Ex. No. : 4.1 **Date:**

Register No.: **Name:**

Factors of a number

Determine the factors of a number (i.e., all positive integer values that evenly divide into a number).

```
def factors(n):
    return [i for i in range(1, n + 1) if n % i == 0]


number = int(input("Enter a number: "))
print(factors(number))
```

```
def factors(n):
    return [i for i in range(1, n + 1) if n % i == 0]


number = int(input("Enter a number: "))
print(factors(number))
```

Ex. No. : 4.2 **Date:**

Register No.: **Name:**

Non Repeated Digit Count

Write a program to find the count of non-repeated digits in a given number N. The number will be passed to the program as an input of type int.

Assumption: The input number will be a positive integer number >= 1 and <= 25000.

Some examples are as below.

If the given number is 292, the program should return 1 because there is only 1 non-repeated digit '9' in this number

If the given number is 1015, the program should return 2 because there are 2 non-repeated digits in this number, '0', and '5'.

If the given number is 108, the program should return 3 because there are 3 non-repeated digits in this number, '1', '0', and '8'.

If the given number is 22, the function should return 0 because there are NO non-repeated digits in this number.

```
def count_non_repeated_digits(n):
    digit_count = {}
    for digit in str(n):
        if digit in digit_count:
            digit_count[digit] += 1
        else:
            digit_count[digit] = 1
    non_repeated_count = sum(1 for count in digit_count.values() if count == 1)
    return non_repeated_count

number = int(input("Enter a number: "))
non_repeated_digits = count_non_repeated_digits(number)
print(f"The count of non-repeated digits in {number} is: {non_repeated_digits}")
```

Ex. No. : 4.3 **Date:**

Register No.: **Name:**

Prime Checking

Write a program that finds whether the given number N is Prime or not. If the number is prime, the program should return 2 else it must return 1.

Assumption: 2 <= N <=5000, where N is the given number.

```
def is_prime(n):
  if n <= 1:
    return 1
  if n == 2:
    return 2
  if n % 2 == 0:
    return 1
  for i in range(3, int(n**0.5) + 1, 2):
    if n % i == 0:
      return 1
  return 2

number = int(input("Enter a number: "))
result = is_prime(number)
print(result)
```

Input Format:

Integer input from stdin.

Output Format:

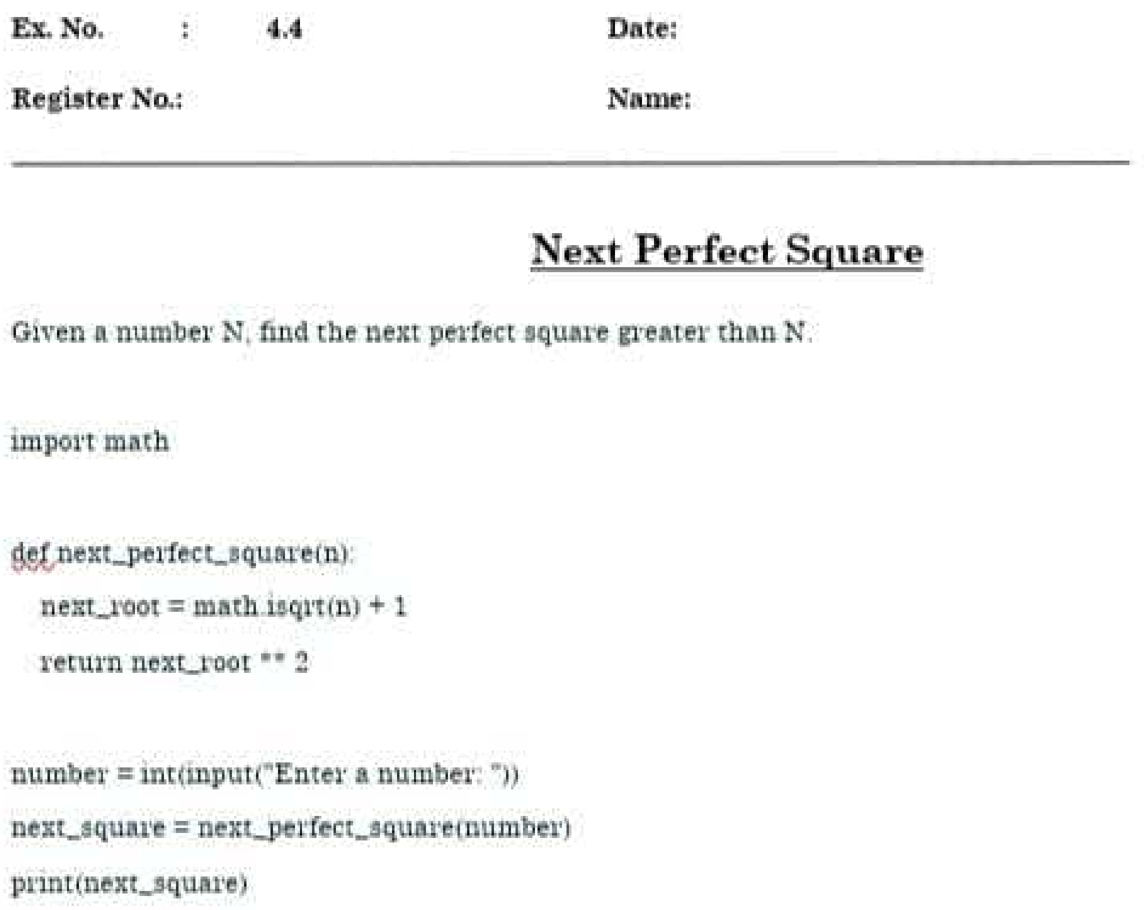
Perfect square greater than N.

Example Input:

10

Output:

16



Ex. No. : 4.4 **Date:**

Register No.: **Name:**

Next Perfect Square

Given a number N, find the next perfect square greater than N.

```
import math

def next_perfect_square(n):
   next_root = math.isqrt(n) + 1
   return next_root ** 2

number = int(input("Enter a number: "))
next_square = next_perfect_square(number)
print(next_square)
```

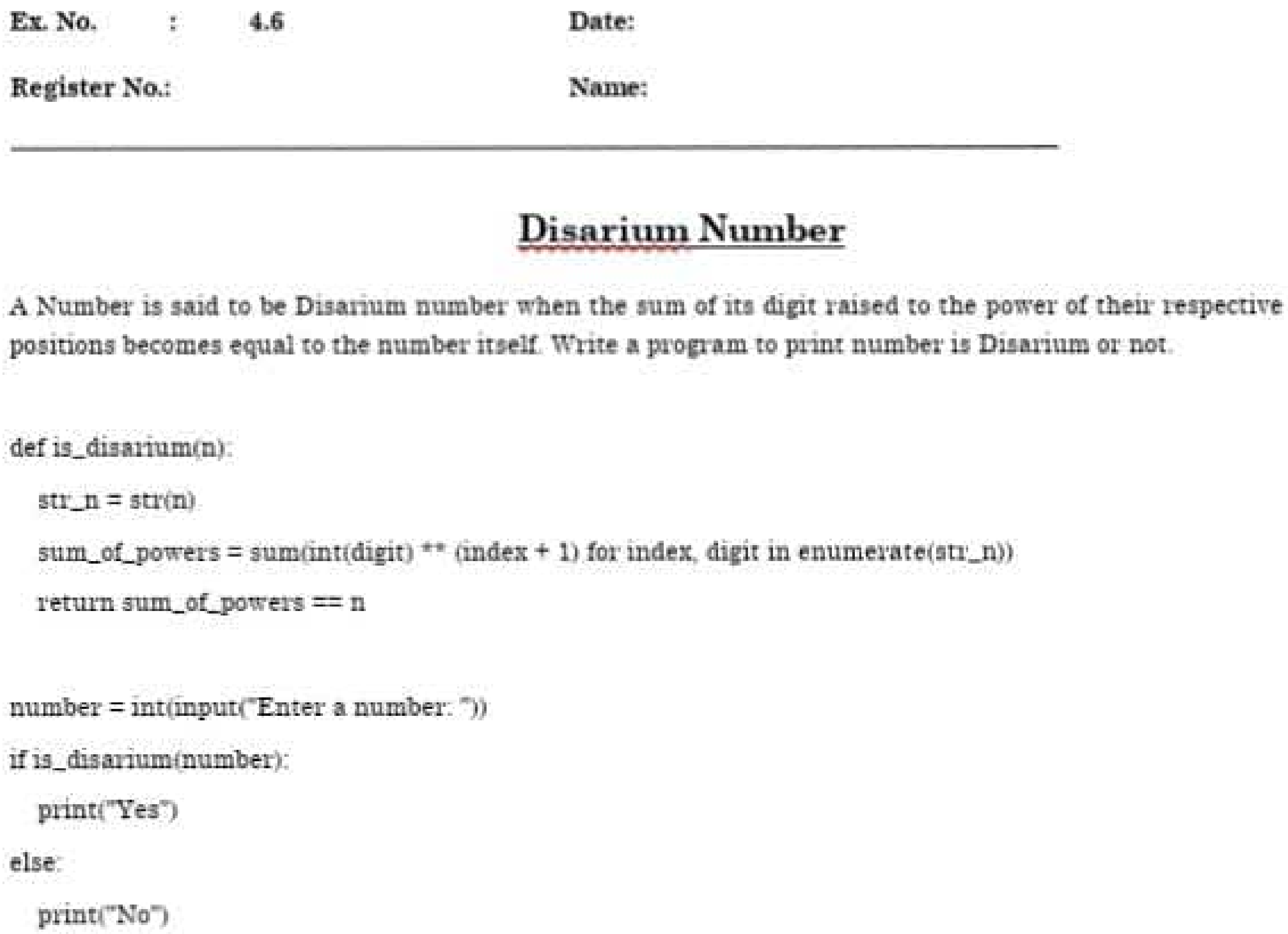
Ex. No. : 4.5 **Date:**

Register No.: **Name:**

Nth Fibonacci

Write a program to return the nth number in the fibonacci series. The value of N will be passed to the program as input.

```
def fibonacci(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    a, b = 0, 1
    for _ in range(2, n):
        a, b = b, a + b
    return b

n = int(input("Enter the position in the Fibonacci series: "))
nth_fibonacci = fibonacci(n)
print(nth_fibonacci)
```



Ex. No.   :   4.6   **Date:**

Register No.:   **Name:**

Disarium Number

A Number is said to be Disarium number when the sum of its digit raised to the power of their respective positions becomes equal to the number itself. Write a program to print number is Disarium or not.

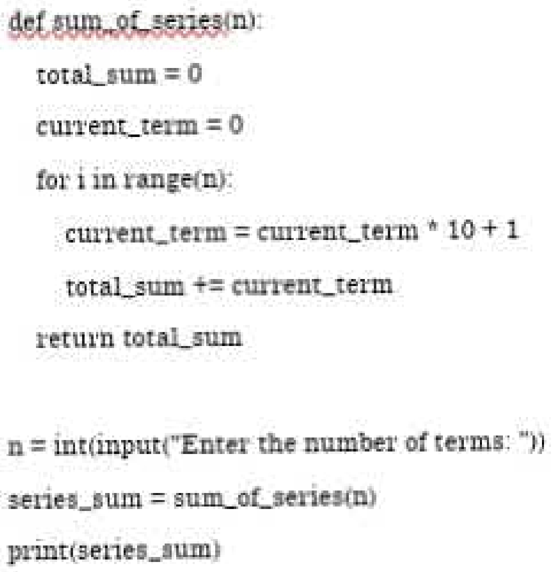
```
def is_disarium(n):
    str_n = str(n)
    sum_of_powers = sum(int(digit) ** (index + 1) for index, digit in enumerate(str_n))
    return sum_of_powers == n

number = int(input("Enter a number: "))
if is_disarium(number):
    print("Yes")
else:
    print("No")
```

Ex. No. : 4.7 **Date:**

Register No.: **Name:**

Sum of Series

Write a program to find the sum of the series 1 +11 + 111 + 1111 + . . . + n terms (n will be given as input from the user and sum will be the output)

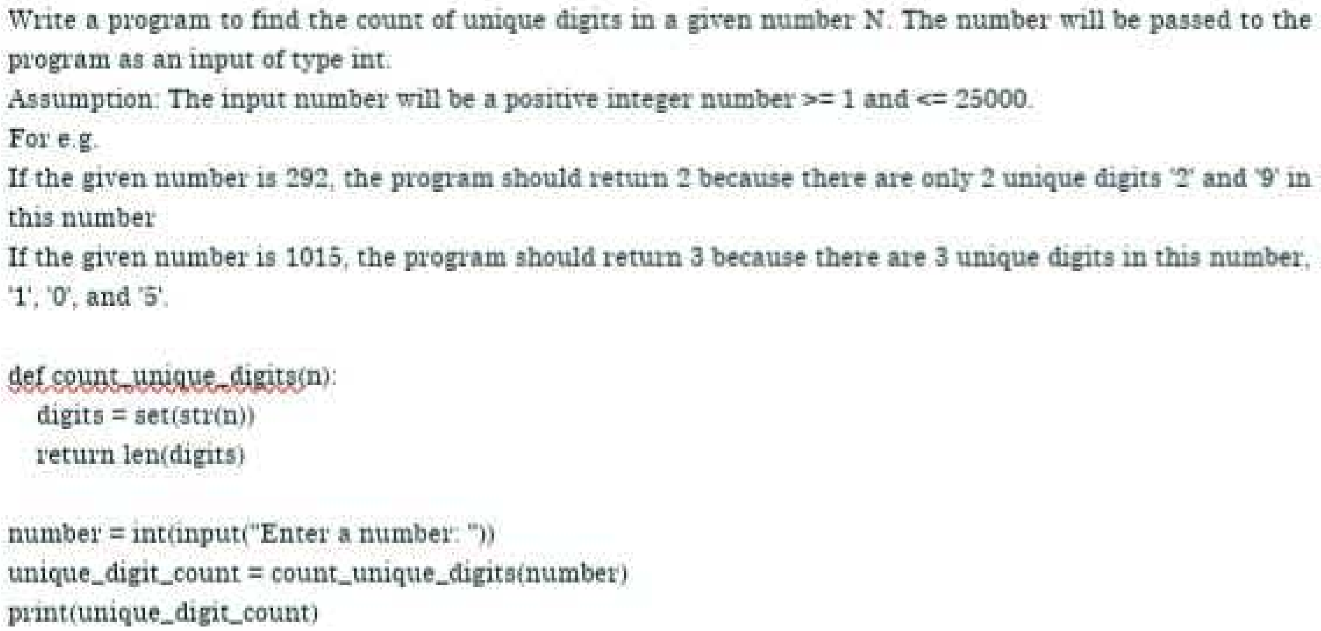


```
def sum_of_series(n):
  total_sum = 0
  current_term = 0
  for i in range(n):
    current_term = current_term * 10 + 1
    total_sum += current_term
  return total_sum

n = int(input("Enter the number of terms: "))
series_sum = sum_of_series(n)
print(series_sum)
```

Ex. No. : 4.8 **Date:**

Register No.: **Name:**

Unique Digit Count



Write a program to find the count of unique digits in a given number N. The number will be passed to the program as an input of type int.
Assumption: The input number will be a positive integer number >= 1 and <= 25000.
For e.g.
If the given number is 292, the program should return 2 because there are only 2 unique digits '2' and '9' in this number
If the given number is 1015, the program should return 3 because there are 3 unique digits in this number, '1', '0', and '5'.

```
def count_unique_digits(n):
   digits = set(str(n))
   return len(digits)

number = int(input("Enter a number: "))
unique_digit_count = count_unique_digits(number)
print(unique_digit_count)
```

Ex. No. : 4.9 **Date:**

Register No.: **Name:**

Product of single digit

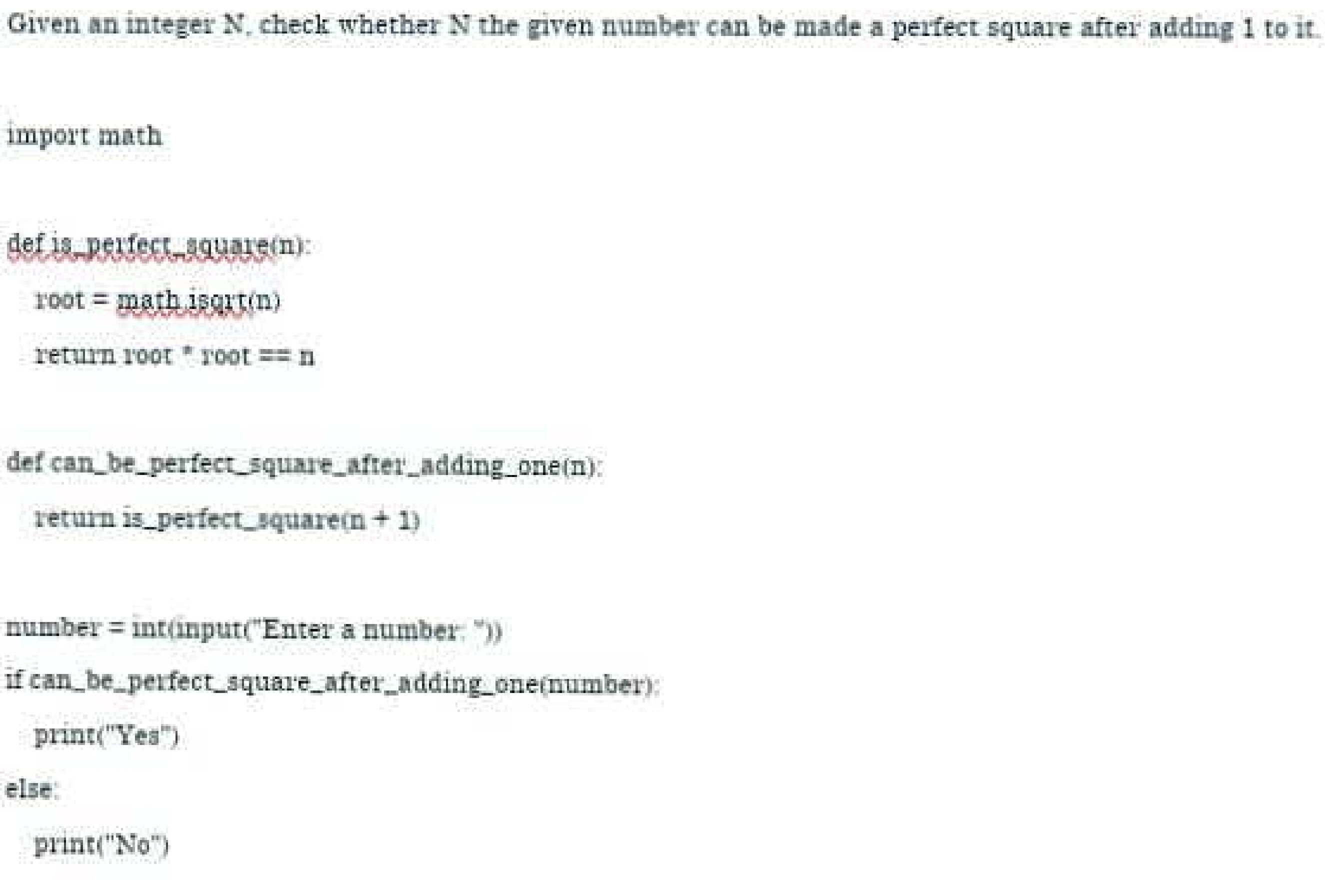
Given a positive integer N, check whether it can be represented as a product of single digit numbers.

```
def can_be_product_of_single_digits(n):
  if n < 10:
     return "Yes"
  for i in range(1, 10):
     for j in range(1, 10):
        if i * j == n:
           return "Yes"
  return "No"

number = int(input("Enter a number: "))
result = can_be_product_of_single_digits(number)
print(result)
```

Ex. No. : 4.10 **Date:**

Register No.: **Name:**

Perfect Square After adding One



Given an integer N, check whether N the given number can be made a perfect square after adding 1 to it.

```
import math

def is_perfect_square(n):
   root = math.isqrt(n)
   return root * root == n

def can_be_perfect_square_after_adding_one(n):
   return is_perfect_square(n + 1)

number = int(input("Enter a number: "))
if can_be_perfect_square_after_adding_one(number):
   print("Yes")
else:
   print("No")
```