Part I

Functions

```
{define {double x}
  {+ x x}}

{define {quadruple x}
  {double {double x}}}

{quadruple 2}
```

$\twoheadrightarrow$ `8`

Functions

```
{+ {define {double x} {+ x x}}
   1}
```

?

No: a function ***definition*** is not an expression

Functions

```
{+ {double 4}
   1}
```

?

Yes: a function ***call*** is an expression

We'll use ***call*** and ***application*** interchangably

Function Definitions

```
{define {triple x}
  {+ x {+ x x}}}
```

A function has

- a name
- an argument name
- a *body*

```
(define-type Body-Exp
  (numE [n : Number])
  (idE [s : Symbol])
  (plusE [l : Body-Exp] [r : Body-Exp])
  (multE [l : Body-Exp] [r : Body-Exp]))
```

?

Function Definitions

```
{define {triple x}
  {+ x {+ x x}}}
```

A function has

- a name
- an argument name
- a *body*

Allow `x` to be an expression, and then

```
{+ x {+ x x}}
```

is also an expression

Functions and Function Calls

- numbers
- identifiers
- addition expressions
    - first and second arguments are expressions
- multiplication expressions
    - first and second arguments are expressions
- function-call expressions
    - a function name and an argument expression

- a function definition
    - a function name, argument name, and body expression

Functions and Function Calls

```
(define-type Exp
  (numE [n : Number])
  (idE [s : Symbol])
  (plusE [l : Exp]
         [r : Exp])
  (multE [l : Exp]
         [r : Exp])
  (appE [s : Symbol]
        [arg : Exp]))

(define-type Func-Defn
  (fd [name : Symbol]
      [arg : Symbol]
      [body : Exp]))
```

Representing Programs

```
(define-type Exp
  (numE [n : Number])
  (idE [s : Symbol])
  (plusE [l : Exp]
         [r : Exp])
  (multE [l : Exp]
         [r : Exp])
  (appE [s : Symbol]
        [arg : Exp]))

(define-type Func-Defn
  (fd [name : Symbol]
      [arg : Symbol]
      [body : Exp]))
```

```
{+ 1 2}
```

```
(plusE (numE 1)
       (numE 2))
```

Representing Programs

```
(define-type Exp
  (numE [n : Number])
  (idE [s : Symbol])
  (plusE [l : Exp]
         [r : Exp])
  (multE [l : Exp]
         [r : Exp])
  (appE [s : Symbol]
        [arg : Exp]))

(define-type Func-Defn
  (fd [name : Symbol]
      [arg : Symbol]
      [body : Exp]))
```

```
{+ x 2}
```

```
(plusE (idE 'x)
       (numE 2))
```

Representing Programs

```
(define-type Exp
  (numE [n : Number])
  (idE [s : Symbol])
  (plusE [l : Exp]
         [r : Exp])
  (multE [l : Exp]
         [r : Exp])
  (appE [s : Symbol]
        [arg : Exp]))

(define-type Func-Defn
  (fd [name : Symbol]
      [arg : Symbol]
      [body : Exp]))
```

```
{define {plus-two x}
  {+ x 2}}
```

```
(fd 'plus-two
    'x
    (plusE (idE 'x)
           (numE 2)))
```

Representing Programs

```
(define-type Exp
  (numE [n : Number])
  (idE [s : Symbol])
  (plusE [l : Exp]
         [r : Exp])
  (multE [l : Exp]
         [r : Exp])
  (appE [s : Symbol]
        [arg : Exp]))

(define-type Func-Defn
  (fd [name : Symbol]
      [arg : Symbol]
      [body : Exp]))
```

```
{plus-two 9}
```

```
(appE 'plus-two
      (numE 9))
```

Representing Programs

```
(define-type Exp
  (numE [n : Number])
  (idE [s : Symbol])
  (plusE [l : Exp]
         [r : Exp])
  (multE [l : Exp]
         [r : Exp])
  (appE [s : Symbol]
        [arg : Exp]))

(define-type Func-Defn
  (fd [name : Symbol]
      [arg : Symbol]
      [body : Exp]))
```

```
{define {double x}
  {+ x x}}

{define {quadruple x}
  {double {double x}}}

{quadruple 2}
```

```
(list (fd 'double 'x
          (plusE (idE 'x)
                 (idE 'x)))
      (fd 'quadruple 'x
          (appE 'double
                (appE 'double
                      (idE 'x)))))

(appE 'quadruple (numE 2))
```

Part 2

Evaluating Function Calls

```
{define {double x}
  {+ x x}}

{double 3}

↠ {+ 3 3}

↠ 6
```

```
interp : (Exp (Listof Func-Defn) -> Number)

get-fundef : (Symbol (Listof Func-Defn) -> Func-Defn)

subst : (Exp Symbol Exp -> Exp)
```