Recursive Thinking

By

Dr. Nadia M. Mohammed

Recursive Thinking

- ***Recursion*** is:
  - A *problem-solving* ***approach***, that can ...
  - Generate *simple solutions* to ...
  - *Certain kinds* of problems that ...
  - Would be *difficult to solve in other ways*

Recursive Thinking: An Example

- Recursion splits a problem Into one or more simpler versions of itself

Strategy for processing nested dolls:

1. if there is only one doll
2. do what it needed for it

   else

3. do what is needed for the outer doll
4. Process the inner nest in the same way

FIGURE 7.1
A Set of Nested Wooden Figures

Recursive Thinking: Another Example

- **Factorial :** A classic example of a recursive procedure is the function used to calculate the factorial of a natural number:

This factorial function can also be described without using recursion by making use of the typical looping constructs found in imperative programming languages:

Pseudocode (iterative):

```
function factorial is:

input: integer n such that n >= 0

output: [n × (n-1) × (n-2) × … × 1]

    1. create new variable called running_total with a value of 1

    2. begin loop
          1. if n is 0, exit loop
          2. set running_total to (running_total × n)
          3. decrement n
          4. repeat loop

    3. return running_total

end factorial
```

Recursive Thinking: Another Example

- **Factorial :** A classic example of a recursive procedure is the function used to calculate the factorial of a natural number:

$$\operatorname{fact}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \operatorname{fact}(n-1) & \text{if } n > 0 \end{cases}$$

Pseudocode (recursive):

```
function factorial is:

input: integer n such that n >= 0

output: [n × (n-1) × (n-2) × … × 1]


    1. if n is 0, return 1
    2. otherwise, return [ n × factorial(n-1) ]


end factorial
```

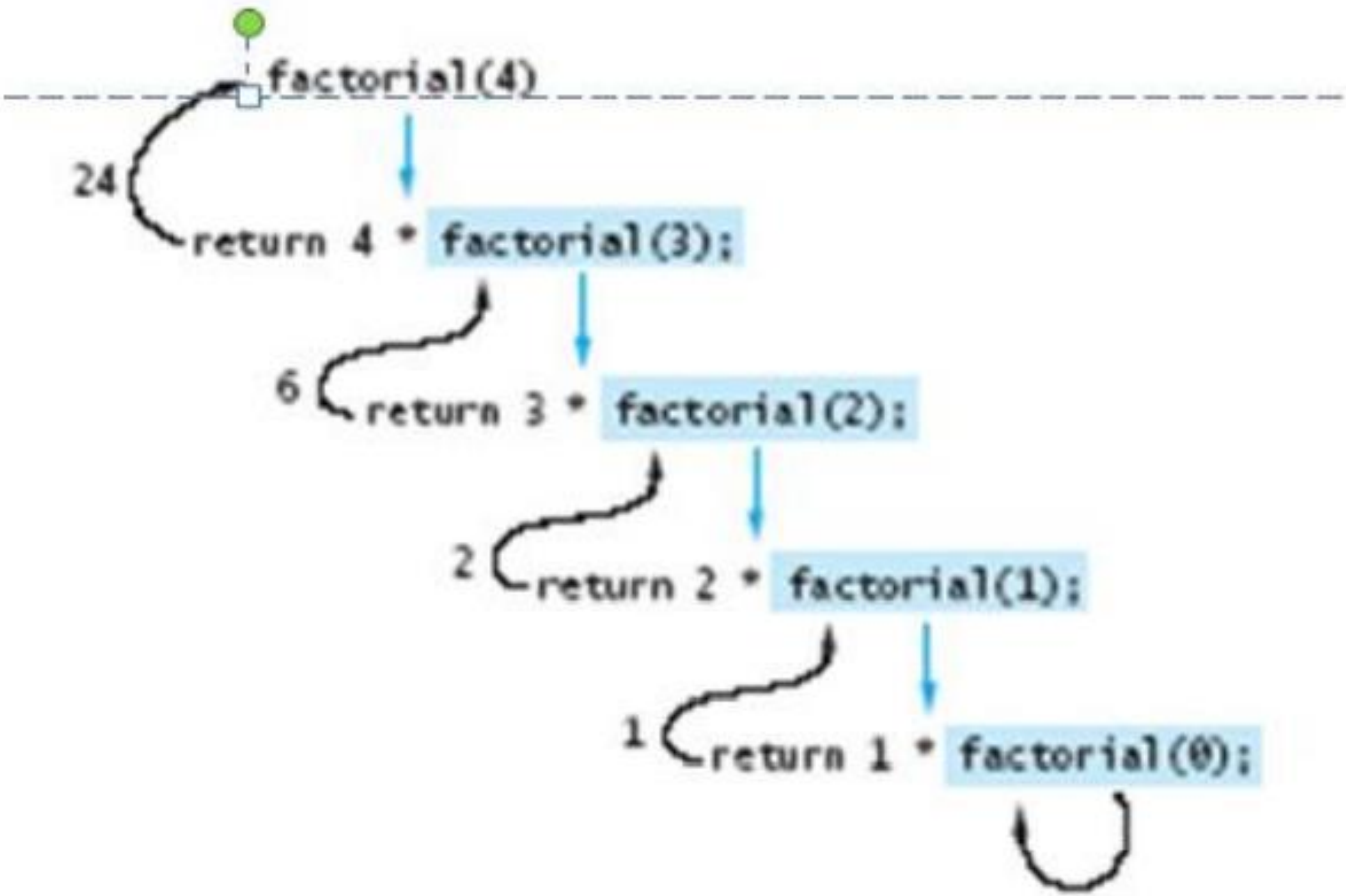
The function can also be written as a recurrence relation:

$$b_n = nb_{n-1}$$
$$b_0 = 1$$

This evaluation of the recurrence relation demonstrates the computation that would be performed in evaluating the pseudocode above:

Computing the recurrence relation for n = 4:

$$
\begin{aligned}
b_4 &= 4 * b_3 \\
&= 4 * (3 * b_2) \\
&= 4 * (3 * (2 * b_1)) \\
&= 4 * (3 * (2 * (1 * b_0))) \\
&= 4 * (3 * (2 * (1 * 1))) \\
&= 4 * (3 * (2 * 1)) \\
&= 4 * (3 * 2) \\
&= 4 * 6 \\
&= 24
\end{aligned}
$$


factorial(4)
24
return 4 * factorial(3);
6
return 3 * factorial(2);
2
return 2 * factorial(1);
1
return 1 * factorial(0);

- HW:

- 1- solve by using **Recursive the** Fibonacci sequenc
- 2- solve by using **Recursive the greatest common divisor (gcd)**
- 3- solve by using **Recursive the** Hanoi Tower