Evaluate:

- evaluate any program.

- break a string into parts.

- then apply.

To identify:

literals - 1, 2, 0.

names - const declaration.

function application - function

:

components.

↳ implement behaviour on how to evaluate them.

Simply everything to literal.

numbers, string, boolean, return val. directly.

name - name in current frame

lookup-symbol: symbol, env.

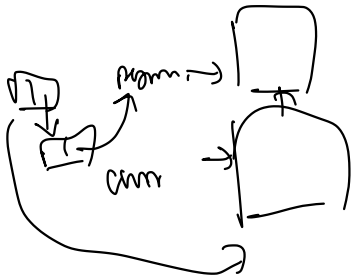
compare w/ this.

list containing all symbols declared in prgm

list of list.

head(list) = env.

tail = higher env.



op. combi — symbol of ops.

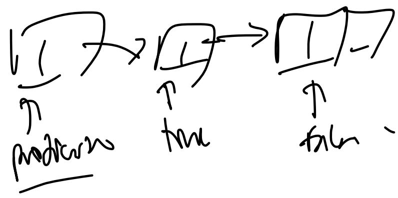
(+, -, *, /)

⇒ convert into function application.

Conditionals.

conditional_predicate (component)

→ head of list



lambda exp:

change into compound function

Sequence: many statements

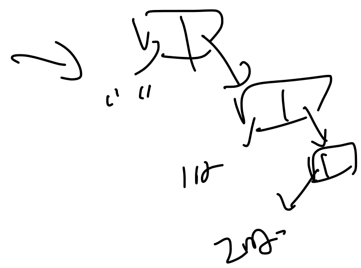
1. No statement → empty sequence.
2. Last statement.
3. Recursive.

↳ eval. first statement, ie return statement

return

use continue

a = == b;
b = == a;



block. { } → adds a frame to curr env.

→ scan out all local def.
init them as unassigned

+ eval. body.

↗ * return : eval. statement that will be returned
also.
- creates a return symbol.
- indicate termination of apply.

function
declaration.

Const var.
declaration.
(assign 1st then)

Const a = 2.
↑
eval 1st.

then assign a = 2 inside env.
↳ undefined

var assignment → eval. expr.
(change
number!)
assign symbol value.
return result. of evaluation

Function
application.

apply function to find regardless that
read arguments.

evals. each argument before applying
function / operator

→ procedure program to return the
result eval. or by use of a
seq of statements.

Apply : only compound & primitive functions

↓

extend env.
creates new frame
evaluate body.

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

S12

Problems:

1. There is a difference in the handling of function declarations between JavaScript on the one hand and the Source Academy and the MCE on the other hand: In JavaScript, the function declarations that appear anywhere in a statement sequence of a block are automatically moved to the beginning of the block. In JavaScript, the following program will produce the value 42:

```
{
    const x = f(8);
    function f(y) {
        return y + 34;
    }
    x;
}
```

because any JavaScript system will move the function declaration to the beginning of the block:

```
{
    function f(y) {
        return y + 34;
    }
    const x = f(8);
    x;
}
```

before the program is evaluated. Verify that this is not the case in the Source Academy or in the MCE. Modify the evaluator MCE 5 of Brief B11 such that it behaves like JavaScript implementations in this respect.

2. The evaluator MCE 5 of Brief B11 does not detect undeclared names. Therefore, the following program runs without error in the MCE:

```
false ? abracadabra(simsalabim) : 42;
```

The Source Academy, on the other hand, gives nice error messages for *any* name that is not declared. Modify the evaluator such that any undeclared name is detected and reported to the user as an error.

Hint: The function `scan_out_declarations` of the given MCE might come in handy.

1)

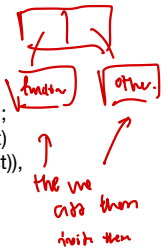
```
function scan_out_declarations(component) {
    return is_sequence(component)
        ? accumulate(append,
                     null,
                     map(scan_out_declarations,
                         sort(component))) // change
        : is_declaration(component)
        ? list(declaration_symbol(component))
        : null;
}

function sort(component) {
    const new_com = sequence_statements(component);
    let temp = null;
    const len = length(com);
    for (let i = 0; i < len; i = i + 1) {
        if (is_tagged_list(list_ref(new_com, k), "function_declaration")) {
            temp = pair(list_ref(new_com, k), remove(list_ref(new_com, k), new_com));
        }
    }
    return temp;
}

function eval_block(component, env) {
    let body = block_body(component);
    const scanned = scan_out_declarations(body);
    const unassigned = list_of_unassigned(scanned);
    if (is_sequence(body)) {
        body = pair("sequence", list(sort(body)));
    }
    return evaluate(body, extend_environment(scanned,
                                             unassigned,
                                             env));
}
```

Soln.

```
function reorder_statements(stmts) {
    // split_statements splits given stmts
    // into pair(function_declarations,
    //           all_other_statements)
    function split_statements(stmts) {
        if (is_null(stmts)) {
            return pair(null, null);
        } else {
            const first_statement = head(stmts);
            const split_rest = split_statements(tail(stmts));
            return is_function_declaration(first_statement)
                ? pair(pair(first_statement, head(split_rest)),
                       tail(split_rest))
                : pair(head(split_rest),
                       pair(first_statement, tail(split_rest)));
        }
    }
    const split = split_statements(stmts);
    return append(head(split), tail(split));
}
```



hoisting

2)

```
function eval_conditional(component, env) {
    evaluate(conditional_consequent(component), env);
    evaluate(conditional_alternative(component), env);
    return
is_truthy(evaluate(conditional_predicate(component),
env))
        ? evaluate(conditional_consequent(component),
env)
        : evaluate(conditional_alternative(component),
env);
}
```

Also scan

for return. at sequence.

is_name

look-up-symbol-value-

making parse-and-eval-

Soln
create frame w/o eval., leave them as special val.
unassigned

If it works and don't spoil other functions