# CW3-2-1: Lambdas – Map & Filter – Generators

## 1. Lambdas

Write a Python function to sort a list of dictionaries using lambda.

Sample list:

```
students = [
    {"name": "John", "age": 20},
    {"name": "Alice", "age": 18},
    {"name": "Bob", "age": 22},
    {"name": "Emily", "age": 19}
]
```

Output:

```
[
    {"name": "Alice", "age": 18},
    {"name": "Emily", "age": 19},
    {"name": "John", "age": 20},
    {"name": "Bob", "age": 22}
]
```

## 2. Lambdas

Use a lambda function to sort a list of strings based on the number of vowels in each string, in descending order.

Sample list:

```
words = ["apple", "banana", "cherry", "date", "elderberry"]
```

Output:

```
["elderberry", "banana", "cherry", "apple", "date"]
```

3. **Map**

Write a Python function to convert a given list of tuples to a list of strings using map function.

Sample list:

```
tuples_list = [('Hello', 'World'), ('Open', 'AI'), ('GPT', '3')]
```

Output:

```
['Hello World', 'Open AI', 'GPT 3']
```

4. **Map & Filter**

Given a list of strings, use the map function to convert each string to uppercase. Then, use the filter function to remove all strings that contain the letter 'a'.

Sample list:

```
strings_list = ["apple", "banana", "Orange", "grape", "kiwi"]
```

Output:

```
['KIWI']
```

## 5. Generators

Define a generator function called **my_range** that produces a sequence of numbers based on the given start, stop, and step values. Check and validate input arguments using Built-in Exceptions.

1. Raise a **ValueError** if any of the parameters (**start**, **stop**, **step**) are not integers.
2. Raise a **ValueError** if the **step** parameter is zero.
3. Raise a **ValueError** if the **step** parameter prevents the sequence from reaching the **stop** value.

Expected Test Result:

```
result = list(my_range(1, 10, 2))  # Output: [1, 3, 5, 7, 9]

result = list(my_range(1.5, 10, 2))
# Raises ValueError: start parameter must be an integer

result = list(my_range(1, 10, 0))
# Raises ValueError: step parameter cannot be zero

result = list(my_range(10, 1, 2))
# Raises ValueError: invalid range, step prevents sequence from
# reaching stop value
```

## 6. Generators

Create a simple generator that generates Fibonacci numbers.

Expected Test Result:

```
fib_gen = fibonacci_generator(10)
for _ in fib_gen:
    print(_, end=' ')
# Output:   0 1 1 2 3 5 8 13 21 34
```

# CW3-2-2: Decorators – Recursion – Itertools

## 7. Decorators

Write a decorator called **do_before** that takes an argument called **func** Executes the **func** function before the wrapped function.

Expected Test Result:

```
def pre_func():
    print("Executing the pre-function...")

# Apply the decorator to a sample function
@do_before(pre_func)
def wrapped_function():
    print("Executing the wrapped function...")

# Call the wrapped function
wrapped_function()

# Output
# Executing the pre-function...
# Executing the wrapped function...
```

## 8. Decorators

Write a decorator called **do_mutli_times** that executes the desired function several times. Leave an argument for your decorator that specifies the number of iterations and it is 2 by default.

Expected Test Result:

```
@do_multi_times()
def print_message():
    print("Hello, World!")

print_message()

# Output:
# Hello, World!
# Hello, World!

@do_multi_times(iterations=3)
def print_message():
    print("Hello, World!")

print_message()

# Output
# Hello, World!
# Hello, World!
# Hello, World!
```

## 9. Recursion

Write a recursive function calculate the **nth** member of **Fibonacci** sequence.

Expected Test Result:

```
# Example usage:
n = 10
fib_number = fibonacci(n)
print(f"The {n}th number of the Fibonacci sequence is: {fib_number}")

# output
# The 10th number of the Fibonacci sequence is: 34
```

**10. Recursion**

Write a recursive function to calculate the factorial of a given positive integer.

Expected Test Result:

```
Input: 5
Output: 120

Input: 6
Output: 720
```

## 11. Itertools

Design a Python program that takes a string and generates all possible permutations of a given string using the **itertools** library. (Generate all possible forms)

Expected Test Result:

```
Enter a string: ABC
ABC
ACB
BAC
BCA
CAB
CBA
```

## 12. Itertools

Design a Python program that focuses on generating all possible combinations of a given list of elements using the **itertools** library.

Expected Test Result:

```
Enter: 123
('1',)
('2',)
('3',)
('1', '2')
('1', '3')
('2', '3')
('1', '2', '3')
```