Transform your code into powerful solutions.

250+ KILLER PYTHON ONE-LINERS

```
duplicate = lambda x: x * 2
```

by Hernando Abella

250+ Killer Python One-Liners

Transform your code into powerful solutions.

duplicate = lambda x: x * 2

By Hernando Abella

Aluna Publishing House is united by our shared passion for education, languages, and technology. Our mission is to provide the ultimate learning experience when it comes to books. We believe that books are not just words on paper; they are gateways to knowledge, imagination, and enlightenment.

Through our collective expertise, we aim to bridge the gap between traditional learning and the digital age, harnessing the power of technology to make books more accessible, interactive, and enjoyable. We are dedicated to creating a platform that fosters a love for reading, language, and lifelong learning. Join us on our journey as we embark on a quest to redefine the way you experience books.

Let's unlock the limitless potential of knowledge, one page at a time.

TABLE OF CONTENTS

264. Calculate the Area of a Sector

This book was written from the ground up to help you master the complexities of this beautiful language. Inside you'll discover a collection of powerful, concise code snippets that will improve the way you write and think.

Together we will deepen the art of creating code to solve complex problems in the most fun, elegant and efficient way!

1. CONVERT CELSIUS TO FAHRENHEIT

The **celsius_to_fahrenheit** function converts Celsius to Fahrenheit.

```
def celsius_to_fahrenheit(celsius): return (celsius * 9/5) + 32

print(celsius_to_fahrenheit(25))

# Output: 77.0°F (25°C converted to Fahrenheit)
```

2. SWAP TWO VARIABLES

The **swap_without_temp** function swaps the values of two variables **a** and **b** without using a temporary variable.

```
def swap_without_temp(a, b):
return b, a
result = swap_without_temp(5, 10)
print(f"After swapping: a = {result[1]}, b = {result[0]}")
# Output: After swapping: a = 10, b = 5
```

3. CONVERT RGB TO HEX

The **rgb_to_hex** function combines the red, green, and blue (RGB) values into a single hexadecimal color code.

```
    def rgb_to_hex(r, g, b): return f"#{((r << 16) + (g << 8) + b):06X}"

    print(rgb_to_hex(0, 51, 255))

    # Output: #0033FF
```

4. TRANSPOSE OF A MATRIX

The **transpose_matrix** function computes the transpose of a given matrix.

```
    def transpose_matrix(matrix):
    return [list(row) for row in zip(*matrix)]
    matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
    ]
    print("\n".join(", ".join(map(str, row)) for row in
transpose_matrix(matrix)))
    # Output:
    # 1, 4, 7
    # 2, 5, 8
    # 3, 6, 9
```

5. CHECK IF DATE IS VALID

The **is_date_valid** function checks if a date is valid.

```
from datetime import datetime
def is_date_valid(val): return datetime.strptime(val, "%B %d, %Y %H:%M:%S") if val else False
print(is_date_valid("December 17, 1995 03:24:00"))
# 1995-12-17 03:24:00
```

6. FIND THE DAY OF YEAR

The **day_of_year** function takes a date as input and calculates the day of the year for that date.

```
from datetime import datetime
def day_of_year(date): return date.timetuple().tm_yday
print(day_of_year(datetime(2024, 10, 1)))
# Output: 275
```

7. CAPITALIZE A STRING

The **capitalize_string** function takes a string as input and capitalizes the first character.

```
    def capitalize_string(s): return s[0].upper() + s[1:] if s else
""

    print(capitalize_string("follow for more"))

    # Output: Follow for more
```

8. FIND THE NUMBER OF DAYS BETWEEN TWO DAYS

The **day_diff** function finds the number of days between two days.

```
from datetime import datetime
def day_diff(date1, date2): return abs((date2 - date1).days) + 1
date1 = datetime(2020, 10, 21)
date2 = datetime(2021, 10, 22)
print(day_diff(date1, date2))
# Output: 367
```

9. FIND THE FRECUENCY OF CHARACTER IN A STRING

The **character_frequency** function finds the frequency of characters in a String.

```
    def character_frequency(string): return {char: string.count(char) for char in set(string)}

    print(", ".join([f"{k}=>{v}" for k, v in character_frequency("hello").items()]))

    # Output: h=>1, e=>1, l=>2, o=>1
```

10. GENERATE RANDOM HEX

The **random_hex** function generates a random integer between 0 and 16777215 (0xffffff in hexadecimal).

```
import random
def random_hex(): return f"#{random.randint(0, 0xFFFFFF):06X}"
print(random_hex())
# Output: e.g., #A58E72
```

11. CREATE RANDOM STRINGS

The **random_string** function generates a random String of lowercase letters with the specified length.

```
import random
import string
def random_string(length): return ''.join(random.choice(string.ascii_lowercase) for _ in range(length))
print(random_string(10))
# Output: e.g., bjmfpwmoqi
```

12. FIND THE ODD OCCURRENCE

The **find_odd** function finds the integer which appears an odd number of times in a given list of integers.

```
    import random
    import functools
    def find_odd(ar):
    return functools.reduce(lambda x, y: x ^ y, ar)
    ar = [random.randint(1, 100) for _ in
range(random.randint(10, 20))]
    print("Input:", " ".join(map(str, ar)))
    print("Result:", find_odd(ar))
    # Input: 25 32 37 63 66 6 4 26 2 38 59 1 30
    # Result: 121
```

13. CHECK IF A NUMBER IS EVEN OR ODD

The **is_even** function checks whether a given number is even or odd.

```
def is_even(num): return num % 2 == 0
print(is_even(2))
# Output: True
```

14. SIMPLE SUM

The **addition** function takes two numbers as input and returns their sum.

```
def addition(a, b): return a + b
result = addition(5, 3)
print(result)
# Output: 8
```

15. PYRAMID PATTERN

The **create_pyramid** function takes the number of rows as an argument and returns an array representing the pyramid pattern.

```
    def create_pyramid(rows):
    return [f"{' ' * (rows - i)}{'*' * (2 * i - 1)}" for i in range(1, rows + 1)]
    pyramid = create_pyramid(5)
    for row in pyramid:
    print(row)
    """
```

```
                    *
                   ***
                  *****
                 *******
                *********
"""
```

16. REVERSE A STRING

The **reverse** function takes a string as input and returns a new string with the characters in reverse order.

```
def reverse(string): return string[::-1]
print(reverse("hello world"))
# Output: dlrow olleh
```

17. CHECK IF ARRAY IS EMPTY

The **is_not_empty** function checks if an array is not empty.

```
def is_not_empty(arr): return isinstance(arr, list) and len(arr) > 0

print(is_not_empty([1, 2, 3]))

# Output: True
```

18. MATCHSTICK COUNT IN STEPS

The **match_houses** function calculates the total number of matchsticks required based on the number of steps provided.

```
    def match_houses(steps): return steps * 6 - (steps - 1) if steps > 0 else 0
    print(match_houses(1))
    # Output: 6
    print(match_houses(4))
    # Output: 21
    print(match_houses(0))
    # Output: 0
```

19. SHUFFLE AN ARRAY

The **shuffle_array** function randomly shuffles the elements of a given array.

```
import random
def shuffle_array(arr):
shuffled = arr[:]
random.shuffle(shuffled)
return shuffled
print(", ".join(map(str, shuffle_array([1, 2, 3, 4]))))
# Output: e.g., 4, 2, 3, 1
```

20. VALIDATE VOWEL SANDWICH

The **is_vowel_sandwich** function validates whether a 3-character string is a vowel sandwich.

def is_vowel_sandwich(string): return len(string) == 3 and string[0] not in 'aeiou' and string[1] in 'aeiou' and string[2] not in 'aeiou'

print(is_vowel_sandwich("bat"))

Output: False

print(is_vowel_sandwich("cat"))

Output: True

print(is_vowel_sandwich("dog"))

Output: False

21. COUNT TRUE VALUES IN BOOLEAN ARRAY

The **count_true** function counts the number of **true** values in a boolean array.

```
def count_true(lst): return sum(1 for x in lst if x)
bool_array = [True, False, True, False, True]
count = count_true(bool_array)
print("Number of true values:", count)
# Output: Number of true values: 3
```

22. GET THE LENGTH OF A STRING

The **get_length** function returns the number of characters in the given string.

```
def get_length(string): return len(string)

print(get_length("Hello, world!"))

# Output: 13
```

23. CALCULATE THE AREA OF A CIRCLE

The **calculate_circle_area** function calculates the area of a circle given its radius.

```
import math
def calculate_circle_area(radius): return math.pi * radius ** 2
print(calculate_circle_area(5))
# Output: 78.53981633974483
```

24. MOVE CAPITAL LETTERS TO FRONT

The **cap_to_front** function moves all capital letters in a word to the front of the word, preserving their original order, and followed by the lowercase letters.

```
    def cap_to_front(s): return ''.join(filter(str.isupper, s)) + ''.join(filter(str.islower, s))
    print(cap_to_front("MoveCapitalLettersToFront"))
    # Output: MCLFoveapitalettersoront
```

25. CHECK IF AN ARRAY IS SPECIAL

The **is_special_array** function determines whether an array is special, where every even index contains an even number and every odd index contains an odd number.

```
def is_special_array(arr): return all(arr[i] % 2 == i % 2 for i in range(len(arr)))

arr = [2, 3, 6, 8, 9]

print(is_special_array(arr))

# Output: False
```

26. VALIDATE NUMBER WITHIN BOUNDS

The **int_within_bounds** function validates whether a number `n` is exclusively within the bounds of **lower** and **upper**.

```
    def int_within_bounds(n, lower, upper): return lower <= n < upper and n % 1 == 0
    print(int_within_bounds(10, 5, 20))
    # Output: True
    print(int_within_bounds(5, 5, 20))
    # Output: False
    print(int_within_bounds(21, 5, 20))
    # Output: False
```

27. GENERATE A RANDOM NUMBER WITHIN A RANGE

The **random_in_range** function generates a random number within the specified range.

```
import random

def random_in_range(min_val, max_val): return random.randint(min_val, max_val)

print(random_in_range(1, 10))

# Output: e.g., 6
```

28. CONVERT SECONDS TO HH:MM:SS FORMAT

The **seconds_to_hhmmss** function converts the given number of seconds into the HH:MM:SS format.

```
def seconds_to_hhmmss(seconds):
hours = seconds // 3600
minutes = (seconds % 3600) // 60
seconds = seconds % 60
return f"{hours:02}:{minutes:02}:{seconds:02}"
print(seconds_to_hhmmss(3660))
# Output: 01:01:00
```

29. GET THE LAST ELEMENT OF AN ARRAY

The **get_last_element** function retrieves the last element of the given array.

```
def get_last_element(arr): return arr[-1]

print(get_last_element([1, 2, 3, 4]))

# Output: 4
```

30. JAZZIFY CHORDS

The **jazzified_arr** function appends the number 7 to the end of every chord in an array. It ignores all chords that already end with 7.

```
    def jazzify(arr): return [x if str(x)[-1] == '7' else x + 7 for x in arr]

    arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]

    jazzified_arr = jazzify(arr)

    print(", ".join(map(str, jazzified_arr)))

    # Output: 8, 9, 10, 11, 12, 13, 7, 15, 16
```

31. CHECK IF ALL ELEMENTS IN AN ARRAY ARE THE SAME

The **test_jackpot** function takes in a string representing the result of a jackpot game and returns true if all elements in the string are the same, indicating a jackpot, and false otherwise.

```
def test_jackpot(result): return len(set(result)) == 1
result = "22222"
print(test_jackpot(result))
# Output: True
```

32. SUM OF NUMBERS UP TO A GIVEN NUMBER

The **add_up** function calculates the sum of all numbers from 1 to the number passed to the function in C#.

```
def add_up(num): return num * (num + 1) // 2

print(add_up(5))

# Output: 15
```

33. SUM ALL NUMBERS IN AN ARRAY

The **sum_array** function calculates the sum of all numbers in a given array.

```
def sum_array(arr): return sum(arr)
print(sum_array([1, 2, 3, 4, 5]))
# Output: 15
```

34. FIND THE MAXIMUM VALUE IN AN ARRAY

The **find_max** function finds the maximum value in a given array.

.

```
def find_max(arr): return max(arr)
print(find_max([10, 5, 8, 20, 3]))
# Output: 20
```

35. GET THE CURRENT DATE IN DD/MM/YYYY FORMAT

The **get_current_date** function returns the current date in the format "DD/MM/YYYY".

```
from datetime import datetime

def get_current_date(): return datetime.now().strftime("%d/%m/%Y")

print(get_current_date())

# Output: e.g., 16/04/2024
```

36. CALCULATE THE POWER OF A NUMBER

The **power** function calculates the power of a given base raised to the specified exponent.

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

print(power(2, 5))

# Output: 32.0
```

37. CONVERT STRING TO NUMBER

The **string_to_number** function converts a string to a floating-point number.

```
def string_to_number(string): return float(string)
print(string_to_number("3.14"))
# Output: 3.14
```

38. MARATHON DISTANCE CHECKER

The **marathon_distance** function helps Mary determine if the marathon she wants to run is exactly 25 miles long by analyzing the lengths listed in small portions on the sign-up sheet. It returns true if the total distance matches 25 miles, otherwise, it returns false.

```
    def marathon_distance(distances): return sum(map(abs, distances)) == 25

    distances = [1, 2, -3, 4, 5, -6, 7, 8, -9, 10]

    print(marathon_distance(distances))

    # Output: False
```

39. COUNT THE NUMBER OF WORDS IN A STRING

The **count_words** function counts the number of words in a given string.

```
def count_words(string): return len(string.split())
print(count_words("Hello world, how are you?"))
# Output: 5
```

40. COUNT ONES IN BINARY REPRESENTATION

The **count_ones** function counts the number of ones in the binary representation of an integer.

```
def count_ones(n): return bin(n).count('1')

result = count_ones(12)

print(result)

# Output: 2
```

41. GET THE CURRENT YEAR

The **current_year** function returns the current year as an integer.

```
from datetime import datetime
def current_year(): return datetime.now().year
print(current_year())
# Output: e.g., 2024
```

42. GENERATE A RANDOM NUMBER BETWEEN 1 AND 10

The **random_1_to_10** function generates a random integer between 1 and 10 (inclusive) using the rand method with a range of 1 to 10.

```
import random
def random_1_to_10(): return random.randint(1, 10)
print(random_1_to_10())
# Output: e.g., 8
```

43. CHECK IF A STRING IS EMPTY

The **empty_string** function checks if a string is empty.

```
def empty_string(string): return not string.strip()
print(empty_string(""))
# Output: True
print(empty_string("Hello, world!"))
# Output: False
```

44. SUM OF INDEX MULTIPLIED ELEMENTS

The **index_multiplier** function calculates the sum of all items in an array, where each item is multiplied by its index (zero-based). If the array is empty, it returns 0.

```
    def index_multiplier(arr): return sum(value * index for index, value in enumerate(arr))

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

    print(index_multiplier(arr))

    # Output: 40
```

45. CALCULATE PROGRESS DAYS

The **progress_days** function calculates the total number of progress days based on an array of miles run every Saturday.

```
    def progress_days(runs): return sum(runs[i] < runs[i + 1] for i in range(len(runs) - 1))

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

    print(progress_days(runs))

    # Output: 3
```

46. CHECK IF A NUMBER IS A MULTIPLE OF 5

The **is_multiple_of_5** function checks if a given number is a multiple of 5.

```
def is_multiple_of_5(num): return num % 5 == 0
print(is_multiple_of_5(10))
# Output: True
print(is_multiple_of_5(7))
# Output: False
```

47. CONVERT MINUTES TO SECONDS

The **mins_to_secs** convert minutes to seconds. It takes a number representing minutes (mins) as input and returns the equivalent number of seconds by multiplying the number of minutes by 60.

```
def mins_to_secs(mins): return mins * 60

print(mins_to_secs(5))

# Output: 300
```

48. FIND THE MAXIMUM VALUE IN AN ARRAY OF OBJECTS

The **FindMaxValue** function finds the maximum value in an array of objects.

```
def find_max_value(arr, key):
return max(item[key] for item in arr)
# Example usage
arr = [
{"name": "Alice", "score": 80},
{"name": "Bob", "score": 95},
{"name": "Charlie", "score": 70}
]
print(find_max_value(arr, "score"))
# Output: 95
```

49. CHECK IF A STRING STARTS WITH A SPECIFIC CHARACTER

The **starts_with_char** function checks if a given string starts with a specific character.

```
def starts_with_char(string, char):
return string.lower().startswith(char.lower())
print(starts_with_char("Hello, world!", 'H'))
# Output: True
print(starts_with_char("Hello, world!", 'h'))
# Output: True
```

50. CONVERT DNA TO RNA

The **dna_to_rna** function takes a DNA sequence as input and converts it into its corresponding RNA sequence.

```
def dna_to_rna(dna):
conversion = {'A': 'U', 'T': 'A', 'C': 'G', 'G': 'C'}
return ''.join(conversion.get(base, '') for base in dna)
print(dna_to_rna("ATTGC"))
# Output: UAAAC
```

51. CHECK IF AN ARRAY CONTAINS A SPECIFIC VALUE

The **contains_value** function checks if a given array contains a specific value.

```
def contains_value(arr, value):
return value in arr
# Example usage
print(contains_value([1, 2, 3, 4, 5], 3))
# Output: True
print(contains_value([1, 2, 3, 4, 5], 6))
# Output: False
```

52. CONVERT AN ARRAY TO A COMMA-SEPARATED STRING

The **array_to_csv** function converts an array to a comma-separated string.

```
def array_to_csv(arr):
return ', '.join(map(str, arr))
# Example usage
print(array_to_csv([1, 2, 3, 4, 5]))
# Output: 1, 2, 3, 4, 5
```

53. CHECK IF A YEAR IS A LEAP YEAR

The **is_leap_year** function takes a year as input and returns true if it's a leap year and false otherwise.

```
    def is_leap_year(year):
    return (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
    # Example usage
    print(is_leap_year(2024))
    # Output: True
    print(is_leap_year(2023))
    # Output: False
```

54. FIND THE INDEX OF AN ELEMENT IN AN ARRAY

The **find_index** function finds the index of an element in an array.

```
def find_index(arr, element):
try:
return arr.index(element)
except ValueError:
return -1
# Example usage
fruits = ["apple", "banana", "orange", "grape"]
print(find_index(fruits, "orange"))
# Output: 2
```

55. CONVERT MINUTES TO HOURS AND MINUTES

The **mins_to_hours_and_mins** function convert minutes to hours and minutes.

```
def mins_to_hours_and_mins(mins):
return f"{mins // 60} hours and {mins % 60} minutes"
# Example usage
print(mins_to_hours_and_mins(150))
# Output: 2 hours and 30 minutes
```

56. CHECK IF AN ARRAY IS SORTED IN ASCENDING ORDER

The **is_sorted_ascending** function takes an array arr as input and checks if every element in the array is greater than or equal to the previous element, ensuring that the array is sorted in ascending order.

```
def is_sorted_ascending(arr):
return all(arr[i] <= arr[i + 1] for i in range(len(arr) - 1))
# Example usage
print(is_sorted_ascending([1, 2, 3, 5, 8]))
# Output: True
print(is_sorted_ascending([1, 5, 3, 8, 2]))
# Output: False
```

57. REMOVE A SPECIFIC ELEMENT FROM AN ARRAY

The **remove_element** function removes a specific element from an array.

```
def remove_element(arr, element):
return [x for x in arr if x != element]
# Example usage
result = remove_element([1, 2, 3, 4, 5], 3)
print(result)
# Output: [1, 2, 4, 5]
```

58. TRUNCATE A STRING TO A GIVEN LENGTH

The **truncate_string** function truncate a string to a given length.

```
def truncate_string(s, max_length):
return s[:max_length] + '...' if len(s) > max_length else s
# Example usage
print(truncate_string("Hello, world!", 5))
# Output: Hello...
```

59. CONVERT VIDEO LENGTH FROM MINUTES TO SECONDS

The **minutes_to_seconds** function takes the length of a video in the format "mm:ss" and converts it into seconds.

```
def minutes_to_seconds(time):
return sum(int(x) * 60 ** i for i, x in enumerate(reversed(time.split(':'))))
# Example usage
print(minutes_to_seconds("02:54"))
# Output: 174
```

60. FIND THE DIFFERENCE BETWEEN TWO DATES IN DAYS

The **DateDifferenceInDays** function finds the difference between two dates in days.

```
from datetime import datetime
def DateDifferenceInDays(date1, date2):
return (date2 - date1).days
# Example usage
startDate = datetime(2023, 8, 1)
endDate = datetime(2023, 8, 10)
print(DateDifferenceInDays(startDate, endDate))
# Output: 9
```

61. CHECK IF A STRING IS A VALID EMAIL ADDRESS

The **IsValidEmail** function returns true if the email string matches the pattern, and false otherwise.

```
import re
def IsValidEmail(email):
return bool(re.match(r'^[^\s@]+@[^\s@]+\.[^\s@]+$', email))
# Example usage
print(IsValidEmail("user@example.com"))
# True
print(IsValidEmail("invalid-email"))
# False
```

62. CONVERT SECONDS TO MINUTES AND SECONDS

The **SecsToMinsAndSecs** function calculate the number of minutes and remaining seconds and then returns a formatted string containing the result.

```
    def SecsToMinsAndSecs(seconds):
    return f"{seconds // 60} minutes and {seconds % 60} seconds"
    # Example usage
    print(SecsToMinsAndSecs(120))
    # Output: 2 minutes and 0 seconds
```

63. GENERATE A FIBONACCI SEQUENCE

The **Fibonacci** generates a Fibonacci sequence.

```
def Fibonacci(n):
    return n if n <= 1 else Fibonacci(n - 1) + Fibonacci(n - 2)
    # Example usage
    fibonacci_sequence = [Fibonacci(i) for i in range(8)]
    print(', '.join(map(str, fibonacci_sequence)))
    # Output: 0, 1, 1, 2, 3, 5, 8, 13
```

64. SPELL OUT A WORD

The **Spelling** function takes a word as input and spells it out by consecutively adding letters until the full word is completed. It then returns an array containing each step of the spelling process.

```
def Spelling(word):
return [word[:i+1] for i in range(len(word))]
# Example usage
word = "example"
spelled_word = Spelling(word)
print(", ".join(spelled_word))
# Output: e, ex, exa, exam, examp, exampl, example
```

65. CHECK IF AN ARRAY CONTAINS ONLY UNIQUE VALUES

The **HasUniqueValues** function converts the array arr to a set using uniq, which removes duplicate elements.

```
def HasUniqueValues(arr):
return len(set(arr)) == len(arr)
# Example usage
print(HasUniqueValues([1, 2, 3, 4, 5]))
# Output: True
print(HasUniqueValues([1, 2, 3, 4, 4]))
# Output: False
```

66. GET THE DAY OF THE WEEK FROM A DATE

The **GetDayOfWeek** function get the day of the week from a date.

```
from datetime import datetime
def GetDayOfWeek(date_str):
date_obj = datetime.strptime(date_str, '%Y-%m-%d')
return date_obj.strftime('%A')
# Example usage
print(GetDayOfWeek("2023-08-02"))
# Output: Wednesday
```

67. CHECK IF A NUMBER IS A POWER OF TWO

The **IsPowerOfTwo** function checks whether a given number is a power of two using bitwise operations.

```
def IsPowerOfTwo(num):
return num & (num - 1) == 0
# Example usage
print(IsPowerOfTwo(16))
# Output: True (16 is 2^4)
print(IsPowerOfTwo(5))
# Output: False
```

68. GENERATE MULTIPLICATION TABLE

The **GenerateMultiplicationTable** function creates a multiplication table of a specified size.

```
    def GenerateMultiplicationTable(size):
    table = [[(row + 1) * (col + 1) for col in range(size)] for row in range(size)]
    return table
    # Example usage
    size = 5
    multiplicationTable = GenerateMultiplicationTable(size)
    for row in multiplicationTable:
    print(", ".join(map(str, row)))
```

69. SHHH WHISPERER

The **Shhh** function takes a sentence as input, removes all capital letters except for the first letter, wraps the sentence in double quotation marks, and adds ", whispered your friend." to the end.

```
    def Shhh(sentence):
    lowered = f'"{sentence[0].lower()}
{sentence[1:].lower()}", whispered your friend.'
    return lowered
    # Example usage
    print(Shhh("HELLO THERE"))
    # Output: "Hello there", whispered your friend.
```

70. GET THE MONTH NAME FROM A DATE

The **GetMonthName** function retrieves the name of the month from a given date.

```
import datetime
def GetMonthName(date):
return date.strftime("%B")
# Example usage
date = datetime.datetime(2023, 8, 2)
print(GetMonthName(date))
# Output: August
```

71. FIND THE BOMB

The **Bomb** function searches for the word "bomb" in a given string. If the word is found, it returns "Duck!!!", indicating a potential danger. Otherwise, it returns "There is no bomb, relax.", reassuring that there is no threat.

```
    import re
    def Bomb(s):
    return "Duck!!!" if re.search(r'\bbomb\b', s,
re.IGNORECASE) else "There is no bomb, relax."
    # Example usage
    s = "The bomb is about to explode."
    print(Bomb(s))
    # Output: Duck!!!
```

72. CONVERT FEET TO METERS

The **FeetToMeters** function converts a given length from feet to meters.

```
def FeetToMeters(feet):
return feet * 0.3048
# Example usage
print(FeetToMeters(10))
# Output: 3.048
```

73. CHECK IF A NUMBER IS A PERFECT SQUARE

The **IsPerfectSquare** function checks if a given number is a perfect square.

```
def IsPerfectSquare(num):
sqrt = num ** 0.5
return sqrt.is_integer()
# Example usage
print(IsPerfectSquare(16))
# Output: True
print(IsPerfectSquare(10))
# Output: False
```

74. CHECK IF A STRING CONTAINS ONLY NUMBERS

The **ContainsOnlyNumbers** function checks if a given string contains only numeric characters.

```
import re
def ContainsOnlyNumbers(s):
return bool(re.match(r'^\d+$', s))
# Example usage
print(ContainsOnlyNumbers("12345"))
# Output: True
print(ContainsOnlyNumbers("12a34"))
# Output: False
```

75. GET THE CURRENT MONTH (0-BASED INDEX)

The **CurrentMonth** retrieves the current month as a 0-based index.

```
from datetime import datetime
def CurrentMonth():
return datetime.now().month - 1
# Example usage
print(CurrentMonth())
# Output: 3 is the current month (0-based index)
```

76. CENTURY FROM YEAR

The **CenturyFromYear** function calculates the century corresponding to a given year. It takes a year as input and returns its corresponding century.

```
def CenturyFromYear(year):
return (year + 99) // 100
# Example usage
print(CenturyFromYear(1905))  # Output: 20
print(CenturyFromYear(1700))  # Output: 17
print(CenturyFromYear(1988))  # Output: 20
print(CenturyFromYear(2000))  # Output: 20
print(CenturyFromYear(2001))  # Output: 21
```

77. CHECK IF A NUMBER IS A PRIME NUMBER

The **IsPrime** function checks if a given number is prime.

```
def IsPrime(num):
    return num > 1 and all(num % i != 0 for i in range(2, int(num ** 0.5) + 1))
    # Example usage
    print(IsPrime(13))
    # Output: True
    print(IsPrime(4))
    # Output: False
```

78. GET THE LAST N ELEMENTS OF AN ARRAY

The **LastNElements** function retrieves the last n elements of an array.

```
def LastNElements(arr, n):
return arr[-n:]
# Example usage
result = LastNElements([1, 2, 3, 4, 5], 3)
print(", ".join(map(str, result)))
# Output: 3, 4, 5
```

79. CONVERT DEGREES TO RADIANS

The **DegToRad** function converts an angle in degrees to radians.

```
import math
def DegToRad(degrees):
return degrees * (math.pi / 180)
# Example usage
print(DegToRad(90))
# Output: 1.5707963267948966
```

80. BINARY LETTER CONVERTER

The **ConvertBinary** function transforms all letters from 'a' to 'm' to 0 and letters from 'n' to 'z' to 1 in a given string.

```
    def ConvertBinary(s):
    return ''.join(['0' if 'a' <= c <= 'm' else '1' for c in s.lower()])
    # Example usage
    print(ConvertBinary("hello"))
    # Output: 00000
    print(ConvertBinary("world"))
    # Output: 11111
    print(ConvertBinary("abcxyz"))
    # Output: 000111
```

81. FIND THE INTERSECTION OF TWO ARRAYS

The **FindIntersection** function finds the intersection of two arrays.

```
def FindIntersection(arr1, arr2):
return list(set(arr1) & set(arr2))
# Example usage
arr1 = [1, 2, 3]
arr2 = [2, 3, 4]
intersection = FindIntersection(arr1, arr2)
print(intersection)
# Output: [2, 3]
```

82. CONVERT DAYS TO YEARS, MONTHS, AND DAYS

The **DaysToYearsMonthsDays** function converts a given number of days to years, months, and remaining days.

```
    def DaysToYearsMonthsDays(days):
    years = days // 365
    months = (days % 365) // 30
    remaining_days = (days % 365) % 30
    return f"{years} years, {months} months, and
{remaining_days} days"
    # Example usage
    print(DaysToYearsMonthsDays(1000))
    # Output: 2 years, 9 months, and 5 days
```

83. CHECK IF AN OBJECT IS EMPTY

The **IsEmptyObject** function checks if a given object has no own properties.

```
def IsEmptyObject(obj):
return len(obj) == 0
# Example usage
print(IsEmptyObject({}))  # Output: True
print(IsEmptyObject({"name": "John", "age": 30}))
# Output: False
```

84. COUNT DECIMAL PLACES

The **GetDecimalPlaces** function takes a number represented as a string and returns the number of decimal places it has. If the number has no decimal places, it returns 0.

```
def GetDecimalPlaces(num):
if "." in num:
return len(num.split(".")[1])
else:
return 0
# Example usage
print(GetDecimalPlaces("3.14159"))
# Output: 5
print(GetDecimalPlaces("100.345"))
# Output: 3
print(GetDecimalPlaces("10.000"))
# Output: 3
print(GetDecimalPlaces("32"))
# Output: 0
```

85. REMOVE WHITESPACE FROM A STRING

The **RemoveWhitespace** function removes all whitespace characters from a given string.

```
def RemoveWhitespace(str):
return "".join(str.split())
# Example usage
print(RemoveWhitespace("  Hello,  world!  "))
# Output: Hello,world!
```

86. FIND THE DIFFERENCE BETWEEN TWO ARRAYS

The **ArrayDifference** function finds the difference between two arrays.

```
def ArrayDifference(arr1, arr2):
return list(set(arr1) - set(arr2))
# Example usage
arr1 = [1, 2, 3]
arr2 = [2, 3, 4]
difference = ArrayDifference(arr1, arr2)
print(", ".join(map(str, difference)))
# Output: 1
```

87. CHECK IF A NUMBER IS A FIBONACCI NUMBER

The **IsFibonacci** function checks if a number is a Fibonacci number.

```
    def IsFibonacci(num):
    return IsPerfectSquare(5 * num * num + 4) or
IsPerfectSquare(5 * num * num - 4)
    def IsPerfectSquare(num):
    sqrt = int(num ** 0.5)
    return sqrt * sqrt == num
    # Example usage
    print(IsFibonacci(5))
    # True
    print(IsFibonacci(6))
    # False
```

88. CONVERT HOURS TO MINUTES

The **HoursToMinutes** function takes a parameter hours and returns the equivalent number of minutes by multiplying hours by 60.

```
    def HoursToMinutes(hours):
    # Multiply hours by 60 to get the equivalent number of minutes
    return hours * 60
    # Example usage
    print(HoursToMinutes(2))
    # Output: 120
```

89. GET THE FIRST N ELEMENTS OF AN ARRAY

The **FirstNElements** function get the first n elements of an array.

```
    def FirstNElements(arr, n):
    # Use list slicing to retrieve the first n elements of the array
    return arr[:n]
    # Example usage
    array = [1, 2, 3, 4, 5]
    first_n_elements = FirstNElements(array, 3)
    print(", ".join(map(str, first_n_elements)))
    # Output: 1, 2, 3
```

90. CHECK IF A NUMBER IS ODD

The **is_odd** function checks if a number is odd.

```
def is_odd(num):
return num % 2 != 0
# Example usage
print(is_odd(5))
# Output: True
print(is_odd(4))
# Output: False
```

91. CALCULATE THE STANDARD DEVIATION OF AN ARRAY OF NUMBERS

The **standard_deviation** function calculates the standard deviation of an array of numbers.

```
import math
def standard_deviation(arr):
mean = sum(arr) / len(arr)
sum_of_squared_differences = sum((x - mean) ** 2 for x in arr)
return math.sqrt(sum_of_squared_differences / len(arr))
# Example usage
print(standard_deviation([1, 2, 3, 4, 5]))
# Output: 1.4142135623730951
```

92. CHECK IF A STRING ENDS WITH A SPECIFIC SUBSTRING

The **ends_with_substring** function checks if a string ends with a specific substring.

```
def ends_with_substring(string, substring):
return string.endswith(substring)
# Example usage
print(ends_with_substring("Hello, world!", "world!"))
# Output: True
print(ends_with_substring("Hello, world!", "Hello"))
# Output: False
```

93. CALCULATE THE SUM OF SQUARES OF AN ARRAY

The **sum_of_squares** function calculates the sum of squares of an array.

```
def sum_of_squares(arr):
return sum(x ** 2 for x in arr)
# Example usage
print(sum_of_squares([1, 2, 3, 4, 5]))
# Output: 55
```

94. CALCULATE PI TO N DECIMAL PLACES

The **my_pi** function takes an integer **n** and returns the mathematical constant PI to n decimal places.

```
import math
def my_pi(n):
return round(math.pi, n)
# Example usage
print(my_pi(5))
# Output: 3.14159
```

95. GENERATE AN ARRAY OF RANDOM NUMBERS

The **random_array** function generates an array of random numbers.

```
import random
def random_array(length):
return [random.randint(0, 99) for _ in range(length)]
# Example usage
random_numbers = random_array(5)
print(random_numbers)
# Output: Array with 5 random numbers, e.g., [23, 45, 67, 11, 88]
```

96. JOIN PATH PORTIONS

The **join_path** function receives two portions of a path and joins them using the "/" separator. If the separator is missing between the portions, it is added before joining.

```
def join_path(portion1, portion2):
return f"{portion1.rstrip('/')}/{portion2.lstrip('/')}"
# Example usage
print(join_path("portion1", "portion2"))
# Output: portion1/portion2
```

97. CONVERT SECONDS TO HOURS, MINUTES, AND SECONDS

The **seconds_to_hours_mins_secs** function convert seconds to hours, minutes, and remaining seconds.

```
    def seconds_to_hours_mins_secs(seconds):
    hours = seconds // 3600
    remaining_seconds = seconds % 3600
    minutes = remaining_seconds // 60
    remaining_minutes = remaining_seconds % 60
    return f"{hours} hours, {minutes} minutes, and {remaining_minutes} seconds"
    # Example usage
    print(seconds_to_hours_mins_secs(7320))
    # Output: "2 hours, 2 minutes, and 0 seconds"
```

98. SIMPLE CALCULATOR

The **calculator** function calculates different operations.

```
    def calculator(num1, op, num2):
    return {'+': num1 + num2, '-': num1 - num2, '*': num1 * num2, '/': num1 / num2}[op]
    # Example usage
    print(calculator(5, '+', 3))    # Addition outputs: 8
    print(calculator(10, '-', 4))   # Subtraction outputs: 6
    print(calculator(6, '*', 2))    # Multiplication outputs: 12
    print(calculator(20, '/', 5))   # Division outputs: 4
```

99. FIND NEMO

The **find_nemo** function function takes a string of words as input and searches for the word "Nemo". If "Nemo" is found, it returns a string indicating the position of "Nemo" in the sentence. If "Nemo" is not found, it returns a message indicating its absence.

```
    def find_nemo(sentence):
    words = sentence.split()
    index = words.index("Nemo") + 1 if "Nemo" in words else -1
    return f"I found Nemo at {index}!" if index > 0 else "I can't find Nemo :("
    # Example usage
    print(find_nemo("I am finding Nemo"))
    # I found Nemo at 3!
```

100. COUNT THE OCCURRENCES OF A CHARACTER IN A STRING

The **count_occurrences** function counts the occurrences of a character in a String.

```
def count_occurrences(string, char):
return string.count(char)
# Example usage
print(count_occurrences("hello world", 'l'))
# Output: 3
```

101. CONVERT VIDEO LENGTH TO SECONDS

The **minutes_to_seconds** function takes the length of a video in the format "mm:ss" and converts it to seconds. If the input format is correct, it returns the length of the video in seconds; otherwise, it returns 0.

```
def minutes_to_seconds(time):
minutes, seconds = map(int, time.split(':'))
return minutes * 60 + seconds if 0 <= seconds < 60 else 0

# Example usage
print(minutes_to_seconds("02:54"))
# Output: 174
```

102. REMOVE DUPLICATES FROM A STRING

The **remove_duplicates_from_string** function removes duplicates from a String.

```
def remove_duplicates_from_string(string):
return ''.join(sorted(set(string), key=string.index))
# Example usage
print(remove_duplicates_from_string("hello"))
# Output: helo
```

103. FIND THE MODE OF AN ARRAY OF NUMBERS

The **mode** function finds the mode of an array of numbers.

```
    def mode(arr):
    occurrences = {}
    for num in arr:
    if num in occurrences:
    occurrences[num] += 1
    else:
    occurrences[num] = 1
    max_occurrences = max(occurrences.values())
    return [num for num, count in occurrences.items() if count
== max_occurrences]
    # Example usage
    numbers = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
    print(f"Mode: {mode(numbers)}")
    # Output: Mode: [4]
```

104. CHECK FOR REPDIGIT

The **is_repdigit** function checks if a given integer is a repdigit, meaning it is composed of the same digit repeated. If the integer is a repdigit, it returns true; otherwise, it returns false.

```
def is_repdigit(num):
return len(set(str(num))) == 1
# Example usage
print(is_repdigit(333))
# Output: True
```

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
```

136. FIND THE AVERAGE OF ODD NUMBERS IN AN ARRAY

The **average_of_odd_numbers** function finds the average of odd numbers in an array.

```
    def average_of_odd_numbers(arr):
    odd_numbers = [num for num in arr if num % 2 != 0]
    return sum(odd_numbers) / len(odd_numbers) if
odd_numbers else 0
    # Test case
    numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    print(average_of_odd_numbers(numbers))
    # Output: 5.0
```

137. COUNT THE LETTERS IN A STRING (CASE-INSENSITIVE)

The **count_letters** function counts the letter in a String (case-insensitve).

```
def count_letters(s):
return {char: s.lower().count(char) for char in s.lower() if char.isalpha()}
# Test case
input_str = "Hello, World!"
result = count_letters(input_str)
for char, count in result.items():
print(f"{char} => {count}")
# h => 1
# e => 1
# l => 3
# o => 2
# w => 1
# r => 1
# d => 1
```

138. CONVERT SECONDS TO DAYS, HOURS, MINUTES, AND SECONDS

The **secs_to_days_hours_mins_secs** function converts seconds to days, hours, minutes, and seconds.

```
def secs_to_days_hours_mins_secs(seconds):
days = seconds // 86400
hours = (seconds % 86400) // 3600
minutes = ((seconds % 86400) % 3600) // 60
secs = ((seconds % 86400) % 3600) % 60
return f"{days} days, {hours} hours, {minutes} minutes, and {secs} seconds"
# Test case
seconds = 100000
print(secs_to_days_hours_mins_secs(seconds))
# 1 days, 3 hours, 46 minutes, and 40 seconds
```

139. CHECK IF A NUMBER IS A PRIME FACTOR OF ANOTHER NUMBER

The **is_prime_factor** function checks if a given number factor is a prime factor of another number num.

```
def is_prime_factor(num, factor):
return num % factor == 0 and all(factor % i != 0 for i in range(2, int(factor**0.5) + 1)) and factor > 1
# Test cases
num = 20
factor = 2
print(is_prime_factor(num, factor))
# Output: True
factor = 3
print(is_prime_factor(num, factor))
# Output: False
```

140. FIND THE LARGEST PRIME FACTOR OF A NUMBER

The **largest_prime_factor** function calculates the largest prime factor of a given number.

```
def largest_prime_factor(num):
if num == 1:
return 1
factor = next((x for x in range(2, int(num**0.5) + 1) if num % x == 0 and all(x % y != 0 for y in range(2, int(x**0.5) + 1))), 0)
return num if factor == 0 else factor
# Test case
print(largest_prime_factor(48))
# Output: 2
```

141. CHECK IF A NUMBER IS A PRONIC SQUARE

The **is_pronic_square** function checks if the square root of the given number num is an integer, indicating that num is a perfect square.

```
def is_pronic_square(num):
sqrt = num ** 0.5
return sqrt == int(sqrt)
# Test cases
print(is_pronic_square(6))
# Output: True
print(is_pronic_square(20))
# Output: False
print(is_pronic_square(21))
# Output: True
```

142. WORLD LANDMASS PROPORTION CALCULATOR

The **area_of_country** function calculates a country's proportion of the total world's landmass based on its name and area.

```
def area_of_country(name, area):
return f"{name} is {area * 100 / 148940000:.2f}% of the total world's landmass"
# Test case
country_name = "Canada"
country_area = 9984670  # in square kilometers
print(area_of_country(country_name, country_area))
# Output: Canada is 6.71% of the total world's landmass
```

143. LOADED DIE DETECTION

The **is_unloaded** function calculates derivatives of frequencies and compares them to a critical value.

```
    def is_unloaded(frequencies):
    return sum(f * (i + 1) ** (f - 1) for i, f in
enumerate(frequencies)) > 11.0705
    # Test case
    frequencies = [0.02, 0.03, 0.05, 0.1, 0.2, 0.6]
    print(is_unloaded(frequencies))
    # Output: True or False
```

144. HAND WASHING DURATION CALCULATOR

The **wash_hands** function calculates the total duration a person spends washing their hands based on the number of times they wash their hands per day (N) and the number of months (nM) they follow this routine.

```
def wash_hands(N, nM):
total_seconds = nM * N * 21 * 30
minutes = total_seconds // 60
seconds = total_seconds % 60
return f"{minutes} minutes and {seconds} seconds"
# Test case
N = 3  # Number of times a person washes their hands per day
nM = 2  # Number of months they follow this routine
print(wash_hands(N, nM))
# Output: 63 minutes and 0 seconds
```

145. CHECK IF A NUMBER IS A HAPPY NUMBER

The **is_happy_number** function checks if a given number is a "happy number" or not.

```
    def is_happy_number(num):
    return num == 1 or num == 7 or num == 10 or (num < 100 and num % 10 == 0 and num % 11 == 0) or (num < 1000 and num % 100 == 0 and num % 111 == 0)
    # Test case
    print("true" if is_happy_number(19) else "false")
    # Output: false
```

146. BILLABLE DAYS BONUS CALCULATOR

The **calculate_bonus** function efficiently computes the bonus using predefined thresholds and rates, ensuring accurate compensation for the employee's performance.

```
    def calculate_bonus(days):
    return 325 * max(days - 32, 0) + 225 * max(days - 40, 0)
+ 50 * max(days - 48, 0)
    # Example usage
    billable_days = 45
    bonus = calculate_bonus(billable_days)
    print(f"Bonus for {billable_days} billable days: {bonus}")
    # Output: Bonus for 45 billable days: 5350
```

147. CALCULATE THE VOLUME OF A SPHERE

The **sphere_volume** function calculates the volume of a sphere given its radius.

```
import math
def sphere_volume(radius):
return (4/3) * math.pi * radius ** 3
# Example usage
volume = sphere_volume(5)
print(volume)
# Output: 523.5987755982989
```

148. DISCOUNTED PRICE CALCULATOR

The **calculate_final_price** function takes the original price and the discount percentage as integers and returns the discounted price.

```
    def calculate_final_price(price, discount):
    return price * (100 - discount) * 0.01
    # Example usage
    final_price = 100  # Final price
    discount = 20  # Discount percentage
    discounted_price = calculate_final_price(final_price, discount)
    print(f"Final price after {discount}% discount: {discounted_price}")
    # Output: Final price after 20% discount: 80.0
```

149. DOUBLE LETTER CHECKER

The **has_double_letters** function takes a word as input and returns true if the word contains two consecutive identical letters, and false otherwise.

```
import re
def has_double_letters(word):
return bool(re.search(r"(\w)\1", word))
# Example usage:
word = "hello"  # Word to check for double letters
has_double_letters = has_double_letters(word)
print(f"Does the word '{word}' have double letters? {has_double_letters}")
# Output: Does the word 'hello' have double letters? True
```

150. FIND THE LENGTH OF THE LONGEST WORD IN A SENTENCE

The **longest_word_length** function finds the length of the longest word in a sentence.

```
def longest_word_length(sentence):
return max(len(word) for word in sentence.split())
# Example usage:
sentence = "The quick brown fox jumped over the lazy dog"
print(longest_word_length(sentence))
# Output: 6
```

151. STOLEN ITEMS LOSS CALCULATOR

The **calculate_losses** function takes a dictionary of stolen items and their values as input and returns the difference between the total value of those items and the policy limit.

```
def calculate_losses(stolen_items):
total_value = sum(stolen_items.values())
return "Lucky you!" if total_value == 0 else total_value
# Example usage:
stolen_items = {
"watch": 100,
"phone": 200,
"wallet": 50
}
losses = calculate_losses(stolen_items)
print(f"Losses: {losses}")
# Losses: 350
```

152. HACKER SPEAK CONVERTER

The **convert_to_hacker_speak** function takes a string as input and returns its hacker speak equivalent, replacing characters 'a', 'e', 'i', 'o', and 's' with '4', '3', '1', '0', and '5' respectively.

```
import re
def convert_to_hacker_speak(input_str):
return re.sub('[aeios]', lambda x: '4' if x.group() == 'a' else '3' if x.group() == 'e' else '1' if x.group() == 'i' else '0' if x.group() == 'o' else '5', input_str)
# Example usage:
input_str = "hello world"
hacker_speak = convert_to_hacker_speak(input_str)
print(f"Original: {input_str}")
# Original: hello world
print(f"Hacker Speak: {hacker_speak}")
# Hacker Speak: h3ll0 w0rld
```

153. FIND THE AREA OF A RECTANGLE

The **rectangle_area** function finds the area of a rectangle.

```
def rectangle_area(length, width):
return length * width
# Example usage:
length = 5
width = 10
area = rectangle_area(length, width)
print(f"Area: {area}")
# Output: Area: 50
```

154. CALCULATE THE SUM OF EVEN NUMBERS IN AN ARRAY

The **sum_of_even_numbers** calculates the sum of even numbers in an array.

```
def sum_of_even_numbers(arr):
return sum(num for num in arr if num % 2 == 0)
# Example usage:
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum = sum_of_even_numbers(arr)
print(f"Sum of even numbers: {sum}")
# Output: Sum of even numbers: 30
```

155. MISSING NUMBER FINDER

The **find_missing_number** function takes an array of numbers as input and returns the missing number. It efficiently calculates the missing number by subtracting the sum of the given array from the sum of all numbers from 1 to 10.

```
def find_missing_number(arr):
return 55 - sum(arr)
# Example usage:
numbers = [1, 2, 3, 4, 6, 7, 8, 9, 10]
missing_number = find_missing_number(numbers)
print(f"Missing number: {missing_number}")
# Output: Missing number: 5
```

156. CALCULATE THE VOLUME OF A CYLINDER

The **cylinder_volume** function calculates the volume of a cylinder.

```
import math
def cylinder_volume(radius, height):
return math.pi * radius ** 2 * height
# Example usage:
volume = cylinder_volume(5, 10)
print(f"Volume of the cylinder: {volume}")
# Output: Volume of the cylinder: 785.398163397448
```

157. BBQ SKEWER ANALYZER

The **bbq_skewers** method takes a list of skewers as input and returns the count of vegetarian and non-vegetarian skewers. A vegetarian skewer contains only vegetables ("-o"), while a non-vegetarian skewer contains at least one piece of meat ("-x").

```
    def bbq_skewers(grill):
    vegetarian_count = sum(1 for skewer in grill if "-x" not in skewer)
    non_vegetarian_count = len(grill) - vegetarian_count
    return [vegetarian_count, non_vegetarian_count]
    # Example usage:
    grill = ["-o", "-o", "-x", "-o", "-x"]  # Grill skewers
    result = bbq_skewers(grill)
    print(f"Vegetarian skewers: {result[0]}, Non-vegetarian skewers: {result[1]}")
    # Output: Vegetarian skewers: 3, Non-vegetarian skewers: 2
```

158. CONVERT DECIMAL NUMBER TO OCTAL

The **decimal_to_octal** function converts decimal number to octal.

```
def decimal_to_octal(num):
return oct(num).lstrip("0o")
# Example usage:
print(decimal_to_octal(27))
# Output: "33"
```

159. COLLATZ SEQUENCE ANALYZER

The **Collatz** method takes a positive integer as input and returns the number of steps required to reach 1 in the Collatz sequence.

```
    def collatz(num):
    count = 0
    while num != 1:
    num = num // 2 if num % 2 == 0 else 3 * num + 1
    count += 1
    return count
    # Example usage:
    num = 27
    steps = collatz(num)
    print(f"Number of steps to reach 1 in Collatz sequence for {num}: {steps}")
    # Output: Number of steps to reach 1 in Collatz sequence for 27: 111
```

160. CHECK IF A STRING IS A VALID PHONE NUMBER (NORTH AMERICAN FORMAT)

The **is_valid_phone_number** function checks if a string is a valid phone number (north american format).

```
import re
def is_valid_phone_number(phone):
return bool(re.match(r'^\d{3}-\d{3}-\d{4}$', phone))
# Example usage:
print(is_valid_phone_number("555-123-4567"))
# Output: True
print(is_valid_phone_number("123-4567"))
# Output: False
```

161. FIND THE SUM OF THE FIRST N NATURAL NUMBERS

The **sum_of_naturals** function finds the sum of the first n natural numbers.

```
def sum_of_naturals(n):
return (n * (n + 1)) // 2
# Example usage:
print(sum_of_naturals(10))
# Output: 55
```

162. VOWEL DASHER

The **dashed** function in the **StringDasher** class takes a string as input and returns the modified string with dashes added around each vowel.

```
    import re
    def dashed(string):
    return re.sub(r'[aeiouAEIOU]', lambda x: f'-{x.group(0)}-', string)
    # Example usage:
    input_str = "hello world"  # Example: Input string
    dashed_string = dashed(input_str)
    print(f"Original: {input_str}")
    # Original: hello world
    print(f"Dashed: {dashed_string}")
    # Dashed: h-e-ll-o- w-o-rld
```

163. FIND THE FACTORS OF A NUMBER (EXCLUDING 1 AND THE NUMBER ITSELF)

The **factors** function finds the factors of a number (excluding 1 and the number itself).

```
def factors(num):
result = []
for i in range(2, num):
if num % i == 0:
result.append(i)
return result
# Example usage:
print(", ".join(map(str, factors(12))))
# Output: 2, 3, 4, 6
```

164. CALCULATE THE AREA OF A TRIANGLE GIVEN THE BASE AND HEIGHT

The **triangle_area** function calculates the area of a triangle given the base and height.

```
def triangle_area(base, height):
return 0.5 * base * height
# Example usage:
area = triangle_area(5, 10)
print(area)
# Output: 25.0
```

165. CHECK IF A STRING IS A VALID SOCIAL SECURITY NUMBER (SSN)

The **is_valid_ssn** function checks whether a given string is a valid Social Security Number (SSN) in the format "XXX-XX-XXXX", where X represents a digit.

```
import re
def is_valid_ssn(ssn):
return bool(re.match(r'^\d{3}-\d{2}-\d{4}$', ssn))
# Test cases
print(is_valid_ssn("123-45-6789"))
# Output: True
print(is_valid_ssn("123-45-678"))
# Output: False
```

166. GENERATE AN ARRAY OF RANDOM NUMBERS WITHIN A RANGE

The **random_array_in_range** function generates an array of random numbers within a range.

```
import random
def random_array_in_range(min_val, max_val, length):
return [random.randint(min_val, max_val) for _ in range(length)]
# Test the function
random_array = random_array_in_range(1, 100, 5)
print(random_array)
# Output will be an array of 5 random numbers between 1 and 100
```

167. XO CHECKER

The **xo_checker** function takes a string as input and returns true if the string contains an equal number of 'x's and 'o's (case-insensitive), and false otherwise.

```
    def xo_checker(string):
    x_count = string.lower().count('x')
    o_count = string.lower().count('o')
    return x_count == o_count
    # Example usage:
    input_str = "xoxoxo"  # Example input string
    result = xo_checker(input_str)
    print(f"String \"{input_str}\" has the same number of 'x's and 'o's: {result}")
    # Output: String "xoxoxo" has the same number of 'x's and 'o's: True
```

168. CHECK IF A STRING IS A VALID IPV4 ADDRESS

The **is_valid_ipv4** function checks whether a given string represents a valid IPv4 address.

```
import re
def is_valid_ipv4(ip):
pattern = r"^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$"
return bool(re.match(pattern, ip))
# Example usage:
print(is_valid_ipv4("192.168.1.1"))
# Output: True
print(is_valid_ipv4("256.0.0.1"))
# Output: False
```

169. CONVERT DECIMAL NUMBER TO HEXADECIMAL

The **decimal_to_hex** function converts decimal number to hexadecimal.

```
def decimal_to_hex(num):
return hex(num).lstrip("0x").upper()
# Example usage:
print(decimal_to_hex(255))
# Output: "FF"
```

170. CHECK IF A STRING IS A VALID DATE (YYYY-MM-DD FORMAT)

The **is_valid_date** function checks whether a given string is a valid date in the format "YYYY-MM-DD".

```
import re
def is_valid_date(date):
return bool(re.match(r'^\d{4}-\d{2}-\d{2}$', date))
# Example usage:
print(is_valid_date("2023-08-02"))
# Output: True
print(is_valid_date("02-08-2023"))
# Output: False
```

171. CHINESE ZODIAC SIGN IDENTIFIER

The **get_chinese_zodiac_sign** function in the **ChineseZodiac** class takes a birth year as input and returns the corresponding Chinese zodiac sign.

```
    def get_chinese_zodiac_sign(birth_year):
    zodiac_signs = ["Monkey", "Rooster", "Dog", "Pig", "Rat", "Ox", "Tiger", "Rabbit", "Dragon", "Snake", "Horse", "Sheep"]
    return zodiac_signs[birth_year % 12]
    # Example usage:
    birth_years = [2021, 2020, 1938, 1951, 1964, 1977, 1990, 2003, 2016, 1969, 1982, 1995]
    for year in birth_years:
    sign = get_chinese_zodiac_sign(year)
    print(f"Year {year}: {sign}")
    # Year 1982: Dog
    # Year 1995: Pig
```

172. CHECK IF A STRING IS A VALID PASSWORD

The **is_valid_password** function checks if a string is a valid password (at least 8 characters, with a digit and special character).

```
import re
def is_valid_password(password):
# Regex pattern for a valid password:
# - At least 8 characters
# - At least one uppercase letter, one lowercase letter, one digit, and one special character from @$!%*?&
pattern = r"^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$"
return bool(re.match(pattern, password))
# Test cases
print(is_valid_password("P@ssw0rd"))
# Output: True
print(is_valid_password("password123"))
# Output: False
```

173. FIND THE NTH FIBONACCI NUMBER

The **fibonacci** function finds the nth fibonacci number.

```
def fibonacci(n):
return n - 1 if n <= 2 else fibonacci(n - 1) + fibonacci(n - 2)
# Test case
print(fibonacci(7))
# Output: 8
```

174. DIVING MINIGAME CHECKER

The **diving_minigame** function takes an array of integers representing the scores and returns **true** if the minigame is possible, and **false** otherwise.

```
    def diving_minigame(arr):
    return bool((lambda score: next((False for score_change in arr if score <= 0 or (score_change < 0 and (score := max(0, score - 2)) < 1) or (score_change >= 0 and (score := min(score + 4, 10)) > 0)), True))(10))
    # Test cases
    print(diving_minigame([0, -4, 0, -4, -5, -2]))  # True
    print(diving_minigame([-4, -3, -4, -3, 5, 2, -5, -20, -42, -4, 5, 3, 5]))  # True
    print(diving_minigame([1, 2, 1, 2, 1, 2, 1, 2, 1, -3, -4, -5, -3, -4]))  # False
    print(diving_minigame([-5, -5, -5, -5, -5, 2, 2, 2, 2, 2, 2, 2, 2]))  # False
```

175. ROGER'S SHOOTING SCORE CALCULATOR

The **roger_shots** function takes an array of strings representing the shots ("Bang!" or "BangBang!") and returns the total score earned by Roger. Each "Bang!" shot contributes 0.5 points to the score, and each "BangBang!

```
    def roger_shots(arr):
    return sum(0.5 for shot in arr if shot in ["Bang!", "BangBang!"])
    # Test cases
    print(roger_shots(["Bang!", "Bang!", "Bang!", "Bang!", "Bang!", "Bang!"]))  # 3.0
    print(roger_shots(["Bang!", "Bang!", "Bang!", "Bang!", "BangBang!"]))  # 2.5
    print(roger_shots(["Bang!", "BangBangBang!", "Boom!", "Bang!", "BangBang!", "BangBang!"]))  # 2.0
    print(roger_shots(["BangBang!", "BangBang!", "BangBang!"]))  # 1.5
    print(roger_shots(["Bang!", "BadaBing!", "Badaboom!", "Bang!", "Bang!", "Bang!", "Bang!", "Bang!"]))  # 3.0
```

```
    print(roger_shots(["BangBang!", "BangBang!", "Bag!",
"Ban!", "Tang!", "Bang!", "Bang!"]))  # 2.0
```

176. MIDDLE CHARACTER OF STRING

The **get_middle** function takes a string as input and returns the middle character(s) of the string.

```
def get_middle(s):
mid = len(s) // 2
return s[mid] if len(s) % 2 != 0 else s[mid - 1:mid + 1]
# Test case
print(get_middle("middle"))
# Output: dd
```

177. CALCULATE THE VOLUME OF A CUBE

The **cube_volume** function calculates the volume of a cube.

```
def cube_volume(side):
return side ** 3
# Test case
print(cube_volume(5))
# Output: 125
```

178. CHECK IF A STRING IS A VALID CREDIT CARD NUMBER (VISA, MASTERCARD, DISCOVER, AMERICAN EXPRESS)

The **is_valid_credit_card** function checks is a string is a valid credit card number (visa, mastercard, discover, american express).

```
import re
def is_valid_credit_card(card):
regex = re.compile(r'^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6(?:011|5[0-9][0-9])[0-9]{12}|3[47][0-9]{13})$')
return bool(regex.match(card.replace("-", "")))
# Test cases
print(is_valid_credit_card("4012-3456-7890-1234"))
# Output: True
print(is_valid_credit_card("1234-5678-9012-3456"))
# Output: False
```

179. CALCULATE THE PERIMETER OF A TRIANGLE

The **triangle_perimeter** function calculates the perimeter of a triangle.

```
def triangle_perimeter(side1, side2, side3):
return side1 + side2 + side3
# Test case
print(triangle_perimeter(5, 10, 7))
# Output: 22
```

180. CHECK IF A NUMBER IS A VAMPIRE NUMBER

The **is_vampire_number** function checks is a number is a vampire number.

```
def is_vampire_number(num):
num_str = str(num)
num_len = len(num_str)
for factor1 in range(10 ** (num_len // 2 - 1), int(num ** 0.5) + 1):
if num % factor1 == 0:
factor2 = num // factor1
if sorted(str(factor1) + str(factor2)) == sorted(num_str):
return True
return False
# Test cases
print(is_vampire_number(1260))
# Output: True
print(is_vampire_number(1250))
# Output: False
```

181. OBSOLETE SUM CONVERTER

The **get_abs_sum** the absolute sum of integers in an array.

```
def get_abs_sum(arr):
return sum(abs(num) for num in arr)
# Test case
arr = [2, -1, 4, 8, 10]
print(get_abs_sum(arr))
# Output: 25
```

182. CHECK IF A NUMBER IS A DUCK NUMBER

The **is_duck_number** function checks whether a given number is a duck number.

```
def is_duck_number(num):
num_str = str(num)
return '0' in num_str and num_str[0] != '0'
# Test cases
print(is_duck_number(1023))
# Output: True
print(is_duck_number(12345))
# Output: False
```

183. GENERATE A RANDOM PASSWORD

The **random_password** function generates a random password of the specified length by creating an array of random alphanumeric characters and joining them together into a string.

```
import random
import string
def random_password(length):
chars = string.ascii_letters + string.digits
return ''.join(random.choice(chars) for _ in range(length))
# Test case
print(random_password(8))
# Output: "3klS0p9x"
```

184. CALCULATE THE AREA OF A TRAPEZOID

The trapezoid_area function calculates the area of a trapezoid.

```
def trapezoid_area(base1, base2, height):
return 0.5 * (base1 + base2) * height
# Test case
print(trapezoid_area(4, 8, 6))
# Output: 36
```

185. CHECK IF A NUMBER IS A KAPREKAR NUMBER

The **is_kaprekar_number** function checks if a given number is a Kaprekar number.

```
def is_kaprekar_number(num):
square = num ** 2
square_str = str(square)
num_str = str(num)
left = int(square_str[:len(num_str)])
right = int(square_str[-len(num_str):])
return left + right == num
# Test cases
print(is_kaprekar_number(9))
# Output: True
print(is_kaprekar_number(297))
# Output: True
print(is_kaprekar_number(45))
# Output: False
```

186. CALCULATE THE VOLUME OF A CONE

The **cone_volume** function calculates the volume of a cone.

```
import math
def cone_volume(radius, height):
return (1 / 3) * math.pi * radius ** 2 * height
# Test case
print(cone_volume(5, 10))
# Output: 261.79938779914943
```

187. CHECK IF A STRING IS A VALID US PHONE NUMBER

The **is_valid_us_phone_number** function checks if a given string represents a valid US phone number.

```
import re
def is_valid_us_phone_number(phone):
pattern = r"^(?:(?:\+1\s?)?(?:\(?\d{3}\)?[\s.-]?)?\d{3}[\s.-]?\d{4})$"
return bool(re.match(pattern, phone))
# Test cases
print(is_valid_us_phone_number("+1 (123) 456-7890"))
# Output: True
print(is_valid_us_phone_number("123-456-7890"))
# Output: True
print(is_valid_us_phone_number("1-800-ABC-DEFG"))
# Output: False
```

188. SASTRY NUMBER CHECKER

The **is_sastry** function etermines whether a given positive integer n is a Sastry number.

```
import math
def is_sastry(num):
return math.sqrt(int(str(num) + str(num + 1))) % 1 == 0
# Test cases
print(is_sastry(183))
# Output: True
print(is_sastry(184))
# Output: False
```

189. FACTOR CHAIN CHECKER

The **factor_chain** function checks if an array forms a factor chain, where each element is a factor of the next consecutive element.

```
    def factor_chain(arr):
    return all(arr[i + 1] % arr[i] == 0 for i in range(len(arr) -
1))
    # Test cases
    chain1 = [1, 2, 4, 8, 16, 32]
    chain2 = [2, 4, 6, 7, 12]
    print(factor_chain(chain1))
    # Output: True
    print(factor_chain(chain2))
    # Output: False
```

190. CALCULATE BOXES IN ALGEBRA SEQUENCE

The **box_seq** function takes a step number as input and calculates the number of boxes in that step of the algebra sequence. Each step increases the number of boxes by either 0 or 2, depending on whether the step number is even or odd, respectively.

```
def box_seq(step):
return step + (step % 2 * 2)
# Example usage:
print(box_seq(5))
# Output: 7
print(box_seq(0))
# Output: 0
print(box_seq(10))
# Output: 10
```

191. CALCULATE THE VOLUME OF A CUBOID

The **cuboid_volume** function calculates the volume of a cuboid.

```
def cuboid_volume(length, width, height):
return length * width * height
# Example usage:
length = 5
width = 10
height = 8
volume = cuboid_volume(length, width, height)
print("Volume of the cuboid:", volume)
# Output: Volume of the cuboid: 400
```

192. TRIANGULAR NUMBER SEQUENCE

The **triangular** calculates the nth triangular number in the triangular number sequence. A triangular number is the sum of all positive integers up to a given integer n.

```
def triangular(n):
return n * (n + 1) // 2
# Example usage:
print(triangular(1))  # Output: 1
print(triangular(2))  # Output: 3
print(triangular(3))  # Output: 6
print(triangular(4))  # Output: 10
print(triangular(5))  # Output: 15
```

193. GENERATE A RANDOM COLOR (HEXADECIMAL FORMAT)

The **random_color_hex** function generates a random color (hexadecimal format)

```
import random
def random_color_hex():
return '#{0:06x}'.format(random.randint(0, 0xFFFFFF))
# Example usage:
print(random_color_hex())
# Output: #6C0F4A
```

194. CALCULATE THE AREA OF A CIRCLE SECTOR

The **circle_sector_area** function calculates the area of a circle sector.

```
import math
def circle_sector_area(radius, angle):
return (angle / 360) * math.pi * radius ** 2
# Example usage:
print(circle_sector_area(5, 90))
# Output: 19.6349540849362
```

195. CALCULATE THE AREA OF A REGULAR POLYGON

The **regular_polygon_area** function calculates the area of a regular polygon.

```
import math
def regular_polygon_area(side_length, num_of_sides):
return (num_of_sides * side_length ** 2) / (4 * math.tan(math.pi / num_of_sides))
# Example usage:
print(regular_polygon_area(5, 6))
# Output: 64.9519052838329
```

196. REMOVE DUPLICATES FROM ARRAY

The **remove_duplicates** function removes duplicates from array.

```
def remove_duplicates(arr):
return list(dict.fromkeys(arr))
# Example usage:
array = [1, 2, 3, 3, 4, 4, 5, 5, 6]
result = remove_duplicates(array)
print(", ".join(map(str, result)))
# Output: 1, 2, 3, 4, 5, 6
```

197. CALCULATE THE AREA OF AN ELLIPSE

The **ellipse_area** function calculates the area of an ellipse.

```
import math
def ellipse_area(a, b):
return math.pi * a * b
# Example usage:
semi_major_axis = 5
semi_minor_axis = 10
area = ellipse_area(semi_major_axis, semi_minor_axis)
print(area)
# Output: 157.07963267948966
```

198. CHECK IF A NUMBER IS A LEYLAND NUMBER

The **is_leyland_number** function checks if a number is a Leyland number.

```
def is_leyland_number(num):
for x in range(2, int(num ** (1/3)) + 1):
for y in range(x + 1, num // x + 1):
if x ** y + y ** x == num:
return True
return False
# Example usage:
print(is_leyland_number(17))
# Output: True
print(is_leyland_number(30))
# Output: True
print(is_leyland_number(100))
# Output: False
```

199. GENERATE A RANDOM UUID

The **random_uuid** function generates a random UUID by substituting placeholders in the UUID template with random hexadecimal values.

```
    import random
    import string
    def random_uuid():
    random_hex = lambda length: ''.join(random.choice(string.hexdigits) for _ in range(length))
    return f"{random_hex(8)}-{random_hex(4)}-4{random_hex(3)}-{random_hex(4)}-{random_hex(12)}".lower()
    # Example usage:
    print(random_uuid())
    # Output: a0f768f5-6bf2-4f6b-a512-c9121ea1b44a
```

200. CHECK IF A STRING IS A VALID IPV6 ADDRESS

The **is_valid_ipv6** function checks is a string is a valid IPV6 address.

```
import re
def is_valid_ipv6(ip):
pattern = r"^(?:[0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}$"
return bool(re.match(pattern, ip))
# Test cases
print(is_valid_ipv6("2001:0db8:85a3:0000:0000:8a2e:0370:7334"))
# Output: True
print(is_valid_ipv6("2001:0db8:85a3::8a2e:0370:7334"))

# Output: True
print(is_valid_ipv6("256.0.0.0"))
# Output: False
```

201. CALCULATE THE AREA OF A PARALLELOGRAM

The **parallelogram_area** function calculates the area of a parallelogram.

```
def parallelogram_area(base_length, height):
return base_length * height
# Test case
print(parallelogram_area(5, 10))
# Output: 50
```

202. CHECK IF A STRING IS A VALID MAC ADDRESS

The **is_valid_mac_address** function checks if a string is a valid MAC Address.

```
import re
def is_valid_mac_address(mac):
pattern = r'^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$'
return bool(re.match(pattern, mac))
# Test cases
print(is_valid_mac_address("00:1A:2B:3C:4D:5E"))
# Output: True
print(is_valid_mac_address("00:1A:2B:3C:4D"))
# Output: False
```

203. CONVERT RGB TO HSL (HUE, SATURATION, LIGHTNESS)

The **rgb_to_hsl** function converts an RGB color value to its corresponding HSL representation (Hue, Saturation, Lightness).

```
import colorsys
def rgb_to_hsl(rgb):
r, g, b = rgb
h, l, s = colorsys.rgb_to_hls(r / 255.0, g / 255.0, b / 255.0)
return h * 360, s * 100, l * 100
# Example usage:
rgb_value = (255, 0, 0)  # Red color
hsl_value = rgb_to_hsl(rgb_value)
print(hsl_value)
# Output: (0.0, 100.0, 50.0)
```

204. CHECK IF A NUMBER IS A PANDIGITAL NUMBER

The **is_pandigital_number** function checks if a given number is a pandigital number.

```
def is_pandigital_number(num):
num_str = str(num)
return len(set(num_str)) == len(num_str) and '0' not in num_str and max(num_str) == str(len(num_str))
# Test cases
print(is_pandigital_number(123456789))
# Output: True
print(is_pandigital_number(987654321))
# Output: True
print(is_pandigital_number(1023456789))
# Output: False
```

205. NEUTRALIZE STRINGS INTERACTION

The **neutralise** function takes two strings composed of '+' and '-' characters and returns a new string that represents the interaction between the two input strings. If the characters at corresponding positions in both strings are the same, it adds that character to the output string. Otherwise, it adds '0' to the output string.

```
def neutralise(s1, s2):
return ''.join(c1 if c1 == c2 else '0' for c1, c2 in zip(s1, s2))
# Test case
s1 = "++-+"
s2 = "+-++"
print(neutralise(s1, s2))
# Output: +00+
```

206. CONVERT YEN TO USD

The **yen_to_usd** function converts Yen (Japanese currency) to USD (American currency) using the exchange rate of 1 USD = 107.5 Yen. It returns the converted amount rounded to two decimal places.

```
def yen_to_usd(yen):
return round(yen / 107.5, 2)
# Test case
yen_amount = 1000
print(yen_to_usd(yen_amount))
# Output: 9.3
```

207. CALCULATE WAR OF NUMBERS

The **war_of_numbers** takes an array of integers, sums up the even and odd numbers separately, and then subtracts the smaller sum from the larger one. It returns the absolute value of this subtraction.

```
def war_of_numbers(arr):
even_sum = sum(x for x in arr if x % 2 == 0)
odd_sum = sum(x for x in arr if x % 2 != 0)
return abs(even_sum - odd_sum)
# Test case
numbers = [1, 2, 3, 4, 5, 6]
print(war_of_numbers(numbers))
# Output: 3
```

208. CALCULATE ITERATED SQUARE ROOT

The **isqrt** function that takes an integer as input and returns the number of times its iterated square root can be taken until the result is less than 2. If the input number is negative, it returns "invalid".

```
import math
def isqrt(n):
if n < 0:
return "invalid"
else:
return str(int(math.log(math.sqrt(n), 2)))
# Example usage
print(isqrt(16))  # Output: 2
print(isqrt(25))  # Output: 2
print(isqrt(256)) # Output: 4
print(isqrt(-25)) # Output: invalid
```

209. DETERMINE ROCK, PAPER, SCISSORS WINNER

The **RPS** function determines the winner in Rock, Paper, Scissors game.

```
    def rps(p1, p2):
    if p1 == p2:
    return "It's a draw"
    elif (p1 == "Rock" and p2 == "Scissors") or (p1 == "Scissors" and p2 == "Paper") or (p1 == "Paper" and p2 == "Rock"):
    return "The winner is p1"
    else:
    return "The winner is p2"
    # Example usage
    print(rps("Rock", "Paper"))
    # Output: The winner is p2
    print(rps("Scissors", "Paper"))
    # Output: The winner is p1
    print(rps("Rock", "Rock"))
    # Output: It's a draw
```

210. REPLACE SAUSAGES WITH "WURST”

The **wurst_is_better** function takes a string representing a sentence or text containing various types of sausages.

```
import re

def wurst_is_better(text):

return
re.sub(r'kielbasa|chorizo|moronga|salami|sausage|andouille|naem|merguez|gurka|snorkers|pepperoni', 'Wurst', text, flags=re.IGNORECASE)

# Example usage

input_text = "I love kielbasa, chorizo, and pepperoni pizza."

print(wurst_is_better(input_text))

# Output: I love Wurst, Wurst, and Wurst pizza.
```

211. UPDATE AGES AFTER YEARS

The **after_n_years** function updates the ages of the people after a specified number of years have passed.

```
    def after_n_years(people, years):
    return {name: age + abs(years) for name, age in people.items()}
    # Example usage
    ages = {
    "John": 30,
    "Alice": 25,
    "Bob": 40
    }
    updated_ages = after_n_years(ages, 5)
    for name, age in updated_ages.items():
    print(f"{name}: {age}")
    # John: 35
    # Alice: 30
    # Bob: 45
```

212. DETECT SYNCOPATION IN MUSIC

The **has_syncopation** function takes a line of music represented as a string, where hashtags '#' represent emphasized beats, and determines if the line of music contains any syncopation, returning true if it does and false otherwise.

```
    def has_syncopation(s):
    return any(c == '#' for idx, c in enumerate(s) if idx % 2 == 1)
    # Example usage
    print(has_syncopation("#.#.#"))
    # Output: True
    print(has_syncopation("##.#.#"))
    # Output: True
    print(has_syncopation("###"))
    # Output: False
```

213. EXTEND VOWELS IN A WORD

The **extend_vowels** function extend the vowels in a word.

```
import re
def extend_vowels(word, num):
if num < 0 or not isinstance(num, int):
return "invalid"
return re.sub(r'[aeiouAEIOU]', lambda x: x.group(0) * (num + 1), word)
# Example usage
print(extend_vowels("hello", 2))
# Output: "heellloo"
print(extend_vowels("world", -1))
# Output: "invalid"
```

214. GENERATE A RANDOM ALPHANUMERIC STRING

The **random_alphanumeric_string** function generates a random alphanumeric string.

```
import random
import string
def random_alphanumeric_string(length):
return ''.join(random.choices(string.ascii_letters + string.digits, k=length))
# Example usage
print(random_alphanumeric_string(8))
# Output: "Yw83XmLb"
```

215. CALCULATE THE AREA OF A REGULAR HEXAGON

The **regular_hexagon_area** function calculates the area of a regular hexagon.

```
import math
def regular_hexagon_area(side_length):
return (3 * math.sqrt(3) * side_length ** 2) / 2
# Example usage
print(regular_hexagon_area(5))
# Output: 64.9519052838329
```

216. CALCULATE CUBE DIAGONAL FROM VOLUME

The **cube_diagonal** function takes the volume of a cube as input and returns the length of the cube's main diagonal, rounded to two decimal places. It uses the formula for the length of the diagonal of a cube, which is the cube root of the volume multiplied by the square root of 3.

```
import math
def cube_diagonal(volume):
return round(math.pow(volume, 1/3) * math.sqrt(3), 2)
# Example usage
volume = 27
print(cube_diagonal(volume))
# Output: 5.2
```

217. BIGINT DECIMAL STRING FORMATTER

The **format_bigint** accepts a BigInt and a desired number of decimals, returning a string representation with the correct precision.

```
    def format_bigint(big_number, decimals):
    big_number_str = str(big_number)
    return big_number_str[:len(big_number_str) - decimals] +
"." + big_number_str[len(big_number_str) - decimals:]
    # Example usage
    number = 12345678901234567890123456789O
    num_decimals = 5
    formatted_number = format_bigint(number,
num_decimals)
    print(formatted_number)
    # Output: 123456789012345678901 23.45678
```

218. CHECK IF A NUMBER IS A REVERSIBLE NUMBER

The **is_reversible_number** function checks if a number is a reversible number.

```
def is_reversible_number(num):
return num + int(str(num)[::-1]) == sum(int(digit) for digit in str(num))
# Test cases
print(is_reversible_number(36))
# Output: True
print(is_reversible_number(45))
# Output: True
print(is_reversible_number(10))
# Output: False
```

219. CALCULATE THE CIRCUMFERENCE OF A CIRCLE

The **circle_circumference** function calculates the circumference of a circle.

```
import math
def circle_circumference(radius):
return 2 * math.pi * radius
# Test case
print(circle_circumference(5))
# Output: 31.41592653589793
```

220. FIND THE SHORTEST WORD IN A STRING

The **shortest_word** function finds the shortest word in a string.

```
def shortest_word(string):
return min(string.split(), key=len)
# Test case
print(shortest_word("This is a test sentence"))
# Output: "a"
```

221. FIND THE LONGEST WORD LENGTH IN A STRING

The **longest_word_length** function finds the longest word length in a string.

```
def longest_word_length(string):
return max(len(word) for word in string.split())
# Test case
print(longest_word_length("This is a test sentence"))
# Output: 8
```

222. FIND THE SUM OF PROPER DIVISORS OF A NUMBER

The **sum_of_proper_divisors** function finds the sum of proper divisors of a number.

```
def sum_of_proper_divisors(num):
return sum(i for i in range(1, num) if num % i == 0)
# Test cases
print(sum_of_proper_divisors(28))
# Output: 28
print(sum_of_proper_divisors(12))
# Output: 16
```

223. CHECK IF A NUMBER IS A UNITARY PERFECT NUMBER

The **is_unitary_perfect_number** function checks if a number is a unitary perfect number.

```
def is_unitary_perfect_number(num):
def gcd(a, b):
return a if b == 0 else gcd(b, a % b)
return num == sum(i for i in range(1, num) if num % i == 0 and gcd(num, i) == 1)
# Test cases
print(is_unitary_perfect_number(18))
# Output: False
print(is_unitary_perfect_number(28))
# Output: True
```

224. CALCULATE THE PERIMETER OF A REGULAR POLYGON

The **RegularPolygonPerimeter** function calculates the perimeter of a regular polygon.

```
def regular_polygon_perimeter(side_length, num_sides):
return side_length * num_sides
# Test case
print(regular_polygon_perimeter(5, 6))
# Output: 30
```

225. CALCULATE THE AREA OF AN EQUILATERAL TRIANGLE

The **equilateral_triangle_area** function calculates the area of an equilateral triangle.

```
import math
def equilateral_triangle_area(side_length):
return (math.sqrt(3) * side_length ** 2) / 4
# Test case
print(equilateral_triangle_area(5))
# Output: 10.825317547305485
```

226. CHECK IF A NUMBER IS A HARSHAD SMITH NUMBER

The **IsHarshadSmithNumber** function checks if a given number is both a Harshad number and a Smith number.

```
def is_harshad_smith_number(num):
return all(num % i != 0 for i in range(2, int(num ** 0.5) + 1)) and sum(int(d) for d in str(num)) == num and sum(int(d) for d in str(num)) == sum(sum(int(d) for d in str(f)) for f in ([i for i in range(2, num) if num % i == 0]))
# Test cases
print(is_harshad_smith_number(22))
# Output: True
print(is_harshad_smith_number(10))
# Output: False
```

227. CHECK IF A NUMBER IS A PERFECT POWER

The **is_perfect_power** function checks if a number is a perfect power.

```
def is_perfect_power(num):
return any(base**exponent == num for base in range(2, int(num**0.5) + 1) for exponent in range(2, int(num**0.5) + 1))
# Test cases
print(is_perfect_power(64))
# Output: True
print(is_perfect_power(25))
# Output: False
```

228. DROP ELEMENTS FROM ARRAY

The **drop** function takes an array and an optional value indicating the number of elements to drop from the beginning of the array. It returns a new array containing the remaining elements after dropping the specified number of elements.

```
    def drop(arr, val=1):
    return arr[val:]
    # Test cases
    print(drop([1, 2, 3]))  # Output: [2, 3]
    print(drop([1, 2, 3], 2))  # Output: [3]
    print(drop([1, 2, 3], 5))  # Output: []
    print(drop([1, 2, 3], 0))  # Output: [1, 2, 3]
    print(drop(["banana", "orange", "watermelon", "mango"],
2))  # Output: ['watermelon', 'mango']
    print(drop([], 2))  # Output: []
```

229. MAXIMUM TOTAL OF LAST FIVE ELEMENTS IN AN ARRAY

The **max_total** function calculates the maximum total of the last five elements in an integer array.

```
def max_total(nums):
return sum(sorted(nums)[-5:])
# Test cases
print(max_total([1, 1, 0, 1, 3, 10, 10, 10, 10, 1]))
# Output: 43
print(max_total([0, 0, 0, 0, 0, 0, 0, 0, 0, 100]))
# Output: 100
print(max_total([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
# Output: 40
print(max_total([12, 8, 73, 1, 24, 11, 88, 39, 2, -47]))
# Output: 236
print(max_total([48, 90, 42, -12, 1, -14, -36, -37, -9, -4]))
# Output: 177
```

230. CALCULATE THE AREA OF A REGULAR PENTAGON

The **regular_pentagon_area** function calculates the area of a regular pentagon.

```
import math
def regular_pentagon_area(side_length):
return (1 / 4) * math.sqrt(5 * (5 + 2 * math.sqrt(5))) * side_length ** 2
# Test case
print(regular_pentagon_area(5))
# Output: 43.01193501472417
```

231. CALCULATE THE VOLUME OF A PYRAMID

The **pyramid_volume** function calculates the volume of a pyramid.

```
def pyramid_volume(base_area, height):
return (1 / 3) * base_area * height
# Test case
print(pyramid_volume(25, 10))
# Output: 83.33333333333333
```

232. CHECK IF A NUMBER IS A WEDDERBURN-ETHERINGTON NUMBER

The **is_wedderburn_etherington_number** function checks if a given number is a Wedderburn-Etherington number.

```
def is_wedderburn_etherington_number(num):
return all(num % i != 0 for i in range(2, num)) and \
functools.reduce(operator.mul, (functools.reduce(operator.mul, range(1, n)) for n in range(2, num)), 1) == \
functools.reduce(operator.mul, range(1, num), 1)
# Test cases
print(is_wedderburn_etherington_number(6))
# Output: True
print(is_wedderburn_etherington_number(12))
# Output: False
```

233. CALCULATE THE SURFACE AREA OF A CUBE

The **cube_surface_area** function calculates the surface area of a cube.

```
def cube_surface_area(side_length):
return 6 * side_length ** 2
# Test case
print(cube_surface_area(5))
# Output: 150
```

234. FIND THE SECOND LARGEST NUMBER IN AN ARRAY

The **second_largest** takes an array of numbers and returns the second largest number.

```
def second_largest(arr):
sorted_arr = sorted(arr, reverse=True)
return sorted_arr[1]
# Test case
arr = [1, 3, 5, 7, 9]
print(second_largest(arr))
# Output: 7
```

235. CALCULATE THE AREA OF A REGULAR OCTAGON

The **regular_octagon_area** function calculates the area of a regular octagon.

```
def regular_octagon_area(side_length):
return 2 * (1 + 2 ** 0.5) * side_length ** 2
# Test
print(regular_octagon_area(5))
# Output: 86.60254037844387
```

236. CHECK IF A NUMBER IS A REPUNIT NUMBER

The **is_repunit_number** function checks if a number is a repunit number.

```
import re
def is_repunit_number(num):
return bool(re.match("^1+$", str(num)))
# Test
print(is_repunit_number(111))
# Output: True
print(is_repunit_number(11))
# Output: False
```

237. CALCULATE THE VOLUME OF AN ELLIPSOID

The **ellipsoid_volume** function calculates the volume of an ellipsoid.

```
import math
def ellipsoid_volume(a, b, c):
return (4 / 3) * math.pi * a * b * c
# Test
print(ellipsoid_volume(5, 3, 2))
# Output: 125.66370614359172
```

238. CHECK IF A STRING IS A VALID URL

The **is_valid_url_alt** function checks if a given string is a valid URL using an alternative approach.

```
import re
def is_valid_url_alt(url):
return bool(re.match(r"^(ftp|http|https):\/\/[^ ]+$", url))
# Test
print(is_valid_url_alt("https://www.example.com"))
# Output: True
print(is_valid_url_alt("invalid url"))
# Output: False
```

239. CHECK IF A STRING IS A VALID TAX IDENTIFICATION NUMBER (TIN)

The **is_valid_tin** function check if a given string is a valid Tax Identification Number (TIN).

```
import re
def is_valid_tin(tin):
return bool(re.match(r"^[A-Z]{2}\d{6}[A-Z\d]{2}$", tin))
# Test
print(is_valid_tin("AB123456CD"))
# Output: True
print(is_valid_tin("invalid tin"))
# Output: False
```

240. CHECK IF A STRING IS A VALID ISBN (INTERNATIONAL STANDARD BOOK NUMBER)

The **is_valid_isbn** function checks if a given string is a valid International Standard Book Number (ISBN).

```
import re
def is_valid_isbn(isbn):
return bool(re.match(r"^(?:\d{9}[\dX]|(?:\d{3}-){2}\d{1}[\dX])$", isbn))
# Test
print(is_valid_isbn("123456789"))
# Output: True
print(is_valid_isbn("invalid isbn"))
# Output: False
```

241. CHECK IF A STRING IS A VALID IP ADDRESS

The **is_valid_ip_address** function checks if a given string is a valid IP address.

```
import socket
def is_valid_ip_address(ip):
try:
socket.inet_aton(ip)
return True
except socket.error:
return False
# Test
print(is_valid_ip_address("192.168.1.1"))
# Output: True
print(is_valid_ip_address("invalid ip"))
# Output: False
```

242. REVERSE A STRING (USING RECURSION)

The **ReverseStringRecursive** function reverse a string (using recursion).

```
    def reverse_string_recursive(s):
    return s if len(s) <= 1 else reverse_string_recursive(s[1:])
+ s[0]
    # Test
    print(reverse_string_recursive("hello"))
    # Output: olleh
```

243. COUNT THE OCCURRENCES OF EACH ELEMENT IN AN ARRAY

The **count_occurrences** function counts the occurrences of each element in an array.

```
from collections import Counter
def count_occurrences(arr):
return dict(Counter(arr))
# Test
print(count_occurrences([1, 2, 1, 3, 2, 4, 1]))
# Output: {1: 3, 2: 2, 3: 1, 4: 1}
```

244. CHECK IF TWO ARRAYS ARE EQUAL

The **arrays_are_equal** function checks if two arrays are equal (shallow comparison).

```
def arrays_are_equal(arr1, arr2):
return arr1 == arr2
# Test
print(arrays_are_equal([1, 2, 3], [1, 2, 3]))
# Output: True
print(arrays_are_equal([1, 2, 3], [1, 2, 4]))
# Output: False
```

245. FIND THE MINIMUM VALUE IN AN ARRAY

The function **find_min_value** finds the minimum value in a given array of numbers.

```
def find_min_value(arr):
return min(arr)
# Test
numbers = [2, 7, 1, 9, 4]
print(find_min_value(numbers))
# Output: 1
```

246. FLATTEN AN ARRAY OF NESTED ARRAYS

The **flatten_array** function flatten an array of nested arrays (using concat).

```
def flatten_array(arr):
return [item for sublist in arr for item in sublist]
# Test
nested_arrays = [[1, 2], [3, 4], [5, 6]]
print(", ".join(map(str, flatten_array(nested_arrays))))
# Output: 1, 2, 3, 4, 5, 6
```

247. FIND THE AVERAGE OF NUMBERS IN AN ARRAY

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

```
def find_average(arr):
return sum(arr) / len(arr)
# Test
numbers = [1, 2, 3, 4, 5]
print(find_average(numbers))
# Output: 3.0
```

248. SUM THE SQUARES OF NUMBERS IN AN ARRAY

The **SumSquares** function sum the squares of numbers in an array.

```
def sum_squares(arr):
return sum(num * num for num in arr)
# Test
numbers = [1, 2, 3, 4, 5]
print(sum_squares(numbers))
# Output: 55
```

249. CHECK IF A STRING IS A PALINDROME (IGNORING NON-ALPHANUMERIC CHARACTERS)

The **is_palindrome_ignoring_non_alphanumeric** function checks if a string is a palindrome (ignoring non-alphanumeric characters).

```
import re
def is_palindrome_ignoring_non_alphanumeric(s):
# Remove non-alphanumeric characters and convert to lowercase
cleaned_str = re.sub(r'[^a-z0-9]', '', s.lower())
# Check if the cleaned string is equal to its reversed version
return cleaned_str == cleaned_str[::-1]
# Test
print(is_palindrome_ignoring_non_alphanumeric("A man, a plan, a canal, Panama!"))
# Output: True
```

250. FIND BOB IN A LIST

The **find_bob** function takes an array of strings representing names and returns the index of the name "Bob" in the array. If "Bob" is not found, it returns -1.

```
def find_bob(names):
try:
return names.index("Bob")
except ValueError:
return -1
# Test cases
names1 = ["Jimmy", "Layla", "Mandy"]
names2 = ["Bob", "Nathan", "Hayden"]
names3 = ["Paul", "Layla", "Bob"]
names4 = ["Garry", "Maria", "Bethany", "Bob", "Pauline"]
print(find_bob(names1))  # -1
print(find_bob(names2))  # 0
print(find_bob(names3))  # 2
print(find_bob(names4))  # 3
```

251. CALCULATE THE VOLUME OF A BOX

The **box_volume** function calculates the volume of a box.

```
def box_volume(length, width, height):
return length * width * height
# Test case
length = 5
width = 3
height = 2
volume = box_volume(length, width, height)
print(f"Volume of the box: {volume}")
# Output: Volume of the box: 30
```

252. MOVE ZEROS TO THE END

The **move_zeros** function converts the given array such that all the zeros are moved to the end while maintaining the order of the non-zero elements.

```
def move_zeros(arr):
non_zero_elements = [x for x in arr if x != 0]
zero_count = arr.count(0)
return non_zero_elements + [0] * zero_count
# Test cases
result = move_zeros([1, 2, 0, 1, 0, 1, 0, 3, 0, 1])
print(','.join(map(str, result)))
# Output: 1,2,1,1,3,1,0,0,0,0
result = move_zeros([9, 0, 0, 9, 1, 2, 0, 1, 0, 1, 0, 3, 0, 1,
9, 0, 0, 0, 0, 9])
print(','.join(map(str, result)))
# Output: 9,9,1,2,1,1,3,1,9,9,0,0,0,0,0,0,0,0,0
# Additional test cases here...
```

253. FIND THE MEDIAN OF NUMBERS IN AN ARRAY

The **find_median** function finds the median of numbers in an array.

```
def find_median(arr):
sorted_arr = sorted(arr)
middle = len(sorted_arr) // 2
return (sorted_arr[middle] + sorted_arr[~middle]) / 2 if len(sorted_arr) % 2 == 0 else sorted_arr[middle]
# Test case
print(find_median([1, 3, 2, 4, 5]))
# Output: 3
```

254. COUNT THE VOWELS IN A STRING

The **count_vowels** function counts the vowels in a string.

```
import re
def count_vowels(string):
return len(re.findall(r'[aeiouAEIOU]', string))
# Test case
print(count_vowels("Hello, how are you?"))
# Output: 7
```

255. CALCULATE VOTE DIFFERENCE

The **get_vote_count** function takes a dynamic object representing the votes, extracts the upvote and downvote counts, and returns the difference between them.

```
    def get_vote_count(votes):
    return (votes.get('upvotes', 0) or 0) -
(votes.get('downvotes', 0) or 0)
    # Test cases
    print(get_vote_count({'upvotes': 13, 'downvotes': 0}))  #
13
    print(get_vote_count({'upvotes': 2, 'downvotes': 33}))  #
-31
    print(get_vote_count({'upvotes': 132, 'downvotes': 132}))
 # 0
    print(get_vote_count({'upvotes': 0, 'downvotes': 0}))  # 0
    print(get_vote_count({'downvotes': 4, 'upvotes': 1}))  # -3
```

256. CHATROOM STATUS

The **chatroom_status** determines the status of a chatroom based on the number of users online. If there are no users, it returns "no one online". If there is only one user, it returns their username followed by "online". If there are two users, it returns both usernames followed by "online". If there are more than two users, it returns the usernames of the first two users followed by the count of remaining users and "more online".

```
    def chatroom_status(users):
    return "no one online" if not users else f"{users[0]} online" if len(users) == 1 else f"{users[0]} and {users[1]} online" if len(users) == 2 else f"{users[0]}, {users[1]} and {len(users) - 2} more online"
    # Test cases
    print(chatroom_status([]))  # "no one online"
    print(chatroom_status(["becky325"]))  # "becky325 online"
```

```
    print(chatroom_status(["becky325", "malcolm888"]))  # "becky325 and malcolm888 online"
    print(chatroom_status(["becky325", "malcolm888", "fah32fa"]))  # "becky325, malcolm888 and 1 more online"
    print(chatroom_status(["paRIE_to"]))  # "paRIE_to online"
    print(chatroom_status(["s234f", "mailbox2"]))  # "s234f and mailbox2 online"
    print(chatroom_status(["pap_ier44", "townieBOY", "panda321", "motor_bike5", "sandwichmaker833", "violinist91"]))  # "pap_ier44, townieBOY and 4 more online"
```

257. FIND THE ASCII VALUE OF A CHARACTER

The **get_ascii_value** function finds the ASCII value of a character.

```
def get_ascii_value(c):
return ord(c)
# Test case
print(get_ascii_value('A'))
# Output: 65
```

258. CHECK IF A STRING IS AN ISOGRAM (NO REPEATING CHARACTERS)

The **IsIsogram** function checks if a string is an isogram (no repeating characters)

```
def is_isogram(string):
return len(set(string.lower())) == len(string)
# Test cases
print(is_isogram("hello"))
# Output: False
print(is_isogram("world"))
# Output: True
```

259. CALCULATE THE HAMMING DISTANCE OF TWO STRINGS (EQUAL LENGTH)

The **hamming_distance** function calculates the hamming distance of two strings (equal length).

```
def hamming_distance(str1, str2):
if len(str1) != len(str2):
raise ValueError("Strings must have equal length")
return sum(c1 != c2 for c1, c2 in zip(str1, str2))
# Test case
print(hamming_distance("karolin", "kathrin"))
# Output: 3
```

260. CALCULATE THE DISTANCE BETWEEN TWO POINTS IN A 2D PLANE

The **calculate_distance** function calculates the distance between two points in a 2D plane.

```
import math
def calculate_distance(point1, point2):
x1, y1 = point1
x2, y2 = point2
return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# Test case
print(calculate_distance([0, 0], [3, 4]))
# Output: 5
```

261. CHECK IF A STRING IS A POSITIVE NUMBER (NO SIGN OR DECIMAL ALLOWED)

The **is_positive_number** function checks if a string is a positive number (no sign or decimal allowed)

```
import re
def is_positive_number(string):
return bool(re.match(r'^\d+$', string))
# Test cases
print(is_positive_number("123"))
# Output: True
print(is_positive_number("-123"))
# Output: False
```

262. FIND THE FIRST NON-REPEATING CHARACTER IN A STRING

The **find_first_non_repeating_character** function finds the first non-repeating character in a string.

```
def find_first_non_repeating_character(string):
for char in string:
if string.count(char) == 1:
return char
return None  # Return None if no non-repeating character is found
# Test case
print(find_first_non_repeating_character("hello"))
# Output: 'h'
```

263. CALCULATE THE AREA OF A KITE

The **area_of_kite** function calculates the area of a kite.

```
def area_of_kite(d1, d2):
return 0.5 * d1 * d2
# Test case
print(f"Area of the kite: {area_of_kite(10, 6)}")
# Output: Area of the kite: 30.0
```

264. CALCULATE THE AREA OF A SECTOR

The **sector_area** function calculates the area of a sector within a circle based on the provided radius and angle.

```
import math
def sector_area(radius, angle):
return math.pi * radius**2 * angle / 360.0
# Test case
print(f"Area of the sector: {sector_area(5, 60)}")
# Output: Area of the sector: 5.235987755982989
```

In every line of code, they have woven a story of innovation and creativity. This book has been your compass in the vast world of Python.

Close this chapter knowing that every challenge overcome is an achievement, and every solution is a step toward mastery.

Your code is the melody that gives life to projects. May they continue creating and programming with passion!

Thank you for allowing me to be part of your journey.

With gratitude,

Hernando Abella

Author of 250+ Killer Python One-Liners

Discover Useful Resources @:

www.hernandoabella.com

www.beat-byte-publishing.com