TIGER ANALYTICS

CODING EXERCISE-PYTHON

Submitted by

CALVIN DAVIS S R

On

15.07.2024

Data Science - Internship

Coding Exercise-Python

Question 1:

Write a program which takes 2 digits, X,Y as input and generates a 2-dimensional array.

The element value in the i-th row and j-th column of the array should be i*j.

Note: i=0,1.., X-1; j=0,1,i-Y-1.

Expected Input:

3,5

Expected Output:

[[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
def generate_2d_array(X, Y):
    array = []
    for i in range(X):
        row = []
        for j in range(Y):
            row.append(i * j)
        array.append(row)
    return array

# Example usage
X = 3
Y = 5
result = generate_2d_array(X, Y)
print(result)
```

Question 2:

Write a program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.

Expected Input:

without,hello,bag,world

Expected Output:

bag,hello,without,world

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
def sort_words(input_string):
    # Split the input string into a list of words
    words = input_string.split(',')
    # Sort the list of words alphabetically
    words.sort()
    # Join the sorted list back into a comma-separated string
    sorted_words = ','.join(words)
    return sorted_words

# Example usage
input_string = "without,hello,bag,world"
output = sort_words(input_string)
print(output)
```

Question 3:

Write a program that accepts a sequence of whitespace separated words as input and prints the words after removing all duplicate words and sorting them alphanumerically.

Expected Input:

hello world and practice makes perfect and hello world again

Expected Output:

again and hello makes perfect practice world

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
def remove_duplicates_and_sort(input_string):
    words = input_string.split()
    unique_words = set(words)
    # Sort the unique words alphanumerically
    sorted_words = sorted(unique_words)
    # Join the sorted list back into a whitespace-separated string
    result = ' '.join(sorted_words)
    return result

# Example usage
input_string = "hello world and practice makes perfect and hello world again"
output = remove_duplicates_and_sort(input_string)
print(output)
```

Question 4:

Write a program, which will find all such numbers between 1000 and 3000 (both included) such that each digit of the number is an even number.

The numbers obtained should be printed in a comma-separated sequence on a single line.

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
def find_even_digit_numbers():
    even_digit_numbers = []
    for number in range(1000, 3001):  # Iterate from 1000 to 3000 (inclusive)
```

```
        number_str = str(number)
        if all(int(digit) % 2 == 0 for digit in number_str):  # Check if all digits are even
            even_digit_numbers.append(number_str)
    return ','.join(even_digit_numbers)

# Example usage
output = find_even_digit_numbers()
print(output)
```

Question 5:

Write a program that accepts a sentence and calculate the number of letters and digits.

Expected Input:

hello world! 123

Expected Output:

LETTERS 10

DIGITS 3

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
def count_letters_and_digits(input_string):
    letters = 0
    digits = 0

    for char in input_string:
        if char.isalpha():  # Check if the character is a letter
            letters += 1
        elif char.isdigit():  # Check if the character is a digit
            digits += 1

    return letters, digits
```

```
# Example usage

input_string = "hello world! 123"

letters, digits = count_letters_and_digits(input_string)

print(f"LETTERS {letters}")

print(f"DIGITS {digits}")
```

Question 6:

Write a program that accepts a sentence and calculate the number of upper case letters and lower case letters.

Expected Input:

Hello world!

Expected Output:

UPPER CASE 1

LOWER CASE 9

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
def count_upper_and_lower_case(sentence):
    upper_case = 0
    lower_case = 0

    for char in sentence:
        if char.isupper():
            upper_case += 1
        elif char.islower():
            lower_case += 1

    return upper_case, lower_case
```

```
# Example usage

sentence = input("Enter a sentence: ")

upper_case, lower_case = count_upper_and_lower_case(sentence)

print(f"UPPER CASE {upper_case}")

print(f"LOWER CASE {lower_case}")
```

Question 7:

Write a program that computes the net amount of a bank account based a transaction log from console input. The transaction log format is shown as following:

D 100

W 200

D means deposit while W means withdrawal.

Expected Input:

D 300

D 300

W 200

D 100

Expected Output:

500

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
def compute_net_amount(transaction_log):
    net_amount = 0

    transactions = transaction_log.split(",")  # Split the input into individual transactions

    for transaction in transactions:
        transaction = transaction.strip()  # Remove any leading/trailing spaces
        if transaction.startswith('D'):
```

```
            amount = int(transaction[2:])  # Extract the deposit amount
            net_amount += amount
        elif transaction.startswith('W'):
            amount = int(transaction[2:])  # Extract the withdrawal amount
            net_amount -= amount

    return net_amount

# Example usage
input_data = input("Enter transaction log (comma-separated): ")
net_amount = compute_net_amount(input_data)
print(net_amount)
```

Question 8:

A website requires the users to input username and password to register. Write a program to check the validity of password input by users.

Following are the criteria for checking the password:

1. At least 1 letter between [a-z]

2. At least 1 number between [0-9]

1. At least 1 letter between [A-Z]

3. At least 1 character from [$#@]

4. Minimum length of transaction password: 6

5. Maximum length of transaction password: 12

Passwords that match the criteria are to be printed, each separated by a comma.

Expected Input:

ABd1234@1,a F1#,2w3E*,2We3345

Expected Output:

ABd1234@1

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
import re

def is_valid_password(password):
    # Criteria check using regular expressions
    if (len(password) < 6 or len(password) > 12):
        return False
    if not re.search("[a-z]", password):
        return False
    if not re.search("[A-Z]", password):
        return False
    if not re.search("[0-9]", password):
        return False
    if not re.search("[$#@]", password):
        return False
    return True

def check_passwords(passwords):
    valid_passwords = []

    password_list = passwords.split(",")  # Split the input into individual passwords

    for password in password_list:
        if is_valid_password(password.strip()):  # Check each password
            valid_passwords.append(password.strip())

    return ",".join(valid_passwords)

# Example usage
```

```
input_data = input("Enter passwords (comma-separated): ")

valid_passwords = check_passwords(input_data)

print(valid_passwords)
```

Question 9:

You are required to write a program to sort the (name, age, height) tuples by ascending order where name is string, age and height are numbers. The tuples are input by console. The sort criteria is:

1: Sort based on name;

2: Then sort based on age;

3: Then sort by score.

The priority is that name > age > score.

Expected Input:

Tom,19,80

John,20,90

Jony,17,91

Jony,17,93

Json,21,85

Expected Output:

[('John', '20', '90'), ('Jony', '17', '91'), ('Jony', '17', '93'), ('Json', '21', '85'), ('Tom', '19', '80')]

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
def sort_tuples(data):
    # Convert the input string to a list of tuples
    tuples = [tuple(item.split(',')) for item in data.split(";")]

    # Sort the list of tuples
    sorted_tuples = sorted(tuples, key=lambda x: (x[0], int(x[1]), int(x[2])))

    return sorted_tuples
```

```
# Example usage

input_data = input("Enter the tuples (comma-separated, with semicolon between tuples): ")

result = sort_tuples(input_data)

print(result)
```

Question 10:

A robot moves in a plane starting from the original point (0,0). The robot can move toward UP, DOWN, LEFT and RIGHT with a given steps.

The numbers after the direction are steps.

The trace of robot movement is shown as the following:

Expected Input:

UP 5

DOWN 3

LEFT 3

RIGHT 2

Expected Output:

Compute the distance from current position after a sequence of movement and original point. If the distance is a float, then just print the nearest integer.

Note:

In case of taking data from the user, it should be in a comma-separated form.

Code:

```
import math


def calculate_final_position(movements):
    x, y = 0, 0  # Starting position at the origin

    # Parse and apply movements
    for movement in movements.split(";"):
        direction, steps = movement.split()
        steps = int(steps)
```

```
        if direction == "UP":
            y += steps
        elif direction == "DOWN":
            y -= steps
        elif direction == "LEFT":
            x -= steps
        elif direction == "RIGHT":
            x += steps

    # Compute the distance from the origin
    distance = math.sqrt(x**2 + y**2)
    return round(distance)  # Round to the nearest integer

# Example usage
input_data = input("Enter movements (comma-separated, with semicolon between movements): ")
distance = calculate_final_position(input_data)
print(distance)
```

Question 11:

Find the continuous occurrence of the string.

Expected Input:

Aabbcdeefffaabbcc

Expected Output:

a2b2c1d1e2f3a2b2c2

Code:

```
def find_continuous_occurrences(s):
    if not s:
        return ""

    result = []
    count = 1
```

```
    previous_char = s[0]

    for char in s[1:]:
        if char == previous_char:
            count += 1
        else:
            result.append(f"{previous_char}{count}")
            previous_char = char
            count = 1

    # Append the last character group
    result.append(f"{previous_char}{count}")

    return ''.join(result)

# Example usage
input_string = input("Enter the string: ")
output = find_continuous_occurrences(input_string)
print(output)
```

Question 12:

Find the pair of alphabets in an alphanumeric string whose sum of numbers in between is always 9

Expected Input 1:

a54b12c

Expected Output:

a,b

Expected Input 2:

a55b234cd9f63de54x3m

Expected Output:

b,c
b,d
d,f
f,d
f,e
e,x

Code:

```
import re

def find_pairs_with_sum_nine(s):
    # Regular expression to extract alphabets and numbers
    parts = re.findall(r'([a-zA-Z])(\d*)', s)
```

```
    # Create a dictionary to store numbers between alphabets
    num_dict = {}
    prev_char = None
    prev_number = 0

    for char, num in parts:
        num = int(num) if num else 0
        if prev_char:
            num_dict.setdefault(prev_char, []).append(num)
        prev_char = char
        prev_number = num

    result_pairs = []

    # Compare each pair of alphabets
    keys = list(num_dict.keys())
    for i in range(len(keys)):
        for j in range(i + 1, len(keys)):
            char1, char2 = keys[i], keys[j]
            sums1 = num_dict[char1]
            sums2 = num_dict[char2]
            # Check for pairs that sum up to 9
            for s1 in sums1:
                for s2 in sums2:
                    if s1 + s2 == 9:
                        result_pairs.append(f"{char1},{char2}")
                        break

    return result_pairs

# Example usage
input_string = input("Enter the alphanumeric string: ")
pairs = find_pairs_with_sum_nine(input_string)
for pair in pairs:
    print(pair)
```

Question 13:

Find how many pairs in a binary number that starts and ends with 1

Expected Input 1:

100101

Expected Output:

2

Expected Input 2:
1001101010010

Expected Output:
15

Code:

```
def count_pairs(binary_str):
    count = 0
    # Find all positions of '1' in the binary string
    ones_positions = [i for i, bit in enumerate(binary_str) if bit == '1']

    # Calculate the number of valid pairs
    num_ones = len(ones_positions)
    if num_ones < 2:
        return 0

    # Number of pairs can be calculated using combinations formula n choose 2
    count = num_ones * (num_ones - 1) // 2
    return count

# Example usage
input_binary = input("Enter the binary number: ")
result = count_pairs(input_binary)
print(result)
```

Question 14:
Find the minimum possible denominations for given valid currency.
(No of currencies used should be minimum)

Expected Input 1:
valid_currency: [1,2,5,10,20,50,100,200,500,2000]
Money: 210

Expected Output:
200-1
10-1

Expected Input 2:
valid_currency: [1,2,5,10,20,50,100,200,500]
Money: 556

Expected Output:
500-1
50-1
5-1

1-1

Expected Input 3:
valid_currency: [1,2,5,10,20,50,100,200,500,2000]
Money: 2000

Expected Output:
2000-1

Expected Input 4:
valid_currency: [1,2,5,10,20,50,100,500,1000]
Money: 210

Expected Output:
100-2
10-1

Expected Input 5:
valid_currency: [1,2,5,10,20,50,100,200,500,1000]
Money: 2000

Expected Output:
1000-2

Code:

```
def get_min_denominations(valid_currency, money):
    valid_currency.sort(reverse=True)  # Sort denominations in descending order
    result = []

    for denom in valid_currency:
        if money == 0:
            break
        if denom <= money:
            count = money // denom
            money -= count * denom
            result.append(f"{denom}-{count}")

    return result

# Example usage
valid_currency = list(map(int, input("Enter valid currencies (comma-separated): ").split(',')))
money = int(input("Enter the amount of money: "))

denominations = get_min_denominations(valid_currency, money)
for denomination in denominations:
    print(denomination)
```

Question 15:

There is a bus travelling from Town A to Town B. There are n stops between them and bus has to make m stops.
Find the numbery of ways in the travel so that no stop is consecutive

Expected Input 1:

n=12
m=4

Expected Output:

Output :126

Expected Input 2:

n = 16
s = 5

Expected Output:

792

Code:

```
import math

def count_non_consecutive_stops(n, m):
    if m > n:
        return 0
    # Calculate combinations C(n - m + 1, m)
    slots = n - m + 1
    ways = math.comb(slots, m)
    return ways

# Example usage
n = int(input("Enter the total number of stops (n): "))
m = int(input("Enter the number of stops to be made (m): "))
result = count_non_consecutive_stops(n, m)
print("Output:", result)
```

Question 16:

A gaming company wants to create an App with multiple games.
The instruction of the games is given. You are asked to write the code to prepare the games, Where inputs will be taken from users. Once the gaming algorithm is prepared then it can be associated with production interface of the App.

Game: Stone Paper Scissor Cut

Each win of a player will be counted as a one point for the player.
The game continues until any of the player scores 5.

Expected Input: Expected Output:

Player A	Player B	Result
Stone	Stone	DRAW
Stone	Paper	Player B wins
Stone	Scissor	Player A wins
Paper	Stone	Player A wins
Paper	Paper	DRAW
Paper	Scissor	Player B wins
Scissor	Scissor	DRAW
Scissor	Stone	Player B wins
Scissor	Paper	Player A wins

Code:

```
def determine_winner(player_a, player_b):
    """ Determine the result of a single round """
    if player_a == player_b:
        return "DRAW"
    elif (player_a == "Stone" and player_b == "Scissor") or \
         (player_a == "Paper" and player_b == "Stone") or \
         (player_a == "Scissor" and player_b == "Paper"):
        return "Player A wins"
    else:
        return "Player B wins"

def play_game():
    """ Play the Stone Paper Scissor Cut game until one player reaches 5 points """
    scores = {"Player A": 0, "Player B": 0}
    target_score = 5

    print("Enter moves for each round. The game ends when one player reaches 5 points.")

    while scores["Player A"] < target_score and scores["Player B"] < target_score:
        player_a = input("Player A move (Stone/Paper/Scissor): ").capitalize()
        player_b = input("Player B move (Stone/Paper/Scissor): ").capitalize()

        if player_a not in ["Stone", "Paper", "Scissor"] or player_b not in ["Stone", "Paper", "Scissor"]:
            print("Invalid input. Please enter Stone, Paper, or Scissor.")
            continue

        result = determine_winner(player_a, player_b)
        print(f"Result: {result}")

        if result == "Player A wins":
            scores["Player A"] += 1
        elif result == "Player B wins":
            scores["Player B"] += 1
```

```
        print(f"Scores - Player A: {scores['Player A']}, Player B: {scores['Player B']}")

    if scores["Player A"] == target_score:
        print("Player A wins the game!")
    elif scores["Player B"] == target_score:
        print("Player B wins the game!")

# Example usage
play_game()
```

Question 17:

Validate Email Address:

a. Check for '@' symbol, it should be only 1
b. Only lower-case letters are allowed
c. Numbers are allowed
e. No symbols allowed other than '.' & '_'

Code:

```
import re


def validate_email(email):
    # Regular expression pattern for validating the email address
    pattern = r'^[a-z0-9]+(?:_[a-z0-9]+)*@[a-z0-9]+(?:\.[a-z0-9]+)*$'

    # Check if the email matches the pattern
    if re.match(pattern, email):
        # Split the email to ensure there is only one '@'
        parts = email.split('@')
        if len(parts) == 2:
            return "Valid email address"
        else:
            return "Invalid email address"
    else:
        return "Invalid email address"
```

```
# Example usage

email = input("Enter the email address to validate: ")

result = validate_email(email)

print(result)
```

Question 18:

Solve the following patterns

a.) Input Description: row count

Expected Input:

4

Expected Output:

```
1
2 * 3
4 * 5 * 6
7 * 8 * 9 * 10
```

b.) Input Description: row count

Expected Input:

4

Expected Output:

```
   *
  * *
 * * *
* * * *
 * * *
  * *
   *
```

c.) Input Description: row count

Expected Input:

4

Expected Output:

```
1
2 * 3
4 * 5 * 6
7 * 8 * 9 * 10
4 * 5 * 6
2 * 3
1
```

d.) Program to print the pattern 'G'
Input Description : No of rows

Expected Input:
7
Expected Output:

```
 ***
*
*
* ***
*   *
*   *
 * * *
```

e.) Input Description: row count (only odd)
Expected Input:
5
Expected Output:

```
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
```

Code:

Pattern of Increasing Numbers in Rows

```
def print_number_pattern(row_count):
    current_number = 1
    for row in range(1, row_count + 1):
        line = []
        for _ in range(row):
            line.append(str(current_number))
            current_number += 1
        print(" * ".join(line))

# Example usage
row_count = int(input("Enter the number of rows: "))
print_number_pattern(row_count)
```

Diamond Pattern with Stars

```
def print_diamond_pattern(row_count):
    if row_count % 2 == 0:
        raise ValueError("Row count must be odd for diamond pattern.")

    middle = row_count // 2

    for i in range(middle + 1):
        print(" " * (middle - i) + "* " * (i + 1))
    for i in range(middle - 1, -1, -1):
        print(" " * (middle - i) + "* " * (i + 1))

# Example usage
row_count = int(input("Enter the number of rows (odd number): "))
print_diamond_pattern(row_count)
```

Pattern with Pyramid and Inverted Pyramid

```
def print_number_pyramid(row_count):
    current_number = 1
    # Print upper half
    for row in range(1, row_count + 1):
        line = []
        for _ in range(row):
            line.append(str(current_number))
            current_number += 1
        print(" * ".join(line))

    current_number -= row_count
    # Print lower half
    for row in range(row_count - 1, 0, -1):
        line = []
        for _ in range(row):
            line.append(str(current_number))
            current_number += 1
        print(" * ".join(line))
        current_number -= row

# Example usage
row_count = int(input("Enter the number of rows: "))
print_number_pyramid(row_count)
```

Pattern of Letter 'G'

```
def print_pattern_g(row_count):
    for i in range(row_count):
        if i == 0:
            print(' ' * (row_count - 3) + '*' * 3)
        elif i == row_count - 1:
            print(' ' * (row_count - 3) + '*' * 3)
        elif i == 1 or i == 2:
```

```
            print('*')
        elif i == 3:
            print('*' + '*' * (row_count - 2) + '*')
        else:
            print('*' + ' ' * (row_count - 2) + '*')

# Example usage
row_count = int(input("Enter the number of rows: "))
print_pattern_g(row_count)
```

Matrix with Borders

```
def print_matrix_with_borders(row_count):
    matrix = [[0] * row_count for _ in range(row_count)]

    # Fill borders with 1
    for i in range(row_count):
        matrix[0][i] = 1
        matrix[row_count - 1][i] = 1
        matrix[i][0] = 1
        matrix[i][row_count - 1] = 1

    # Print the matrix
    for row in matrix:
        print(' '.join(map(str, row)))

# Example usage
row_count = int(input("Enter the row count (only odd): "))
print_matrix_with_borders(row_count)
```

Question 19:

Cyclic rotation:

Case 1: first element moves to last and rest all the elements move one step to left
Case 2: last element moves to first and rest all the element move one step to right

Input 1 Description: 1 - first to last 2- last to first
Input 2 Description : string
Input 3 Description : no of times

Expected Input 1:

1
'happy'
2

Expected Output:

Appyh
ppyha

Expected Input 2:

2
'happy'
3

Expected Output:

yhapp
pyhap

Code:

```
def cyclic_rotation(direction, s, times):
    n = len(s)
    rotated_strings = []

    for _ in range(times):
        if direction == 1:  # Case 1: First element moves to last
            s = s[1:] + s[0]
        elif direction == 2:  # Case 2: Last element moves to first
            s = s[-1] + s[:-1]
        rotated_strings.append(s)

    return rotated_strings

# Example usage
direction = int(input("Enter 1 for first to last or 2 for last to first: "))
s = input("Enter the string: ")
times = int(input("Enter the number of times to rotate: "))

result = cyclic_rotation(direction, s, times)
for r in result:
    print(r)
```

Question 20:

In a pathology lab test, there are n number of samples for testing the health condition of a patient, each slide has 5 components, Sugar level, Blood pressure, Heartbeat rate, weight and fat percentage,
based on input as provided by the patient's blood report.

1. Create a sample input for a healthy patient as follows:

"Sugar level":15, "Blood pressure":32, "Heartbeat rate":71, "weight":65, "fat percentage":10.

2. Get values from the user and compare inputs with healthy patient data. If the patient data is not matching with the healthy
patient’s data, provide a warning.

3. Provide difference in readings to the patient.

Expected Output:

Sugar level:56

Blood pressure:120

Heartbeat rate:45

weight:67

fat percentage:67

{'Sugar level': -41, 'Blood pressure': -88, 'Heartbeat rate': 26, 'weight': -2, 'fat percentage': -57}

Sugar level -41

The sugar level is 41 less than the ideal value

Blood pressure -88

Blood pressure is 88 less than the ideal value

Heartbeat rate 26

Heartbeat rate is 26 more than the ideal value

weight -2

weight is 2 less than the ideal value

fat percentage 57

fat percentage is 57 less than the ideal value

Code:

```
def compare_patient_data(healthy_data, patient_data):
    differences = {}
    for key in healthy_data:
        difference = patient_data[key] - healthy_data[key]
        differences[key] = difference
        if difference != 0:
            direction = "more" if difference > 0 else "less"
            print(f"{key} {difference}")
            print(f"The {key.lower()} is {abs(difference)} {direction} than the ideal value\n")
    return differences

# Healthy patient data
healthy_data = {
    "Sugar level": 15,
    "Blood pressure": 32,
```

```
    "Heartbeat rate": 71,
    "weight": 65,
    "fat percentage": 10
}

# Get patient's test results from the user
patient_data = {}
patient_data["Sugar level"] = int(input("Enter Sugar level: "))
patient_data["Blood pressure"] = int(input("Enter Blood pressure: "))
patient_data["Heartbeat rate"] = int(input("Enter Heartbeat rate: "))
patient_data["weight"] = int(input("Enter weight: "))
patient_data["fat percentage"] = int(input("Enter fat percentage: "))

# Compare the data and print the differences
differences = compare_patient_data(healthy_data, patient_data)
```

Question 21:

Check whether the given number is Armstrong number or not
Armstrong number: 153 => 1^3+5^3+3^3=153 (If summing each digit to the power of number of digits results to the same number then it is a Armstrong number)

1634 => 1^4+6^4+3^4+4^4 =1634 1^4 => digit^(number of digits)

Expected Input:

1634

Expected Output:

Armstrong number

Question 22:

Convert Decimal to binary (Without inbuilt function)

Expected Input 1:

12

Expected Output:

1100

Expected Input 2:

20

Expected Output :

10100

Code:

```
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)

    if sum_of_powers == number:
```

```
        return True
    else:
        return False

# Example usage
number = int(input("Enter a number: "))
if is_armstrong(number):
    print(f"{number} is an Armstrong number")
else:
    print(f"{number} is not an Armstrong number")
```

Question 23:

Find whether the given number is perfect number or not.
Any number can be perfect number in Python, if the sum of its positive divisors excluding the number itself is equal to that number.

For example, 6 is a perfect number in Python because 6 is divisible by 1, 2, 3 and 6.
So, the sum of these values are: 1+2+3 = 6 (Remember, we have to exclude the number itself. That's why we haven't added 6 here). Some of the perfect numbers are 6, 28, 496, 8128 and 33550336 so on.

Expected Input:

28

Expected Output:

Perfect number

Code:

```
def is_perfect_number(number):
    # Calculate the sum of divisors
    sum_of_divisors = sum(i for i in range(1, number) if number % i == 0)

    # Check if the sum of divisors is equal to the number
    if sum_of_divisors == number:
        return True
    else:
        return False

# Example usage
number = int(input("Enter a number: "))
if is_perfect_number(number):
    print(f"{number} is a Perfect number")
else:
    print(f"{number} is not a Perfect number")
```