent> <alternative>)

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

(**if** <predicate> <consequent> <alternative>)

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

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

(**if** <predicate> <consequent> <alternative>)

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

(**define** <name> <expression>)

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

(**if** <predicate> <consequent> <alternative>)

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

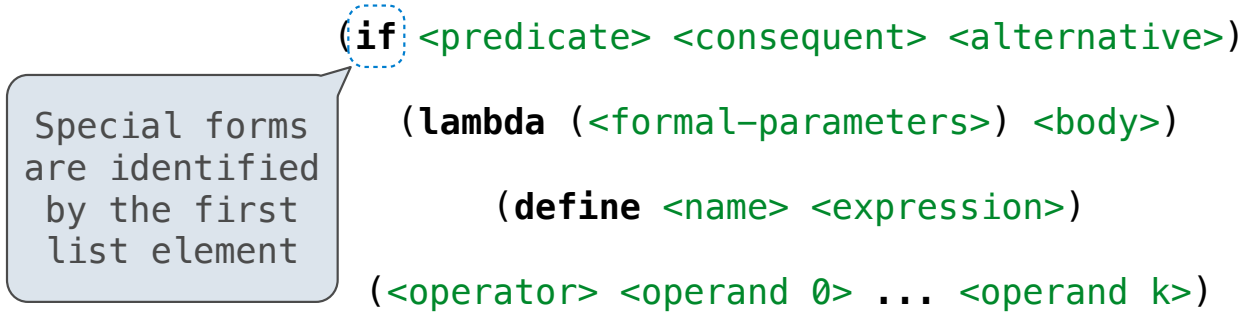
(**define** <name> <expression>)

(<operator> <operand 0> **...** <operand k>)

Scheme Evaluation

The scheme_eval function choose behavior based on expression form:

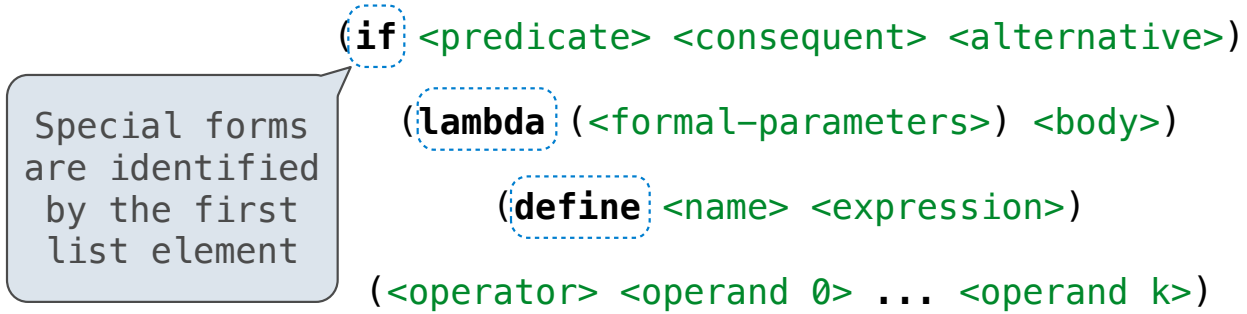
- Symbols are looked up in the current environment
- Self-evaluating expressions are returned as values
- All other legal expressions are represented as Scheme lists, called combinations



Scheme Evaluation

The scheme_eval function choose behavior based on expression form:

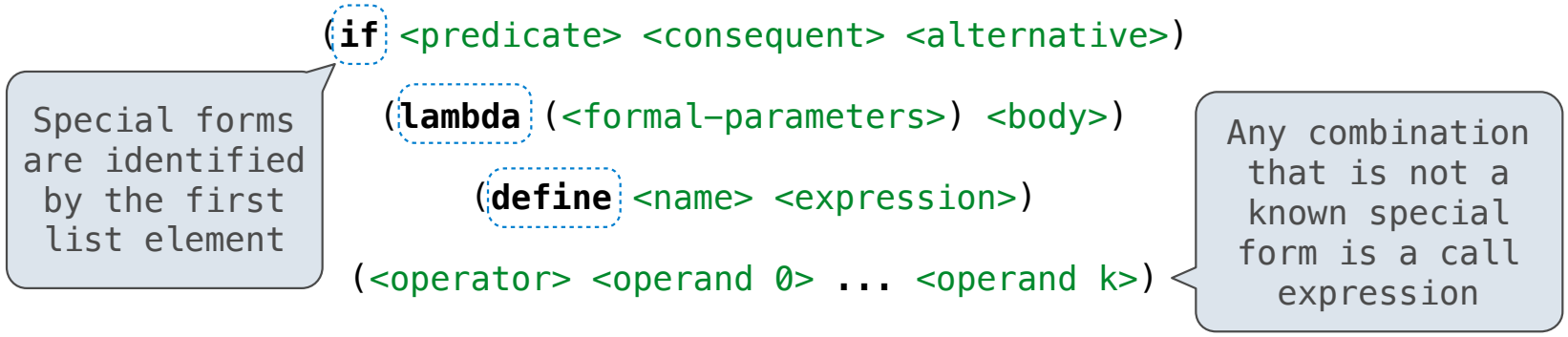
- Symbols are looked up in the current environment
- Self-evaluating expressions are returned as values
- All other legal expressions are represented as Scheme lists, called combinations



Scheme Evaluation

The scheme_eval function choose behavior based on expression form:

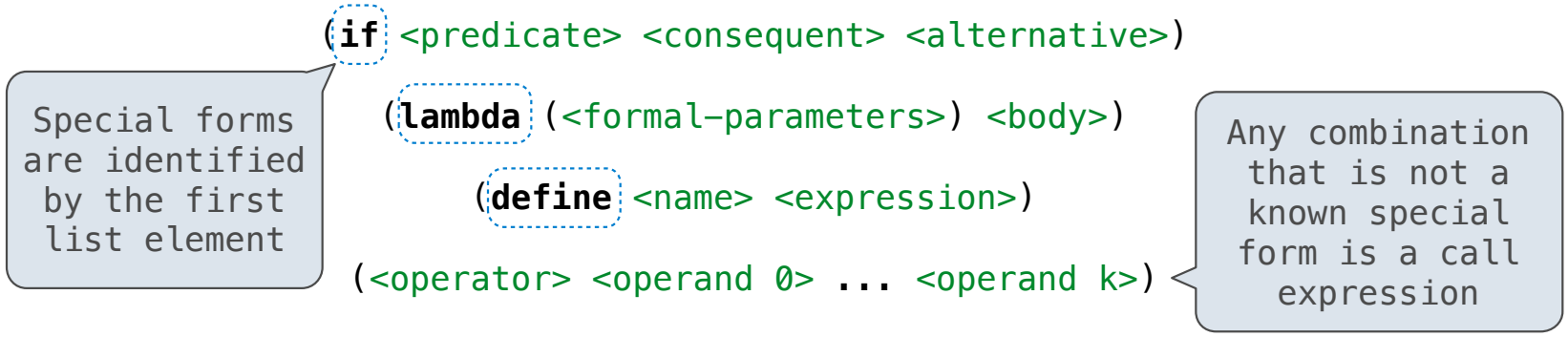
- Symbols are looked up in the current environment
- Self-evaluating expressions are returned as values
- All other legal expressions are represented as Scheme lists, called combinations



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

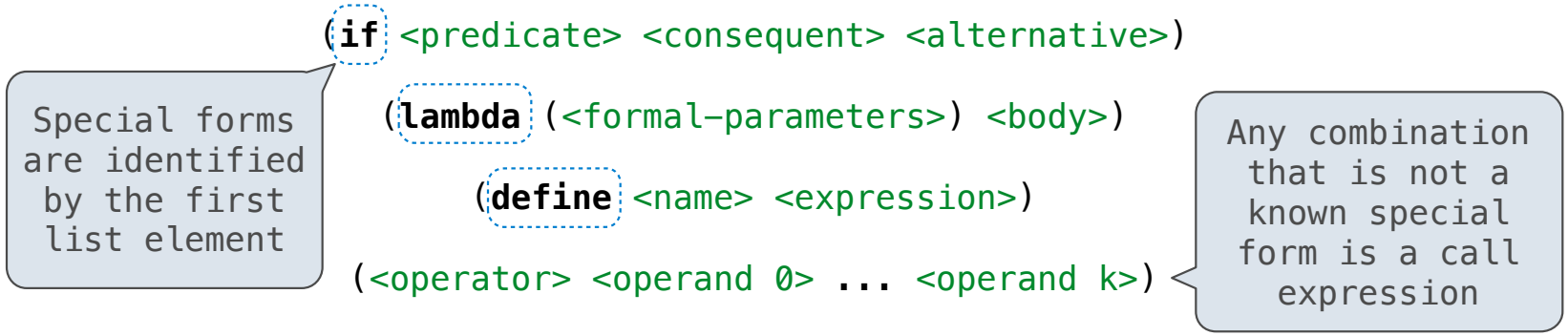



```
(define (demo s) (if (null? s) '(3) (cons (car s) (demo (cdr s))) ))
```

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))
```

Logical Forms

Logical Special Forms

Logical Special Forms

Logical forms may only evaluate some sub-expressions

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)
- **Cond** expression: (**cond** (<p1> <e1>) **...** (<pn> <en>) (else <e>))

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)
- **Cond** expression: (**cond** (<p1> <e1>) **...** (<pn> <en>) (else <e>))

The value of an if expression is the value of a sub-expression:

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)
- **Cond** expression: (**cond** (<p1> <e1>) **...** (<pn> <en>) (else <e>))

The value of an if expression is the value of a sub-expression:

- Evaluate the predicate.

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)
- **Cond** expression: (**cond** (<p1> <e1>) **...** (<pn> <en>) (else <e>))

The value of an if expression is the value of a sub-expression:

- Evaluate the predicate.
- Choose a sub-expression: <consequent> or <alternative>.

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)
- **Cond** expression: (**cond** (<p1> <e1>) **...** (<pn> <en>) (else <e>))

The value of an if expression is the value of a sub-expression:

- Evaluate the predicate.
- Choose a sub-expression: <consequent> or <alternative>.
- Evaluate that sub-expression in place of the whole expression.

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)
- **Cond** expression: (**cond** (<p1> <e1>) **...** (<pn> <en>) (else <e>))

The value of an if expression is the value of a sub-expression:

- Evaluate the predicate.
- Choose a sub-expression: <consequent> or <alternative>.

do_if_form

- Evaluate that sub-expression in place of the whole expression.

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)
- **Cond** expression: (**cond** (<p1> <e1>) **...** (<pn> <en>) (else <e>))

The value of an if expression is the value of a sub-expression:

- Evaluate the predicate.
- Choose a sub-expression: <consequent> or <alternative>.

do_if_form

- Evaluate that sub-expression in place of the whole expression.

scheme_eval

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)
- **And** and **or**: (**and** <e1> **...** <en>), (**or** <e1> **...** <en>)
- **Cond** expression: (**cond** (<p1> <e1>) **...** (<pn> <en>) (else <e>))

The value of an if expression is the value of a sub-expression:

- Evaluate the predicate.
- Choose a sub-expression: <consequent> or <alternative>.

do_if_form

- Evaluate that sub-expression in place of the whole expression.

scheme_eval

(Demo)

Quotation

Quotation

Quotation

The quote special form evaluates to the quoted expression, which is not evaluated

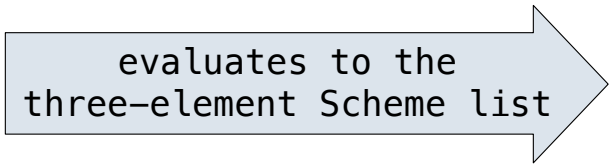
Quotation

The quote special form evaluates to the quoted expression, which is not evaluated

```
(quote <expression>)
```

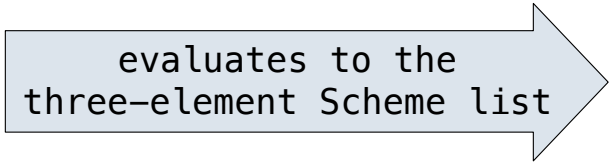
Quotation

The quote special form evaluates to the quoted expression, which is not evaluated

`(quote <expression>)` `(quote (+ 1 2))`  `(+ 1 2)`

Quotation

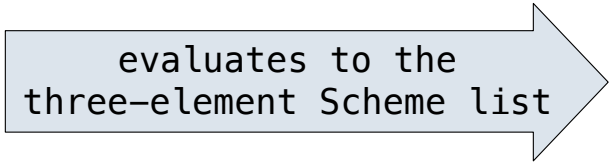
The quote special form evaluates to the quoted expression, which is not evaluated

`(quote <expression>)` `(quote (+ 1 2))`  `(+ 1 2)`

The <expression> itself is the value of the whole quote expression

Quotation

The quote special form evaluates to the quoted expression, which is not evaluated

`(quote <expression>)` `(quote (+ 1 2))`  `(+ 1 2)`

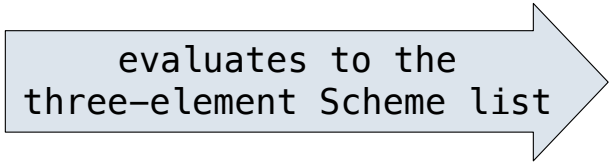
The <expression> itself is the value of the whole quote expression

'<expression> is shorthand for (quote <expression>)

Quotation

The quote special form evaluates to the quoted expression, which is not evaluated

```
(quote <expression>)     (quote (+ 1 2))
```




```
(+ 1 2)
```

The <expression> itself is the value of the whole quote expression

'<expression> is shorthand for (quote <expression>)

```
(quote (1 2))        is equivalent to        '(1 2)
```

Quotation

The quote special form evaluates to the quoted expression, which is not evaluated

(quote <expression>) (quote (+ 1 2)) evaluates to the three-element Scheme list (+ 1 2)

The <expression> itself is the value of the whole quote expression

'<expression> is shorthand for (quote <expression>)

(quote (1 2)) is equivalent to '(1 2)

The scheme_read parser converts shorthand ' to a combination that starts with quote

Quotation

The quote special form evaluates to the quoted expression, which is not evaluated

(quote <expression>) (quote (+ 1 2)) evaluates to the three-element Scheme list (+ 1 2)

The <expression> itself is the value of the whole quote expression

'<expression> is shorthand for (quote <expression>)

(quote (1 2)) is equivalent to '(1 2)

The scheme_read parser converts shorthand ' to a combination that starts with quote

(Demo)

Lambda Expressions

Lambda Expressions

Lambda Expressions

```
Lambda expressions evaluate to user-defined procedures
```

Lambda Expressions

Lambda expressions evaluate to user-defined procedures

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

Lambda Expressions

Lambda expressions evaluate to user-defined procedures

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

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

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
        self.body = body
        self.env = env
```

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
        self.env = env
```

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 expression
        self.env = env
```

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 expression
        self.env = env ..... A Frame instance
```

Frames and Environments

Frames and Environments

A frame represents an environment by having a parent frame

Frames and Environments

A frame represents an environment by having a parent frame

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

Frames and Environments

A frame represents an environment by having a parent frame

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

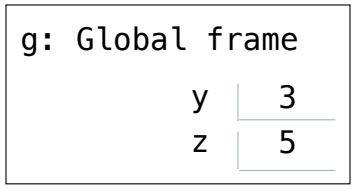
In Project 4, Frames do not hold return values

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

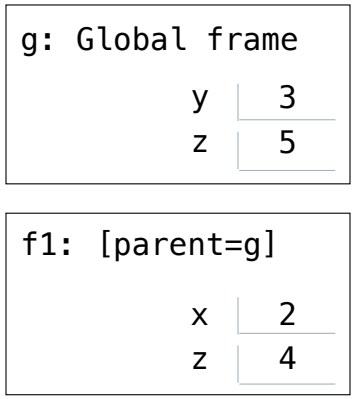


Frames and Environments

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

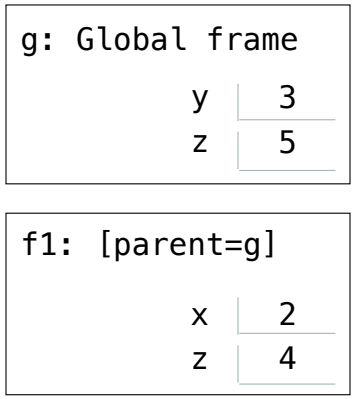


Frames and Environments

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)

Define Expressions

Define Expressions

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

```
(define <name> <expression>)
```

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

```
(define <name> <expression>)
```

1. Evaluate the <expression>

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

```
(define <name> <expression>)
```

1. Evaluate the <expression>
2. Bind <name> to its value in the current frame

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

```
(define <name> <expression>)
```

1. Evaluate the <expression>
2. Bind <name> to its value in the current frame

```
(define x (+ 1 2))
```

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

```
(define <name> <expression>)
```

1. Evaluate the <expression>
2. Bind <name> to its value in the current frame

```
(define x (+ 1 2))
```

Procedure definition is shorthand of define with a lambda expression

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

```
(define <name> <expression>)
```

1. Evaluate the <expression>
2. Bind <name> to its value in the current frame

```
(define x (+ 1 2))
```

Procedure definition is shorthand of define with a lambda expression

```
(define (<name> <formal parameters>) <body>)
```

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

```
(define <name> <expression>)
```

1. Evaluate the <expression>
2. Bind <name> to its value in the current frame

```
(define x (+ 1 2))
```

Procedure definition is shorthand of define with a lambda expression

```
(define (<name> <formal parameters>) <body>)
(define <name> (lambda (<formal parameters>) <body>))
```

Applying User-Defined Procedures

Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

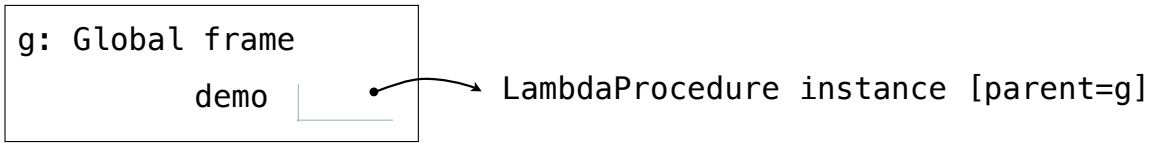
```
(define (demo s) (if (null? s) '(3) (cons (car s) (demo (cdr s)))))
```

Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

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



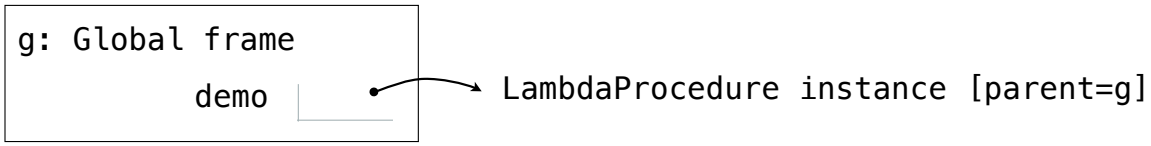
Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

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

(demo (list 1 2))
```



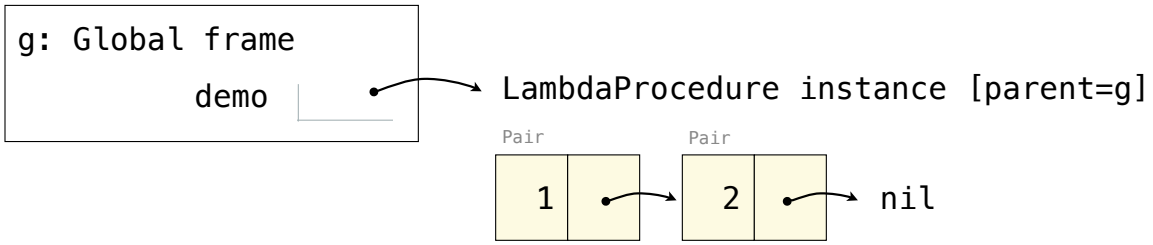
Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

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

(demo (list 1 2))
```



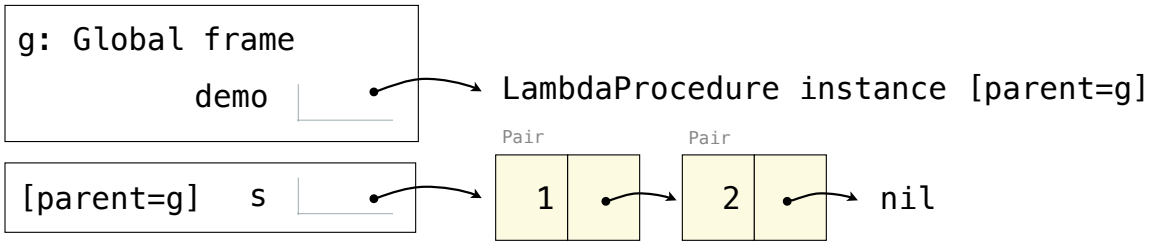
Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

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

(demo (list 1 2))
```



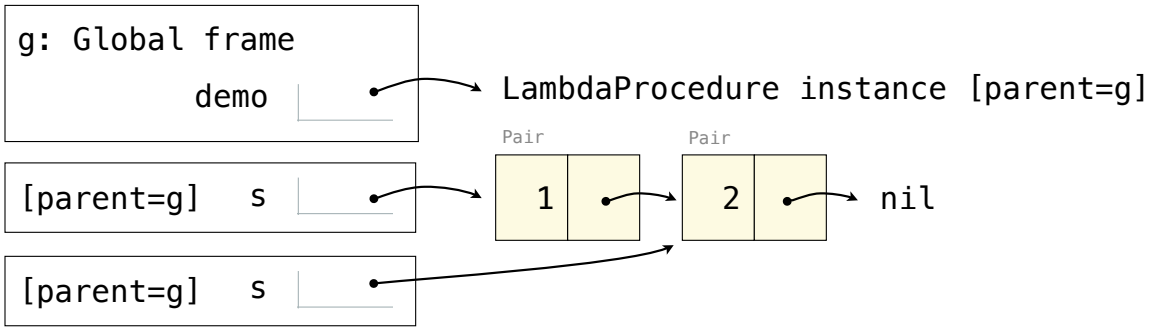
Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

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

(demo (list 1 2))
```



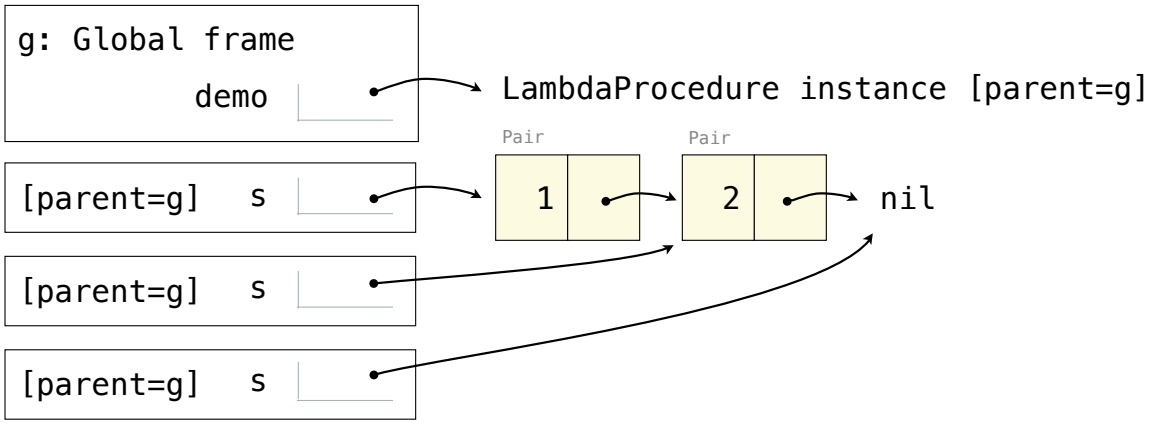
Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

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

(demo (list 1 2))
```



Eval/Apply in Lisp 1.5

Eval/Apply in Lisp 1.5

```
apply[fn;x;a] =
      [atom[fn] → [eq[fn;CAR] → caar[x];
                   eq[fn;CDR] → cdar[x];
                   eq[fn;CONS] → cons[car[x];cadr[x]];
                   eq[fn;ATOM] → atom[car[x]];
                   eq[fn;EQ] → eq[car[x];cadr[x]];
                   T → apply[eval[fn;a];x;a]];
       eq[car[fn];LAMBDA] → eval[caddr[fn];pairlis[cadr[fn];x;a]];
       eq[car[fn];LABEL] → apply[caddr[fn];x;cons[cons[cadr[fn];
                                                     caddr[fn]];a]]]
eval[e;a] = [atom[e] → cdr[assoc[e;a]];
      atom[car[e]] →
                [eq[car[e],QUOTE] → cadr[e];
                eq[car[e];COND] → evcon[cdr[e];a];
                T → apply[car[e];evlis[cdr[e];a];a]];
      T → apply[car[e];evlis[cdr[e];a];a]]
```