# Lec 5 : Col100

## let Keyword & Scoping rules

**Scope :**

- The scope of a name begins from its definition and ends where the corresponding scope ends.
- Scopes end with definition of functions
- Scopes end with the keyword end in any let ---- in ---- end

# Scope Rules

- Scopes may be disjoint → a scope can't span two disjoint scopes
- Scopes may be nested
  - → no partial overlaps allowed

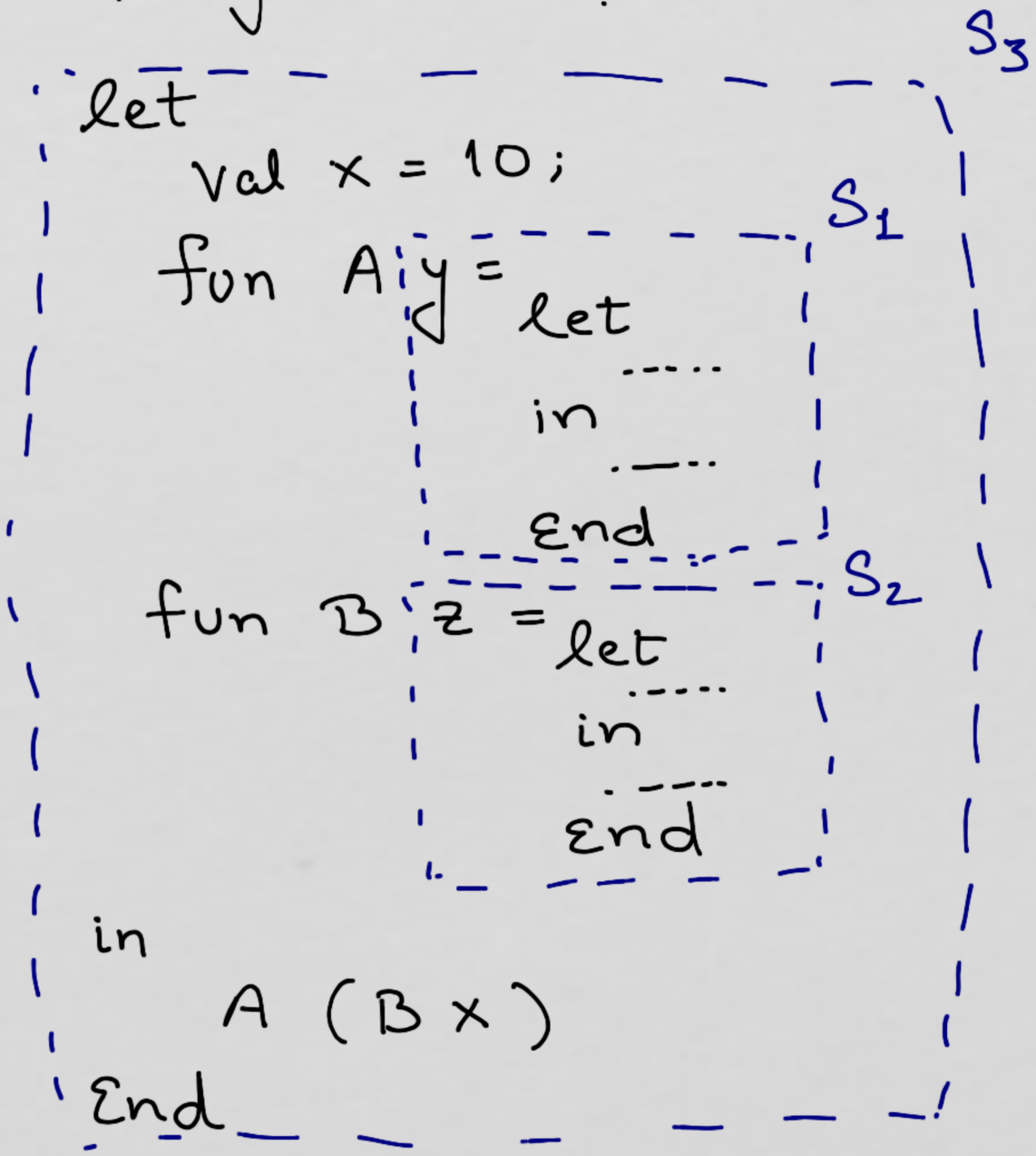
Eg:

```
fun sum_of_squares (x,y)
= let fun sq(a) = a*a;
  in
  sq(x) + sq(y)
  End;
```

Scope of x

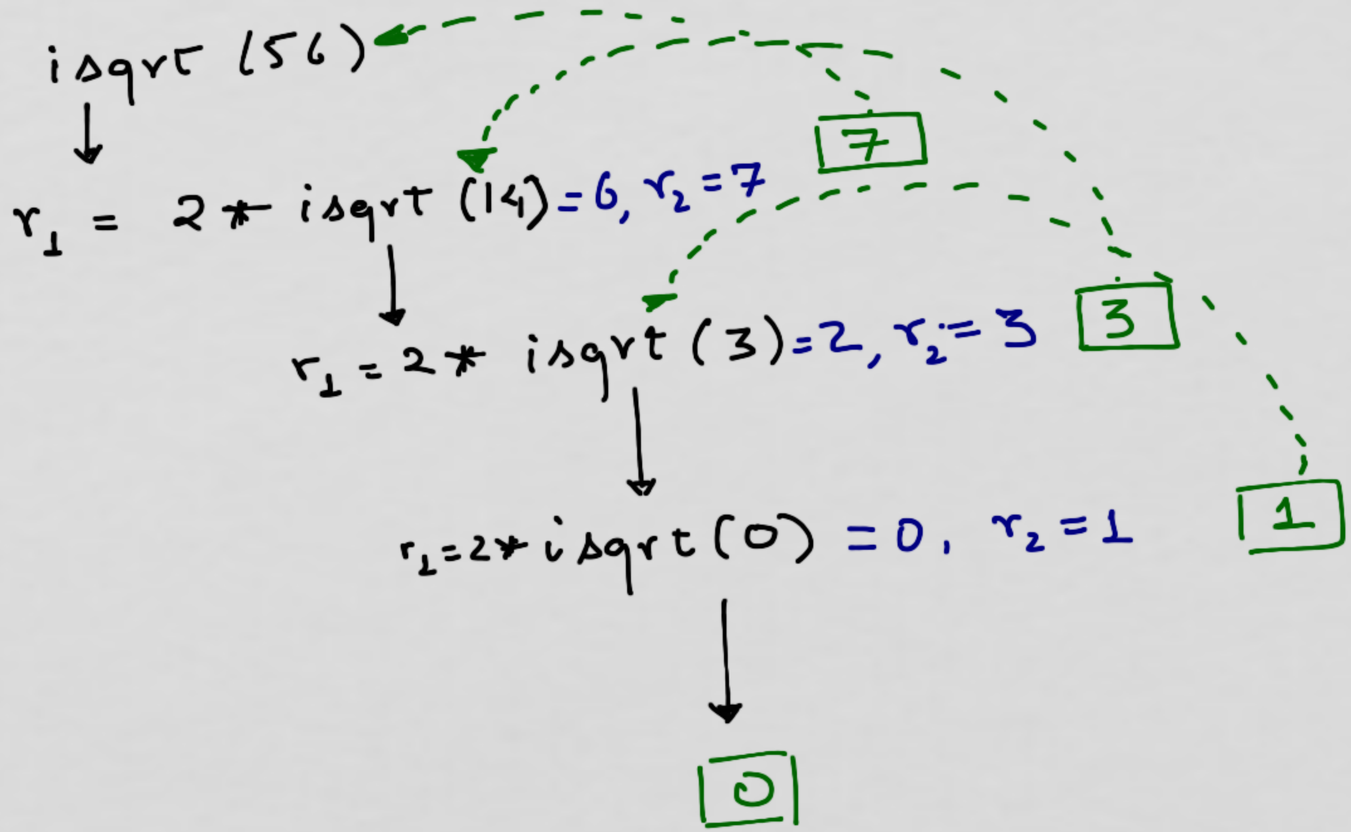
Scope of a?

# Disjoint, Nested Scopes




$S_1$ C $S_3$

$S_1$ & $S_2$ are disjoint




## SML Code

```
fun isqrt (n) =
    if (n=0) then 0
    else let val r1 = 2 * isqrt (ndiv4)
         in
             let val r2 = r1 + 1
             in
                 if (r2*r2) <= n then r2
                 else r1
             end
         end;
```

Another eg:

Perfect Numbers

Math behind it

$$n = \sum \{ k \mid 0 < k < n,\ k \mid n \}$$

$$n = \sum_{k=1}^{n-1} ij\,divisor(k)$$

I can safely change the upper bound to $n \operatorname{div} 2$ → why?

$$n = \sum_{k=1}^{n \operatorname{div} 2} ij\,divisor(k)$$

TASK: Check if a given number n is a perfect number

Our algorithm will be named

fun perject (n)

: $\mathbb{P} \longrightarrow \mathbb{B}$

$$ij\,divisor(k) = \begin{cases} k & ij\ \ k \mid n \\ 0 & otherwise \end{cases}$$

$$\sum_{l}^{u} ij\,divisor(k) = \begin{cases} 0 & ij\ \ l > u \\ ij\,divisor(l) + \sum_{l+1}^{u} ij\,divisor(k) & otherwise \end{cases}$$

```
Exception nonpositive;

fun perfect(n) =
    if n<=0 then raise nonpositive
    Else let fun sum_divisors (l,u) =
                 if l>u then 0
                 Else let fun ifdivisor(k) =
                              if n mod k =0 then k
                              Else 0
                      in
                      ifdivisor(l) +
                      sum_divisors(l+1,u)
                      in
                      n = sum_divisors(1, n div 2)
```

# Complexity Analysis

- Efficiency of an algorithm both in terms of space and time

fact(n) = if n=0 then 1
else n * fact(n-1)

Q: How many multiplications required for a problem of size n?

```
fact(n) = if n=0 then 1
          else n * fact(n-1)
```

Let $T(\text{fact}(n))$ be the number of multiplication operations required for size n

Then,

$$T(\text{fact}(n)) = \begin{cases} 0 & \text{if } n = 0 \\ 1 + T(\text{fact}(n-1)) & \text{otherwise} \end{cases}$$

So for $n > 0$, by telescoping

$$T(\text{fact}(n)) = 1 + T(\text{fact}(n-1))$$

$$= 1 + 1 + T(\text{fact}(n-2))$$

$$\vdots$$

$$= \underbrace{1 + 1 + \cdots}_{n \text{ times}} + T(\text{fact}(0))$$

$$= n$$

Similar expression for space & the number of invocations!

Why does one care?

→ Computers are becoming faster each passing year!

- Consider matrix multiplication

$\sim n^3$ operations

Assume ten fastest processor $\sim 10^9$ ops/sec

- when $n = 10$

time taken $= 10^3 \times 10^{-9} = 10^{-6}$ seconds

- when $n = 10^5$

time taken $\sim 10^6$ seconds $\sim$ 250 hrs

imagine

$$\tau(f(n)) = 1 + \tau\left(\frac{f(n)}{2}\right)$$

and

$$\tau(f(1)) = 1$$

Assume $f(n) = 2^m$

then

$$\tau(f(n)) = 1 + \tau(2^{m-1})$$

$$= 1 + 1 + \tau(2^{m-2})$$

$$\cdots$$

$$= 1 + 1 + \ldots + \tau(2^0)$$

$$= m + 1 = \log_2 n + 1$$

## New notion

Order of growth

: an independent measure of the resources required by a computational process

i.e. if n measures the size of the problem then we can measure resources reqd. by the algorithm as a function $R(n)$.

We say that $R(n)$ has an order of growth $O(f(n))$

i.e. $R(n) \leq K f(n)$

for some $K$

whenever $n \geq n_0$

Eg. for fact(n)

Space req $= n$

# of multiplications $= n$

# of func. calls $= n+1$

→ each of these have an order of growth $O(n)$

tells us which of the two competing algorithms will win eventually in the long run

for eg: however large K is

$K n$ will at some point onwards

will always be smaller than

$n^2$ or $2^n$