National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2021/2022

R8
Mutable Data and Environment Model

Problems:

1. The following is the make_withdraw function shown in Lecture L7 and L8:

```
function make_withdraw(balance) {

    function withdraw(amount) {
        if (balance >= amount) {
            balance = balance - amount;
            return balance;
        } else {
            return "Insufficient funds";
        }
    }
    return withdraw;
}
```

```
mw (bal, pw)
    let attempt = 0.
    wd
      ↳ attempts ≥ 3
            ? disabled
            : try !== pw
              ? attempt += 1
                : wrong.
            : attempt = 0
              ↳ resetting.
```

Modify the make_withdraw function so that it creates password-protected accounts. That is, for example,

```
const acc = make_withdraw(100, "my_password");
acc(30, "his_passcode"); // returns "Wrong password; no withdraw"
acc(30, "my_password");  // returns 70
```

Moreover, to further increase security, we want an account to be disabled if it has been accessed three consecutive times with incorrect passwords. For example,

```
const acc = make_withdraw(100, "my_password");
acc(30, "his_passcode"); // returns "Wrong password; no withdraw"
acc(30, "my_password");  // returns 70
acc(10, "sesame");       // returns "Wrong password; no withdraw"
acc(15, "canola");       // returns "Wrong password; no withdraw"
acc(25, "olive");        // returns "Wrong password; no withdraw"
acc(30, "my_password");  // returns "Account disabled"
acc(30, "his_passcode"); // returns "Account disabled"
```

whats env model?

why do we care?

useful info?

↓

name : value binding.

|| scoping

(1) its dynamic!

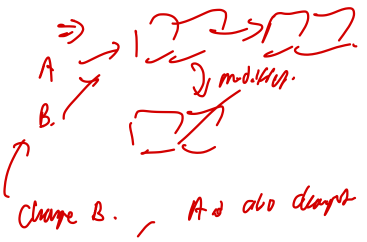
values ⇒ can be changed

(2) shared variables.

eg. destructive append.

env model = scoping + these.

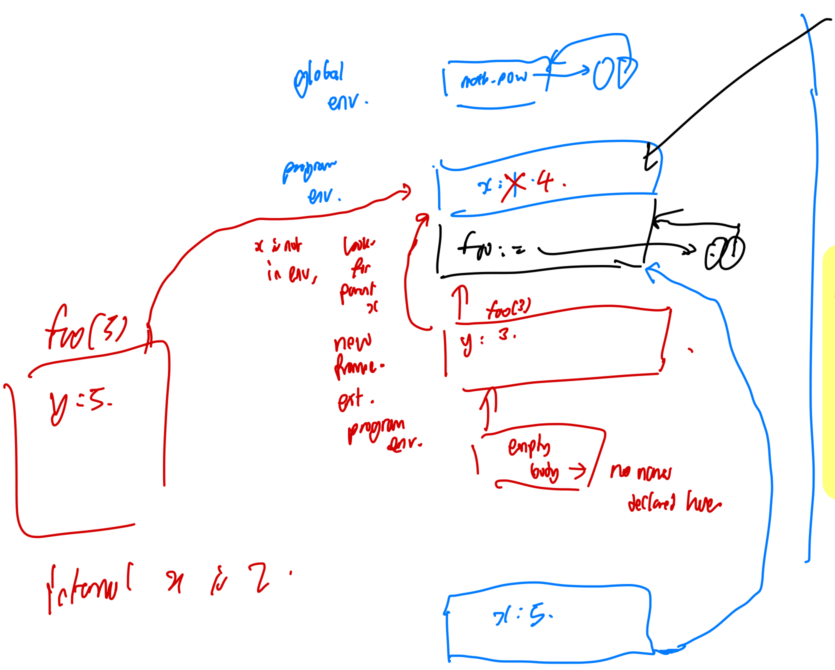
ext. of scoping rules



∴ shared variable

Conclusion? ← eg. github merges.

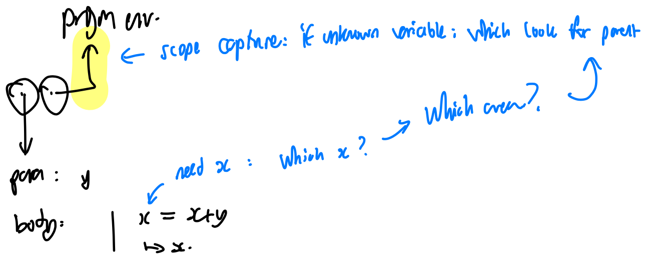
conflicts — must be resolved



```
let x=1.
function foo(y).
  x = x+y.
  return x.
foo(3);
{
let x=5.
foo(x);
}.
```

← only create new frame

foo ::=


prgm env.
← scope capture: if unknown variable: which look for parent
para : y
body:
x = x+y
↦ x.
need x : which x? → which env?

2. Consider the following program that makes a calculator function for a final price after adding a fixed commission, considering a given tax rate. What is the result of evaluating the program? Draw the environment diagram for the program according to the environment model. Show all the frames that are created during the program evaluation. Show the final value of each binding.

```
let commission = 25; // my commission in dollars

// return a calculator for total price
// total price = (commission + cost) * (1 + tax_rate)

function make_price_calculator(tax_rate) {
    function calculator(cost) {
        return (commission + cost) * (1 + tax_rate);
    }
    return calculator;
}

const calc = make_price_calculator(0.07);
commission = 125;
calc(75);
```

body.

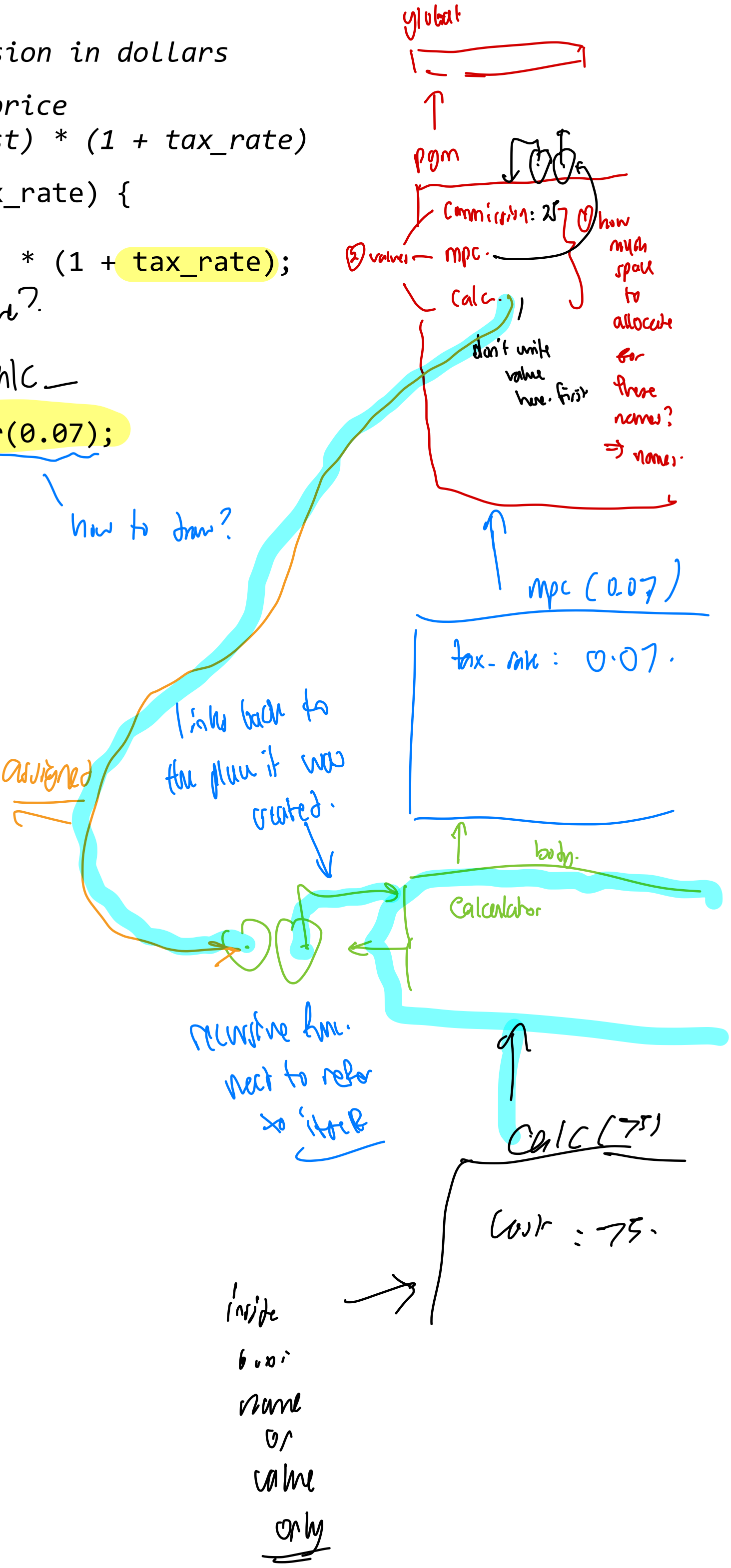
recursive?

calc

must equal to return value.

how to draw?

cost.


global
pgm
commission: 25
① how much space to allocate for these names?
⇒ names.
② values
mpc.
calc.
don't write value here. First
mpc (0.07)
tax_rate : 0.07.
link back to the place it was created.
assigned
body.
calculator
recursive fun. need to refer to itself
calc (75)
cost : 75.
inside box: name or value only

Ans in lecture.

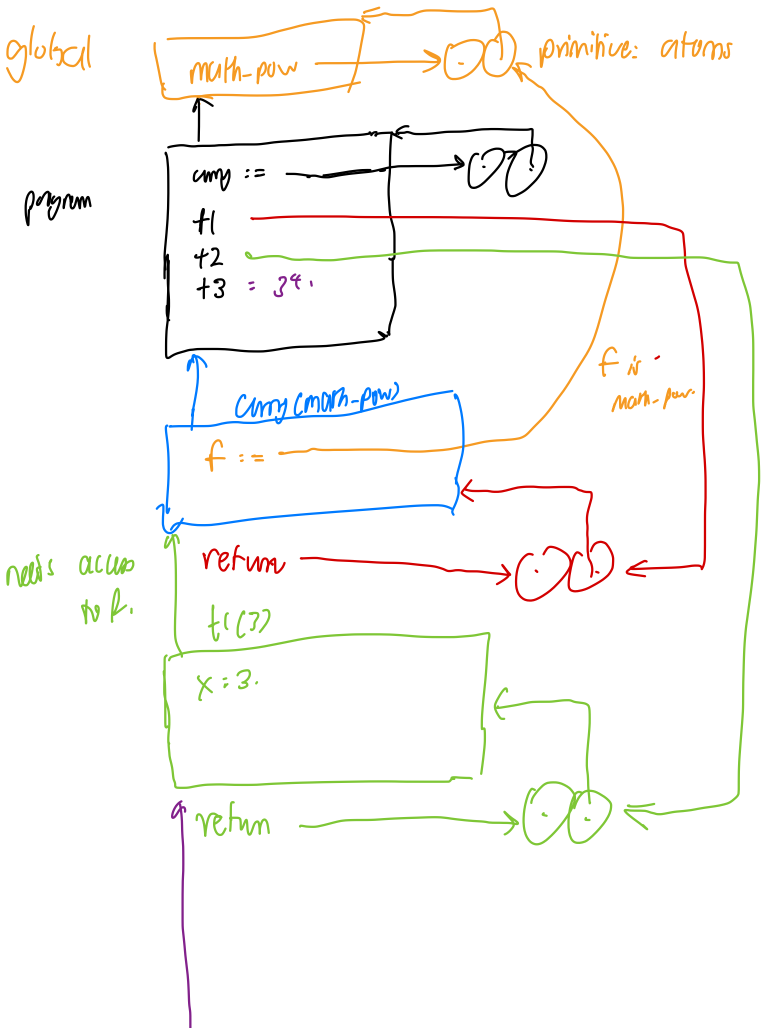
3. What is the result of evaluating the following program? Draw the environment diagram for the program according to the environment model. Show all the frames that are created during the program evaluation. Show the final value of each binding.

   The environment model distinguishes between **primitive** and **pre-declared** functions, and this distinction is laid down in the Specification of Source §3—2021 edition. Calls of primitive functions do not create any frames; they simply directly produce their result. Calls of pre-declared functions create new frames.

```
function curry(f) {
    return x => y => f(x, y);
}
(curry(math_pow))(3)(4);
```

   The program shows an example of the technique of *currying*, which converts a function that takes multiple arguments into a sequence of functions that each takes a single argument (https://en.wikipedia.org/wiki/Currying).

```
const f1 = curry(math_pow)
const f2 = f1(3);
const f3 = f2(4);
```



f2(4).

y = 3

return math.pow(3, 4).

$= 3^4$.