Lambda , Maps Filter & Reduce

- Given a list let's see how to double each element of the given list. Using map()

a = [1, 2, 3, 4]

#Expected Output: [2, 4, 6, 8]

- Use filter() and lambda to extract all even numbers from a list of integers.

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

#Expected Output: [2, 4, 6, 8, 10]

- Use reduce() and lambda to find the longest word in a list of strings.

from functools import reduce

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

#Expected Output: 'banana'

- Use map() to square each number in the list and round the result to one decimal place.

my_floats = [4.35, 6.09, 3.25, 9.77, 2.16, 8.88, 4.59]

#Expected Output: [18.9, 37.1, 10.6, 95.5, 4.7, 78.9, 21.1]

- Use filter() to select names with 7 or fewer characters from the list.

my_names = ["olumide", "akinremi", "josiah", "temidayo", "omoseun"]

#Expected Output: ['olumide', 'josiah', 'omoseun']

- Use reduce() to calculate the sum of all numbers in a list. [1, 2, 3, 4, 5]

Solutions :

1) Double each element

a = [1, 2, 3, 4]

doubled = list(map(lambda x: x * 2, a))   # [2, 4, 6, 8]

2) Evens with filter

numbers = [1,2,3,4,5,6,7,8,9,10]

evens = list(filter(lambda x: x % 2 == 0, numbers)) # [2,4,6,8,10]

3) Longest word with reduce

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

longest = reduce(lambda acc, w: acc if len(acc) >= len(w) else w, words) # 'banana'

4) Square and round to 1 decimal

my_floats = [4.35, 6.09, 3.25, 9.77, 2.16, 8.88, 4.59]

squared_rounded = list(map(lambda x: round(x**2, 1), my_floats)) # [18.9, 37.1, 10.6, 95.5, 4.7, 78.9, 21.1]

5) Names with <= 7 chars

my_names = ["olumide", "akinremi", "josiah", "temidayo", "omoseun"]

short_names = list(filter(lambda n: len(n) <= 7, my_names)) # ['olumide', 'josiah', 'omoseun']

6) Sum with reduce

nums = [1,2,3,4,5]

total = reduce(lambda a,b: a+b, nums)

All And Any

1. Check if All Numbers are Positive. Given a list of integers, determine if all numbers are positive. Using all()

Input : numbers = [1, 2, 3, 4, 5]

#Expected Output : True

2. Check if Any Number is Even. Given a list of integers, check if any number is even. Using any()

Input: numbers = [1, 3, 5, 7, 8]

#Expected Output: True

3. Determine if any number in a list is divisible by 5 an print.

<u>Solutions :</u>

1. numbers1 = [1, 2, 3, 4, 5]

   all_positive = all(n > 0 for n in numbers1) # True

2. numbers2 = [1, 3, 5, 7, 8]

   any_even = any(n % 2 == 0 for n in numbers2) # True

3. numbers3 = [2, 11, 14, 25, 33]

   if any(n % 5 == 0 for n in numbers3):

    print("At least one number is divisible by 5")

<u>Enumerate</u>

1. Using below list and enumerate(), print index followed by value.

Input: fruits = ["apple", "banana", "cherry"]
Output:

```
0 apple
1 banana
2 cherry
```

2. Using below dict and enumerate, print key followed by value

Input: person = {"name": "Alice", "age": 30, "city": "New York"}

Output:
```
name: Alice
age: 30
city: New York
```

3. Given the list `fruits = ["apple", "banana", "cherry", "date", "elderberry"]`, use `enumerate()` to create a list of tuples where each tuple contains the index and the corresponding fruit, but only for even indices.

Output:

```
[(2, 'banana'), (4, 'date')]
```

Solutions :

```
# Index then value
fruits = ["apple", "banana", "cherry"]
for i, f in enumerate(fruits):
    print(i, f)

# Dict key: value
person = {"name": "Alice", "age": 30, "city": "New York"}
for k, v in person.items():
    print(f"{k}: {v}")

# Even indices only
fruits2 = ["apple", "banana", "cherry", "date", "elderberry"]
even_index_fruits = [(i, f) for i, f in enumerate(fruits2) if i % 2 == 0]
print(even_index_fruits)  # [(0,'apple'), (2,'cherry'), (4,'elderberry')]
```

Max() , Min()

1. Find the Maximum and Minimum Values in a List

```
numbers = [1, 32, 63, 14, 5, 26, 79, 8, 59, 10]
```

2. Given a set of numbers, find the maximum and minimum values.

```
setn = {5, 10, 3, 15, 2, 20}
```

3. Write a Python function that takes a list of strings as input and returns a tuple containing the shortest and longest word from the list, in that order. If there are multiple words of the same shortest or longest length, return the first shortest/longest word found.

```
Input: words = ["apple", "banana", "kiwi", "grapefruit", "orange"]
Output: ('kiwi', 'grapefruit')
```

Solutions :

```
numbers = [1, 32, 63, 14, 5, 26, 79, 8, 59, 10]
print(max(numbers), min(numbers))  # 79 1

setn = {5, 10, 3, 15, 2, 20}
print(max(setn), min(setn))  # 20 2

def shortest_longest(words):
    shortest = min(words, key=len)
    longest = max(words, key=len)
    return shortest, longest

words_list = ["apple", "banana", "kiwi", "grapefruit", "orange"]
print(shortest_longest(words_list))  # ('kiwi', 'grapefruit')
```

Exception Handling

1. Write a Python program that attempts to divide two numbers a = 10 b = 0
and handles a ZeroDivisionError if the denominator is zero. Divide a by b and handle the exception and print the error

2. Apply exception handling to below code and handle an exception if the index is out of range.
```
my_list = [1, 2, 3]
print(my_list[5])
```

3. Correct this below code with appropriate exception handlings. And finally print “Execution completed”
```
def safe_divide(a,b):
    result = a / b
    print(f"Result: {result}")


safe_divide(1,0)
safe_divide(1,”a”)
```

Solutions :

```
# ZeroDivisionError handling
a, b = 10, 0
try:
    print(a / b)
except ZeroDivisionError as e:
    print(f"Error: {e}")

# Index out of range
my_list = [1, 2, 3]
try:
    print(my_list[5])
except IndexError as e:
    print(f"Index error: {e}")

# Robust safe_divide
def safe_divide(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        print("Cannot divide by zero")
    except TypeError:
        print("Both arguments must be numbers")
    else:
        print(f"Result: {result}")
    finally:
        print("Execution completed")

safe_divide(1, 0)
safe_divide(1, "a")
```

<u>**Decorator**</u>

1. Write a function that appends 1 to 1000 numbers to a list and add a decorator to that function to calculate the start and end time. Calculate the total time taken and print.

2. Create a parameterised decorator retry that retries a function a specified number of times.

```
@retry(3)
    def may_fail(name):
        print(f"Hello, {name}!")
```

3. Create a decorator validate_positive for below function that ensures the argument passed to a function is positive.

```
    def square_root(x):
  return x ** 0.5
```

4. Create a decorator cache that caches the result of a function based on its arguments.

```
@cache
   def expensive_computation(x):
  print("Performing computation...")
  return x * x
expensive_computation(5)
expensive_computation(5)
```

Write a cache decorator for it to check if the calculation is already performed then return the result.

5. Create a decorator requires_permission that checks if a user has the ‘admin’ permission before allowing access to a function, if a different user then responds “Access denied”.

```
 def delete_user(user, user_id):
  print(f"User {user_id} deleted by {user['name']}")

user1 = {'name': 'Alice', 'permissions': ['admin']}
user2 = {'name': John, 'permissions': ['dev']}
user3 = {'name': 'Kurt', 'permissions': ['test'']}
```

Solutions :

```
# Timing
def timed(fn):
    @wraps(fn)
    def wrapper(*args, **kwargs):
        start = time.time()
        result = fn(*args, **kwargs)
        end = time.time()
        print(f"{fn.__name__} took {end - start:.4f}s")
        return result
    return wrapper

@timed
def append_numbers():
    return [i for i in range(1, 1001)]

# Retry
def retry(n):
    def deco(fn):
        @wraps(fn)
        def wrapper(*args, **kwargs):
            last_exc = None
            for _ in range(n):
                try:
                    return fn(*args, **kwargs)
                except Exception as e:
                    last_exc = e
            raise last_exc
        return wrapper
    return deco

@retry(3)
def may_fail(name):
    print(f"Hello, {name}!")

# Validate positive
def validate_positive(fn):
    @wraps(fn)
    def wrapper(x):
        if x <= 0:
            raise ValueError("Value must be positive")
        return fn(x)
    return wrapper

@validate_positive
def square_root(x):
    return x ** 0.5

# Cache
def cache(fn):
    memo = {}
    @wraps(fn)
    def wrapper(*args):
        if args in memo:
            return memo[args]
        memo[args] = fn(*args)
        return memo[args]
    return wrapper

@cache
def expensive_computation(x):
    print("Performing computation...")
    return x * x

# Permission check
def requires_permission(permission):
    def deco(fn):
        @wraps(fn)
        def wrapper(user, *args, **kwargs):
            if permission in user.get("permissions", []):
                return fn(user, *args, **kwargs)
            print("Access denied")
        return wrapper
    return deco

@requires_permission('admin')
def delete_user(user, user_id):
    print(f"User {user_id} deleted by {user['name']}")

user1 = {'name': 'Alice', 'permissions': ['admin']}
user2 = {'name': 'John',  'permissions': ['dev']}
user3 = {'name': 'Kurt',  'permissions': ['test']}
```

GENERATOR

1. Write a code using generator can be used to produce an infinite sequence of Fibonacci numbers
Of 10 numbers

Output:

```
0
```

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

2. Write a generator function called infinite_multiples(n) that yields multiples of the given base value indefinitely.

Input n=3

Output:
```
3
6
9
12
15
…
```

3. Write a generator function called repeat_word(word, times) that yields the given character char a specified number of times.

word = “hello”
times = 5

Solutions :

```
# Fibonacci (first 10)
def fib():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

g = fib()
for _ in range(10):
    print(next(g))

# Infinite multiples
def infinite_multiples(n):
    k = 1
    while True:
        yield n * k
        k += 1

# Repeat word
def repeat_word(word, times):
    for _ in range(times):
        yield word
```

FILE HANDLING

1 . Write a Python program to read the entire content of a file named sample.txt and display it.

2. Write a Python program to count the number of words in a file named words.txt

3.Create a program to write the string “Hello, Python!” into a file named output.txt.

4. Write a Python program to create a CSV file named students.csv with columns Name, Roll Number, and Marks. Add at least three entries

```
data = [
    ["Name", "Roll Number", "Marks"],
    ["Alice", "101", "85"],
    ["Bob", "102", "90"],
    ["Charlie", "103", "88"]
]
```

5. From a file with 100+ lines. Write a code using a generator to fetch all the data from the file.

Solutions :

```
# Read entire file
print(Path("sample.txt").read_text())
```

```
# Count words
word_count = len(Path("words.txt").read_text().split())
print(word_count)

# Write string
Path("output.txt").write_text("Hello, Python!")

# Write CSV
data = [
    ["Name", "Roll Number", "Marks"],
    ["Alice", "101", "85"],
    ["Bob", "102", "90"],
    ["Charlie", "103", "88"],
]
with open("students.csv", "w", newline="") as f:
    csv.writer(f).writerows(data)

# Stream big file
def stream_file(path):
    with open(path, "r") as f:
        for line in f:
            yield line.rstrip()

for line in stream_file("bigfile.txt"):
    pass  # process line
```

CLASS

1. Define a class Person with attributes name and age. Create an instance of this class and print its attributes.

2. Problem: Write a Python class named BankAccount with attributes like account_number, balance, and customer_name, and methods like deposit, withdraw, and check_balance.

3. Create a class Book with a class method from_string() that creates a Book instance from a string. And print both attributes of the class

```
book = Book.from_string("Python Programming, John Doe")
```

4. Create a base class Animal with a method sound(). Create subclasses Dog and Cat that overrides the sound() method and call those methods.

5. Write a code to perform multiple inheritance.

Solutions :

```
# Person
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
p = Person("Nitin", 30)
print(p.name, p.age)

# BankAccount
class BankAccount:
    def __init__(self, account_number, balance, customer_name):
        self.account_number = account_number
        self.balance = balance
        self.customer_name = customer_name
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        if amount > self.balance:
            raise ValueError("Insufficient funds")
        self.balance -= amount
    def check_balance(self):
        return self.balance

# Book.from_string
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
    @classmethod
    def from_string(cls, s):
        title, author = [part.strip() for part in s.split(",")]
        return cls(title, author)
book = Book.from_string("Python Programming, John Doe")
print(book.title, book.author)

# Inheritance
class Animal:
    def sound(self):
        return "..."
class Dog(Animal):
    def sound(self):
        return "Woof"
class Cat(Animal):
    def sound(self):
        return "Meow"

# Multiple inheritance example
class A: pass
class B: pass
class C(A, B): pass
```

Modules

1. Using datetime, add a week and 12 hours to a date. Given date: March 22, 2020, at 10:00 AM. print original date time and new date time

2. Code to get the dates of yesterday, today, and tomorrow.

3. Write a code snippet using os module, to get the current working directory and print and create a folder “test”. List all the files and folders in the current working directory and remove the directory “test” that was created.

4. Write a Python program to rename a file from old_name.txt to new_name.txt.

5. Create a file and Write a Python program to get the size of a file named example.txt

6. Convert the string "Feb 25 2020 4:20PM" into a Python datetime object

O/P: 2020-02-25 16:20:00

7. Subtract 7 days from the date 2025-02-25 and print the result.

O/P: New date: 2025-02-18

8. Format the date 2020-02-25 as "Tuesday 25 February 2020"

Solutions :

```
# Add 1 week + 12 hours
dt = datetime(2020, 3, 22, 10, 0)
new_dt = dt + timedelta(weeks=1, hours=12)
print(dt, new_dt)

# Yesterday, today, tomorrow
today = date.today()
yesterday = today - timedelta(days=1)
tomorrow = today + timedelta(days=1)
print(yesterday, today, tomorrow)

# CWD, make/remove folder, list
cwd = os.getcwd()
os.makedirs("test", exist_ok=True)
print(os.listdir(cwd))
os.rmdir("test")

# Rename file
os.rename("old_name.txt", "new_name.txt")
```

```
# File size
print(os.path.getsize("example.txt"))

# Parse date string
dt2 = datetime.strptime("Feb 25 2020 4:20PM", "%b %d %Y %I:%M%p")
print(dt2)

# Subtract 7 days
print(date(2025, 2, 25) - timedelta(days=7))  # 2025-02-18

# Format date
print(datetime(2020, 2, 25).strftime("%A %d %B %Y"))
```