**Python Lab 9**

**Name: LUY PATWA**

**Roll number: 24BEC065**

**Div:1 (F2)**

*Q1*

*Def prime\_factors(n, divisor=2):*

*# Base case: If n becomes 1, return an empty list*

*If n <= 1:*

*Return []*

*# If n is divisible by the current divisor, it’s a prime factor*

*If n % divisor == 0:*

*Return [divisor] + prime\_factors(n // divisor, divisor)*

*# Otherwise, increment the divisor to check the next number*

*Return prime\_factors(n, divisor + 1)*

*# Input from the user*

*Try:*

*Num = int(input(“Enter a positive integer to find its prime factors: “))*

*If num <= 0:*

*Print(“Please enter a positive integer.”)*

*Else:*

*Factors = prime\_factors(num)*

*Print(f”Prime factors of {num} are: {factors}”)*

*Except ValueError:*

*Print(“Invalid input. Please enter a valid integer.”)*

*Output*

*Enter a positive integer to find its prime factors: 56*

*Prime factors of 56 are: [2, 2, 2, 7]*

*Enter a positive integer to find its prime factors: 97*

*Prime factors of 97 are: [97]*

*Enter a positive integer to find its prime factors: 120*

*Prime factors of 120 are: [2, 2, 2, 3, 5]*

*Q2*

*Def find\_binary(n):*

*# Base case: if n is 0 or 1, return it as a string*

*If n <= 1:*

*Return str(n)*

*Else:*

*# Recursive call: Divide n by 2 and find the remainder*

*Return find\_binary(n // 2) + str(n % 2)*

*# Input from the user*

*Try:*

*Num = int(input(“Enter a positive integer to find its binary equivalent: “))*

*If num < 0:*

*Print(“Please enter a positive integer.”)*

*Else:*

*Print(f”Binary equivalent of {num} is: {find\_binary(num)}”)*

*Except ValueError:*

*Print(“Invalid input. Please enter a valid integer.”)*

*Output*

*Enter a positive integer to find its binary equivalent: 10*

*Binary equivalent of 10 is: 1010*

*Enter a positive integer to find its binary equivalent: 25*

*Binary equivalent of 25 is: 11001*

*Enter a positive integer to find its binary equivalent: 0*

*Binary equivalent of 0 is: 0*

*Enter a positive integer to find its binary equivalent: 1*

*Binary equivalent of 1 is: 1*

*Q3*

*Def count\_vowels(s):*

*# Base case: if the string is empty, return 0*

*If len(s) == 0:*

*Return 0*

*# Check if the first character is a vowel*

*Vowels = ‘aeiouAEIOU’*

*If s[0] in vowels:*

*Return 1 + count\_vowels(s[1:])*

*Else:*

*Return count\_vowels(s[1:])*

*# Input from the user*

*String = input(“Enter a string: “)*

*# Call the recursive function*

*Vowel\_count = count\_vowels(string)*

*# Output the result*

*Print(“Number of vowels in the string:”, vowel\_count)*

*Output*

*Enter a string: Hello World*

*Number of vowels in the string: 3*

*Enter a string: Python Programming*

*Number of vowels in the string: 5*

*Q4*

*Def reverse\_list(lst):*

*# Base case: if the list is empty or has only one element*

*If len(lst) <= 1:*

*Return lst*

*Else:*

*# Recursive case: reverse the rest of the list and append the first element to the end*

*Return reverse\_list(lst[1:]) + [lst[0]]*

*# Example input*

*Numbers = [1, 2, 3, 4, 5]*

*Print(“Original List:”, numbers)*

*# Reversing using the recursive function*

*Reversed\_numbers = reverse\_list(numbers)*

*Print(“Reversed List:”, reversed\_numbers)*

*Output*

*Original List: [1, 2, 3, 4, 5]*

*Reversed List: [5, 4, 3, 2, 1]*

*Q5*

*Def multiply(a, b):*

*# Base case: if b is 0, the result is 0*

*If b == 0:*

*Return 0*

*# If b is positive, recursively add ‘a’*

*Elif b > 0:*

*Return a + multiply(a, b – 1)*

*# If b is negative, convert to positive and negate the result*

*Else:*

*Return -multiply(a, -b)*

*# Taking input using keyword arguments*

*A = int(input(“Enter value for a: “))*

*B = int(input(“Enter value for b: “))*

*# Calling the recursive function*

*Result = multiply(a=a, b=b)*

*# Display the result*

*Print(f”The product of {a} and {b} is: {result}”)*

*Output*

*Enter value for a: 5*

*Enter value for b: 3*

*The product of 5 and 3 is: 15*

*Q6*

*def sanitize\_list(lst, index=0):*

*# Base case: if index reaches the end of the list, return the list*

*if index == len(lst):*

*return lst*

*# Replace negative value with 0*

*if lst[index] < 0:*

*lst[index] = 0*

*# Recursive call for the next index*

*return sanitize\_list(lst, index + 1)*

*# Example usage*

*my\_list = [4, -3, 7, -1, 0, 9, -8, 6]*

*print("Original List:", my\_list)*

*sanitized\_list = sanitize\_list(my\_list)*

*print("Sanitized List:", sanitized\_list)*

*Output*

*Original List: [4, -3, 7, -1, 0, 9, -8, 6]*

*Sanitized List: [4, 0, 7, 0, 0, 9, 0, 6]*

*Q7*

*Def recursive\_sum(numbers):*

*# Base case: if the list has only one number, return that number*

*If len(numbers) == 1:*

*Return numbers[0]*

*# Recursive case: sum the first number with the sum of the rest of the list*

*Else:*

*Return numbers[0] + recursive\_sum(numbers[1:])*

*Def find\_average(numbers):*

*If len(numbers) == 0:*

*Return 0 # Return 0 if the list is empty to avoid division by zero*

*Total\_sum = recursive\_sum(numbers)*

*Return total\_sum / len(numbers)*

*# Example usage*

*Numbers\_list = [10, 20, 30, 40, 50]*

*Average = find\_average(numbers\_list)*

*Print(f”The average of {numbers\_list} is: {average}”)*

*Output*

*The average of [10, 20, 30, 40, 50] is: 30.0*

*Q8*

*Def string\_length(s):*

*# Base case: if the string is empty, return 0*

*If s == “”:*

*Return 0*

*Else:*

*# Recursive case: reduce the string by one character and add 1*

*Return 1 + string\_length(s[1:])*

*# Input from user*

*User\_input = input(“Enter a string: “)*

*Length = string\_length(user\_input)*

*Print(f”The length of the string is: {length}”)*

*Output*

*Enter a string: Hello World*

*The length of the string is: 11*