Name - Bhavya Mahawar
batch - 80
sapid-590029516

1. Write a Python function to find the maximum and minimum numbers from a sequence of numbers. (Note: Do not use built-in functions.)

```
def find_min_max(numbers):
    if not numbers:
        return None, None # Handle empty sequence case

    min_val = numbers[0]
    max_val = numbers[0]

    for num in numbers:
        if num < min_val:
            min_val = num
        if num > max_val:
            max_val = num

    return max_val, min_val

# Example :
sequence1 = [3, 1, 4, 1, 5, 9, 2, 6]
max_num, min_num = find_min_max(sequence1)
print(f"Sequence: {sequence1}")
print(f"Maximum number: {max_num}")
print(f"Minimum number: {min_num}")

sequence2 = [-10, -5, 0, 5, 10]
max_num, min_num = find_min_max(sequence2)
print(f"\nSequence: {sequence2}")
print(f"Maximum number: {max_num}")
print(f"Minimum number: {min_num}")

sequence3 = [7]
max_num, min_num = find_min_max(sequence3)
print(f"\nSequence: {sequence3}")
print(f"Maximum number: {max_num}")
print(f"Minimum number: {min_num}")

sequence4 = []
max_num, min_num = find_min_max(sequence4)
print(f"\nSequence: {sequence4}")
print(f"Maximum number: {max_num}")
print(f"Minimum number: {min_num}")
```

2. Write a Python function that takes a positive integer and returns the sum of the cube of all the positive integers smaller than the specified number.

```
def sum_cubes_smaller(n):
    if not isinstance(n, int) or n <= 0:
        raise ValueError("Input must be a positive integer.")

    total_sum = 0
    for i in range(1, n):
        total_sum += i**3
    return total_sum

# Example :
print(f"Sum of cubes smaller than 4: {sum_cubes_smaller(4)}")
print(f"Sum of cubes smaller than 1: {sum_cubes_smaller(1)}")
print(f"Sum of cubes smaller than 5: {sum_cubes_smaller(5)}")
```

3. Write a Python function to print 1 to n using recursion. (Note: Do not use loop)

```
def print_1_to_n(n):
    if n <= 0:
        return
    else:
        print_1_to_n(n - 1)
        print(n)

# Example:
```

```
print("Printing 1 to 5:")
print_1_to_n(5)

print("\nPrinting 1 to 0 (should print nothing):")
print_1_to_n(0)

print("\nPrinting 1 to 3:")
print_1_to_n(3)
```

4. Write a recursive function to print Fibonacci series upto n terms.

```
def fibonacci_recursive(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        series = fibonacci_recursive(n - 1)
        series.append(series[-1] + series[-2])
        return series

def print_fibonacci_series(n):
    if n <= 0:
        print("Please enter a positive integer.")
    else:
        fib_series = fibonacci_recursive(n)
        print(f"Fibonacci series up to {n} terms: {fib_series}")

# Example :
print_fibonacci_series(7)
print_fibonacci_series(1)
print_fibonacci_series(2)
print_fibonacci_series(0)
```

5. Write a lambda function to find volume of cone.

```
import math

volume_of_cone = lambda r, h: (1/3) * math.pi * (r**2) * h

# Example :
radius1 = 3
height1 = 7
print(f"Volume of cone with radius {radius1} and height {height1}: {volume_of_cone(radius1, height1):.2f}")

radius2 = 5
height2 = 10
print(f"Volume of cone with radius {radius2} and height {height2}: {volume_of_cone(radius2, height2):.2f}")

radius3 = 2.5
height3 = 6.2
print(f"Volume of cone with radius {radius3} and height {height3}: {volume_of_cone(radius3, height3):.2f}")
```

6. Write a lambda function which gives tuple of max and min from a list. Sample input: [10, 6, 8, 90, 12, 56] Sample output: (90,6)

```
get_max_min_tuple = lambda input_list: (max(input_list), min(input_list))

sample_input = [10, 6, 8, 90, 12, 56]

output = get_max_min_tuple(sample_input)
print(f"Sample input: {sample_input}")
print(f"Sample output: {output}")

# EXAMPLE:
another_list = [5, 2, 9, 1, 7]
print(f"\nInput list: {another_list}")
print(f"Output (max, min): {get_max_min_tuple(another_list)}")
```

7. Write functions to explain mentioned concepts: a. Keyword argument

b. Default argument c. Variable length argument

```
# a
def greet_person(name, message):
    print(f"Hello, {name}! {message}")
greet_person(name="Alice", message="How are you?")
greet_person(message="Nice to meet you!", name="Bob") # Order does not matter with keyword arguments

print("\n--- Keyword Arguments Example --- ")
def create_user(username, email, age):
    print(f"User created: Username={username}, Email={email}, Age={age}")
create_user(username="john_doe", age=30, email="john@example.com")
```

```
# b
def greet_with_default(name, greeting="Hello"):
    print(f"{greeting}, {name}!")
greet_with_default("Charlie")

greet_with_default("David", "Hi")

print("\n--- Default Arguments Example ---")
def configure_settings(theme="dark", font_size=12, show_sidebar=True):
    print(f"Settings: Theme={theme}, Font Size={font_size}, Show Sidebar={show_sidebar}")
configure_settings()
configure_settings(theme="light", font_size=14)
configure_settings(show_sidebar=False, theme="blue")
```

```
# c
def sum_all_numbers(*args):
    total = 0
    for num in args:
        total += num
    print(f"Sum of {args}: {total}")
sum_all_numbers(1, 2, 3)
sum_all_numbers(10, 20, 30, 40, 50)
sum_all_numbers()

print("\n--- Variable Length Keyword Arguments (**kwargs) Example ---")
def display_info(**kwargs):
    print("Displaying Info:")
    for key, value in kwargs.items():
        print(f"  {key}: {value}")

display_info(name="Eve", age=25, city="New York")
display_info(product="Laptop", price=1200.50)
display_info()

print("\n--- Combined *args and **kwargs Example ---")
def process_data(action, *data_items, **options):
    print(f"Action: {action}")
    print(f"Data Items: {data_items}")
    print(f"Options: {options}")

process_data("report", 10, 20, 30, format="pdf", detailed=True)
process_data("log", "error", "warning")
```

8. Write a program to check whether all the values in a dictionary are same or not using lambda function.

```
all_values_same = lambda d: len(set(d.values())) <= 1

# Example dictionaries
dict1 = {'a': 10, 'b': 10, 'c': 10}
dict2 = {'x': 5, 'y': 10, 'z': 5}
dict3 = {'p': 'hello', 'q': 'hello'}
dict4 = {'m': True, 'n': False}
dict5 = {'single': 42}
dict6 = {}

# Examples
print(f"Dictionary: {dict1} -> All values same: {all_values_same(dict1)}")
print(f"Dictionary: {dict2} -> All values same: {all_values_same(dict2)}")
print(f"Dictionary: {dict3} -> All values same: {all_values_same(dict3)}")
print(f"Dictionary: {dict4} -> All values same: {all_values_same(dict4)}")
print(f"Dictionary: {dict5} -> All values same: {all_values_same(dict5)}")
print(f"Dictionary: {dict6} -> All values same: {all_values_same(dict6)}")
```

9. Write a program to create two lists and generate a dictionary with keys from list1 and values from list2.

```
# Define two lists
list1 = ['name', 'age', 'city']
list2 = ['Alice', 30, 'New York']
dictionary_from_lists = dict(zip(list1, list2))

print(f"List 1 (keys): {list1}")
print(f"List 2 (values): {list2}")
print(f"Generated Dictionary: {dictionary_from_lists}")

print("\n--- Example with lists of different lengths ---")
list_keys_short = ['a', 'b']
list_values_long = [1, 2, 3, 4]
dict_short_keys = dict(zip(list_keys_short, list_values_long))
print(f"List of keys (short): {list_keys_short}")
print(f"List of values (long): {list_values_long}")
print(f"Dictionary (short keys): {dict_short_keys}")

list_keys_long = ['x', 'y', 'z', 'w']
list_values_short = [10, 20]
dict_short_values = dict(zip(list_keys_long, list_values_short))
print(f"\nList of keys (long): {list_keys_long}")
print(f"List of values (short): {list_values_short}")
print(f"Dictionary (short values): {dict_short_values}")
```

Github repo link-https://github.com/mahawarbhavya/pythonexperiments.git