S2A: Substitution Model, Recursion, Iteration

CS1101S AY20/21 Sem 1

Studio 2D

Lee Wei Min

Table of Contents

- Review of missions
- Substitution model
- Applicative order vs normal order
- Recursive processes vs iterative

Additional Information

- Don't worry about the math
- Plagiarism check
- Don’t abuse unsubmits!
- 24 hours, please follow up
- Class part – last week not counted

Mission Review

- Code style -> will start penalising poor organisation
- Code comments
- Counting up vs down
- “Compressing” information in parameters
- Functional abstraction
- Autograder
- For details, please see individual comments

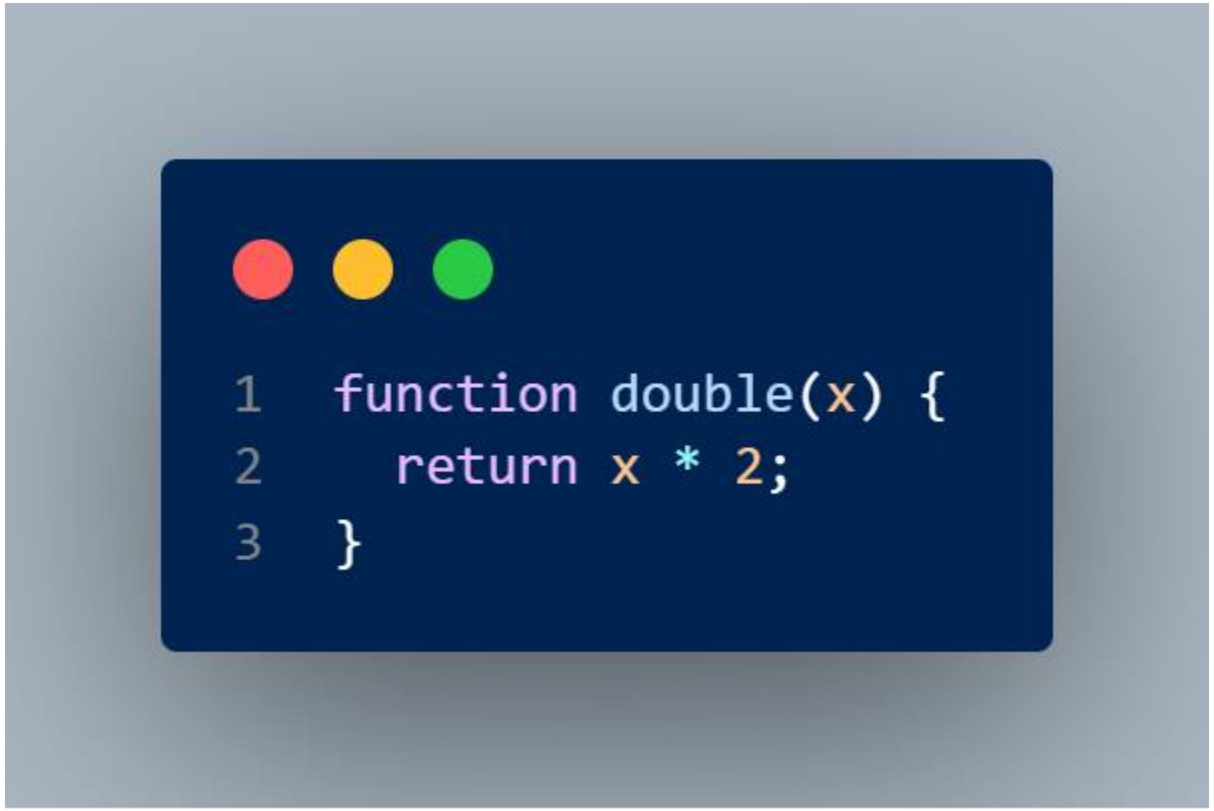
How to prepare for RA??

Substitution Rule

To evaluate an application

- Evaluate the operator to get procedure
- Evaluate the operands to get arguments
- Apply the procedure to the arguments
  - Copy the body of the procedure, substituting the arguments supplied for the formal parameters of the procedure.
  - Evaluate the resulting new body.

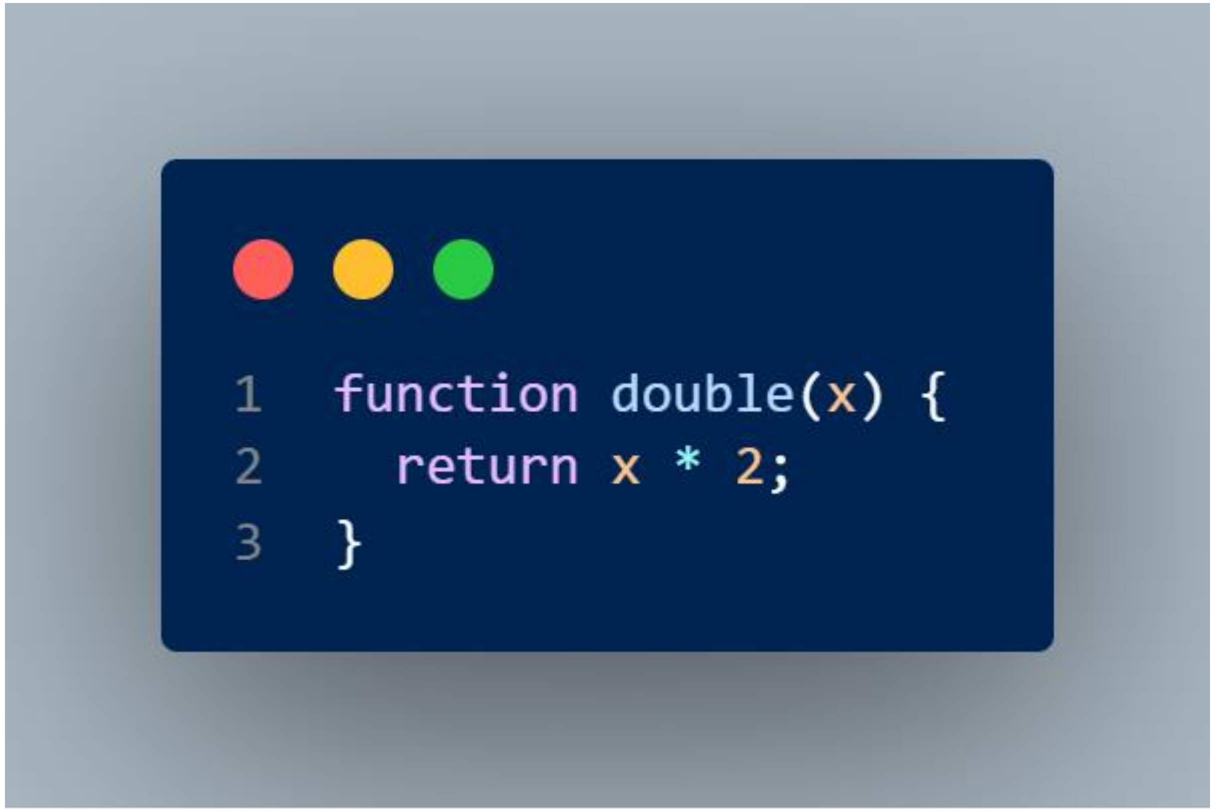
Applicative Order



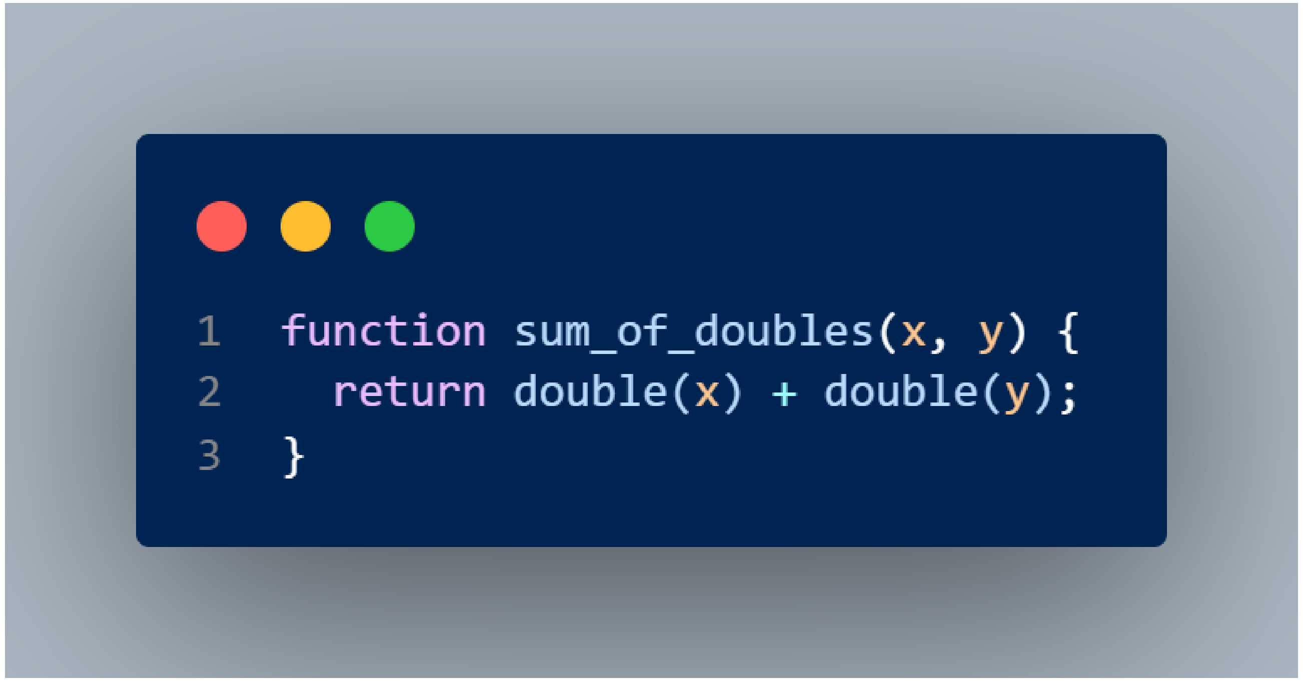
```
function double(x) {
  return x * 2;
}
```

Substitution in Action: double(2)

1. Evaluate the body of function -> x * 2
2. Evaluate the arguments -> x = 2
3. Replace parameters by arguments -> 2 * 2
4. Return 4 (* is a primitive function)



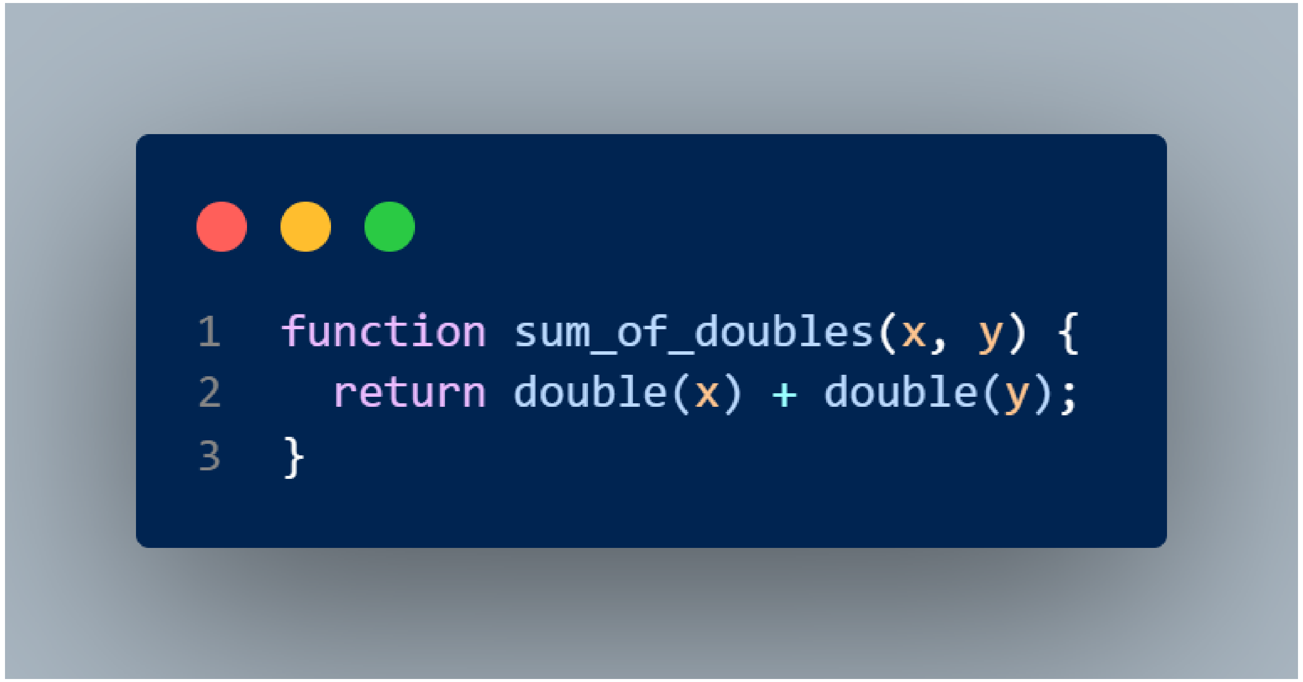
```
function double(x) {
  return x * 2;
}
```



```
function sum_of_doubles(x, y) {
  return double(x) + double(y);
}
```

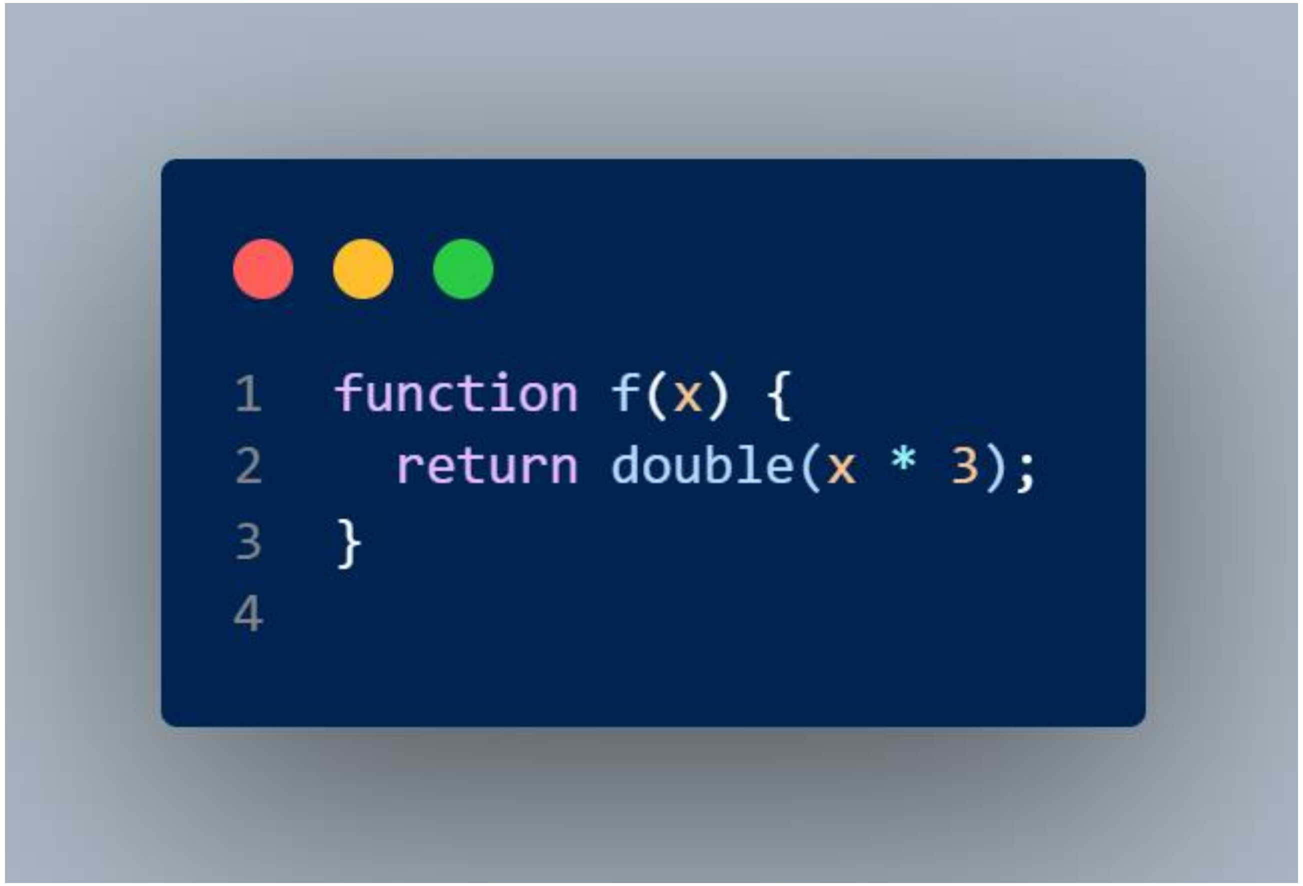
Substition in Action: sum_of_doubles(1, 2)

1. Evaluate the body of function -> double(x) + double(y)
2. Evaluate the arguments -> x = 1, y = 2
3. Replace parameters by arguments -> double(1) + double(2)
4. Same thing again – see previous slide! -> 1 * 2 + 2 * 2
5. Primitive functions + operator precedence -> return 6



```
function sum_of_doubles(x, y) {
  return double(x) + double(y);
}
```

Normal Order

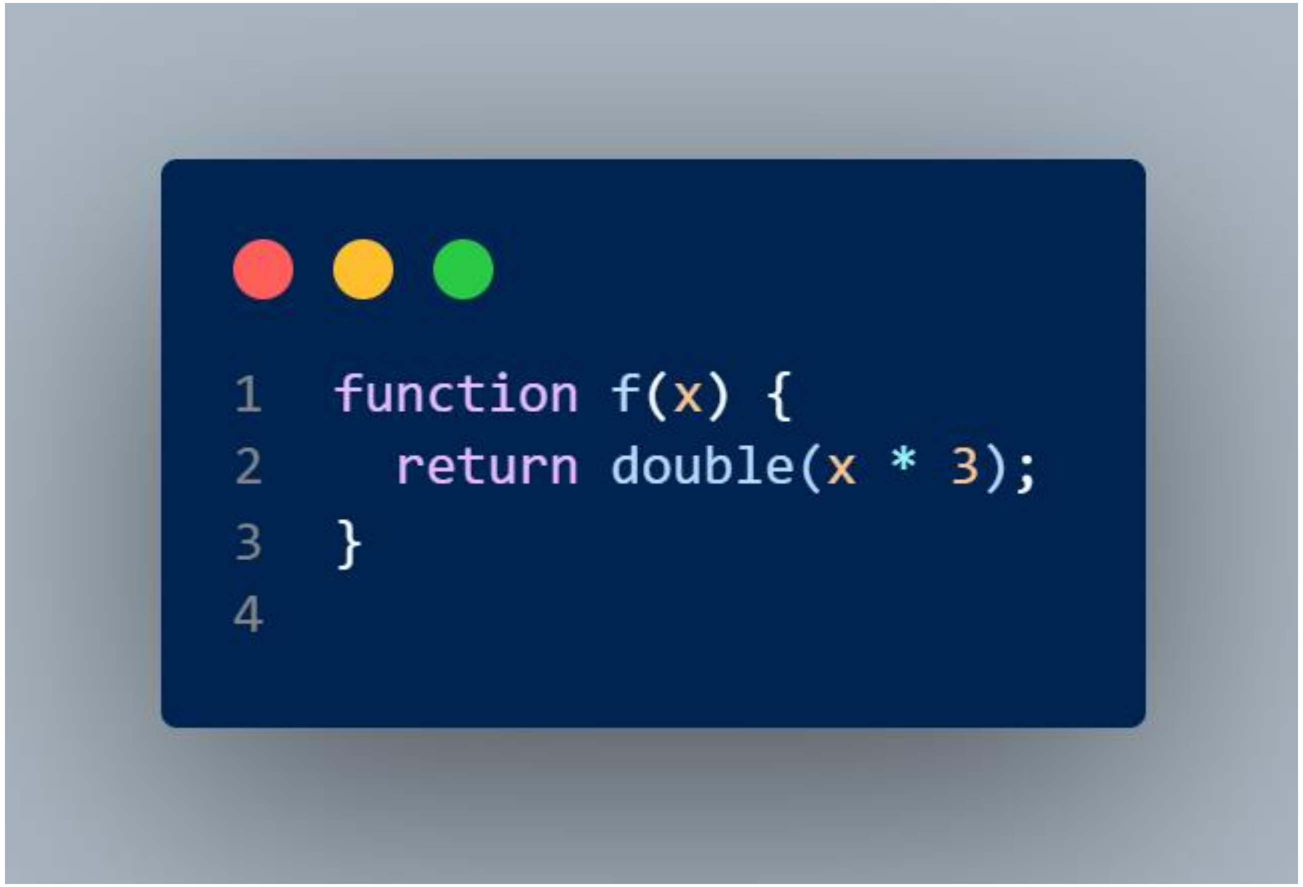


```
function f(x) {
  return double(x * 3);
}

```

Substitution in Action: f(2)

1. Evaluate the body of function -> double(x * 3)
2. Evaluate the arguments -> x = 2
3. Replace parameters by arguments -> double(2 * 3)
4. Avoid evaluating argument -> 2 * 3 * 2
5. Primitive functions + operator precedence -> return 12

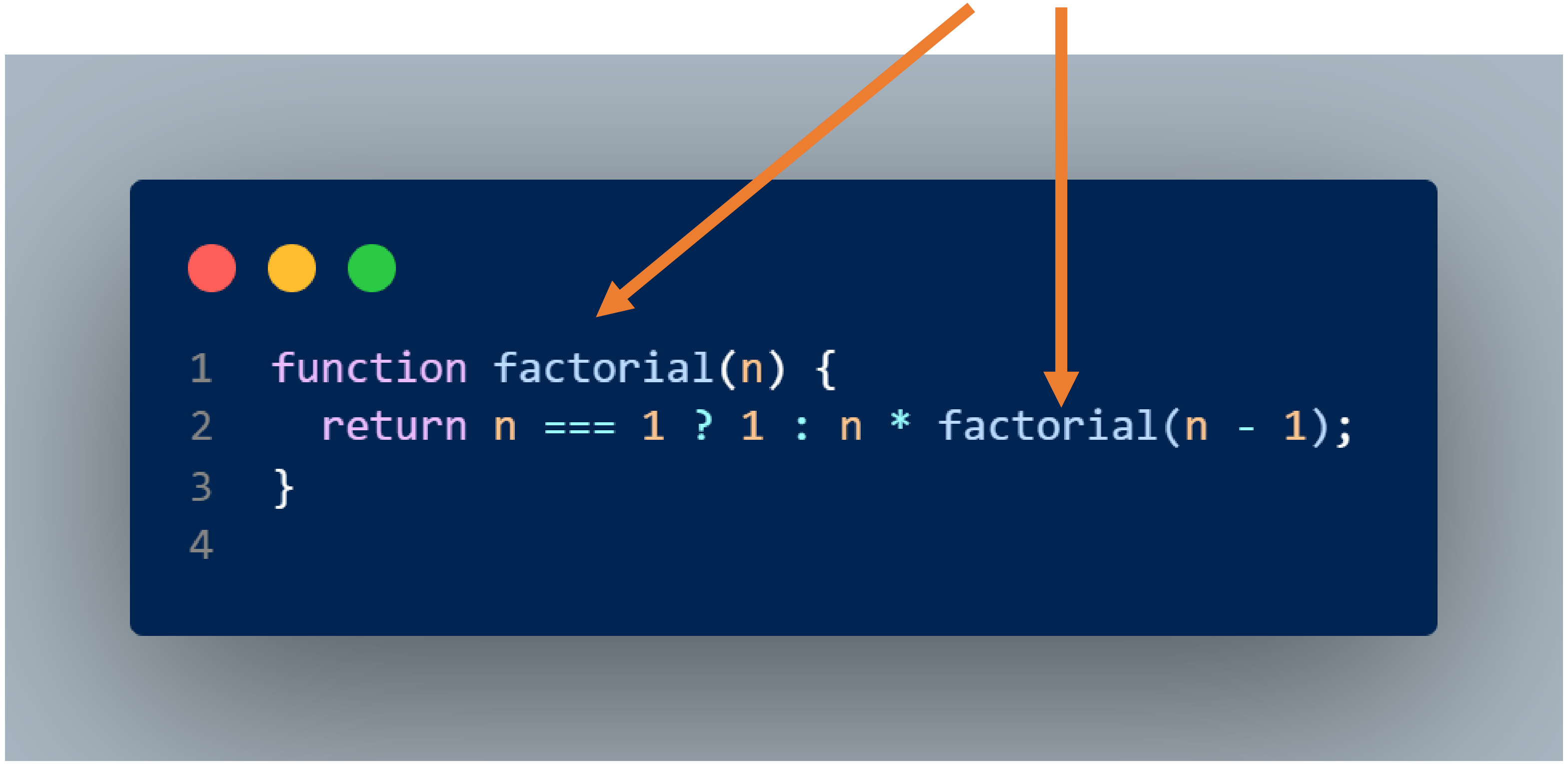


```
function f(x) {
  return double(x * 3);
}
```

Recursion

Recursive Function

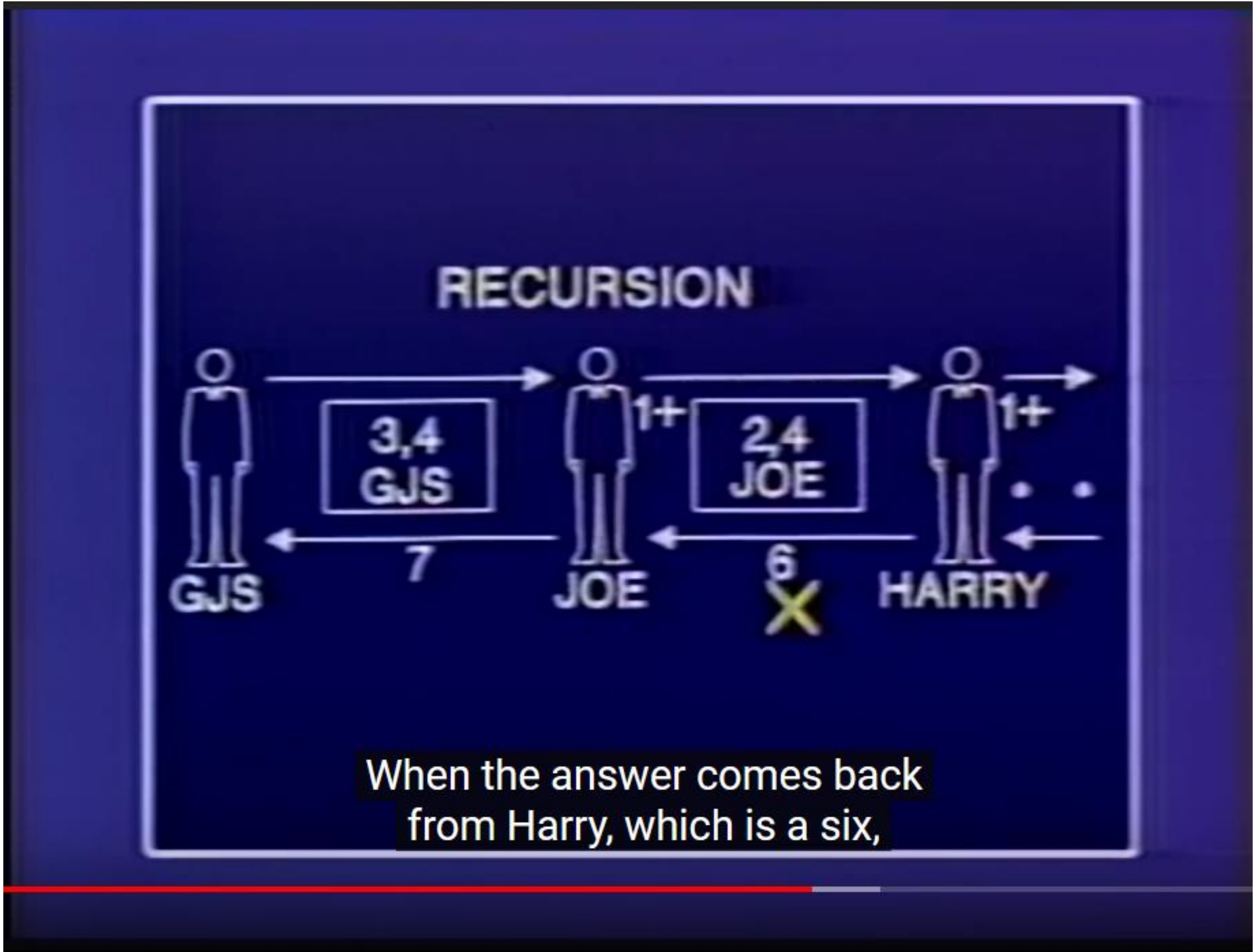
function declaration refers to itself



```
function factorial(n) {
  return n === 1 ? 1 : n * factorial(n - 1);
}
```

Recursive Process

- How the function application is evaluated
- Deferred operations

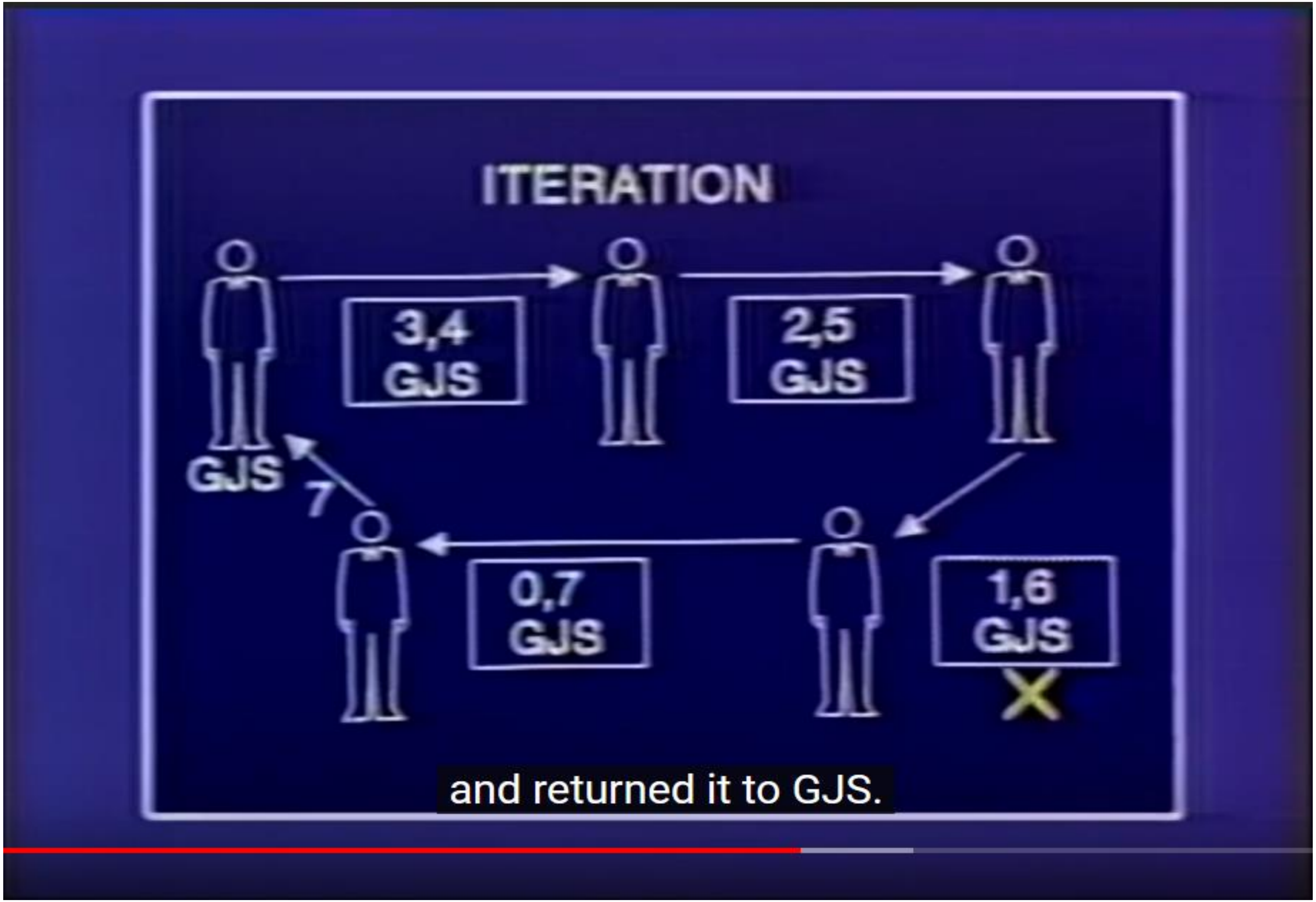


RECURSION

When the answer comes back
from Harry, which is a six,

Iterative Process

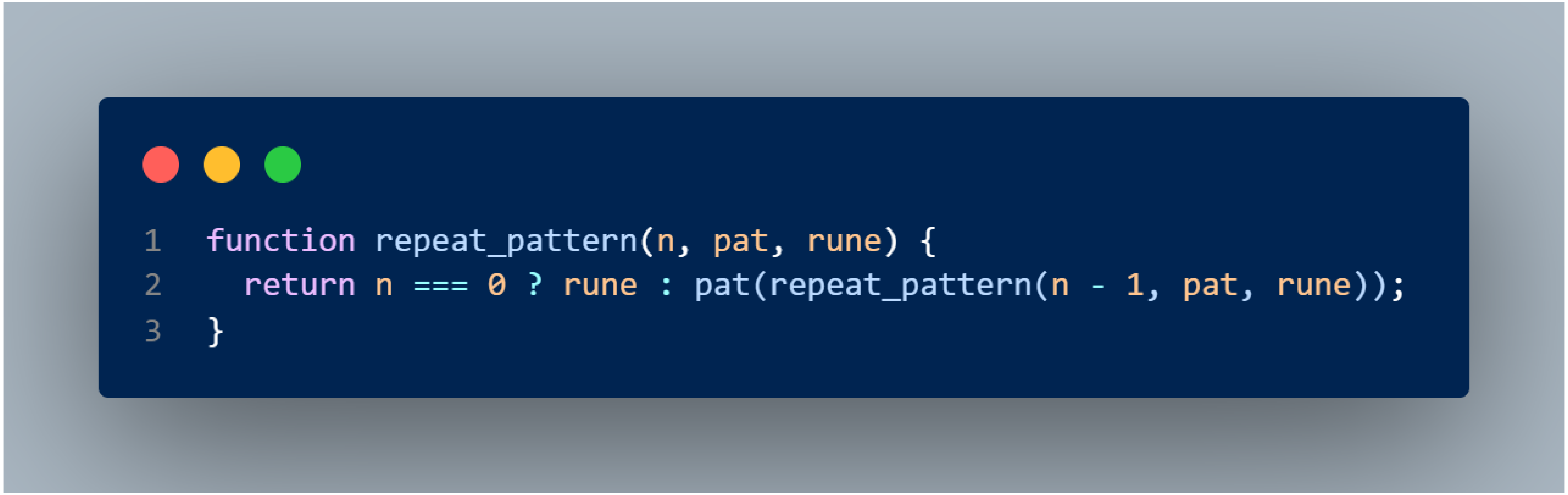
- State can be summarized by a fixed number of state variables
- Rule for updating state variables
- End test for termination
- No deferred operations



ITERATION

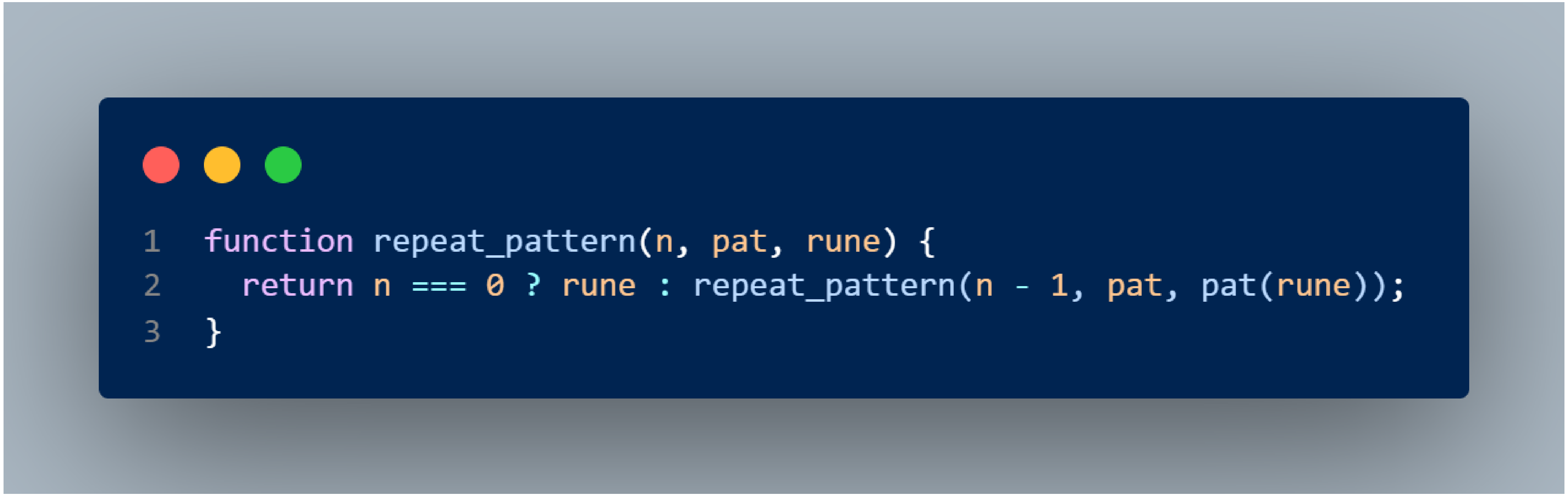
and returned it to GJS.

Recursive or Iterative?



```
function repeat_pattern(n, pat, rune) {
  return n === 0 ? rune : pat(repeat_pattern(n - 1, pat, rune));
}
```

Recursive or Iterative?



```
function repeat_pattern(n, pat, rune) {
  return n === 0 ? rune : repeat_pattern(n - 1, pat, pat(rune));
}
```

Studio

Resources

- Lecture
- Textbook: 1.1.5, 1.2.1