Lecture 3:

Recursion and Induction on natural numbers

A natural number is

either

- 0, or

- 1 + k, where

k is a natural number

0

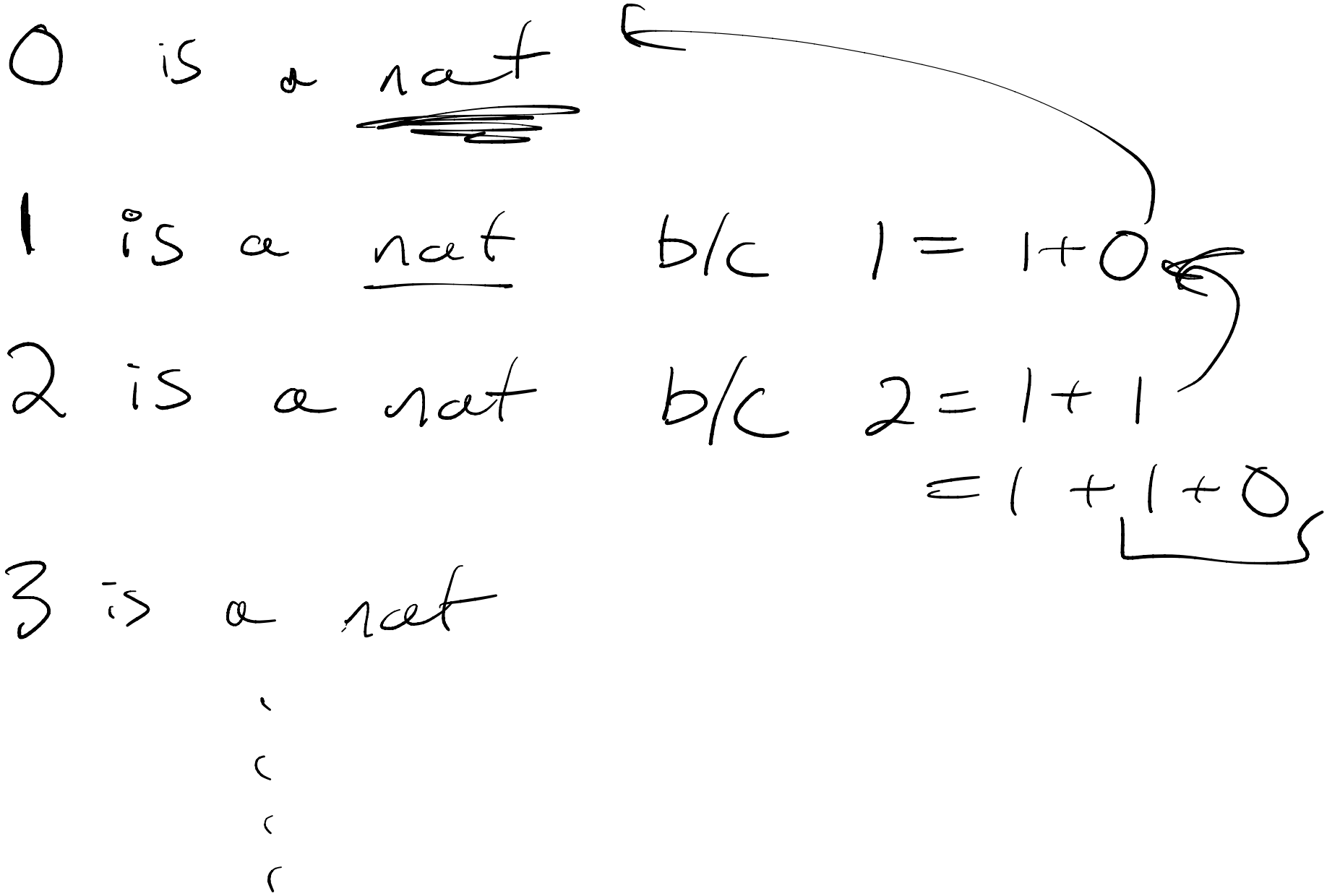
1

2

3

...

→ and that's it!


0 is a nat
1 is a nat b/c 1 = 1+0
2 is a nat b/c 2 = 1+1
= 1 + 1+0
3 is a nat

Recursion on a natural number

Purpose: double a number

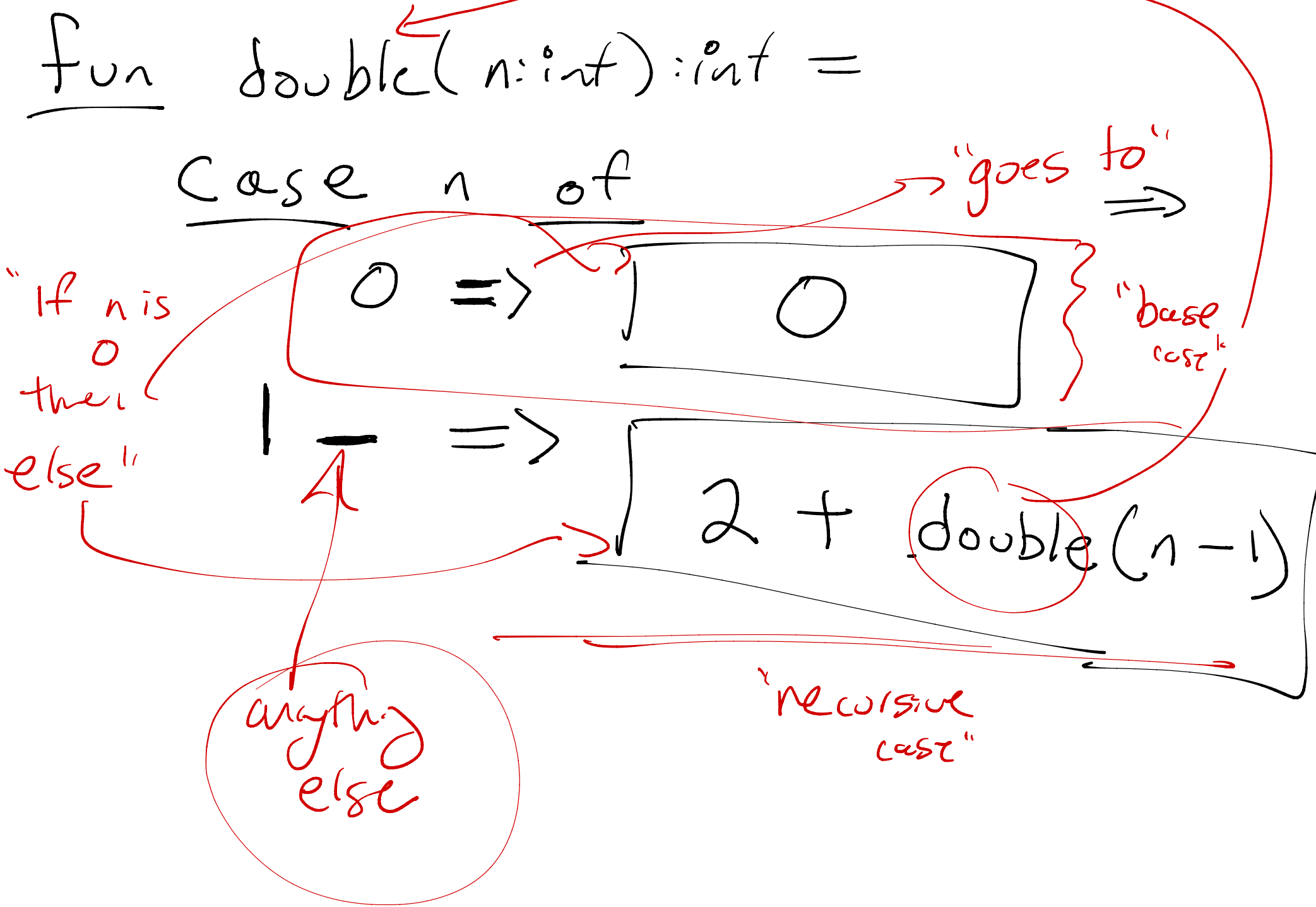
Examples: double(2) = 4

double(3) = 6

double(4) = 8 . . . .

```
fun double(n:int):int = 2*n
```

use int to represent nat


fun double(n:int):int =
case n of
0 => 0
"goes to"
"base case"
"If n is 0 then"
_ => 2 + double(n-1)
anything else
"recursive case"

double (2)

↦ case 2 of 0 ⇒ 0 | _ ⇒ 2 + double (2 − 1)

↦ 2 + double (2 − 1)

↦ 2 + double (1)

↦ 2 + (case 1 of 0 ⇒ 0 | _ ⇒ 2 + double (1 − 1))

↦ 2 + (2 + double (1 − 1))

↦ 2 + 2 + double (0)

↦ 2 + 2 + (case 0 of 0 ⇒ 0 | _ ⇒ double (0 − 1))

↦ 2 + (2 + 0)

↦ 2 + 2

↦ 4

double (1 + 1 + 1 + 1 + 1 + ... + 0)

= 2 + 2 + 2 + 2 + 2 + ... + 0

double (n1)

$\mapsto$ case n1 of 0 $\Rightarrow$ 0 | _ $\overset{2+}{\Rightarrow}$ double (n1 - 1)

$\mapsto$ 2 + double (n1 - 1)

$\mapsto$ 2 + double (n2)

.

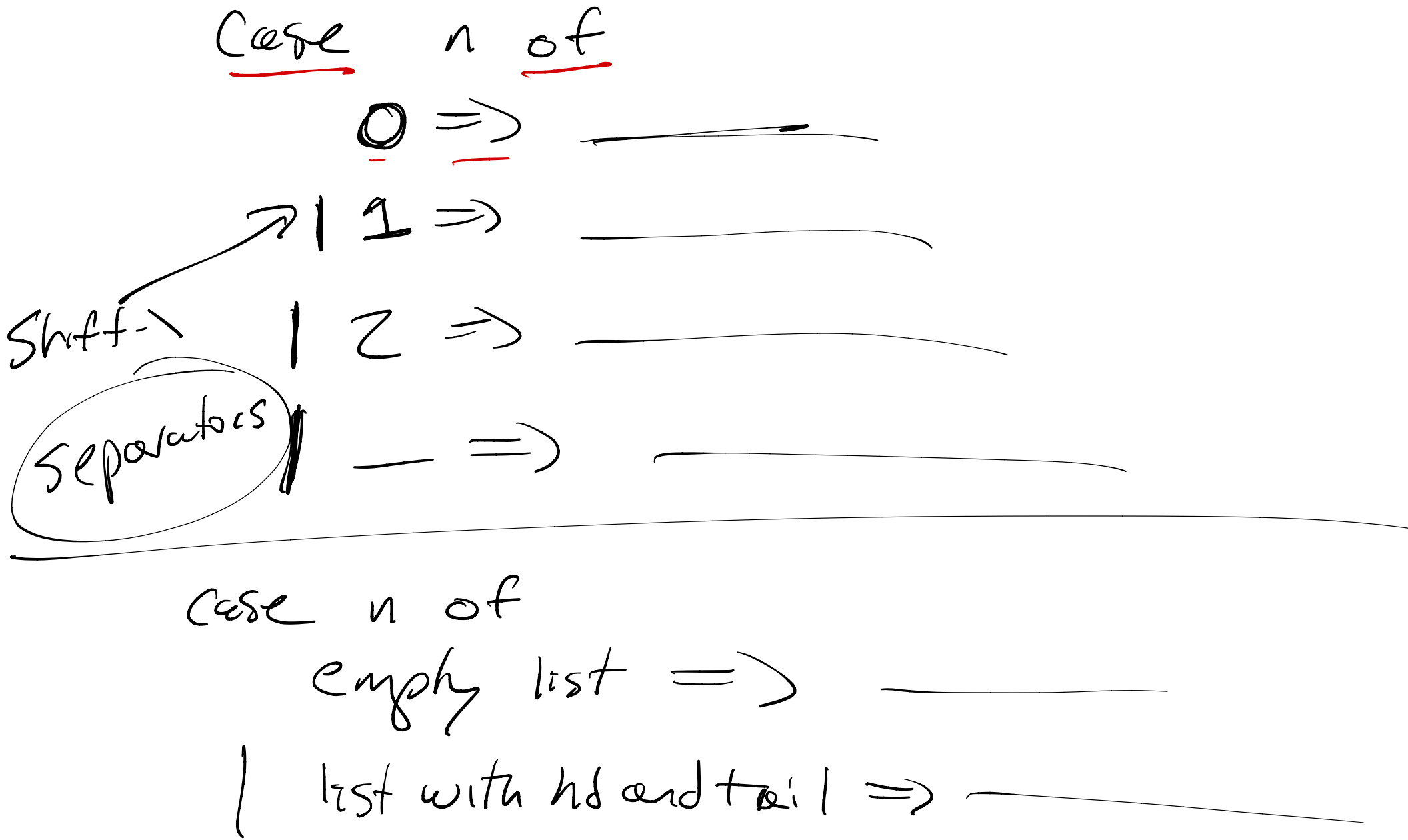
.

.

.

.

.


Case n of
0 =>
| 1 =>
Shift-\
| 2 =>
separators
| _ =>
case n of
empty list =>
| list with hd and tail =>

To step

[case n of
  0 ⇒ e_0
  1 — ⇒ e_1]

(1) step n until it is a value

(2) if n is 0, step to e_0

(3) if n is not 0, steps to e_1

Goal: compute 2^n for a natural number n

Examples: $exp(2) = 4 = 2^2$

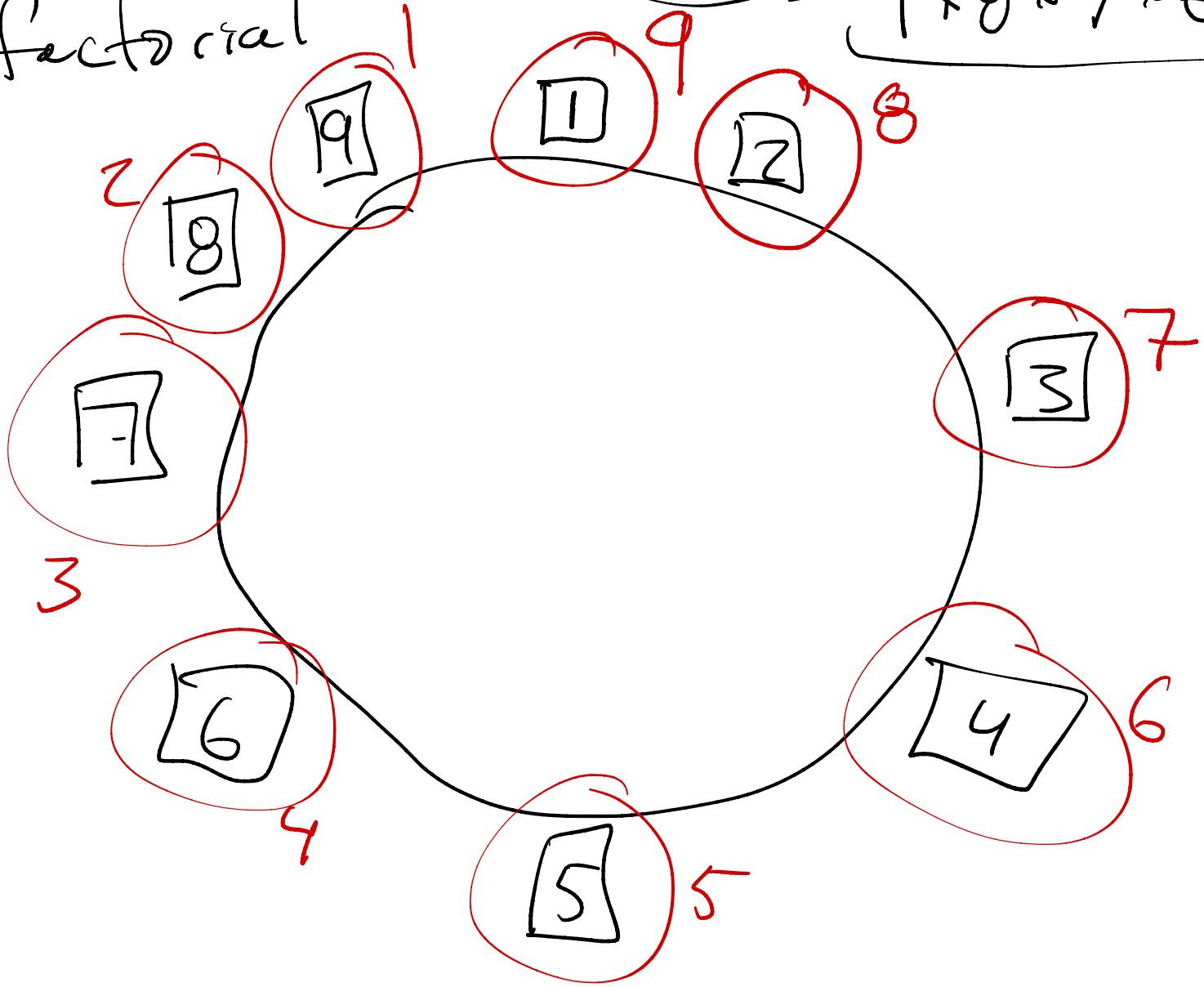
$exp(3) = 8 = 2^3$

$exp(4) = 16 = 2^4$

```
fun exp(n : int) : int =
  case n of
    0 => 1
  | _ => 2 * exp(n-1)
```

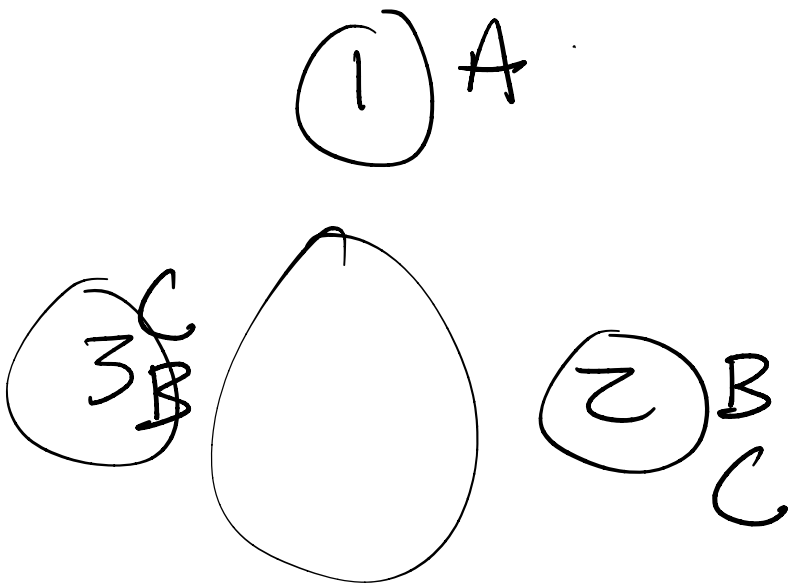
$\underbrace{2 * 2 * 2 * 2 * \ldots * 2 * 1}_{n \text{ of these}}$

"9 factorial" → $9 \times 8 \times 7 \times 6 \cdots \times 2 \times 1$



Buffet

How many arrangements of 9 people are there around the table?



A
B
C

1	2	3
A	B	C
A	C	B
B	A	C
B	C	A
C	A	B
C	B	A

```
fun fact (n:int):int =
    case n of
      0 => 1
    | n => n * fact(n-1)
```

---

fact(9)

↦ 9 * fact(8)

↦ 9 * 8 * fact(7)

↦ 9 * 8 * 7 * fact(6) . . . _

(* Goal: Given a number n,
Compute the string

booooo...oo
n letters *)

step 1

(* e.g. ghost(0) = "b"
ghost(2) = "boo"
ghost(4) = "boooo" *)

step 3

```
fun ghost(n : int) : string =
  case n of
    0 => "b"
  | _ => ghost(n-1) ^ "o"
```

Step 3

ghost(n-1) : string

ghost(2)

↦ ghost(1) ^ "o"

↦ (ghost(0) ^ "o") ^ "o"

↦ ("b" ^ "o") ^ "o" ⟼ "boo"

```
fun double(n) : int =
  case n of
    0 => 0
  | _ => 2 + double(n-1)
```

"recursive call"

```
fun ghost(n) : string =
  case n of
    0 => "b"
  | _ => ghost(n-1) ^ "o"
```

int

```
fun f(n:int) : T =
  case n of
    0 => e0 : T
  | _ => e1 using f(n-1) : T
```

int

string

int

T

string

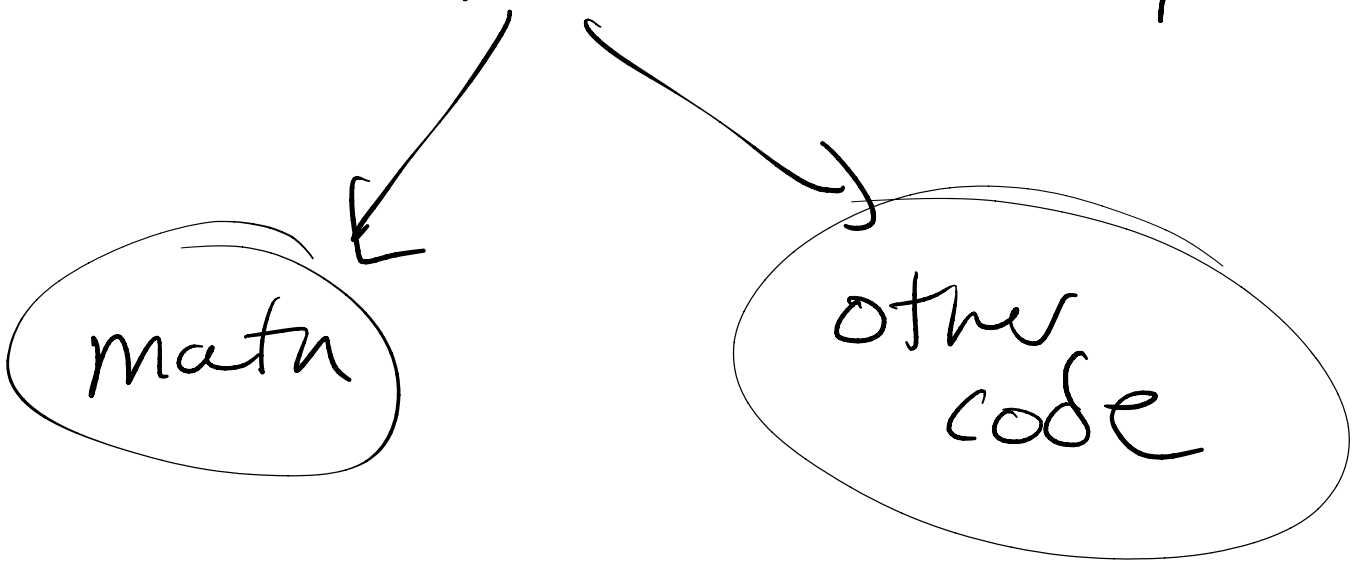
template

Methodology → How to design a function

1. Purpose in a comment (* comment *)
2. Name and type of function
3. Examples
4. Write body (using templates)
5. Tests

Specification of behavior:

Say <u>what</u> a function is supposed to compute



```
fun exp(n:int):int=
  case n of
    0 => 1
  | _ => 2 * exp(n-1)
```

Theorem:

For all natural numbers n,

$$exp(n) = 2^n$$

math

Case for 0:

To show: $exp(0) = 2^0$ { plug 0 in for n

Proof $exp(0)$

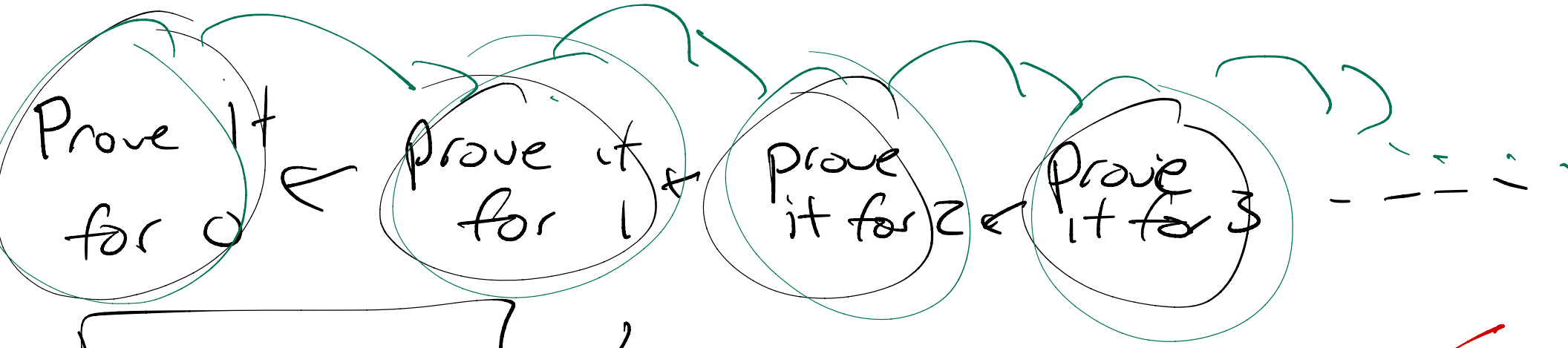
$\mapsto$ case 0 of 0 => 1 | ...

$\mapsto 1$

$= 2^0$

math

How do you prove something for all nat numbers?



Induction

- Prove it for 0 ✓
- Prove it for 1+k, assuming it is true for k

```
fun exp(n:int):int=
  case n of
    0 => 1
  | _ => 2 * exp(n-1)
```

Theorem:

For all natural numbers n,

$$exp(n) = 2^n$$

math

Case for $1+k$:

Inductive hypothesis: $exp(k) = 2^k$

To show: $exp(1+k) = 2^{1+k}$

Proof: $exp(1+k)$

$\mapsto 2 * exp((1+k)-1) \mapsto 2 * exp(k)$

$= 2 * 2^k$

$= 2^{1+k}$

$e_1 == e_2$ means

either (1) both e_1 and e_2 have the same value,

or (2) both raise the same exception,

or (3) both infinite loop