- Installing OCaml } see CS162 github (section / sec 01)
- Running OCaml }
- Mutation vs binding
- Crash course on recursion for CS162.
  - Recipe
  - Demo: list length
  - ~~Exercise: list append~~
  - Exercise: tree mirroring

Mutation vs Binding

Consider:

<Python>

```
x = 1
x = x+1
print(x)
```

<OCaml>

```
let x = 1 in
let x = x + 1 in
print_int(x)
```

<OCaml>

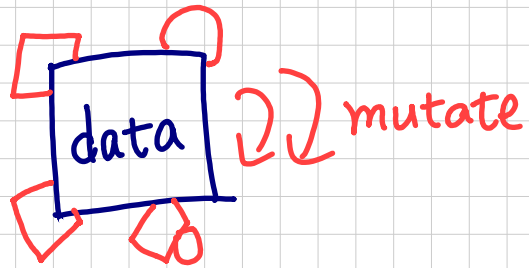
```
let x = 1 in
let y = (let x = x+1 in x) in
print_int(x)
```

In OCaml, "variables" don't vary.

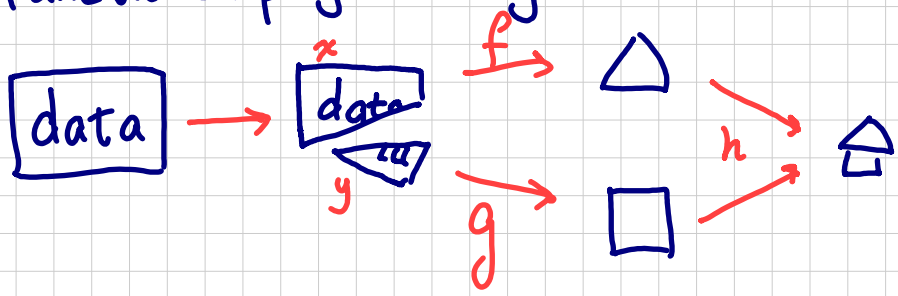
They aren't memory boxes!

Instead, they're names for values that are immutable.

Imperative programming



Functional programming



Loops

```
for i in range(100):
    do something
        - count += 1
        - arr[i] = ...
```

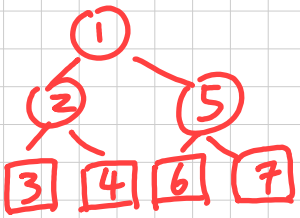
```
for i in rang(100):
    do nothing effectful
```

||

no OP

Recursion: compose solutions to smaller problems into sol'n to a big problem

Recipes for recursion (for CS162)



Examples

1. Identify the structure of the input ← FREE! data types
2. Decompose the structure in the most obvious way. FREE! ← pattern matching
3. Handle the base case.
4. Call f on the recursive sub-structure.
5. Assume the results of 4 is correct.

   compose the results into an overall sol'n.

(i) Binary trees — mirror

(ii) either leaf [x] or

or data & L & R.
(int) (tree) (tree)

(iii) Handle leaf case.

(iv) Call f on L & R

mirror([x]) = [x]

(v) Assume f(L) & f(R)
Solves the problem for
Now, construct sol'n
using data, f(L), f(R)

mirror(L) = J
mirror(R) = 9