1. let в Scheme используется для создания локальных переменных и выполнения выражений внутри локальной области видимости.
let полезна, когда нужны временные переменные для выполнения вычислений или работы с данными в локальной области видимости, изолируя их от глобального контекста программы, помогают избежать конфликта имён и улучшить читаемость кода

```
(define circle-area radius)
    ((let ((pi 3.14) (area (* pi (*radius radius))))
        (display "Площадь равна ")
        (display area)))
(circle-area 3)
```

2

1)
$$\begin{tikzcd}
a \arrow[r] & (1) \arrow[r] \arrow[d] & (2) \arrow[r] \arrow[dl] & (3) \arrow[r] & 5 \\
 & (3) \arrow[d] & & & \\
 & (4) \arrow[r] & 5 & &
\end{tikzcd}$$
Задаётся a

2)
$$\begin{tikzcd}
b \arrow[r] & (2) \arrow[r] \arrow[d] & (3) \arrow[r] & 5 & \\
 & (1) \arrow[r] & (2) \arrow[r] \arrow[d] & (3) \arrow[r] \arrow[dl] & 5 \\
 & & (3) \arrow[d] & & \\
 & & (4) \arrow[r] & 5 &
\end{tikzcd}$$
Задаётся в

3)
$$\begin{tikzcd}
c \arrow[r] & (3) \arrow[r] \arrow[d] & 5 & & & \\
 & (2) \arrow[r] \arrow[dr] & (3) \arrow[r] & 5 & & \\
 & & (1) \arrow[r] & (2) \arrow[r] \arrow[d] & (3) \arrow[r] \arrow[dl] & 5 \\
 & & & (3) \arrow[d] & & \\
 & & & (4) \arrow[r] & 5 &
\end{tikzcd}$$
Задаётся с

3. Аппликативный порядок вычислений (Call by-value)
Это стратегия вычисления, при которой аргументы вычисляются перед применением функции, и только затем в неё передаются

Порядок следующий:
1) Вычисление всех аргументов ф-ии
2) Применение ф-ии к аргументам

(define (square x) (* x x))

(square (+ 2 5)) ; Сначала (+ 2 5), а затем (* 7 7)

В нормальном было бы
(* (+ 2 5) (+ 2 5))

Преимущества:
1) Предсказуемость. Порядок вычисления чёткий, что облегчает отладку
2) Избегание лишних вычислений (если один аргумент исп. неск. раз) не используемые аргументы не вычисл.

Недостатки:
1) Не позволяет реализовать оптимизации, если нельзя вычисл.
2) Если вычисл. аргументов дорогостоящее, а они не исп. в ф-ии, то неэффективно