Q.Given an integer n, return *true* if it is a power of two. Otherwise, return *false*.

An integer n is a power of two, if there exists an integer x such that n == 2x.

**Example 1:** Input: n = 1

Output: true

**Example 2:** Input: n = 16

Output: true

**Example 3:** Input: n = 3

Output: false

Ans: To determine if an integer `n` is a power of two, we can check if it satisfies the condition `n == 2^x` for some integer `x`. There are a couple of approaches to solve this problem.

One way is to use bit manipulation. In binary representation, a power of two will have only one bit set to `1`, with all other bits set to `0`. For example, `2^0 = 1` is represented as `0001`, `2^1 = 2` is represented as `0010`, `2^2 = 4` is represented as `0100`, and so on. We can use the bitwise AND operation to check if `n` has only one bit set to `1`. If that is the case, then `n` is a power of two.

Here's the implementation in Python:

```python

def isPowerOfTwo(n):

if n <= 0:

return False

return n & (n - 1) == 0

```

In this implementation, we first check if `n` is less than or equal to 0. If it is, then `n` cannot be a power of two, so we return `False`. Otherwise, we perform the bitwise AND operation between `n` and `n - 1`. If the result is 0, it means that `n` has only one bit set to `1`, and hence, `n` is a power of two. Otherwise, it is not a power of two, so we return `False`.

Let's test the function with the provided examples:

```python

print(isPowerOfTwo(1)) # Output: True

print(isPowerOfTwo(16)) # Output: True

print(isPowerOfTwo(3)) # Output: False

```

The function produces the expected output for the given examples.

Q. Given a number n, find the sum of the first natural numbers.

**Example 1:**

Input: n = 3

Output: 6

**Example 2:**

Input : 5

Output : 15

Ans: To find the sum of the first `n` natural numbers, we can use the formula for the sum of an arithmetic series: `sum = (n \* (n + 1)) / 2`.

Here's the implementation in Python:

```python

def sumOfFirstNNumbers(n):

return (n \* (n + 1)) // 2

```

In this implementation, we calculate the sum using the formula `(n \* (n + 1)) / 2`. The use of integer division `//` ensures that the result is an integer.

Let's test the function with the provided examples:

```python

print(sumOfFirstNNumbers(3)) # Output: 6

print(sumOfFirstNNumbers(5)) # Output: 15

```

The function produces the expected output for the given examples.

Q. Given a positive integer, N. Find the factorial of N.

**Example 1:**

Input: N = 5

Output: 120

**Example 2:**

Input: N = 4

Output: 24

Ans: To find the factorial of a positive integer `N`, we can use a simple iterative approach.

Here's the implementation in Python:

```python

def factorial(N):

result = 1

for i in range(1, N + 1):

result \*= i

return result

```

In this implementation, we initialize the variable `result` to 1. Then, we iterate from 1 to `N` (inclusive) using a for loop. In each iteration, we multiply `result` by the current value of `i`. Finally, we return the computed `result`, which represents the factorial of `N`.

Let's test the function with the provided examples:

```python

print(factorial(5)) # Output: 120

print(factorial(4)) # Output: 24

```

The function produces the expected output for the given examples.

Q. Given a number N and a power P, the task is to find the exponent of this number raised to the given power, i.e. N^P.

**Example 1 :**

Input: N = 5, P = 2

Output: 25

**Example 2 :** Input: N = 2, P = 5

Output: 32

Ans: To calculate the exponent of a number `N` raised to the power `P`, we can use the `\*\*` operator in Python.

Here's the implementation:

```python

def power(N, P):

return N \*\* P

```

In this implementation, we use the `\*\*` operator to raise `N` to the power `P`. The result is then returned.

Let's test the function with the provided examples:

```python

print(power(5, 2)) # Output: 25

print(power(2, 5)) # Output: 32

```

The function produces the expected output for the given examples.

Q. Given an array of integers **arr**, the task is to find maximum element of that array using recursion.

**Example 1:**

Input: arr = {1, 4, 3, -5, -4, 8, 6}; Output: 8

**Example 2:**

Input: arr = {1, 4, 45, 6, 10, -8}; Output: 45

Ans: To find the maximum element of an array using recursion, we can use a divide-and-conquer approach. The idea is to divide the array into two halves, find the maximum element in each half recursively, and then compare the maximums to determine the overall maximum.

Here's the implementation in Python:

```python

def findMax(arr, start, end):

if start == end:

return arr[start]

mid = (start + end) // 2

max1 = findMax(arr, start, mid)

max2 = findMax(arr, mid + 1, end)

return max(max1, max2)

def getMaxElement(arr):

return findMax(arr, 0, len(arr) - 1)

```

In this implementation, the `findMax` function takes the array `arr`, the starting index `start`, and the ending index `end` as parameters. It uses recursion to divide the array into halves until the base case is reached, which is when `start` is equal to `end`. In the base case, the function simply returns the element at index `start`.

In the recursive case, the function calculates the midpoint index `mid` by taking the average of `start` and `end`. Then, it recursively calls `findMax` for the left half of the array (from `start` to `mid`) and the right half of the array (from `mid + 1` to `end`). The maximum values returned from the two recursive calls are compared using the `max` function, and the larger value is returned as the maximum element of the current subarray.

The `getMaxElement` function is a helper function that calls `findMax` with the initial `start` and `end` values of the entire array.

Let's test the function with the provided examples:

```python

arr1 = [1, 4, 3, -5, -4, 8, 6]

print(getMaxElement(arr1)) # Output: 8

arr2 = [1, 4, 45, 6, 10, -8]

print(getMaxElement(arr2)) # Output: 45

```

The function produces the expected output for the given examples.

Q. Given first term (a), common difference (d) and a integer N of the Arithmetic Progression series, the task is to find Nth term of the series.

**Example 1:**

Input : a = 2 d = 1 N = 5 Output : 6 The 5th term of the series is : 6

**Example 2:**

Input : a = 5 d = 2 N = 10 Output : 23 The 10th term of the series is : 23

Ans: To find the Nth term of an arithmetic progression (AP) series given the first term (a), common difference (d), and the value of N, we can use the formula:

Nth term = a + (N - 1) \* d

Here's the implementation in Python:

```python

def findNthTerm(a, d, N):

return a + (N - 1) \* d

```

In this implementation, we calculate the Nth term using the formula mentioned above. We add the product of `(N - 1)` and `d` to the first term `a` to obtain the Nth term.

Let's test the function with the provided examples:

```python

print(findNthTerm(2, 1, 5)) # Output: 6

print(findNthTerm(5, 2, 10)) # Output: 23

```

The function produces the expected output for the given examples.

Q. Given a string S, the task is to write a program to print all permutations of a given string.

**Example 1:**

***Input:***

S = “ABC”

***Output:***

“ABC”, “ACB”, “BAC”, “BCA”, “CBA”, “CAB”

**Example 2:**

***Input:***

S = “XY”

***Output:***

“XY”, “YX”

Ans: To generate all permutations of a given string, you can use a backtracking algorithm. Here's a Python program that solves this task:

```python

def permute(s):

# Convert the string to a list of characters

char\_list = list(s)

# Helper function to generate permutations

def backtrack(start):

# If we have reached the end of the string, print the permutation

if start == len(char\_list) - 1:

print(''.join(char\_list))

return

# Iterate over the remaining characters and swap each with the start character

for i in range(start, len(char\_list)):

# Swap the characters

char\_list[start], char\_list[i] = char\_list[i], char\_list[start]

# Recursively generate permutations for the remaining characters

backtrack(start + 1)

# Restore the original order by swapping back

char\_list[start], char\_list[i] = char\_list[i], char\_list[start]

# Start the backtracking algorithm

backtrack(0)

# Test the function

s = "ABC"

print("Permutations of", s)

permute(s)

s = "XY"

print("Permutations of", s)

permute(s)

```

Output:

```

Permutations of ABC

ABC

ACB

BAC

BCA

CBA

CAB

Permutations of XY

XY

YX

```

The `permute` function takes a string `s` as input and converts it into a list of characters `char\_list`. It defines a helper function called `backtrack` that performs the actual backtracking algorithm.

The `backtrack` function takes a parameter `start` that represents the starting index of the current permutation. If `start` is equal to the last index of the list (`len(char\_list) - 1`), it means we have reached the end of the string and found a permutation, so we print it.

Otherwise, the function iterates over the remaining characters in the string (starting from `start`) and swaps each character with the character at the starting index. This generates a new permutation. Then, it recursively calls itself with `start + 1` to generate permutations for the remaining characters.

After generating the permutations for the remaining characters, the function restores the original order by swapping the characters back. This is important to ensure that all permutations are generated correctly.

Finally, in the main part of the program, we test the `permute` function with the given examples and print the permutations.

Q. Given an array, find a product of all array elements.

**Example 1:**

Input : arr[] = {1, 2, 3, 4, 5} Output : 120 **Example 2:**

Input : arr[] = {1, 6, 3} Output : 18

Ans: To find the product of all elements in an array, you can iterate over the elements and multiply them together. Here's a Python program that solves this task:

```python

def product\_of\_array(arr):

product = 1

for num in arr:

product \*= num

return product

# Test the function

arr1 = [1, 2, 3, 4, 5]

print("Product of array:", product\_of\_array(arr1))

arr2 = [1, 6, 3]

print("Product of array:", product\_of\_array(arr2))

```

Output:

```

Product of array: 120

Product of array: 18

```

The `product\_of\_array` function takes an array `arr` as input. It initializes a variable `product` to 1, which will store the final product of the elements.

The function then iterates over each element in the array using a for loop. For each element, it multiplies it with the current value of `product`. This way, it accumulates the product of all elements in the array.

Finally, the function returns the computed product.

In the main part of the program, we test the `product\_of\_array` function with the given examples and print the results.