Exercise 4.1

As an exercise, draw a stack diagram for print_n called with s = 'Hello' and n=2. Then write a function called do_n that takes a function object and a number, n, as arguments, and that calls the given function n times.

In [1]:
```python
# Answer for Question 1

# Part 1: Stack diagram explanation (see explanation below)

# Part 2: do_n function implementation
def do_n(func, n):
    if n <= 0:
        return
    func()
    do_n(func, n - 1)

# Example usage
def say_hello():
    print("Hello")

do_n(say_hello, 3)
```

```
Hello
Hello
Hello
```

Exercise 4.2.

What is the output of the following program?

In [2]:
```python
def recurse(n, s):
    if n == 0:
        print(s)
    else:
        recurse(n-1, n+s)

recurse(3, 0)
```

```
6
```

Question 2

Draw a stack diagram that shows the state of the program when it prints the result. Draw a stack diagram that shows the state of the program when it prints the result.

#> Answer for Question 2

Stack diagram (top to bottom):

Frame	n	s
recurse	0	6

Frame	n	s
recurse	1	5
recurse	2	3
recurse	3	0
main		

Question 3

What would happen if you called this function like this: recurse(-1, 0)?

In [4]:
```
# Answer for Question 3
recurse(-1,0)
#If you call recurse(-1, 0), it will result in infinite recursion and eve
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
Cell In[4], line 2
      1 # Answer for Question 3
----> 2 recurse(-1,0)
      3 #If you call recurse(-1, 0), it will result in infinite recursion and eventually cause a RuntimeError due to maximum recursion depth being exceeded.

Cell In[2], line 5, in recurse(n, s)
      3     print(s)
      4 else:
----> 5     recurse(n-1, n+s)

Cell In[2], line 5, in recurse(n, s)
      3     print(s)
      4 else:
----> 5     recurse(n-1, n+s)

    [... skipping similar frames: recurse at line 5 (2974 times)]

Cell In[2], line 5, in recurse(n, s)
      3     print(s)
      4 else:
----> 5     recurse(n-1, n+s)

RecursionError: maximum recursion depth exceeded
```

Question 4

Write a docstring that explains everything someone would need to know in order to use this function (and nothing else).

In [5]:
```
# Answer for Question 4

def recurse(n, s):
    """
```

```
    Recursively adds the values of n to s, decrementing n each time,
    until n reaches 0. Then prints the final sum.

    Parameters:
    n (int): The starting number and number of recursions. Must be non-ne
    s (int): The initial sum value.

    Returns:
    None. The function prints the final sum.

    Raises:
    RuntimeError: If n is negative, causing infinite recursion.

    Example:
    >>> recurse(3, 0)
    6
    """
    if n == 0:
        print(s)
    else:
        recurse(n-1, n+s)
```

Exercise 4.3

Use recursion to print the first n terms of the Fibonacci series.

In [7]:
```
# Answer for Exercise 4.3

def fibonacci(n, a=0, b=1, count=0):
    if count < n:
        print(a, end=' ')
        return fibonacci(n, b, a + b, count + 1)
    print()  # Print a newline at the end

# Example usage
n = int(input("Enter the number of Fibonacci terms to print: "))
fibonacci(n)
```

```
0 1 1 2 3 5 8 13 21 34
```

Exercise 4.4

Use recursion to evaluate the factorial of a number.

In [8]:
```
# Answer for Exercise 4.4

def factorial(n):
    # Base case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    # Recursive case: n! = n * (n-1)!
    else:
        return n * factorial(n - 1)

# Test the function
number = int(input("Enter a non-negative integer to calculate its factori
result = factorial(number)
print(f"The factorial of {number} is: {result}")
```

```
The factorial of 4 is: 24
```