Q1

1.1

*Expressions:*

Primitive atomic => 1

Non primitive atomic => (let ((x 3)) (\* x 5))

Non primitive compound => (lambda (x y) (+ x y 3))

Values:

Primitive atomic => #t

Non primitive atomic => ‘hello

Non primitive compound => #<closure>

1.2 special form is a unique syntax which its semantics does not follow any existing structure like procedures or operators semantics.

e.g. the form “define”

1.3 free variable is a variable where all of its references in a given expression appear without their declaration

e.g. (

(lambda (x) x)

Y

)

Y is a free variable here.

1.4 Symbolic expression -

1.5

a. (define myFunc

(lambda x y z

(+ x y z)

)

)

Can be “sugared” into this syntax:

(define (myFunc x y z) (+ x y z))

b. (cons 5 6)

Can be “sugared” into this syntax:

‘(5 . 6)

1.6

L1 and L0 are equivalent. Each usage of the vars declared in define can be replaced with the bound value of the define var (at all relevant places)

We can write a parser which takes this L1 program: (define x 5) (+ x 3)

and returns the following L0 one: (+ 5 3)

1.7

L2 and L20 are not equivalent. Without define we would not be able to create function “names” hence no recursive procedures, i.e. only finite number of repeating functions call

1.8

PrimOp advantage – this way the primitive procedures are stored as var references to the actual procedures in the meta language and available in the global environment.

When the interpreter starts running those primitive procedures are already defined

Closures advantage –

1.9

The order we compute the collection items does not matter in Map since a computation done on a single item has no side effects on any other item.

In Reduce however, we can give the following contradictory example:

[3, 4].reduce((acc, curr)=> acc % curr, 5) returns 2, whereas

[4, 3].reduce((acc, curr)=> acc % curr, 5) = 1