**Part 1**

1. Let is a special form - because he is not evaluated like regular compound expression.

Compound expression is evaluated by the rules:

1. evaluate the sub-expressions recursively
2. apply the value of the operator on the values of the operands.
3. Applying wrong parameter types to expression: (\* #f #t)
4. Evaluating free variable: (+ x 5) – when this is our whole program.
5. Errors while computing expression: (/ 5 0)
6. Evaluation of compound expression when his operator is not of compatible type: (1 2)
7. TODODODODO
8. Because we don’t place values but exp and we compute only when we needed.

Normal faster 🡪 (L3 (define loop (lambda (x) (loop x)))

(define g (lambda (x) 5))

(g (loop 0)))

Because applicative will enter infinite loop and never ends.

Applicative faster 🡪

(define square (lambda (x) (\* x x)))

(define sum-of-squares (lambda (x y) (+ (square x) (square y))))

(define f (lambda (a) (sum-of-squares (+ a 1) (\* a 2))) (f 5)

Normal eval computes several times the same expression, like (5 \* 2)

While applicative computes them only once.