Introduction to Types

Q1: How do we "construct" data?

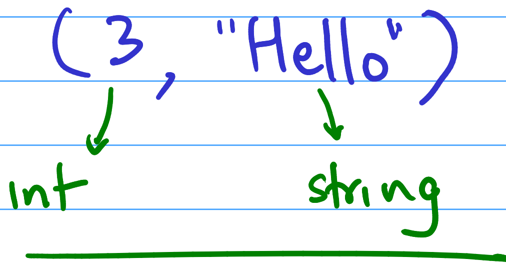
Q2: How do we "deconstruct" data?

Nothing to do with "destructors"/"finalizers".

Foo() / ~Foo()

- Pairs, tuples, variants, records, lists
- Pattern matching
- ADT (algebraic data type) & recursion

Pairs



---

fst : 'a * 'b → 'a
snd : 'a * 'b → 'b

"Accessors"

(3, "Hello") : constructor

Q: How to access the elements of a pair?

```
let myfst (p) = let ((p1), (p2)) = p in
  p1
```

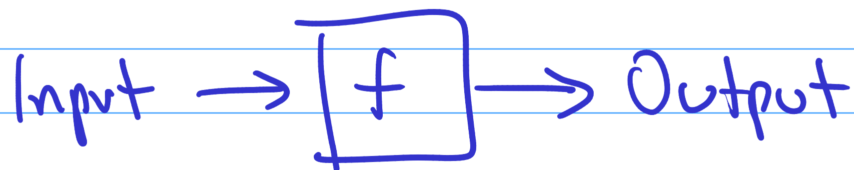
Only way for the pattern match to be successful is if p is a pair

Deconstructor / Pattern match

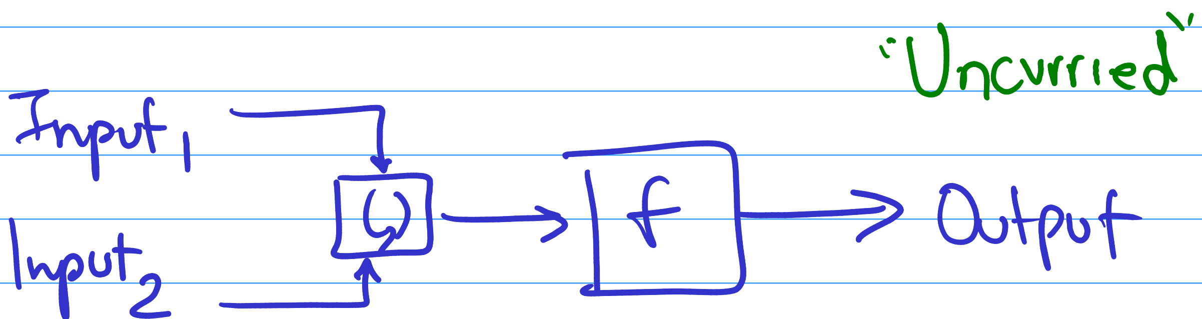
Bind p1 to the first element of the pair

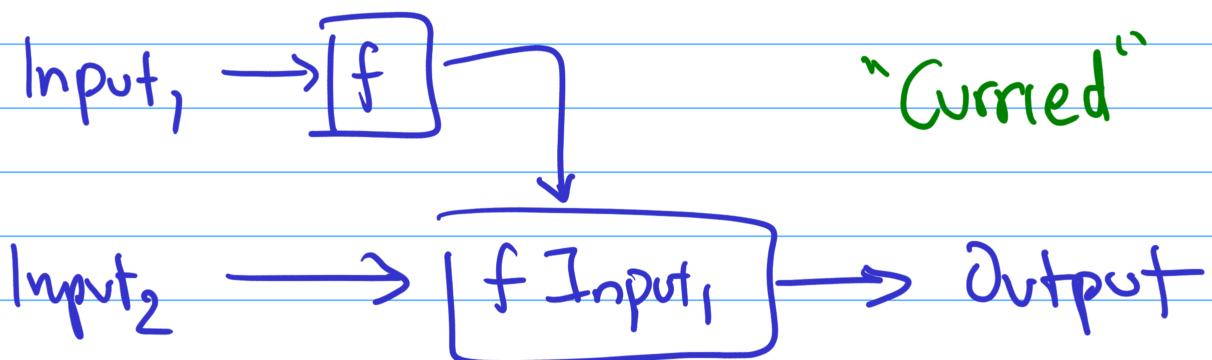
Bind p2 to the second element.

Technically, every fn in OCaml takes a single input & produces a single thing as output.



Q: How to devise functions with multiple inputs?





Lists

$[3; 4; 5]$ → Syntactic sugar

Constructors
- $[\,]$ → Empty list
- $a :: l$ → Take the list l & stick a in front of it

$[3; 4; 5] = 3 :: (4 :: (5 :: [\,]))$

Q: Check if a list is empty?

```
let is_empty l = (List.length l) = 0
```

1. Who defines List.length?
2. Requires $O(n)$ time

```
let is_empty l =
  match l with
  | [] -> true
  | hd::tl -> false
```

```
let car l =
  match l with
  | [] -> 0
  | hd::tl -> hd
```

car 3::(4::[])

⇓

3

Records

```
type person = {name : string; dob : int}

let p = {name = "Mukund"; dob = 1988}
```

(the `{` is labelled: Constructor)

```
p.name        p.dob
```

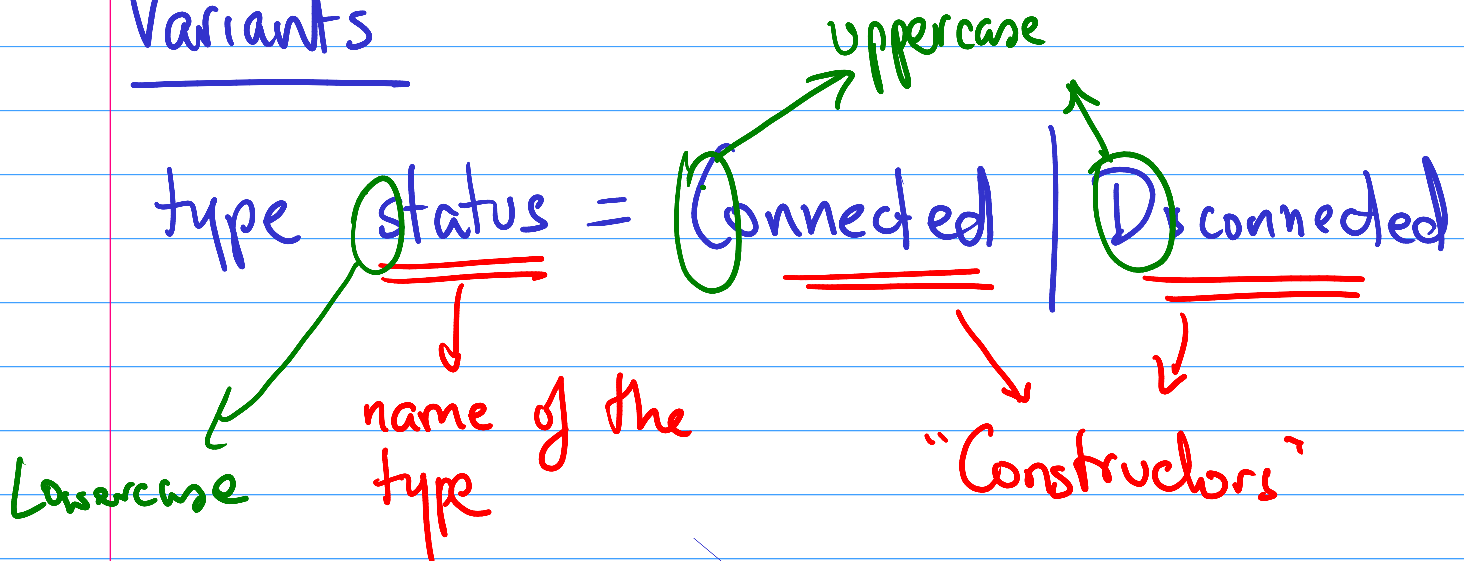
(both are labelled: Accessors)

```
let {name = pname; dob = pdob} = p
  in pname
```

(the pattern is labelled: Deconstructor)

Also, algebraic data type.

Variants

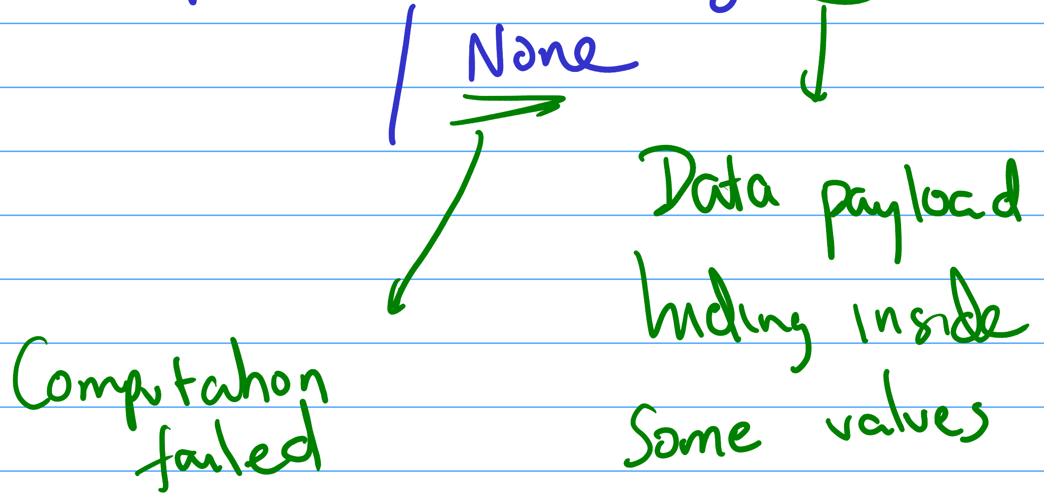


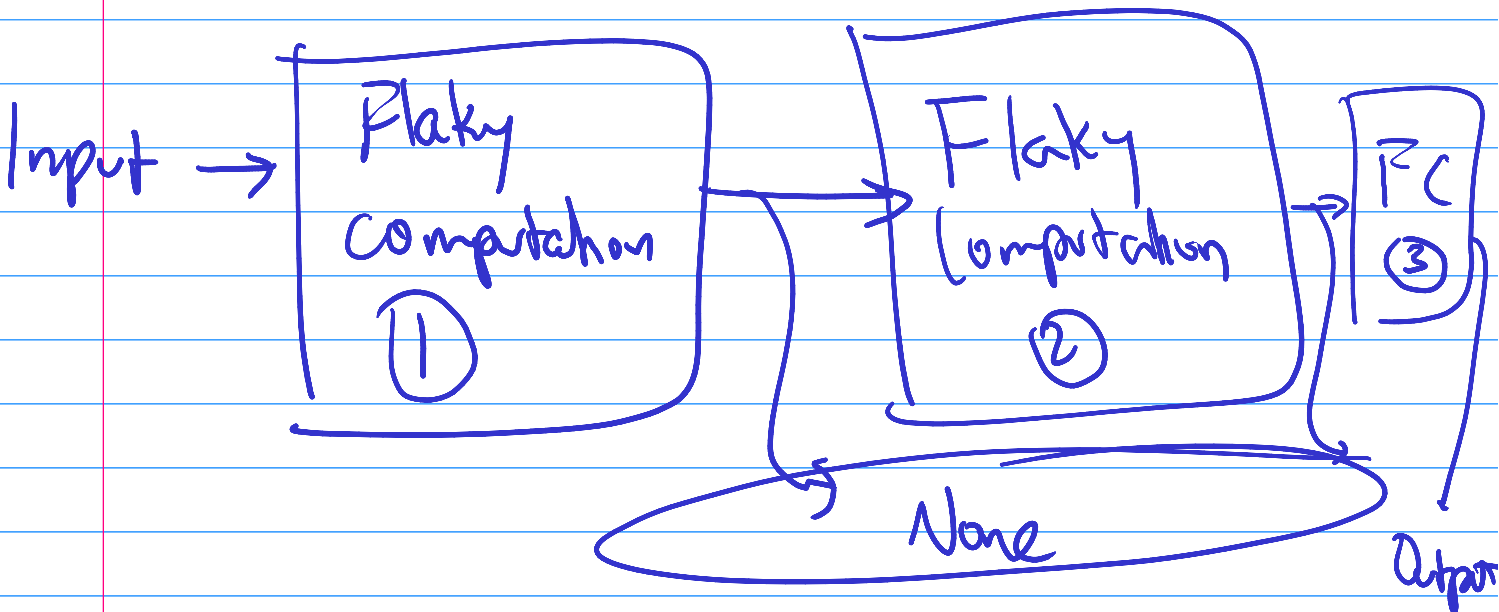
Claim 1: Connected ≠ Disconnected

Claim 2: Every value of type status is either Connected or Disconnected.

Option types

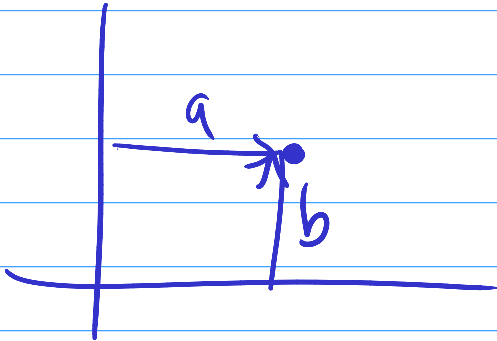
```
type 'a option = Some of 'a
```

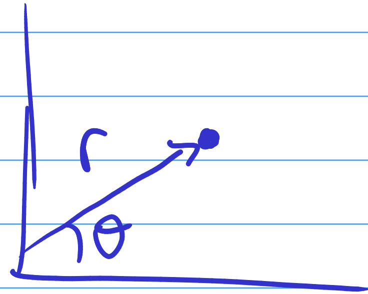




Maybe monad

Complex Numbers





Cartesian Polar

type complex = Cartesian of int * int
| Polar of int * int

There are 4×10^{18} integers in our world.

There are $16 * 10^{36}$ Cartesian complex numbers.

There are $16 * 10^{36}$ Polar c. #s

In total, $32 * 10^{36}$ c. #s.