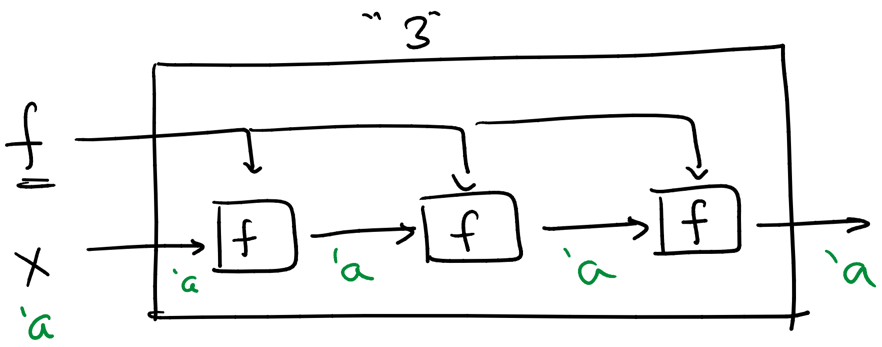
Lecture 9

Tuesday, September 21, 2021 1:58 PM

HW1 Q5.b

"What does the number "3" mean?"



"3" is the act of doing something 3 times.

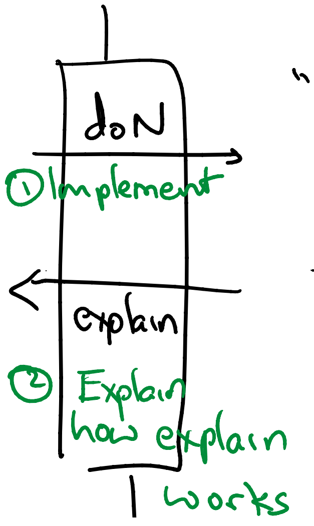
something → f

$$\boxed{\text{"3"} : ('a \to 'a) \to 'a \to 'a}$$

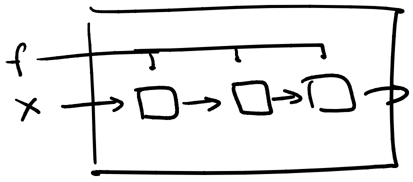
Two interpretations

3 the int

$\boxed{0000\ 0011}$



"3" the funny fn



how explain works

$(+) : int \to int \to int$

add : funny_fn → funny_fn → f . f

3 Define add

1 Ref cells

2 Mutable records

3 print_endline ; Base.stdio.printf ;
read_line [Stdio ; Imperative Prog. 1]

4 Imperative Programming 2

The "Semicolon"

1 let x = 3 ;; in utop

1 Finished typing stmt

② Please evaluate

Read | Eval Print Loop

① $\to$ ②

Only needed in utop

---

② let l = [1; 2; 3]

$\underbrace{;}$ $\underbrace{;}$ → Delimits elements of a list

---

(c)

```
int x = 5;
int y = 3;
x = y;
y = x+3;
```

"and-then"

x = 3, y = 6

```
let x = ref 5
let y = ref 3
(x := !y;
 y := !x + 3)
```

Order of evaluation is left unspecified

$e_1 ; e_2$ [Sequencing Statements]

① First evaluate e_1. Throw away its value.

② Now evaluate e_2. Return its value.

Imperative Programming 3 : Loops

Ⓒ
```
while (b) {
    // do_something
}
```

```
while bexp do
    e
done
```

```
let x=ref 0 in
while !x < 10 do
Stdio.printf "%d\n" !x;
x := !x + 1
done;;
```

Condition changing over time

"and-then" sequencing operation

"Doing" something.

whop syntactic trivia

---

① For loops

(1) for loops

```
for x = 0 to 10 do
Stdio.printf "%d\n" x
done
```

```
0
1
2
:
10
```

```
for x = 10 downto 0 do
Stdio.printf "%d\n" x
done
```

```
10
9
8
:
0
```

for var = int expr $\frac{\text{to}}{\text{downto}}$ int expr

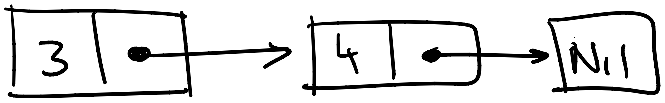
do

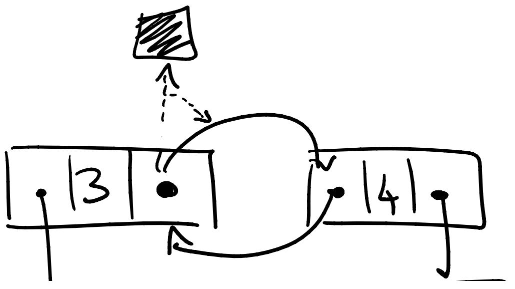
e

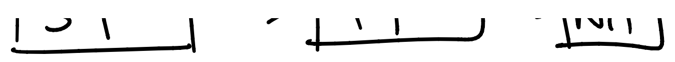
done

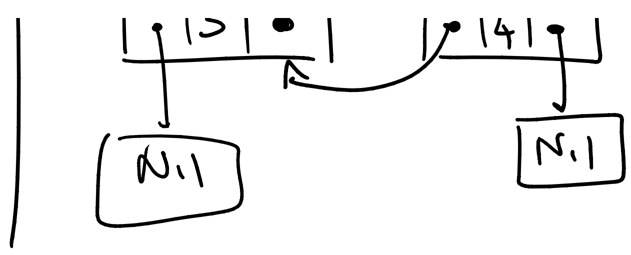
Challenge: Implement a doubly-linked list

[3; 4]









```
type dll = { mutable prev : dll option ;
value : int ; mutable next : dll option }
```

```
let dll3 = { prev=None; value=3;
next=None }
```

```
let dll4 = { prev=None; value=4;
next=None }
```

"Tying the Knot"

```
let rec even x =
    if x=0 then true
    else not (even (x-1))
```

Challenge: Can we define even without rec?