Lecture 5. Funclang

September 21, 2018

Overview

- What is function?
- Lambda Expression
- Recursive Function
- High Order Function
- Curry
- List
- Control Structure: if then else
- Semantics of FuncLang

Syntax used for functional programming language

(lambda (x)(+x x))

Lambda Expression and call

- A tool for defining anonymous function, a language features

```
(
  lambda      //Lambda special function for defining functions
  (x)         //List of formal parameter names of the function
   x          //Body of the function
)
```

- Compare to the notion of procedures and methods in ALGOL family of languages: C, C++, C#, Java (syntax):
  - not specify the name of the function
  - formal parameter name only, no types precede or follow
  - no explicit return is needed
- Compare to the notion of procedures and methods in ALGOL family of languages: C, C++, C#, Java (semantics):
  Procedures and methods: proxy of the location of the subsection of code section
  - adjust the environment
  - jump to the location

  Lambda abstraction:
  - generation of the runtime values
  - each of the runtime values can be used multiple times reuse

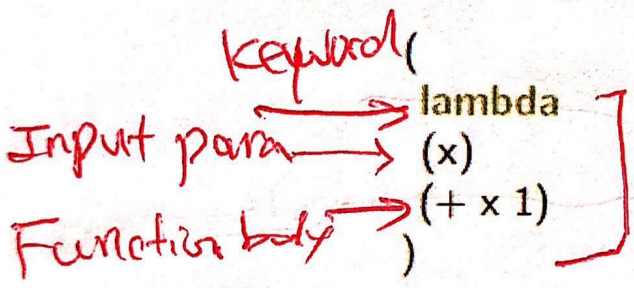
Abstraction in Programming Languages

- Variable: fixed abstraction – you cannot change functionality

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Function (procedure, method): **parameterization for computation** – you can reuse the functionality for different concrete input: the ability to define a procedure, and the ability to call a procedure.

Examples: Lambda Expression




```
//Lambda special form for defining functions
//List of formal parameter names of the function
//Body of the function
```

(lambda (x y) (+ x y))

Examples: Calling the Lambda function

```
(
  (lambda (x) x)
  1
)
```

actual parameter

```
//Begin function call syntax
//Operator: function being called
//Operands: list of actual parameters
//End function call syntax
```

```
(
  (lambda (x y) (+ x y))
  1 1
)
```

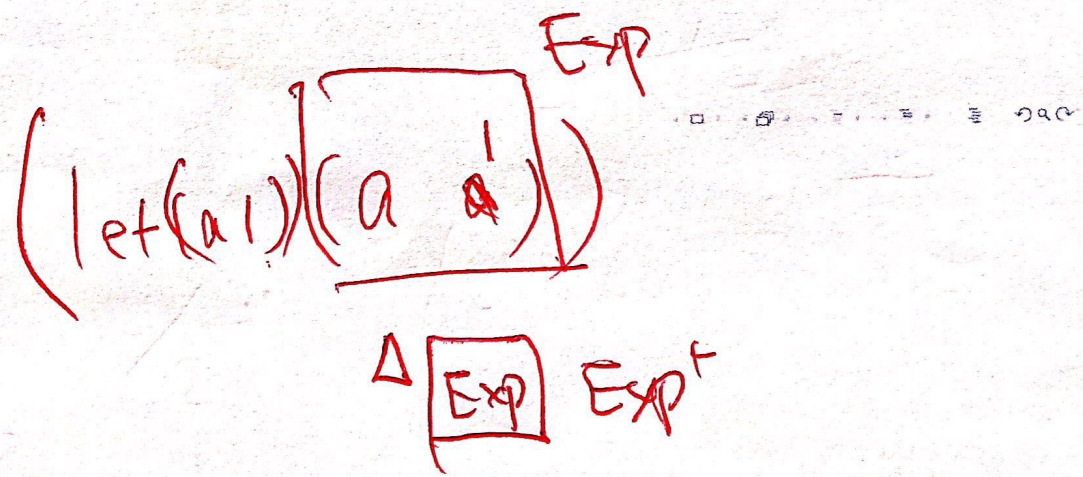
Examples: Combine with Let and Define

```
(let
   (( identity  (lambda (x) x)))          //Naming the function
   ( identity  1)                         //Function call
)
```

```
$ (define square (lambda (x) (* x x)))
$ (square 1.2)
1.44
```

Grammar for FuncLang

Program	::=	DefineDecl* Exp?	*Program*
DefineDecl	::=	(define Identifier Exp)	*Define*
Exp	::=		*Expressions*
		Number	*NumExp*
	|	(+ Exp Exp+)	*AddExp*
	|	(- Exp Exp+)	*SubExp*
	|	(* Exp Exp+)	*MultExp*
	|	(/ Exp Exp+)	*DivExp*
	|	Identifier	*VarExp*
	|	(let ((Identifier Exp)+) Exp)	*LetExp*
	|	(Exp Exp+)	*CallExp*
	|	(lambda (Identifier+) Exp)	*LambdaExp*
Number	::=	Digit	*Number*
	|	DigitNotZero Digit+	
Digit	::=	[0-9]	*Digits*
DigitNotZero	::=	[1-9]	*Non-zero Digits*
Identifier	::=	Letter LetterOrDigit*	*Identifier*
Letter	::=	[a-zA-Z$_]	*Letter*
LetterOrDigit	::=	[a-zA-Z0-9$_]	*LetterOrDigit*



In-class Exercise

- Write five lambda abstraction and calls, discuss with your neighbors
- Select your favorite functions

Pair and List

Cons(2,(list 3))

(2,3)

list

Cons

1. Pair: 2 tuple (fst, snd)
2. List: empty list, or 2 tuple
3. a list is a pair, a pair is not necessarily a list
4. Lists are constructed by using the cons keyword, as is shown here:
   > (cons 1 (list))
   (1)

(list 1)

(list 2 3 4)

1, (list)

2, (list 3 4)

List and its Operations

- list: creating a list
- cons: constructing a pari
- null?: check if a list is a null
- car: get the first element of a pair
- cdr: get the second element of a pair

Built-in Functions List

1. List: constructor for a list; parameters: values used to initialize a list, e.g., (list 1 1 1 1 1)
2. car: takes a pair or a list as an argument, returns the first element, e.g., (car (list 11 1))
3. cdr: take a pair or a list as an argument, returns the second element, e.g., (cdr (list 342))
4. cons: If the second value is a list, it produces a new list with the first value appended to the front of the second value list. Otherwise, it produces a pair of two argument values. e.g., (cons 541 (list 342))
5. null? The function null? takes a single argument and evaluates to #t if that argument is an emptylist.

Examples: Using Built-In Functions

```
(define cadr
  (lambda (lst)
    (car (cdr lst))
  )
)
```

```
(define caddr
  (lambda (lst)
    (car (cdr (cdr lst)))
  )
)
```

if (cond) true false expr

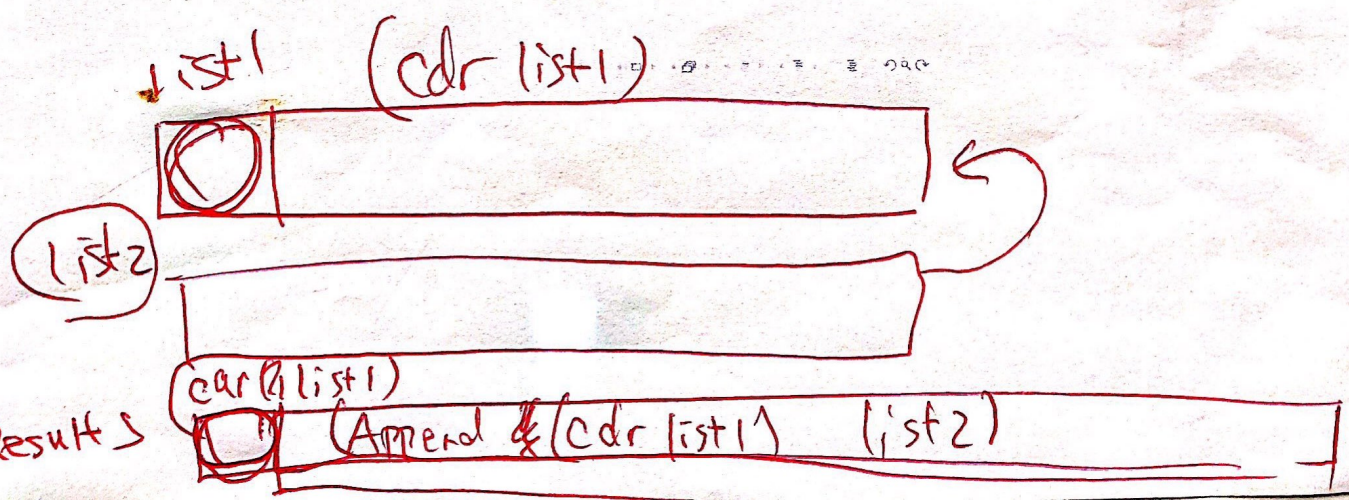
Recursive Function

- Recursive function mirror the definition of the input data type

$List := (list) \mid (cons\ \texttt{val}\ List),\ \texttt{where val} \in \texttt{Value}$

```
(define append
  (lambda (lst1 lst2)
    (if (null? lst1) lst2
      (if (null? lst2) lst1
        (cons (car lst1) (append (cdr lst1) lst2)))
      )
    )
  )
```

First element, second element



In-class Exercise

- Write one recursive function in funclang and discuss with your neighbours
- Select your favorite functions

if (cond) true false expr

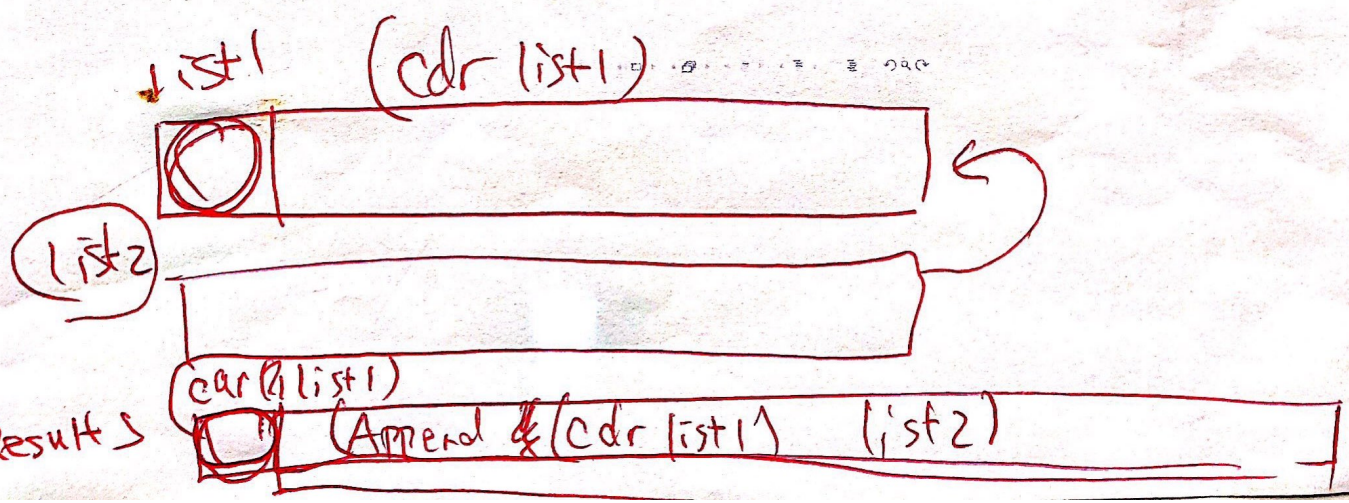
Recursive Function

- Recursive function mirror the definition of the input data type

$List := (list) \mid (cons\ \texttt{val}\ List),\ \texttt{where val} \in \texttt{Value}$

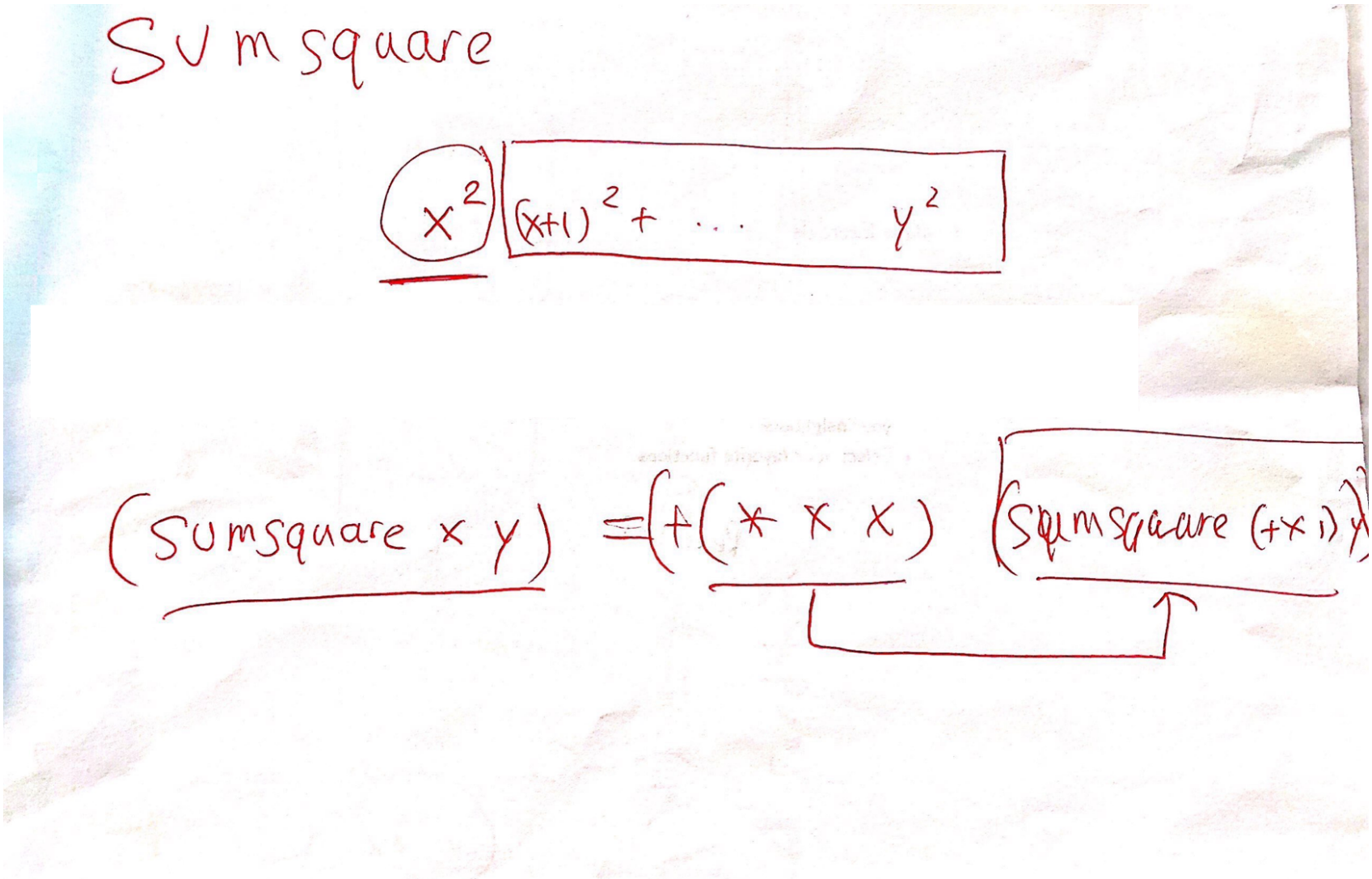
```
(define append
  (lambda (lst1 lst2)
    (if (null? lst1) lst2
      (if (null? lst2) lst1
        (cons (car lst1) (append (cdr lst1) lst2)))
      )
    )
  )
```

First element, second element



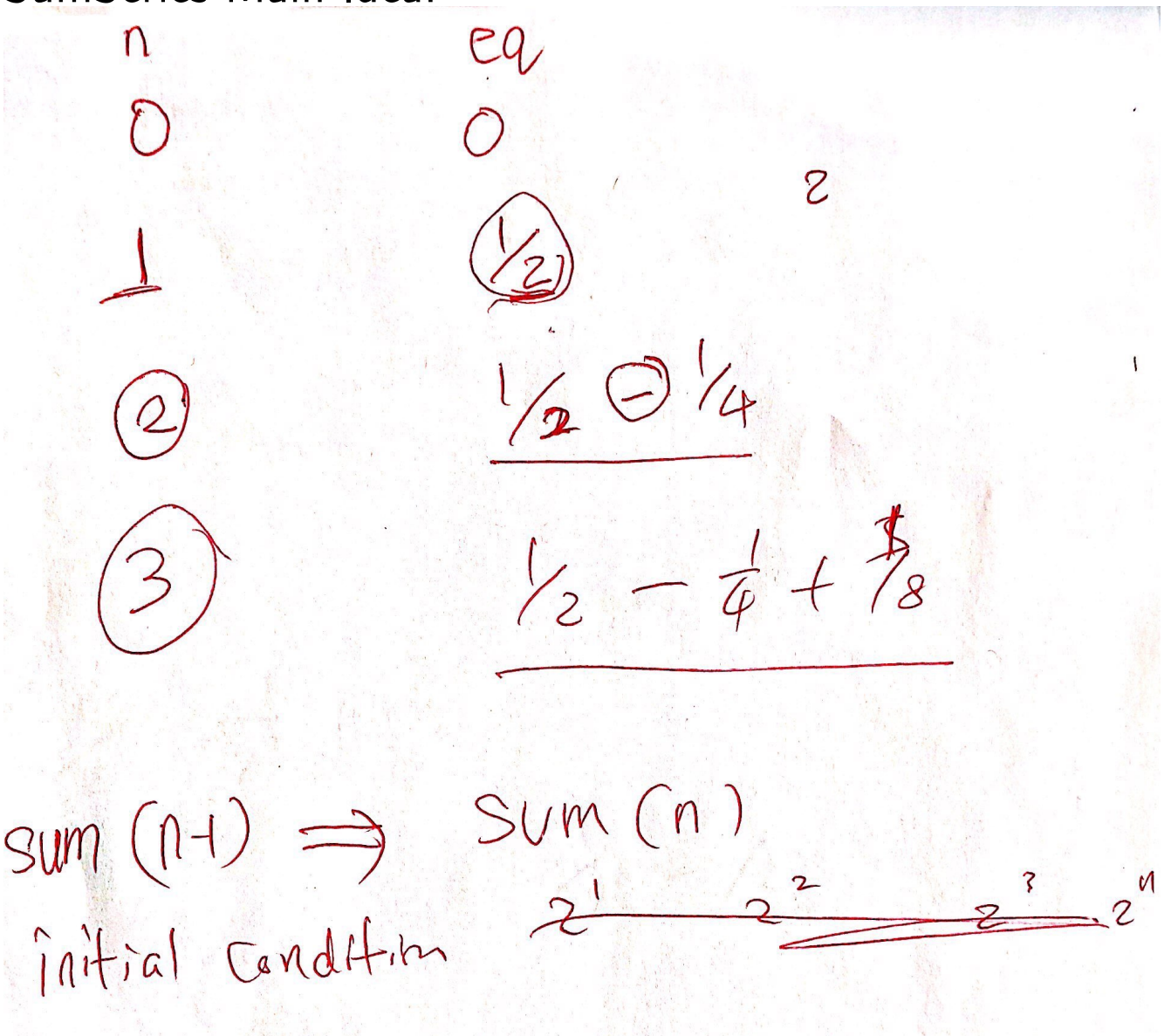
Exercise 5.2.2 from Rajan's book

SumSquares Main Idea:



Exercise 5.2.3 from Rajan's book

SumSeries Main Idea:



High Order Function

First-class function

- a function that accepts a function as argument or return a function as value

```
(define addthree
   (lambda (x)(+ x 3))
)
(define returnone
   (lambda (f) (f 1))
)
$(returnone addthree)
```

parameterize computation

→ Choice of Computation

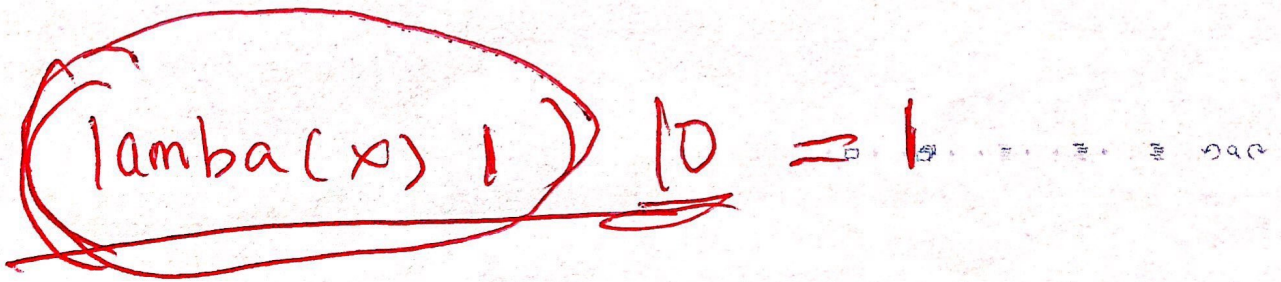
High Order Function

Function definition:

```
(lambda
   (c)
   (lambda (x) c)
)
```

Function call:

```
( (lambda
   (c)
   (lambda (x) c)
  )
1
)
```



(lambda (x) (+ x 1))

High Order Function with Data Structures

apair

pair (3 4)

Rajan's book page 94:

```
(define pair
  (lambda (fst snd)
    (lambda (op)
      (if op fst snd)
    )
  )
)
(define apair (pair 3 4))
(define first (lambda (p) (p #t)))
$ (first apair)
```

3

(lambda (op)
  (if op 3 4))

second ?

(pair 3 4) # +

(lambda (op)
  (if (op) 3 4)) #+

if (#+) 3 4

3

High Order Functions

Exercise:
Write a high order function that takes a list and an operator. Apply the operator to any elements in the list and then return the list

Currying

```
leteexp  returns [LeteExp ast]
  locals [ArrayList<String> names = new ArrayList<String>(), ArrayList<Exp> value_exps = new ArrayList<Exp>()] :
        '(' Lete
        key=numexp
        '(' ( '(' id=Identifier e=exp ')' { $names.add($id.text); $value_exps.add($e.ast); } )+ ')'
        body=exp
        ')' { $ast = new LeteExp($names, $value_exps, $body.ast, $key.ast); }
    ;
```