Recursion

Logic Algorithm

Tri Hadiah Muliawati, Yunia Ikawati

Politeknik Elektronika Negeri Surabaya
Department of Informatics and Computer Engineering,

Subject:

1. Overview Recursion
2. Flowchart and pseudocode

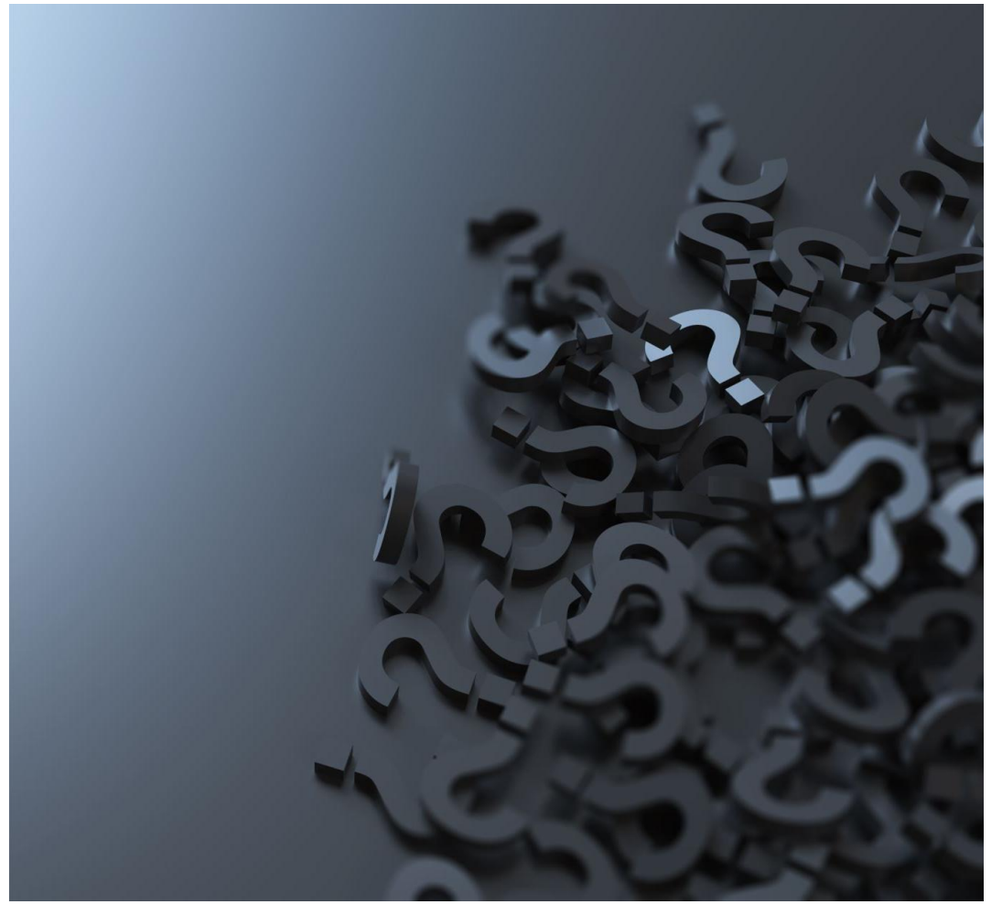
Objectives:

1. Students able to understand the underlying concept of Recursion
2. Students able to use flowchart and pseudocode to implement Recursion

Overview

Recursion

- In computer science, recursion is a way of thinking about and solving problems
- It’s actually one of the central ideas of CS
- In recursion, the solution depends on solutions to smaller instances of the same problem
- Recursive function is a special type of function which calls itself and repeats its behavior until some condition is met to return a result.



Recursive solutions

- When creating a recursive solution, there are a few things we want to keep in mind:
  - We need to break the problem into smaller pieces of itself
  - We need to define a “base case” to stop recursion
  - The smaller problems we break down into need to eventually reach the base case

Example

```
def compute(Input):
    if Input <= 2:
                return Input
    else:
        return Input + compute(Input-1)

def main():
    print(compute(10))

main()
```

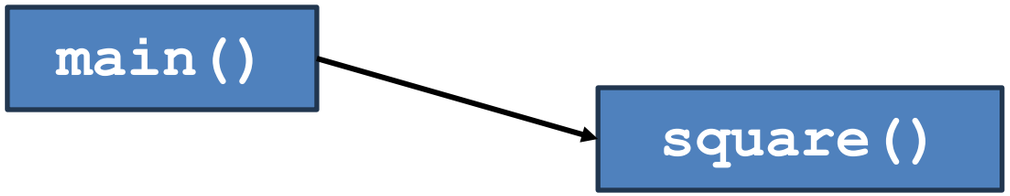
This is called a base case or base condition. If this condition is met, the recursion will stop.

You can see that the `compute()` function calls itself.

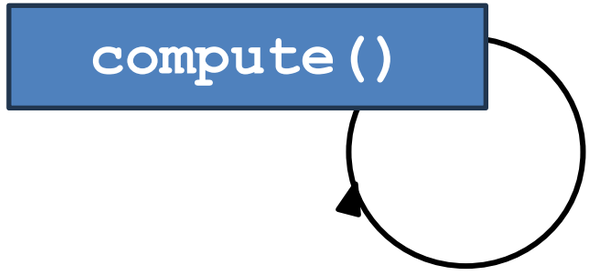
This is where the recursion occurs.

Normal vs Recursive Functions

- So far, we've had functions call other functions
- For example, `main()` function calls the `square()` function



- A recursive function, however, `compute()` function calls itself

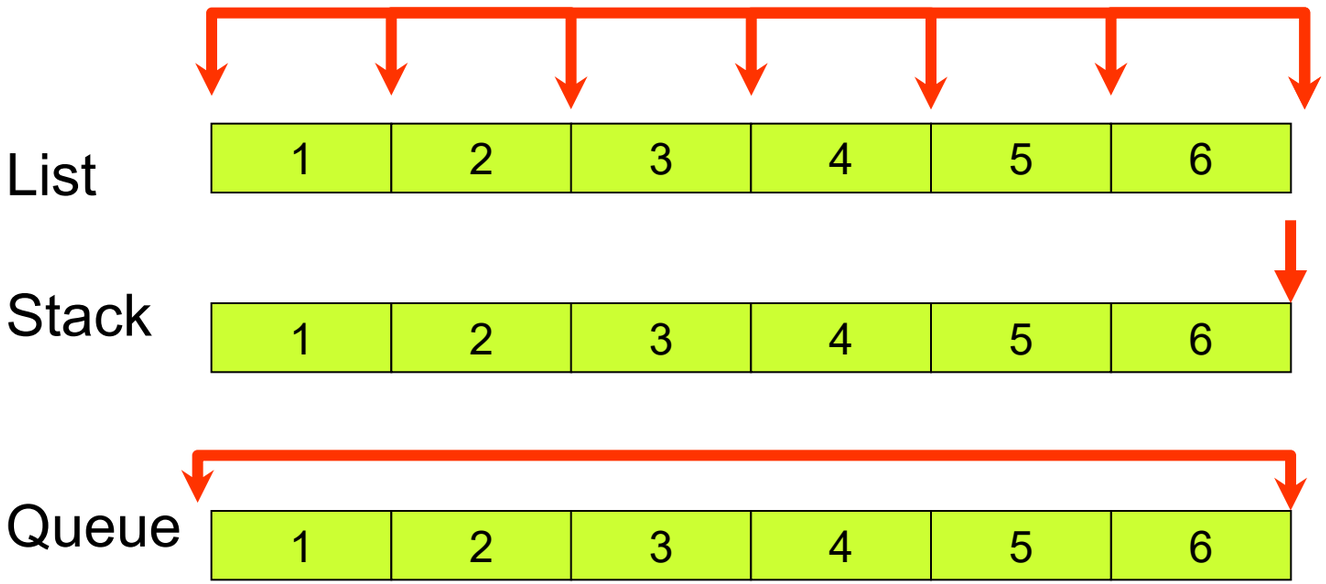


Visualizing Recursion

- To help visualize recursion, we will use a common concept called the Stack and Queue
- Stack – a container that allows push (to put item into stack) and pop (to obtain and remove item from stack)
- Queue – a container that allows enqueue (to put item into queue) and dequeue (to obtain and remove item from queue)

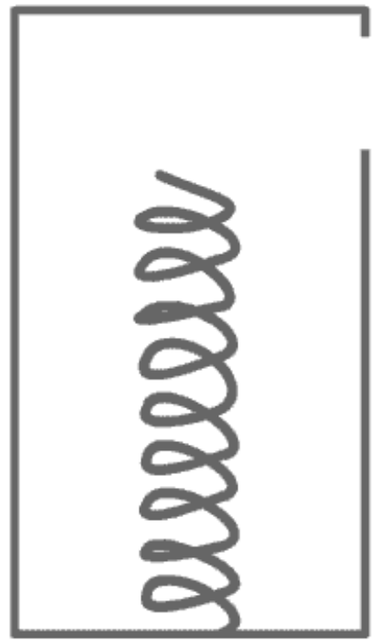
List, Stack and Queue

- In a List, insertion and deletion of elements can be carried out at any position
- However, adding and deleting elements to the stack/queue is done in the leading position or the backward position.



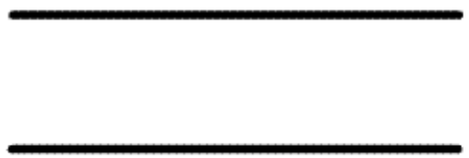
Stack

- Adding and deleting elements is done on the list element located at the front
- Removed element is the most recently added element
- Another name： LIFO (Last In First Out)
- PUSH operation: Adds an element to a stack
- POP operation: Removes an element from a stack



Queue

- Adding data is done at one end of a queue, while deleting data is done at the other end
- Removed data is the earliest data added
- Another name： FIFO (First In First Out)
- ENQUEUE operation: Adds data to a queue
- DEQUEUE operation: Removes data from a queue



Flowchart and Pseudocode

Politeknik Elektronika Negeri Surabaya
Department of Informatics and Computer Engineering,

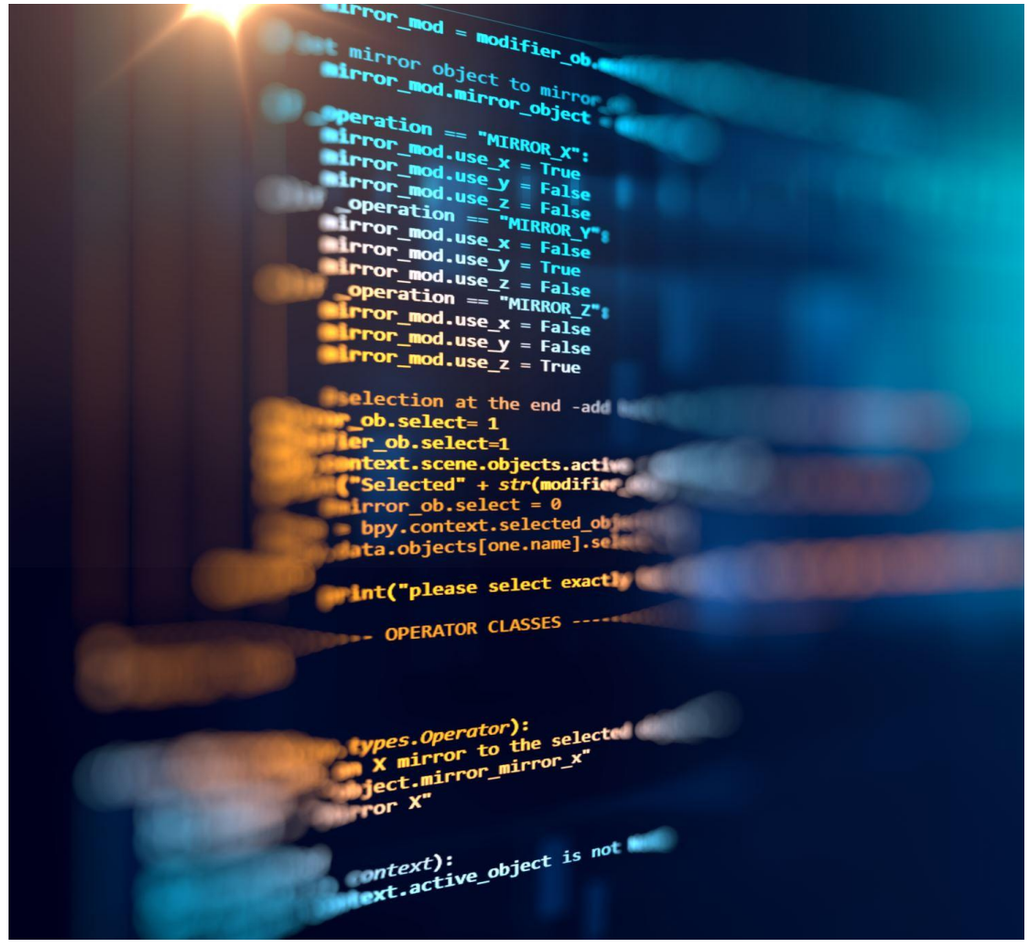
Let’s Try

Create program to calculate factorial of a certain number

e.g 4! = 4 × 3 × 2 × 1 = 24

What is the input, process, and output?

Note: We can use either recursive or non-recursive approach to solve this problem.



Answer (Non-recursive approach)

Create program to calculate factorial of a certain number

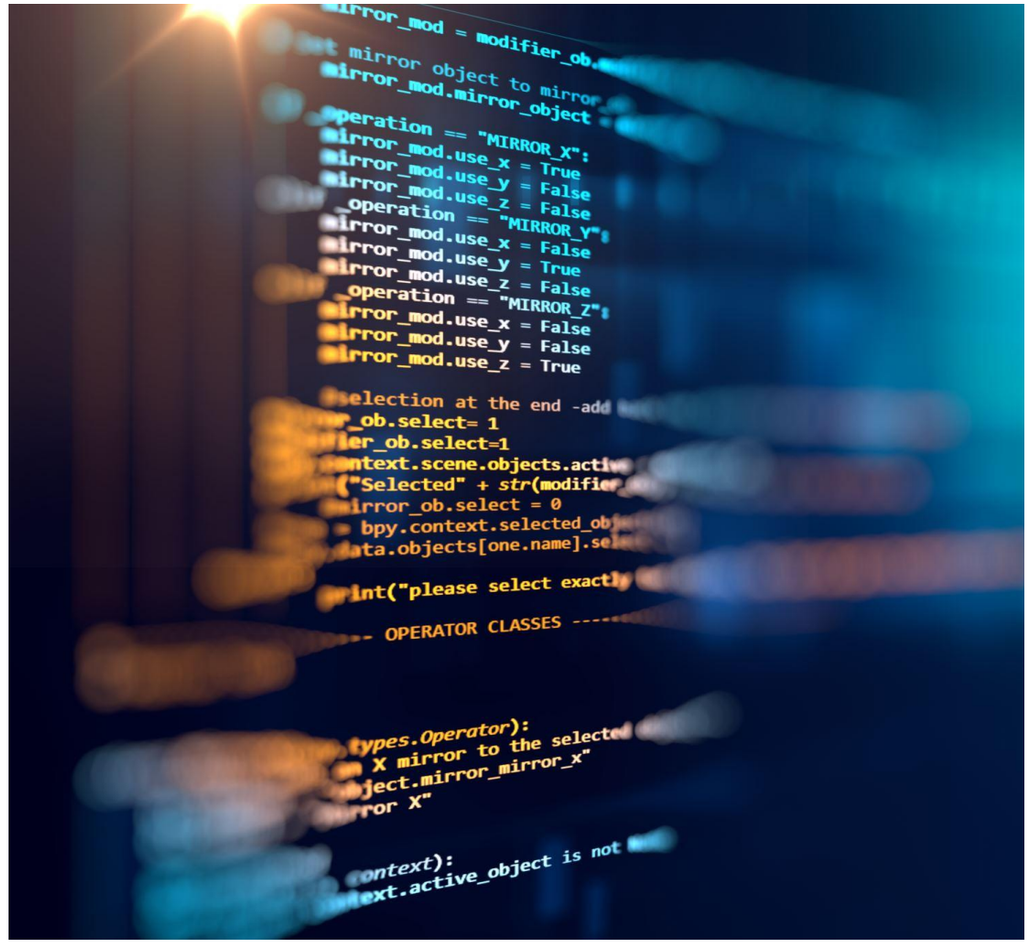
e.g 4! = 4 × 3 × 2 × 1 = 24

What is the input, process, and output?

Input:
Input number (`n`), set `i` =1, set `fact` = 1

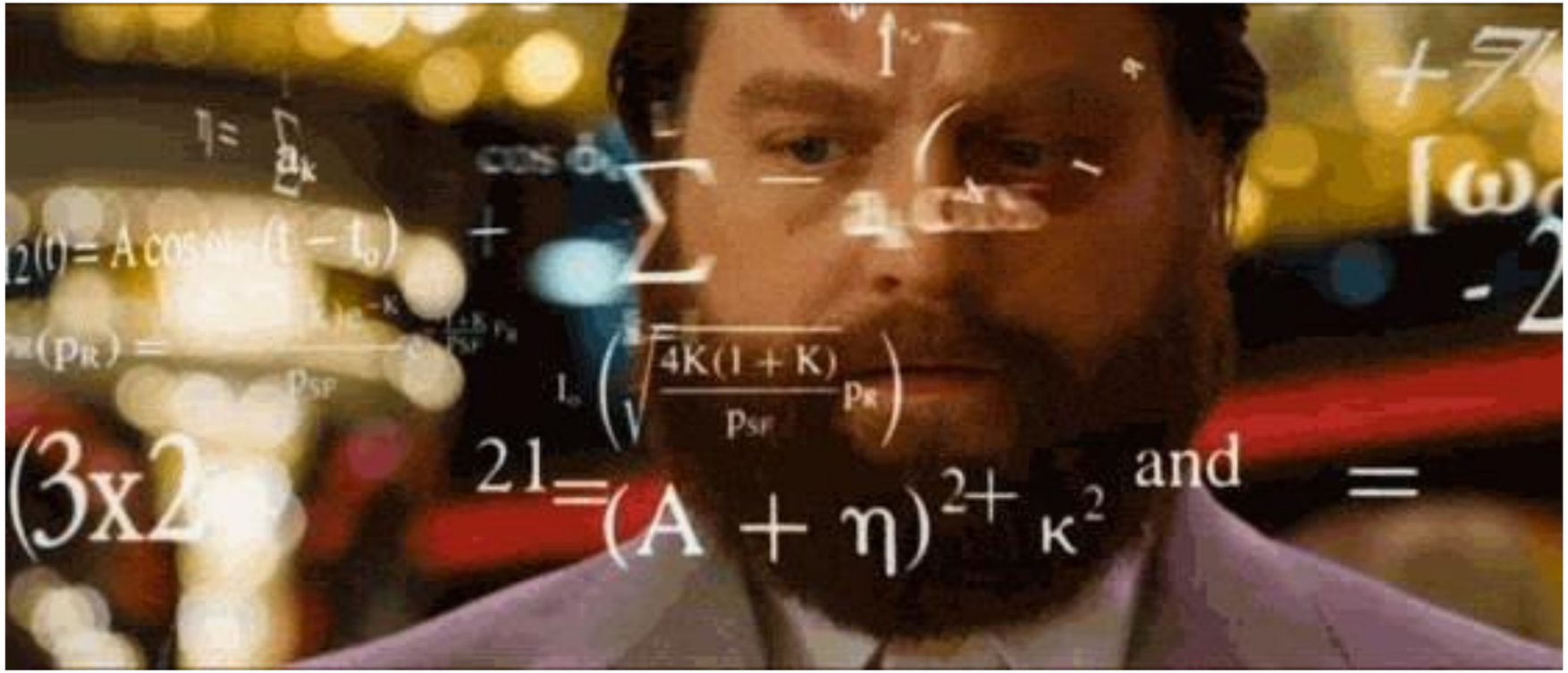
Process:
for `i` = 1 until `n`, calculate `fact` = `fact * i`

Output:
Display `fact`



Let’s Try

Pseudocode ? Flowchart ?

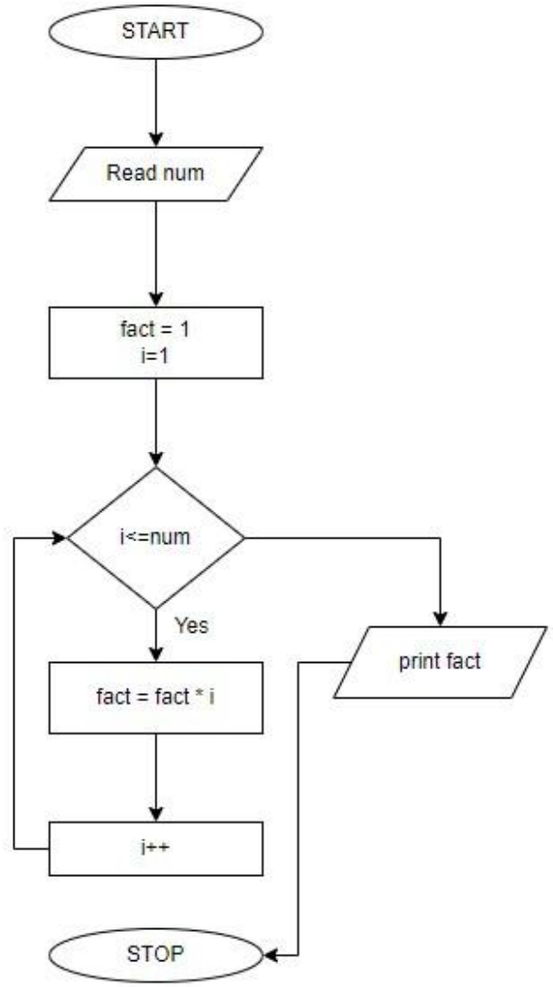


Answer

Pseudocode

```
1. Start
2. INPUT n
3. SET i = 1, fact = 1
4. if i <= n otherwise STEP 6
   Calculate fact = fact * I
   i=i+1 back to step 3
5. DISPLAY fact
6. STOP
```

Flowchart



Answer (Recursive approach)

Create program to calculate factorial of a certain number

1. Break the problem into smaller pieces and look for recurring patterns

`factorial(1)`	1	`1`	1
`factorial(2)`	2 * 1	`2 * factorial(1)`	2
`factorial(3)`	3 * 2 * 1	`3 * factorial(2)`	6
`factorial(4)`	4 * 3 * 2 * 1	`4 * factorial(3)`	24

Some of the result of function call (number(s) inside the square box) is replaceable with the result of previous function call

Answer (Recursive approach)

Create program to calculate factorial of a certain number

1. Break the problem into smaller pieces and look for recurring patterns

`factorial(1)`	1	`factorial(1)`	1
`factorial(2)`	2*1	`2 * factorial(1)`	2
`factorial(3)`	3*2*1	`3 * factorial(2)`	6
`factorial(4)`	4*3*2*1	`4 * factorial(3)`	24
`factorial(n)`	n*(n-1)*(n-2)*...*1	`n * factorial(n-1)`	

Answer (Recursive approach)

Create program to calculate factorial of a certain number

2. Define the base case / base condition

`factorial(1)`	1	`1`	1
`factorial(2)`	2 * 1	`2 * factorial(1)`	2
`factorial(3)`	3 * 2 * 1	`3 * factorial(2)`	6
`factorial(4)`	4 * 3 * 2 * 1	`4 * factorial(3)`	24

The result of function call is not replaceable with the result of previous function call when input is equal to 1

Answer (Recursive approach)

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

```
The factorial of 3 is 6
```

Base case

Recursive case

Answer (Recursive approach)

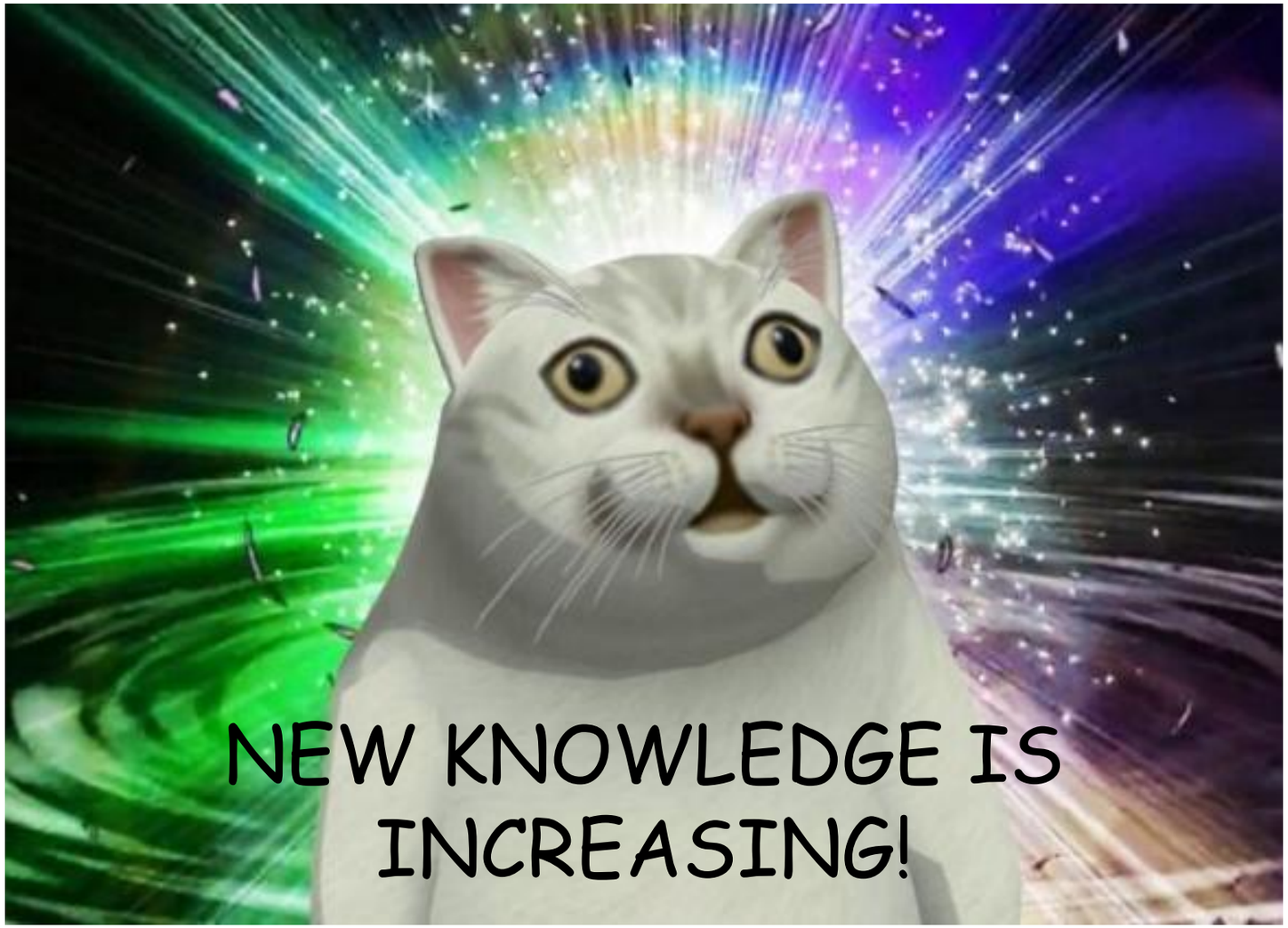
```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

```
factorial(3)          # 1st call with 3
3 * factorial(2)      # 2nd call with 2
3 * 2 * factorial(1)  # 3rd call with 1
3 * 2 * 1             # return from 3rd call as number=1
3 * 2                 # return from 2nd call
6                     # return from 1st call
```

Summary

- Recursion is a way of thinking about and solving problems in which a function calls itself.
- Base case is required for a recursive function to stop calling itself. Without a base case, recursive function will run in an endless loop.


NEW KNOWLEDGE IS
INCREASING!

Reference

1. Karl, Beecher. "Computational Thinking: A Beginner's Guide to Problem-Solving and Programming." Swindon, UK: BCS, The Chartered Institute for IT (2017).
2. CSE 1300 - Introduction to Computing Principles, Kennesaw State University
3. Computer Science (CS) and Computational Thinking (CT) : Pattern Recognition in the AYA curriculum, Miami University

Latihan

Buatlah study kasus sederhana tentang recursive dan jelaskan program dan outputnya?

THANK YOU!