105. Convert Binary Number to Decimal

The **`binary_to_decimal`** function converts a binary number to decimal.

```
def binary_to_decimal(binary):
    return sum(int(bit) * 2**index for index, bit in
enumerate(reversed(binary)))

# Example usage
print(binary_to_decimal("1101"))
# Output: 13
```

106. Check if an Array is Sorted in Descending Order

The **sorted_descending** function checks if an array is sorted in descending order.

```
def sorted_descending(arr):
    return all(arr[i] >= arr[i + 1] for i in range(len(arr) - 1))

# Example usage
print(sorted_descending([5, 4, 3, 2, 1]))
# Output: True
print(sorted_descending([1, 5, 3, 8, 2]))
# Output: False
```

107. Find the Average of Even Numbers in an Array

The **average_of_even_numbers** function finds the average of even numbers in an array.

```
def average_of_even_numbers(arr):
    even_numbers = [num for num in arr if num % 2 == 0]
    return sum(even_numbers) / len(even_numbers) if even_numbers else 0

# Example usage
print(average_of_even_numbers([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
# Output: 6.0
```

108. Capitalize the First Letter of Each Word in a String

The **`capitalize_words`** capitalizes the first letter of each word in a String.

```
def capitalize_words(string):
    return string.title()

# Example usage
print(capitalize_words("hello world"))
# Output: "Hello World"
```

109. Check if an Array is a Subset of Another Array

The **`is_subset`** function checks if an array is a subset of another array.

```
def is_subset(arr1, arr2):
    return all(item in arr2 for item in arr1)

# Example usage
print(is_subset([1, 2, 3], [2, 3, 4, 5, 6]))
# Output: False
print(is_subset([1, 2, 3], [2, 3, 1, 5, 6]))
# Output: True
```

110. Find the Minimum and Maximum Numbers in an Array

The **`min_max`** function finds the minimum and maximum numbers in an array.

```
def min_max(arr):
    return min(arr), max(arr)

# Example usage
result = min_max([10, 5, 25, 3, 15])
print(f"{{ min: {result[0]}, max: {result[1]} }}")
# Output: { min: 3, max: 25 }
```

111. Validate Zip Code

The **`is_valid`** function validates whether a given string is a valid zip code.

```
import re

def is_valid(zip_code):
    return bool(re.match(r'^\d{5}$', zip_code))

# Example usage
print(is_valid("12345"))
# Output: True
print(is_valid("1234"))
# Output: False
print(is_valid("123456"))
# Output: False
print(is_valid("12 45"))
# Output: False
```

112. Remove Null Values from a List.

The **`remove_null`** function removes null values from a list.

```
def remove_null(arr):
    return [item for item in arr if item is not None]

# Example usage
array = [1, None, 2, 3, None, 4, None]
result = remove_null(array)
print(f"Output: {result}")
# Output: [1, 2, 3, 4]
```

113. Maurice's Racing Snails

Maurice and Steve engage in a snail race, each owning three snails of different speeds: slow (s), medium (m), and fast (f). Although Maurice's snails are generally faster, Steve's strategy poses a challenge. In each of the three rounds, they pit their snails against each other strategically. Maurice's plan involves sacrificing his slower snails strategically to ensure victory. This function evaluates Maurice's plan, determining if he wins at least 2 out of 3 games against Steve's snails.

```
def maurice_wins(m_snails, s_snails):
    wins = sum(m > s for m, s in zip(m_snails, s_snails))
    return wins >= 2

# Example usage:
print(maurice_wins([1, 2, 3], [3, 2, 1]))
# Output: True
```

114. Calculate the Sum of Cubes of an Array

The **sum_of_cubes** function calculates the sum of cubes of an array.

```
def sum_of_cubes(arr):
    return sum(x ** 3 for x in arr)

# Example usage:
arr = [1, 2, 3, 4, 5]
sum_result = sum_of_cubes(arr)
print(f"Output: {sum_result}")
# Output: 225
```

115. Shuffle the Characters of a String

The **`shuffle_string`** function suffle the characters of a String.

```
import random

def shuffle_string(s):
    shuffled_chars = list(s)
    random.shuffle(shuffled_chars)
    return ''.join(shuffled_chars)

# Example usage:
s = "hello"
shuffled_string = shuffle_string(s)
print(f"Output: {shuffled_string}")
# Output: oelhl
```

116. Find the Nth Fibonacci Number (recursive)

The **`fibonacci`** function finds the nth fibonacci number (recursive).

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)

# Example usage:
n = 7
result = fibonacci(n)
print(f"Output: {result}")
# Output: 13
```

117. Symmetry Checker

The **`is_symmetrical`** function determines whether a given integer is symmetrical, meaning it reads the same backward as forward.

```
def is_symmetrical(num):
    return str(num) == str(num)[::-1]

num = 12321
print(is_symmetrical(num))
# Output: True
```

118. Maximum Triangle Edge Calculator

The **next_edge** function provides a concise function for determining the maximum possible length of the third edge of a triangle, given the lengths of the other two sides as integers.

```
def next_edge(side1, side2):
    return side1 + side2 - 1

side1 = 5
side2 = 7
print(next_edge(side1, side2))
# Output: 11
```

119. Calculate the Perimeter of a Rectangle

The **`rectangle_perimeter`** function calculates the perimeter of a rectangle given its width and height.

```
def rectangle_perimeter(width, height): return 2 * (width + height)

print(rectangle_perimeter(5, 10))
# Output: 30
```

120. Find the Longest Common Prefix in an Array of Strings

The **`longest_common_prefix`** function finds the longest common prefix among an array of strings.

```
def longest_common_prefix(strs): return "" if not strs else "".join(c for i, c
in enumerate(strs[0]) if all(s[i] == c for s in strs))

# Test
strs = ["apple", "apricot", "appetizer"]
print(longest_common_prefix(strs))
# Output: "ap"
```

121. Greeting Function with Conditional Message

The **say_hello_bye** function introduces a one-liner function that takes a string name and a number (either 0 or 1) as input. Depending on the value of the number, it either greets the person with "Hello" or bids farewell with "Bye", while ensuring the name's first letter is capitalized. The function utilizes a ternary conditional operator for succinctness and clarity.

```
def say_hello_bye(name, num): return f"Hello {name.capitalize()}" if num == 1
else f"Bye {name.capitalize()}"

name = "Alice"
num = 2
print(say_hello_bye(name, num))
# Output: Bye Alice
```

122. Find the First Non-Repeated Character in a String

The **`first_non_repeated_char`** function finds the first non-repeated character in a String.

```
def first_non_repeated_char(s): return next((c for c in s if s.count(c) == 1), None)

s = "abacabad"
print(first_non_repeated_char(s))
# Output: c
```

123. One-Liner Bitwise Operations in Python

These **bitwise_and**, **bitwise_or**, and **bitwise_xor** functions work similarly to the lambda expressions but use the def keyword for function definition. They take two integers n1 and n2 as input and return the result of the respective bitwise AND, OR, and XOR operations between them.

```
def bitwise_and(n1, n2):
    return n1 & n2

def bitwise_or(n1, n2):
    return n1 | n2

def bitwise_xor(n1, n2):
    return n1 ^ n2

num1, num2 = 5, 3

print(bitwise_and(num1, num2))  # Output: 1
print(bitwise_or(num1, num2))   # Output: 7
print(bitwise_xor(num1, num2))  # Output: 6
```

124. Calculate the Exponential of a Number

The **`exponential`** function calculates the result of raising a given base to a specified exponent using the exponentiation operator (**).

```
def exponential(base, exponent):
    return base ** exponent

result = exponential(2, 3)
print(result)
# Output: 8
```

125. Check if a String is an Anagram of Another String

The **`is_anagram`** function checks if a string is an anagram of another string.

```
def is_anagram(str1, str2):
    return sorted(str1) == sorted(str2)

print(is_anagram("listen", "silent"))
# Output: True
print(is_anagram("hello", "world"))
# Output: False
```

126. Compact Phone Number Formatter

The **`format_phone_number`** function offers a concise function to format an array of 10 numbers into a phone number string in the standard (XXX) XXX-XXXX format.

```
def format_phone_number(numbers):
    return "({}{}{}) {}{}{}-{}{}{}{}".format(*numbers)

numbers = [5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
print(format_phone_number(numbers))
# Output: (555) 555-5555
```

127. Check if a Number is a Neon Number

The **`is_neon_number`** function checks if a number is a neon number.

```
def is_neon_number(num):
    sum_of_digits = 0
    square = num * num

    while square:
        sum_of_digits += square % 10
        square //= 10

    return sum_of_digits == num

print(is_neon_number(9))
# Output: True
```

128. Recursive Right Shift Mimicker

The **shift_to_right** function employs recursion to repeatedly divide the first integer by 2^y times, emulating the right shift operation. This approach offers a compact and elegant solution for performing right shifts between two given integers.

```
def shift_to_right(x, y):
    return x if y < 1 else shift_to_right(x // 2, y - 1)

x = 16
y = 2
print(shift_to_right(x, y))
# Output: 4
```

129. Check if a Number is a Disarium Number

The **is_disarium_number** function checks if a number is a disarium number.

```
def is_disarium_number(num):
    num_str = str(num)
    disarium_sum = sum(int(digit) ** (i + 1) for i, digit in
enumerate(num_str))
    return disarium_sum == num

# Test cases
print(is_disarium_number(89))
# Output: True
print(is_disarium_number(135))
# Output: True
print(is_disarium_number(23))
# Output: False
```

130. Remove Vowels from a String

The **`remove_vowels`** function removes vowels from a string.

```
import re

def remove_vowels(string):
    return re.sub(r'[aeiouAEIOU]', '', string)

# Test case
print(remove_vowels("Hello, World!"))
# Output: "Hll, Wrld!"
```

131. Generate an Array of Consecutive Numbers

The **`consecutive_numbers`** function generates an array of consecutive numbers.

```
def consecutive_numbers(start, end_num):
    return list(range(start, end_num + 1))

# Test case
result = consecutive_numbers(1, 5)
print(", ".join(map(str, result)))
# Output: 1, 2, 3, 4, 5
```

132. Check if a Number is a Pronic Number

The **`is_pronic_number`** function checks if a number is a pronic number.

```
def is_pronic_number(num):
    n = int(num ** 0.5)
    return n * (n + 1) == num

# Test cases
print(is_pronic_number(6))
# Output: True
print(is_pronic_number(20))
# Output: True
print(is_pronic_number(7))
# Output: False
```

133. Check if a String is a Pangram

The **`is_pangram`** function checks if a string is a pangram.

```
def is_pangram(s):
    letters = set(c.lower() for c in s if c.isalpha())
    return len(letters) == 26

# Test cases
print(is_pangram("The quick brown fox jumps over the lazy dog"))
# Output: True
print(is_pangram("Hello, World!"))
# Output: False
```

134. Reverse the Order of Words in a Sentence

The **reverse_sentence** function reverses the order of words in a sentence.

```
def reverse_sentence(sentence):
    words = sentence.split()
    reversed_sentence = ' '.join(reversed(words))
    return reversed_sentence

# Test case
print(reverse_sentence("Hello, how are you doing?"))
# Output: "doing? you are how Hello,"
```

135. Calculate the Hypotenuse of a Right-Angled Triangle

The **`calculate_hypotenuse`** function calculates the hypotenuse of a right-angled triangle.

```
import math

def calculate_hypotenuse(a, b):
    return math.sqrt(a ** 2 + b ** 2)

# Test case
print(calculate_hypotenuse(3, 4))
# Output: 5.0
```