BÀI 1

def sum\_of\_numbers(n):

if n == 1:

return 1

else:

return n + sum\_of\_numbers(n - 1)

print(sum\_of\_numbers(7))

Quy trình các bước cho `sum\_of\_numbers(7)`:

\*\*Bước 1: Gọi hàm `sum\_of\_numbers(7)`\*\*

- Kiểm tra điều kiện cơ sở: \( n = 7 \), không thỏa mãn điều kiện cơ sở.

- Thực hiện câu lệnh trong `else`: `return 7 + sum\_of\_numbers(6)`.

\*\*Bước 2: Hàm `sum\_of\_numbers(6)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 6 \), không thỏa mãn điều kiện cơ sở.

- Thực hiện câu lệnh trong `else`: `return 6 + sum\_of\_numbers(5)`.

\*\*Bước 3: Hàm `sum\_of\_numbers(5)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 5 \), không thỏa mãn điều kiện cơ sở.

- Thực hiện câu lệnh trong `else`: `return 5 + sum\_of\_numbers(4)`.

\*\*Bước 4: Hàm `sum\_of\_numbers(4)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 4 \), không thỏa mãn điều kiện cơ sở.

- Thực hiện câu lệnh trong `else`: `return 4 + sum\_of\_numbers(3)`.

\*\*Bước 5: Hàm `sum\_of\_numbers(3)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 3 \), không thỏa mãn điều kiện cơ sở.

- Thực hiện câu lệnh trong `else`: `return 3 + sum\_of\_numbers(2)`.

\*\*Bước 6: Hàm `sum\_of\_numbers(2)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 2 \), không thỏa mãn điều kiện cơ sở.

- Thực hiện câu lệnh trong `else`: `return 2 + sum\_of\_numbers(1)`.

\*\*Bước 7: Hàm `sum\_of\_numbers(1)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 1 \), điều kiện cơ sở được thỏa mãn.

- Hàm trả về `1`.

Các hàm được thực hiện lần lượt theo thứ tự từ dưới lên trên (từ bước 7 lên bước 1):

- `sum\_of\_numbers(1)` trả về `1`

- `sum\_of\_numbers(2)` trả về `2 + 1 = 3`

- `sum\_of\_numbers(3)` trả về `3 + 3 = 6`

- `sum\_of\_numbers(4)` trả về `4 + 6 = 10`

- `sum\_of\_numbers(5)` trả về `5 + 10 = 15`

- `sum\_of\_numbers(6)` trả về `6 + 15 = 21`

- `sum\_of\_numbers(7)` trả về `7 + 21 = 28`

Vì vậy, kết quả cuối cùng là `28`, đây là tổng các số từ 1 đến 7.

Bài 2

def fibonacci(n):

if n <= 1:

return n

else:

return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(8))

```### Quy trình các bước cho `fibonacci(8)`:

\*\*Bước 1: Gọi hàm `fibonacci(8)`\*\*

- Kiểm tra điều kiện cơ sở: \( n = 8 \), không thỏa mãn điều kiện cơ sở

- Thực hiện câu lệnh trong `else`: `return fibonacci(7) + fibonacci(6)`.

\*\*Bước 2: Hàm `fibonacci(7)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 7 \), không thỏa mãn điều kiện cơ

- Thực hiện câu lệnh trong `else`: `return fibonacci(6) + fibonacci(5)`.

\*\*Bước 3: Hàm `fibonacci(6)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 6 \), không thỏa mãn điều kiện cơ sở

- Thực hiện câu lệnh trong `else`: `return fibonacci(5) + fibonacci(4)`.

\*\*Bước 4: Hàm `fibonacci(5)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 5 \), không thỏa mãn điều kiện cơ sở

- Thực hiện câu lệnh trong `else`: `return fibonacci(4) + fibonacci(3)`.

\*\*Bước 5: Hàm `fibonacci(4)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 4 \), không thỏa mãn điều kiện cơ sở

- Thực hiện câu lệnh trong `else`: `return fibonacci(3) + fibonacci(2)`.

\*\*Bước 6: Hàm `fibonacci(3)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 3 \), không thỏa mãn điều kiện cơ sở

- Thực hiện câu lệnh trong `else`: `return fibonacci(2) + fibonacci(1)`.

\*\*Bước 7: Hàm `fibonacci(2)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 2 \), không thỏa mãn điều kiện cơ sở

- Thực hiện câu lệnh trong `else`: `return fibonacci(1) + fibonacci(0)`.

\*\*Bước 8: Hàm `fibonacci(1)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 1 \), thỏa mãn điều kiện cơ sở

- Hàm trả về `1`.

\*\*Bước 9: Hàm `fibonacci(0)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 0 \), thỏa mãn điều kiện cơ sở

- Hàm trả về `0`.

Các hàm được thực hiện lần lượt theo thứ tự từ dưới lên trên (từ bước 9 lên bước 1):

- `fibonacci(0)` trả về `0`.

- `fibonacci(1)` trả về `1`.

- `fibonacci(2)` trở thành `1 + 0 = 1`.

- `fibonacci(3)` trở thành `1 + 1 = 2`.

- `fibonacci(4)` trở thành `2 + 1 = 3`.

- `fibonacci(5)` trở thành `3 + 2 = 5`.

- `fibonacci(6)` trở thành `5 + 3 = 8`.

- `fibonacci(7)` trở thành `8 + 5 = 13`.

- `fibonacci(8)` trở thành `13 + 8 = 21`.

Vì vậy, kết quả cuối cùng khi gọi `fibonacci(8)` là `21`, đây là số Fibonacci thứ 8.

BÀI 3

def power(x, n):

if n == 0:

return 1

else:

return x \* power(x, n - 1)

print(power(2, 6))

```

### Quy trình các bước cho `power(2, 6)`:

\*\*Bước 1: Gọi hàm `power(2, 6)`\*\*

- Kiểm tra điều kiện cơ sở: \( n = 6 \), không thỏa mãn điều kiện cơ sở (\( n = 0 \)).

- Thực hiện câu lệnh trong `else`: `return 2 \* power(2, 5)`.

\*\*Bước 2: Hàm `power(2, 5)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 5 \), không thỏa mãn điều kiện cơ sở (\( n = 0 \)).

- Thực hiện câu lệnh trong `else`: `return 2 \* power(2, 4)`.

\*\*Bước 3: Hàm `power(2, 4)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 4 \), không thỏa mãn điều kiện cơ sở (\( n = 0 \)).

- Thực hiện câu lệnh trong `else`: `return 2 \* power(2, 3)`.

\*\*Bước 4: Hàm `power(2, 3)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 3 \), không thỏa mãn điều kiện cơ sở (\( n = 0 \)).

- Thực hiện câu lệnh trong `else`: `return 2 \* power(2, 2)`.

\*\*Bước 5: Hàm `power(2, 2)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 2 \), không thỏa mãn điều kiện cơ sở (\( n = 0 \)).

- Thực hiện câu lệnh trong `else`: `return 2 \* power(2, 1)`.

\*\*Bước 6: Hàm `power(2, 1)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 1 \), không thỏa mãn điều kiện cơ sở (\( n = 0 \)).

- Thực hiện câu lệnh trong `else`: `return 2 \* power(2, 0)`.

\*\*Bước 7: Hàm `power(2, 0)` được gọi\*\*

- Kiểm tra điều kiện cơ sở: \( n = 0 \), điều kiện cơ sở được thỏa mãn.

- Hàm trả về `1`.

Các hàm được thực hiện lần lượt theo thứ tự từ dưới lên trên (từ bước 7 lên bước 1):

- `power(2, 0)` trả về `1`.

- `power(2, 1)` trở thành `2 \* 1 = 2`.

- `power(2, 2)` trở thành `2 \* 2 = 4`.

- `power(2, 3)` trở thành `2 \* 4 = 8`.

- `power(2, 4)` trở thành `2 \* 8 = 16`.

- `power(2, 5)` trở thành `2 \* 16 = 32`.

- `power(2, 6)` trở thành `2 \* 32 = 64`.

Vì vậy, kết quả cuối cùng khi gọi `power(2, 6)` là `64`, đây là giá trị của \(2^6\).

Bài 4

def thap\_ha\_noi(n, A, C, B):

if n == 1:

print(f"Chuyển đĩa 1 từ cột {A} sang cột {B}")

else:

thap\_ha\_noi(n - 1, A, B, C)

print(f"Chuyển đĩa {n} từ cột {A} sang cột {B}")

thap\_ha\_noi(n - 1, C, A, B)

\*\*Bước 1: Gọi hàm `thap\_ha\_noi(4, "A", "C", "B")`\*\*

- Kiểm tra điều kiện cơ sở: \( n = 4 \), không thỏa mãn điều kiện cơ sở (\( n \neq 1 \)).

- Thực hiện câu lệnh trong `else`.

\*\*Bước 2: Gọi hàm `thap\_ha\_noi(3, "A", "B", "C")`\*\*

- Bước này thực hiện bởi câu lệnh `thap\_ha\_noi(n - 1, A, B, C)` trong phần `else` của hàm.

- Gọi lại hàm với \( n = 3 \) và cột trung gian là "B" thay vì "C".

- Điều này sẽ lần lượt chuyển các đĩa từ cọc "A" sang "B", sử dụng "C" làm cọc trung gian.

\*\*Bước 3: Gọi hàm `thap\_ha\_noi(2, "A", "C", "B")`\*\*

- Tương tự như trên, giờ là với \( n = 2 \) và cọc trung gian là "C".

- Lần này sẽ chuyển các đĩa từ cọc "A" sang "C", sử dụng "B" làm cọc trung gian.

\*\*Bước 4: Gọi hàm `thap\_ha\_noi(1, "A", "B", "C")`\*\*

- Khi \( n = 1 \), điều kiện cơ sở được thỏa mãn và câu lệnh trong phần `if` được thực thi.

- In ra thông điệp chuyển đĩa 1 từ cọc "A" sang "B".

\*\*Bước 5: Trở lại bước 3:\*\*

- Sau khi xử lý xong cho \( n = 1 \), quay trở lại hàm gọi trước đó với \( n = 2 \).

- Tiếp tục quy trình này cho tới khi tất cả các đĩa được chuyển từ cọc "A" sang "B", thông qua "C" là cọc trung gian.

\*\*Bước 6: Kết thúc:\*\*

- Khi tất cả các đĩa đã được chuyển từ cọc "A" sang "B", quá trình kết thúc và không có cuộc gọi hàm nào tiếp theo.

- Chuỗi các lệnh in đã hoàn thành việc hiển thị quá trình chuyển đĩa từ "A" sang "B" theo quy tắc của bài toán tháp Hà Nội.

Vậy, khi gọi `thap\_ha\_noi(4, "A", "C", "B")`, bạn sẽ nhận được kết quả là cách chuyển tất cả 4 đĩa từ cọc "A" sang "B" theo quy tắc của bài toán tháp Hà Nội.