CS339: Abstractions and Paradigms for Programming

Recursion and Iteration

Manas Thakur
CSE, IIT Bombay

Autumn 2025

Example: Newton's method for computing square roots

- Start with a guess and "improve" the guess until it is "good enough"
- Square root of 2:

Guess [y]	Quotient [x/y]	Average [(y+x/y)/2]
1	2/1 = 2	(1+2)/2 = 1.5
1.5	2/1.5 = 1.3333	1.4167
1.4167	1.4118	1.4142
1.4142	…	…

- Say we stop when the square of the guess is equal to the number up to three decimal places.

Example: Newton's square root [Cont.]

```
(define (sqrt-iter guess x)
  (if (good-enough? guess x)
      guess
      (sqrt-iter (improve guess x) x)))

(define (improve guess x)
  (average guess (/ x guess)))

(define (average x y)
  (/ (+ x y) 2))

(define (good-enough? guess x)
  (< (abs (- (square guess) x)) 0.001))

(define (sqrt x)
  (sqrt-iter 1.0 x))
```

```
(define (abs x)
  (if (< x 0)
      (- x)
      x))

(define (square x)
  (* x x))
```

- Can you identify two "bad" things about this code?

Example: Newton's square root [Cont.]

```
(define (average x y) ...)

(define (square x) ...)

(define (abs x) ...)

(define (sqrt x)
  (define (improve guess)
    (average guess (/ x guess)))
  (define (good-enough? guess)
    (< (abs (- (square guess) x)) 0.001))
  (define (sqrt-iter guess)
    (if (good-enough? guess)
        guess
        (sqrt-iter (improve guess))))
  (sqrt-iter 1.0))
```

Packaged together → Namespace Abstraction

Unnecessary arguments gone → Lexical Scoping

What's the secret behind `sqrt-"iter"`?

Let's look at the processes generated by procedures

- Factorial of a number:

$$fact(n) = \begin{cases} 1 & , n = 1 \\ n * fact(n-1) & , o/w \end{cases}$$

- A procedure to compute the same:

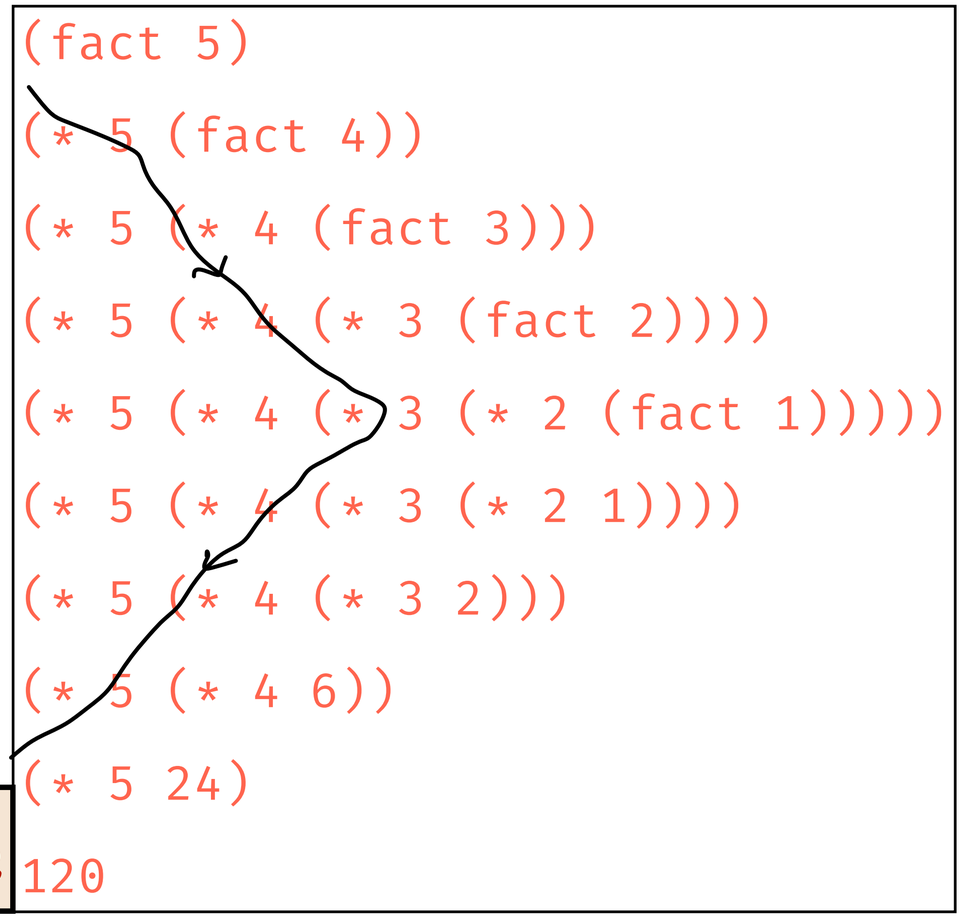
```
(define (fact n)
  (if (= n 1)
      1
      (* n (fact (- n 1)))))
```

The generated process for `fact(5)`

```
(define (fact n)
  (if (= n 1)
      1
      (* n (fact (- n 1)))))
```

Time: O(n)

Space: O(n)



Recursive Process

How about this one?

- Another way to compute factorial:

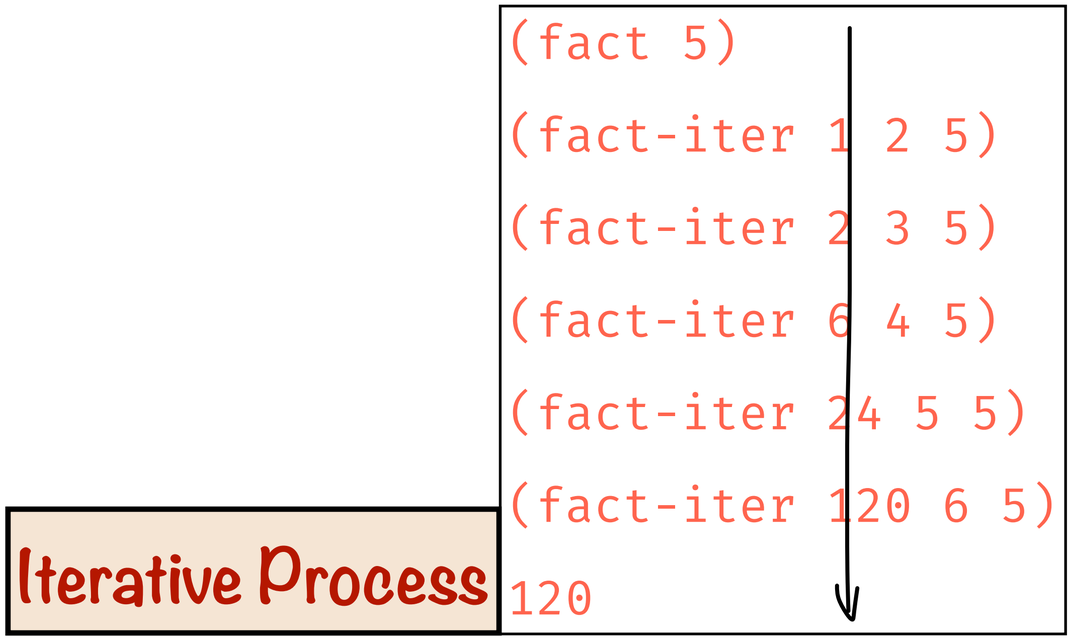
```
(define (fact n)
  (define (fact-iter prod ctr n)
    (if (> ctr n)
        prod
        (fact-iter (* ctr prod)
                   (+ ctr 1)
                   n)))
  (fact-iter 1 1 n))
```

The generated process for `fact(5)`

```
(define (fact n)
  (define (fact-iter prod ctr n)
    (if (> ctr n)
        prod
        (fact-iter (* ctr prod)
                   (+ ctr 1)
                   n)))
  (fact-iter 1 1 n))
```

Time: O(n)

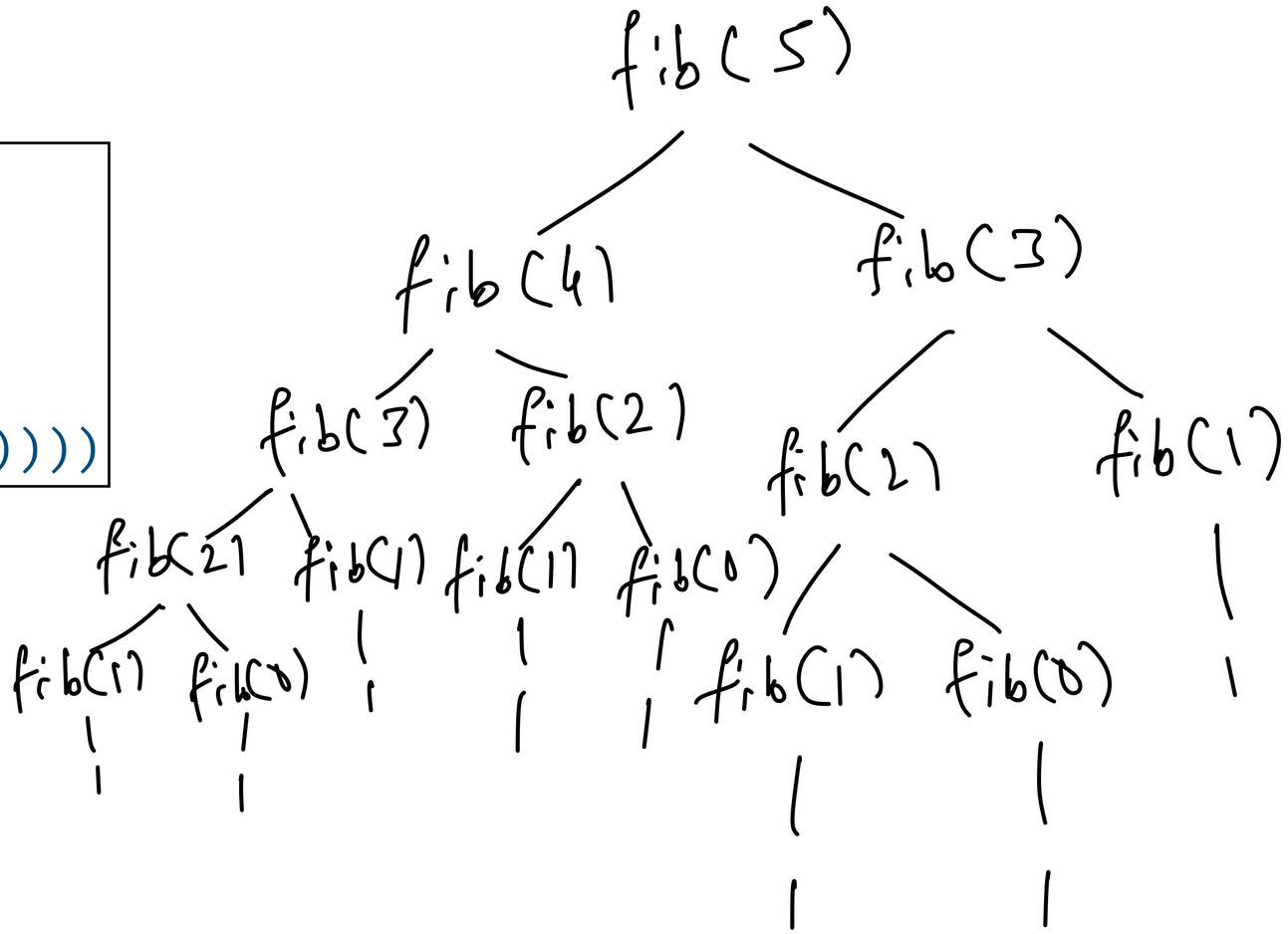
Space: O(1)



Another recursive process

- Fibonacci numbers:

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1))
                 (fib (- n 2))))))
```



Tree-Recursive Process

Recursive vs Iterative Processes

```
(fact 5)
(* 5 (fact 4))
(* 5 (* 4 (fact 3)))
(* 5 (* 4 (* 3 (fact 2))))
(* 5 (* 4 (* 3 (* 2 (fact 1)))))
(* 5 (* 4 (* 3 (* 2 1))))
(* 5 (* 4 (* 3 2)))
(* 5 (* 4 6))
(* 5 24)
120
```

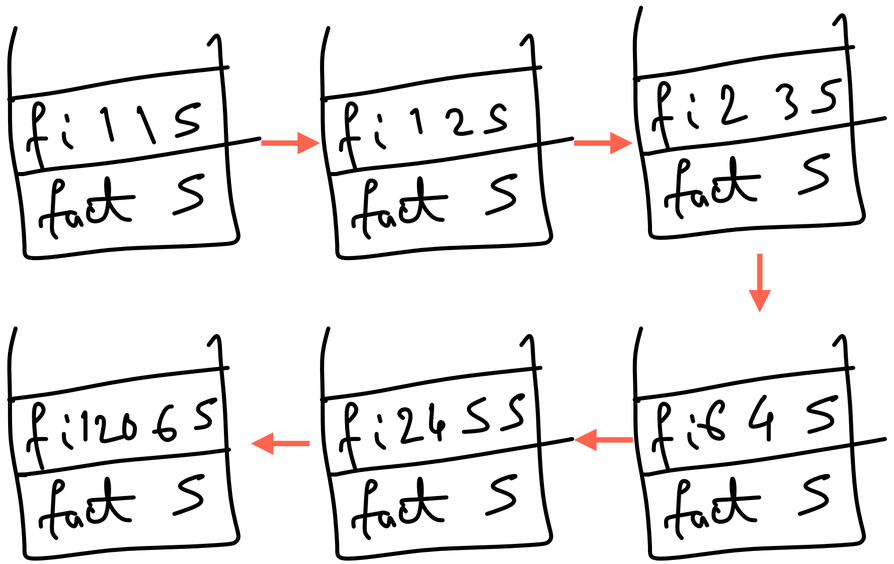
- Recursive: Grow then shrink.
- Recursive: Require more space.
- Iterative: State variables.
- Iterative: Can be resumed easily.
- Recursive: More *bureaucratic*.
- But even an iterative process generated by a recursive procedure requires more space!

```
(fact 5)
(fact-iter 1 2 5)
(fact-iter 2 3 5)
(fact-iter 6 4 5)
(fact-iter 24 5 5)
(fact-iter 120 6 5)
120
```

Tail-Call Optimization

- Iteration without looping constructs is expensive in space.
- But we can avoid returning when the recursive call is the tail!
- Saves stack space and makes iteration (nearly) as efficient as imperative languages with looping constructs.

```
(define (fact n)
  (define (fact-iter prod ctr n)
    (if (> ctr n)
        prod
        (fact-iter (* ctr prod)
                   (+ ctr 1)
                   n)))
  (fact-iter 1 1 n))
```



Lab Modus Operandi

- Each lab has to be done individually.
- Only lab desktops. Fixed seat. No mobile phones.
- TAs would clarify your doubts and evaluate by seeing your code as well as asking questions. Their judgment would be final. We would rotate TAs.
- You can skip one lab; more than that would cause loss of marks
- Maintain a silent atmosphere in the lab.
- DO NOT CHEAT.
- Your day would start by installing the *sicp* package in DrRacket.
- Batch 1A (23B0901–23B0996) today!