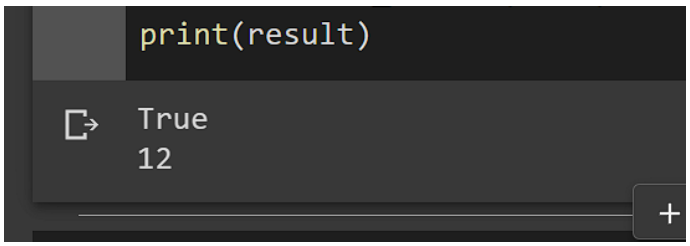
APP WEEK-10 LAB

Q1. Calculate the following using Lambda calculus:
a. T AND F
b. 3 * 4

Code:

```
# T AND F
T = lambda x: lambda y: x
F = lambda x: lambda y: y
result = (T(F))(1)(2)
print(bool(result))

# 3 * 4
# Helper function to convert Church encoding to a number
def from_church(n):
    return n(lambda x: x + 1)(0)
succ = lambda n: lambda f: lambda x: f(n(f)(x))
three = lambda f: lambda x: f(f(f(x)))
four = lambda f: lambda x: f(f(f(f(x))))
mult = lambda n: lambda m: lambda f: n(m(f))
result = from_church(mult(three)(four))
print(result)
```

SnapShot:



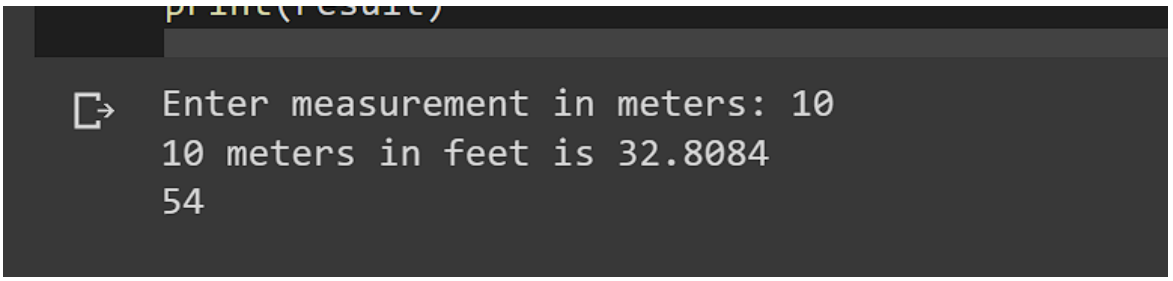
Q2. Lambda functions
a. Write a lambda function to convert measurements from meters to feet.
b. Write a lambda function in Python to implement the following lambda expression:
(λλλλ. λλλλ. (ff + mm)aa)(λλλλ. xx2)(bb)
Note: You need to write a nested lambda function for implementing f+m where f takes the square
function (which takes argument x) passed as a parameter. The above expression calculates a2+b.

Code:

```
#A
meters_to_feet = lambda m: m * 3.28084
meter = int(input("Enter measurement in meters: "))
feet = meters_to_feet(meter)
print(f"{meter} meters in feet is {feet}")
```

```
#B
square = lambda x: x ** 2
double = lambda x: x * 2
sum_of_squares = lambda f: lambda m: lambda a: f(square(a)) + m(double(a))
result = (sum_of_squares(double))(square)(3)  # a = 3, b = 3
print(result)
```

SnapShot:



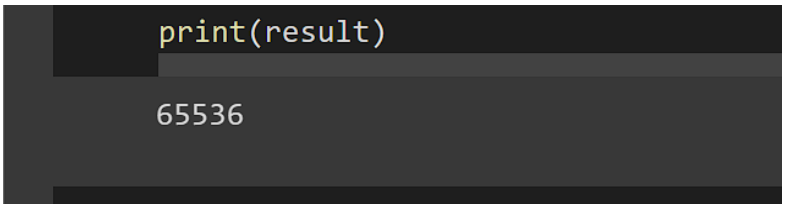
Q3. Passing and returning a function as an argument

Define a function 'square' for squaring a number. Define a function named 'twice' that takes a function f as an argument and returns f(f(x)). Using 'twice' and 'square' create a function 'quad' that takes n as an argument and returns n4. 'quad' should not be defined explicitly. It should only be created as a variable which is then assigned a function.

Code:

```
def square(x):
    """Function to square a number"""
    return x ** 2

def twice(f):
    """
    Function that takes a function f as an argument and returns f(f(x)).
    Here, f is a function that takes one argument (x).
    """
    def g(x):
        return f(f(x))
    return g

# Creating the quad function using the twice and square functions
quad = twice(twice(square))

# Testing the quad function with an input of 2
result = quad(2)
print(result)
```

SnapShot:

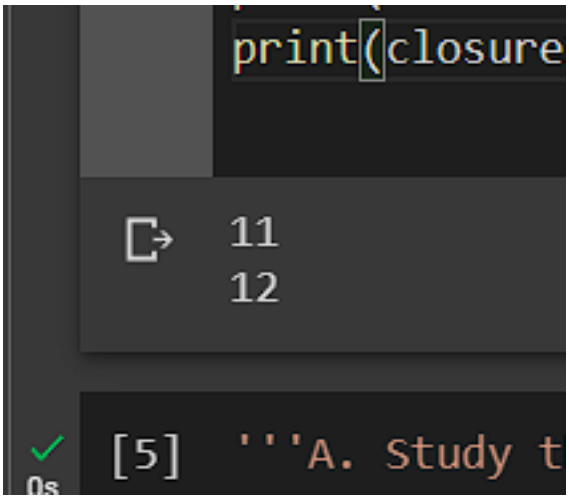


Q4. Closure
A Closure is a function object that remembers values in enclosing scopes even if they are not present
in memory. We have a closure in Python when a nested function references a value in its enclosing
scope.

Code:

```
def outer_function(x):
    def inner_function(y):
        return x + y
    return inner_function

# Create closures
closure1 = outer_function(1)
closure2 = outer_function(2)

# Call closures
print(closure1(10))
print(closure2(10))
```

SnapShot:



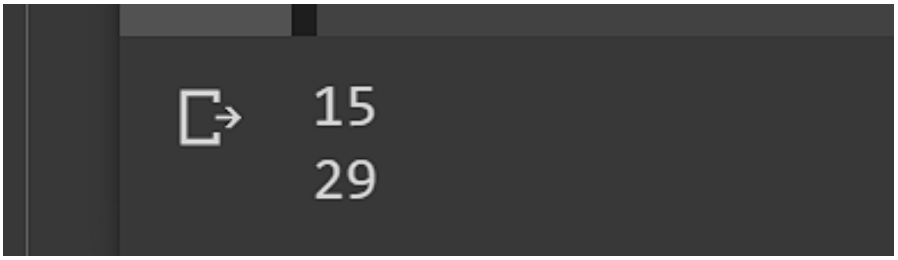
Q5.
A. Study the following program by executing it:
def multiplier_of(n):
def
multiplier(number):
return number*n
return multiplier

B. In a lottery system, random number is chosen by retrieving the number from a random index from a list of random numbers. Write a program to choose a random number in this way. You
must use nested functions – the inner function chooses a number from a random index and the
outer function generates a random list of numbers. The outer function takes n as a parameter where
is the maximum number that can be put in the random list. (Your code should be similar to the
program in 5a)

Code:

```
#A
def multiplier_of(n):
    def multiplier(number):
        return number*n
    return multiplier

times_3 = multiplier_of(3)
print(times_3(5))  # Output: 15

#B
import random

def random_num_gen(n):
    nums = [random.randint(1, n) for i in range(10)]  # Generating a random list of 10 numbers between 1 and n
    def random_num():
        return nums[random.randint(0, 9)]  # Choosing a random number from the list
    return random_num

num_generator = random_num_gen(100)
print(num_generator())  # Output: A random number between 1 and 100
```

SnapShot:

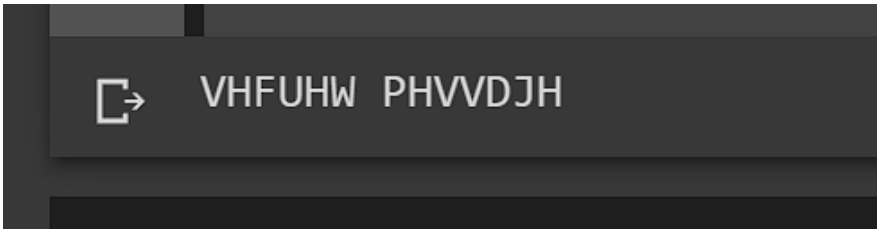


Q6. Map
A secret message needs to be sent. Use the map function to encrypt the message using Caesar cipher.

Code:

```
def caesar_cipher(char, shift):
    if char.isalpha():
        alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        char_index = alpha.index(char.upper())
        new_index = (char_index + shift) % len(alpha)
        new_char = alpha[new_index]
        return new_char if char.isupper() else new_char.lower()
    else:
        return char

message = "SECRET MESSAGE"
shift = 3

encrypted_message = ''.join(map(lambda char: caesar_cipher(char, shift), message))
print(encrypted_message)  # Output: VHFXHU PHVVDJH
```

SnapShot:



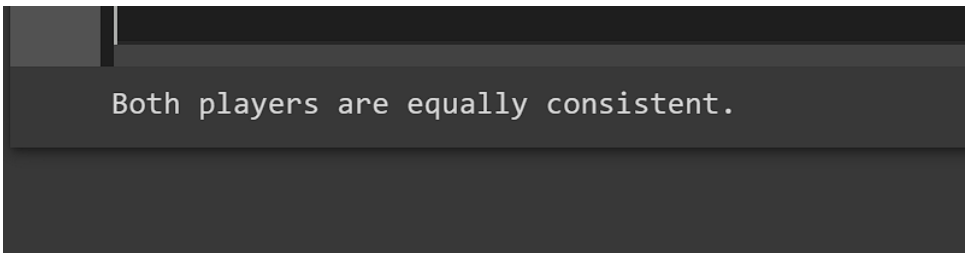
Q7. Reduce
Given runs scored by 2 players in a series of matches, write a Python program using reduce function to find who is the better player of the two in terms of maintaining consistency. (You need to find SD).

Code:

```
from functools import reduce
import math

# runs scored by two players in a series of matches
player1_runs = [10, 20, 30, 40, 50]
player2_runs = [15, 25, 35, 45, 55]

# function to calculate the average of a list of numbers
def average(numbers):
    return reduce(lambda x, y: x + y, numbers) / len(numbers)

# function to calculate the standard deviation (SD) of a list of numbers
def std_deviation(numbers):
```

```
    avg = average(numbers)
    squared_diff = reduce(lambda x, y: x + y, map(lambda x: (x - avg) ** 2, numbers))
    return math.sqrt(squared_diff / len(numbers))

# calculate the SD for each player's runs scored
player1_sd = std_deviation(player1_runs)
player2_sd = std_deviation(player2_runs)

# determine which player is more consistent based on their SD
if player1_sd < player2_sd:
    print("Player 1 is more consistent.")
elif player1_sd > player2_sd:
    print("Player 2 is more consistent.")
else:
    print("Both players are equally consistent.")
```

SnapShot:

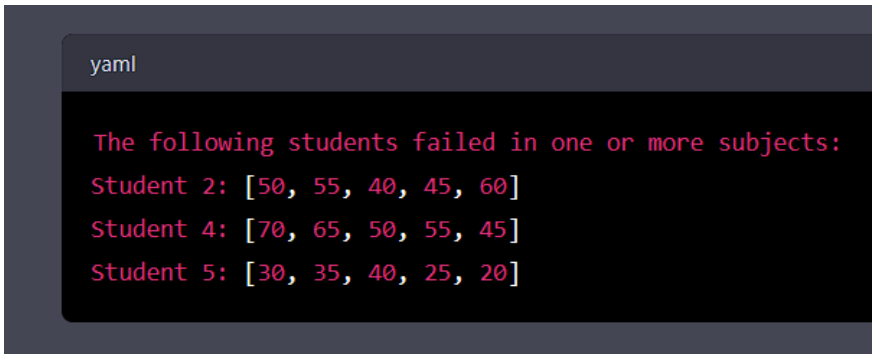


Q8. Filter
The marks scored by a class of students in 5 different subjects are stored in a list of lists. Using the
filter function, write a program to find the students who failed in one or more subjects.

Code:

```
# List of lists representing marks of each student in 5 different subjects
marks = [[80, 85, 70, 60, 75], [50, 55, 40, 45, 60], [90, 95, 80, 85, 75], [70, 65, 50, 55, 45],
[30, 35, 40, 25, 20]]

# Function to check if a student has failed in one or more subjects
def has_failed(marks):
    for mark in marks:
        if mark < 40:
            return True
    return False

# Using filter to find the students who failed in one or more subjects
failed_students = list(filter(has_failed, marks))

# Printing the list of failed students
print("The following students failed in one or more subjects:")
for i, student in enumerate(failed_students):
    print(f"Student {i+1}: {student}")
```

SnapShot:

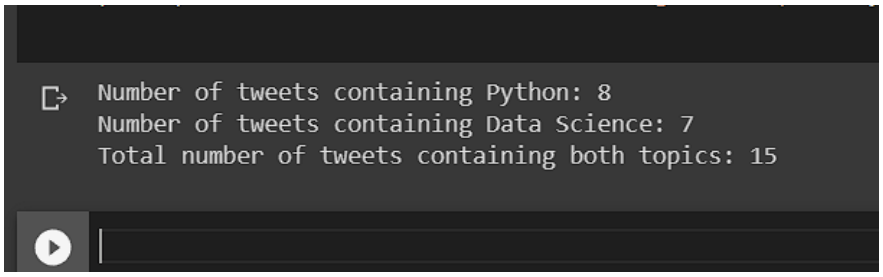


Q9. Map+reduce+filter
Given two trending topics and a bunch of tweets, write a Python program to count the number of
tweets that contain each topic. You need to do this by putting together map(), reduce() and filter()
functions.

Code:

```
from functools import reduce

# example trending topics
topic1 = "Python"
topic2 = "Data Science"

# example tweets list
tweets = [
    "Just wrote my first Python script!",
    "Data Science is my passion",
    "I can't wait to learn more about Python and Data Science",
    "I hate Python",
    "Data Science is not for me",
    "Python vs R - Which one is better for Data Science?",
    "Just got a job as a Data Scientist and I'm excited to use Python",
    "I wish I could go back to using Java instead of Python for Data Science",
    "Data Science is the future",
    "Python is the future",
    "I'm learning Python and Data Science at the same time, it's challenging but fun!"
]

# helper function to count the number of tweets containing a given topic
def count_tweets_containing_topic(topic):
    return len(list(filter(lambda tweet: topic.lower() in tweet.lower(), tweets)))

# use map() to count the number of tweets containing each topic
counts = list(map(count_tweets_containing_topic, [topic1, topic2]))

# use reduce() to sum up the counts
total_count = reduce(lambda x, y: x + y, counts)

# print the results
print(f"Number of tweets containing {topic1}: {counts[0]}")
print(f"Number of tweets containing {topic2}: {counts[1]}")
print(f"Total number of tweets containing both topics: {total_count}")
```

SnapShot:

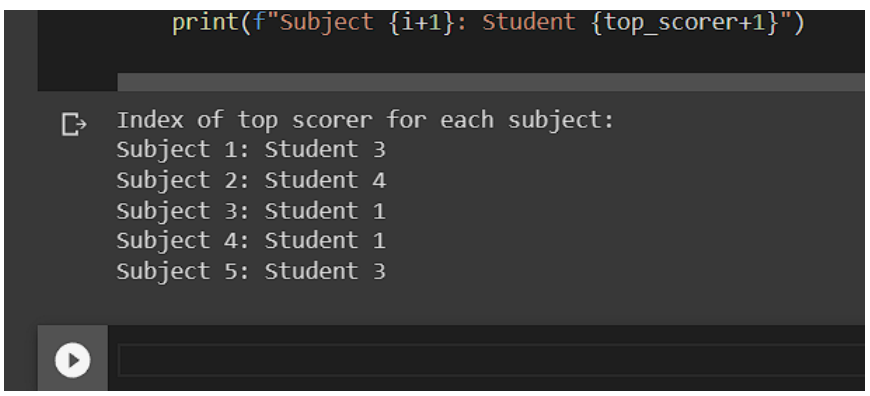


Q10. Given the list of scores secured by the students as a multidimension list Extracting Toppers From a Student Record print the index of top scorer in each subject using function tools

Code:

```
from functools import reduce

# sample multidimensional list of scores
scores = [
    [90, 87, 94, 91, 88],
    [84, 92, 85, 90, 87],
    [92, 89, 90, 86, 93],
    [88, 92, 87, 90, 84],
    [89, 90, 91, 86, 92]
]

# get the index of top scorer for each subject
top_scorers = list(map(lambda subject: reduce(lambda a, b: a if scores[a][subject] >
scores[b][subject] else b, range(len(scores))), range(len(scores[0]))))

# print the index of top scorer for each subject
print("Index of top scorer for each subject: ")
for i, top_scorer in enumerate(top_scorers):
    print(f"Subject {i+1}: Student {top_scorer+1}")
```

SnapShot:

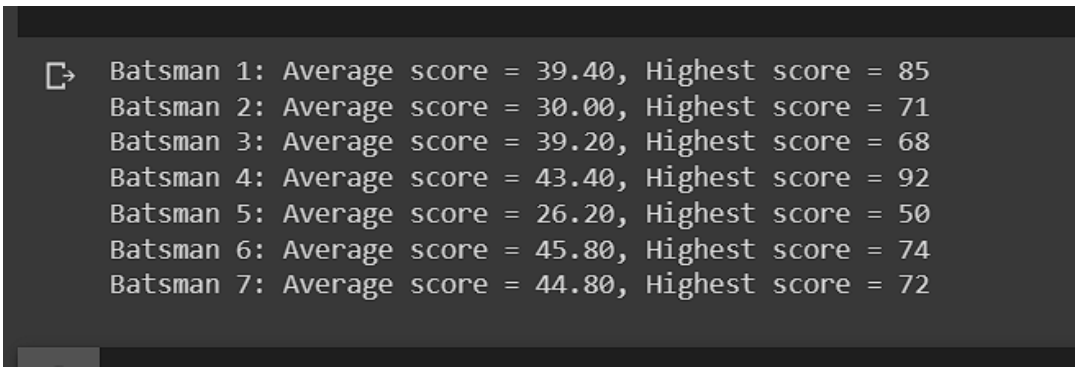


Q11. Given the list scores secured by the batsmans over a certain matches, compute the average score of individual batsman, also the individual highest score of the batsman.

Code:

```
from collections import defaultdict

# Define the list of scores
scores = [
    [42, 0, 31, 52, 12, 74, 56],
    [11, 25, 13, 17, 30, 40, 19],
    [85, 71, 63, 92, 50, 37, 68],
    [3, 7, 21, 19, 10, 28, 9],
    [56, 47, 68, 37, 29, 50, 72]
]

# Define a function to compute the average score and highest score of each batsman
def compute_stats(scores):
    # Create a dictionary to store the scores of each batsman
    bat_dict = defaultdict(list)

    # Loop over the scores for each match and store them in the dictionary
    for match_scores in scores:
        for i, score in enumerate(match_scores):
            bat_dict[i].append(score)

    # Loop over the scores for each batsman and compute the average and highest score
    for bat, bat_scores in bat_dict.items():
        avg_score = sum(bat_scores) / len(bat_scores)
        max_score = max(bat_scores)
        print(f"Batsman {bat + 1}: Average score = {avg_score:.2f}, Highest score = {max_score}")

# Call the function to compute the stats for the scores list
compute_stats(scores)
```

SnapShot:



Q12. Calculate the number of grains of wheat on a chessboard given that the number on each square doubles. There once was a wise servant who saved the life of a prince. The king promised to pay whatever the servant could dream up. Knowing that the king loved chess, the servant told the king he would like to have grains of wheat. One grain on the first square of a chess board, with the number of grains doubling on each successive square. There are 64 squares on a chessboard (where square 1 has one grain, square 2 has two grains, and so on).
Write code that shows:
how many grains were on a given square, and the total number of grains on the chessboard.

Code:

```
num_squares = 64
grains_on_square = [2**n for n in range(num_squares)]
total_grains = sum(grains_on_square)

print("Grains on each square:", grains_on_square)
print("Total grains on the chessboard:", total_grains)
```

SnapShot: