140+ Basic Python Programs

This resource can assist you in preparing for your interview

Piush Kumar Sharma

11/26/23, 4:53 AM Basic Python Program - Jupyter Notebook

Program 1

Write a Python program to print "Hello Python".

In [1]:
```
print("Hell
o Python")
```

```
Hello Python
```

Program 2

Write a Python program to do arithmetical operations addition and division.

In [2]:
```
# Addition
num1 = float(input("Enter the first number for addition: "))
num2 = float(input("Enter the second number for addition: "))
sum_result = num1 + num2
print(f"sum: {num1} + {num2} = {sum_result}")
```

```
Enter the first number for addition: 5
Enter the second number for addition: 6
sum: 5.0 + 6.0 = 11.0
```

In [3]:
```
# Division
num3 = float(input("Enter the dividend for division: "))
num4 = float(input("Enter the divisor for division: "))
if num4 == 0:
    print("Error: Division by zero is not allowed.")
else:
    div_result = num3 / num4
    print(f"Division: {num3} / {num4} = {div_result}")
```

```
Enter the dividend for division: 25
Enter the divisor for division: 5
Division: 25.0 / 5.0 = 5.0
```

Program 3

Write a Python program to find the area of a triangle.

In [4]:
```
# Input the base and height from the user
base = float(input("Enter the length of the base of the triangle: "))
height = float(input("Enter the height of the triangle: "))

# Calculate the area of the triangle
area = 0.5 * base * height

# Display the result
print(f"The area of the triangle is: {area}")
```

```
Enter the length of the base of the triangle: 10
Enter the height of the triangle: 15
The area of the triangle is: 75.0
```


Program 4

Write a Python program to swap two variables.

In [5]:

```
# Input two variables
a = input("Enter the value of the first variable (a): ")
b = input("Enter the value of the second variable (b): ")
# Display the original values
print(f"Original values: a = {a}, b = {b}")
# Swap the values using a temporary variable
temp = a
a = b
b = temp
# Display the swapped values
print(f"Swapped values: a = {a}, b = {b}")
```

```
Enter the value of the first variable (a): 5
Enter the value of the second variable (b): 9
Original values: a = 5, b = 9
Swapped values: a = 9, b = 5
```

Program 5

Write a Python program to generate a random number.

In [6]:

```
import random
print(f"Random number: {random.randint(1, 100)}")
```

```
Random number: 89
```

Program 6

Write a Python program to convert kilometers to miles.

In [7]:

```
kilometers = float(input("Enter distance in kilometers: "))
# Conversion factor: 1 kilometer = 0.621371 miles
conversion_factor = 0.621371
miles = kilometers * conversion_factor
print(f"{kilometers} kilometers is equal to {miles} miles")
```

```
Enter distance in kilometers: 100
100.0 kilometers is equal to 62.137100000000004 miles
```

Program 7

Write a Python program to convert Celsius to Fahrenheit.

In [8]:

```
celsius = float(input("Enter temperature
```

```
in Celsius: ")) 2
3
# Conversion formula: Fahrenheit =
(Celsius * 9/5) + 32
4
fahrenheit = (celsius * 9/5) + 32
5
6
print(f"{celsius} degrees Celsius is equal
to {fahrenheit} degrees Fahr
```

```
Enter temperature in Celsius: 37
37.0 degrees Celsius is equal to 98.6 degrees Fahrenheit
```

Program 8

Write a Python program to display calendar.

```
In [9]:
1
import calendar
2
3
year = int(input("Enter
year: ")) 4
month = int(input("Enter
month: ")) 5
6
cal =
calendar.month(year,
month) 7
print(cal)
```

```
Enter year: 2023
Enter month: 11
  November 2023
Mo Tu We Th Fr Sa Su
  1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Program 9

Write a Python program to solve quadratic equation.

The standard form of a quadratic equation is:

$$\text{□□□□}^2 + \text{□□□□} + \text{□□} = 0$$

where

a, b and c are real numbers and

$\text{□□} \neq 0$

The solutions of this quadratic equation is given by:

$$(-\text{□□} \pm (\text{□□}^2 - 4\text{□□□□})^{1/2})/(2\text{□□})$$

In [10]:

```
import math

# Input coefficients
a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))

# Calculate the discriminant
discriminant = b**2 - 4*a*c

# Check if the discriminant is positive, negative, or zero
if discriminant > 0:
    # Two real and distinct roots
    root1 = (-b + math.sqrt(discriminant)) / (2*a)
    root2 = (-b - math.sqrt(discriminant)) / (2*a)
    print(f"Root 1: {root1}")
    print(f"Root 2: {root2}")
elif discriminant == 0:
    # One real root (repeated)
    root = -b / (2*a)
    print(f"Root: {root}")
else:
    # Complex roots
    real_part = -b / (2*a)
    imaginary_part = math.sqrt(abs(discriminant)) / (2*a)
    print(f"Root 1: {real_part} + {imaginary_part}i")
    print(f"Root 2: {real_part} - {imaginary_part}i")
```

```
Enter coefficient a: 1
Enter coefficient b: 4
Enter coefficient c: 8
Root 1: -2.0 + 2.0i
Root 2: -2.0 - 2.0i
```

Program 10

Write a Python program to swap two variables without temp variable.

In [11]:

```
a = 5
b = 10

# Swapping without a temporary variable
a, b = b, a

print("After swapping:")
print("a =", a)
print("b =", b)
```

```
After swapping:
a = 10
b = 5
```

Program 11

Write a Python Program to Check if a Number is Positive, Negative or Zero.

```
In [12]:
num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

```
Enter a number: 6.4
Positive number
```

Program 12

Write a Python Program to Check if a Number is Odd or Even.

```
In [13]:
num = int(input("Enter a number: "))
if num%2 == 0:
    print("This is a even number")
else:
    print("This is a odd number")
```

```
Enter a number: 3
This is a odd number
```

Program 13

Write a Python Program to Check Leap Year.

```
In [14]:
year = int(input("Enter a year: "))

# divided by 100 means century year (ending with 00)
# century year divided by 400 is leap year
if (year % 400 == 0) and (year % 100 == 0):
    print("{0} is a leap year".format(year))

# not divided by 100 means not a century year
# year divided by 4 is a leap year
elif (year % 4 ==0) and (year % 100 != 0):
    print("{0} is a leap year".format(year))

# if not divided by both 400 (century year) and 4 (not century year)
# year is not leap year
```

```
else:
16
```

```
 print("{0} is not a leap
year".format(year))
```

```
Enter a year: 2024
2024 is a leap year
```


Program 14

Write a Python Program to Check Prime Number.

Prime Numbers:

A prime number is a whole number that cannot be evenly divided by any other number except for 1 and itself. For example, 2, 3, 5, 7, 11, and 13 are prime numbers because they cannot be divided by any other positive integer except for 1 and their own value.

```
In [15]:
1
num = int(input("Enter a number: "))
2
3
# define a flag variable
4
flag = False
5
6
if num == 1:
7
 print(f"{num}, is not a prime number")
8
elif num > 1:
9
 # check for factors
10
 for i in range(2, num):
```

```
11
 if (num % i) == 0:
12
 flag = True # if factor is found, set
flag to True 13
 # break out of loop
14
 break
15
16
 # check if flag is True
17
if flag:
18
 print(f"{num}, is not a prime number")
19
else:
20
 print(f"{num}, is a prime number")
```

```
Enter a number: 27
27, is not a prime number
```

Program 15 ¶

Write a Python Program to Print all Prime Numbers in an Interval of 1-10.

In [20]:

```
# Python program to display all the prime numbers within an interval

lower = 1
upper = 10

print("Prime numbers between", lower, "and", upper, "are:")

for num in range(lower, upper + 1):
 # all prime numbers are greater than 1
 if num > 1:
 for i in range(2, num):
 if (num % i) == 0:
 break
 else:
 print(num)
```

```
Prime numbers between 1 and 10 are:
2
3
5
7
```

Program 16

Write a Python Program to Find the Factorial of a Number.

In [21]:

```
num = int(input("Enter a number: "))

factorial = 1

if num <0:
 print("Factirial does not exist for negative numbers")
elif num == 0:
 print("Factorial of 0 is 1")
else:
 for i in range(1, num+1):
 factorial = factorial*i
 print(f'The factorial of {num} is {factorial}')
```

```
Enter a number: 4
The factorial of 4 is 24
```

Program 17

Write a Python Program to Display the multiplication Table.

In [22]:

```
num = int(input("Display multiplication table of: "))
for i in range(1, 11):
    print(f"{num} X {i} = {num*i}")
```

```
Display multiplication table of: 19
19 X 1 = 19
19 X 2 = 38
19 X 3 = 57
19 X 4 = 76
19 X 5 = 95
19 X 6 = 114
19 X 7 = 133
19 X 8 = 152
19 X 9 = 171
19 X 10 = 190
```

Program 18

Write a Python Program to Print the Fibonacci sequence.

Fibonacci sequence:

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, typically starting with 0 and 1. So, the sequence begins with 0 and 1, and the next number is obtained by adding the previous two numbers. This pattern continues indefinitely, generating a sequence that looks like this:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, and so on.

Mathematically, the Fibonacci sequence can be defined using the following recurrence relation:

$$F(0) = 0 \quad F(1) = 1 \quad F(n) = F(n-1) + F(n-2) \quad \text{for } n > 1$$

In [23]:

```
nterms = int(input("How many terms? "))

# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
# if there is only one term, return n1
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
# generate fibonacci sequence
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

```
How many terms? 10
Fibonacci sequence:
0
1
1
2
3
5
8
13
21
34
```

Program 19

Write a Python Program to Check Armstrong Number?

Armstrong Number:

It is a number that is equal to the sum of its own digits, each raised to a power equal to the number of digits in the number.

For example, let's consider the number 153:

It has three digits (1, 5, and 3).

1^3 5 +

^{3} 3^3 1 + 125 + 27 153

If we calculate + , we get , which is equal to .

So, 153 is an Armstrong number because it equals the sum of its digits raised to the power of the number of digits in the number.

Another example is 9474:

It has four digits (9, 4, 7, and 4).

^{4} 4^4 7^4 4^4 6561 + 256 + 2401 + 256

9 + + +

If we calculate , we get , which is also 9474
equal to .

Therefore, 9474 is an Armstrong number as well.

In [25]:

```
num = int(input("Enter a number: "))

# Calculate the number of digits in num
num_str = str(num)
num_digits = len(num_str)

# Initialize variables
sum_of_powers = 0
temp_num = num

# Calculate the sum of digits raised to the power of num_digits
while temp_num > 0:
    digit = temp_num % 10
    sum_of_powers += digit ** num_digits
    temp_num //= 10

# Check if it's an Armstrong number
if sum_of_powers == num:
    print(f"{num} is an Armstrong number.")
else:
    print(f"{num} is not an Armstrong number.")
```

```
Enter a number: 9474
9474 is an Armstrong number.
```

Program 20

Write a Python Program to Find Armstrong Number in an Interval.

In [26]:

```
# Input the interval from the user
lower = int(input("Enter the lower limit of the interval: "))
upper = int(input("Enter the upper limit of the interval: "))

for num in range(lower, upper + 1): # Iterate through the numbers i
    order = len(str(num)) # Find the number of digits in 'num'
    temp_num = num
    sum = 0

    while temp_num > 0:
        digit = temp_num % 10
        sum += digit ** order
        temp_num //= 10

    # Check if 'num' is an Armstrong number
    if num == sum:
        print(num)
```

```
Enter the lower limit of the interval: 10
Enter the upper limit of the interval: 1000
153
370
371
407
```

Program 21

Write a Python Program to Find the Sum of Natural Numbers.

Natural numbers are a set of positive integers that are used to count and order objects. They are the numbers that typically start from 1 and continue indefinitely, including all the whole numbers greater than 0. In mathematical notation, the set of natural numbers is often denoted as "N" and can be expressed as:

$$\mathbb{N} = 1, 2, 3, 4, 5, 6, 7, 8, \ldots$$

In [27]:
```
limit = int(input("Enter the limit: "))

# Initialize the sum
sum = 0

# Use a for loop to calculate the sum of natural numbers
for i in range(1, limit + 1):
    sum += i

# Print the sum
print("The sum of natural numbers up to", limit, "is:", sum)
```

```
Enter the limit: 10
The sum of natural numbers up to 10 is: 55
```


Program 22

Write a Python Program to Find LCM.

Least Common Multiple (LCM):

LCM, or Least Common Multiple, is the smallest multiple that is exactly divisible by two or more numbers.

Formula:

For two numbers a and b, the LCM can be found using the formula:

$$\text{LCM}(\mathbb{N}, \mathbb{N}) = \frac{|\mathbb{N} \cdot \mathbb{N}|}{\text{GCD}(\mathbb{N}, \mathbb{N})}$$

For more than two numbers, you can find the LCM step by step, taking the LCM of pairs of numbers at a time until you reach the last pair.

Note: GCD stands for Greatest Common Divisor.

In [1]:
```
# Python Program to find the L.C.M. of two input number

def compute_lcm(x, y):
    if x > y: # choose the greater number
        greater = x
    else:
```

```
    greater = y
    while(True):
        if((greater % x == 0) and (greater % y == 0)):
            lcm = greater
            break
        greater += 1
    return lcm

num1 = int(input('Enter the number: '))
num2 = int(input('Enter the number: '))

print("The L.C.M. is", compute_lcm(num1, num2))
```

```
Enter the number: 54
Enter the number: 24
The L.C.M. is 216
```

Program 23

Write a Python Program to Find HCF.

Highest Common Factor(HCF):

HCF, or Highest Common Factor, is the largest positive integer that divides two or more numbers without leaving a remainder.

Formula:

For two numbers a and b, the HCF can be found using the formula:

$$HCF(a, b) = GCD(a, b)$$

For more than two numbers, you can find the HCF by taking the GCD of pairs of numbers at a time until you reach the last pair.

Note: GCD stands for Greatest Common Divisor.

In [2]:

```
# Python program to find H.C.F of two numbers

# define a function
def compute_hcf(x, y):

    # choose the smaller number
    if x > y:
        smaller = y
    else:
        smaller = x
    for i in range(1, smaller+1):
        if((x % i == 0) and (y % i == 0)):
            hcf = i
    return hcf

num1 = int(input('Enter the number: '))
num2 = int(input('Enter the number: '))

print("The H.C.F. is", compute_hcf(num1, num2))
```

```
Enter the number: 54
Enter the number: 24
The H.C.F. is 6
```

Program 24

Write a Python Program to Convert Decimal to Binary, Octal and Hexadecimal.

How can we manually convert a decimal number to binary, octal and hexadecimal?

Converting a decimal number to binary, octal, and hexadecimal involves dividing the decimal number by the base repeatedly and noting the remainders at each step. Here's a simple example:

Let's convert the decimal number 27 to binary, octal, and hexadecimal.

1. **Binary:**

   Divide 27 by 2. Quotient is 13, remainder is 1. Note the remainder.
   Divide 13 by 2. Quotient is 6, remainder is 1. Note the remainder.
   Divide 6 by 2. Quotient is 3, remainder is 0. Note the remainder.
   Divide 3 by 2. Quotient is 1, remainder is 1. Note the remainder.
   Divide 1 by 2. Quotient is 0, remainder is 1. Note the remainder.

   Reading the remainders from bottom to top, the binary representation of 27 is 11011.
2. **Octal:**

   Divide 27 by 8. Quotient is 3, remainder is 3. Note the remainder.
   Divide 3 by 8. Quotient is 0, remainder is 3. Note the remainder.

   Reading the remainders from bottom to top, the octal representation of 27 is 33.
3. **Hexadecimal:**

   Divide 27 by 16. Quotient is 1, remainder is 11 (B in hexadecimal). Note the remainder.

   Reading the remainders, the hexadecimal representation of 27 is 1B.

So, in summary:

Binary: 27 in decimal is 11011 in binary.
Octal: 27 in decimal is 33 in octal.
Hexadecimal: 27 in decimal is 1B in hexadecimal.

```
In [3]:
1
dec_num = int(input('Enter a
decimal number: ')) 2
3
print("The decimal value of",
dec_num, "is:") 4
print(bin(dec_num), "in
binary.")
5
print(oct(dec_num), "in
octal.")
6
print(hex(dec_num), "in
hexadecimal.")
```

```
Enter a decimal number: 27
```

```
The decimal value of 27 is:
0b11011 in binary.
0o33 in octal.
0x1b in hexadecimal.
```

Program 25

Write a Python Program To Find ASCII value of a character.

ASCII value:

ASCII, or American Standard Code for Information Interchange, is a character encoding standard that uses numeric values to represent characters. Each ASCII character is assigned a unique 7-bit or 8-bit binary number, allowing computers to exchange information and display text in a consistent way. The ASCII values range from 0 to 127 (for 7-bit ASCII) or 0 to 255 (for 8-bit ASCII), with each value corresponding to a specific character, such as letters, digits, punctuation marks, and control characters.

In [4]:
```
1
char = str(input("Enter the character: "))
2
print("The ASCII value of '" + char + "' is", ord(char))
```

```
Enter the character: P
The ASCII value of 'P' is 80
```

Program 26

Write a Python Program to Make a Simple Calculator with 4 basic mathematical operations.

In [5]:
```
1  # This function adds two numbers
2  def add(x, y):
3      return x + y
4
5  # This function subtracts two numbers
6  def subtract(x, y):
7      return x - y
8
9  # This function multiplies two numbers
10 def multiply(x, y):
11     return x * y
12
13 # This function divides two numbers
14
```

```python
def divide(x, y):
    return x / y


print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

while True:
    # take input from the user
    choice = input("Enter choice(1/2/3/4): ")

    # check if choice is one of the four options
    if choice in ('1', '2', '3', '4'):
        try:
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
        except ValueError:
            print("Invalid input. Please enter a number.")
            continue

        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))
        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))
        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))
        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))

        # check if user wants another calculation
        # break the while loop if answer is no
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation == "no":
```

```
        break
54
    else:
55
        print("Invalid Input")
```



```
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 1
Enter first number: 5
Enter second number: 6
5.0 + 6.0 = 11.0
Let's do next calculation? (yes/no): yes
Enter choice(1/2/3/4): 2
Enter first number: 50
Enter second number: 5
50.0 - 5.0 = 45.0
Let's do next calculation? (yes/no): yes
Enter choice(1/2/3/4): 3
Enter first number: 22
Enter second number: 2
22.0 * 2.0 = 44.0
Let's do next calculation? (yes/no): yes
Enter choice(1/2/3/4): 4
Enter first number: 99
Enter second number: 9
99.0 / 9.0 = 11.0
Let's do next calculation? (yes/no): no
```

Program 27

Write a Python Program to Display Fibonacci Sequence Using Recursion.

Fibonacci sequence:

The Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones, usually starting with 0 and 1. In mathematical terms, it is defined by the recurrence relation (F(n) = F(n-1) + F(n-2)), with initial conditions (F(0) = 0) and (F(1) = 1). The sequence begins: 0, 1, 1, 2, 3, 5, 8, 13, 21, and so on. The Fibonacci sequence has widespread applications in mathematics, computer science, nature, and art.

In [9]:

```
# Python program to display the Fibonacci sequence

def recur_fibo(n):
 if n <= 1:
 return n
 else:
 return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = int(input("Enter the number of terms (greater than 0): ")) 10
# check if the number of terms is valid
if nterms <= 0:
 print("Plese enter a positive integer")
else:
 print("Fibonacci sequence:")
 for i in range(nterms):
 print(recur_fibo(i))
```

```
Enter the number of terms (greater than 0): 8
Fibonacci sequence:
0
1
1
2
3
5
8
13
```

Program 28

Write a Python Program to Find Factorial of Number Using Recursion.

The factorial of a non-negative integer (n) is the product of all positive integers less than or equal to (n). It is denoted by (n!) and is defined as:

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

For example:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$0!$ is defined to be 1.

Factorials are commonly used in mathematics, especially in combinatorics and probability, to count the number of ways a set of elements can be arranged or selected.

In [11]:

```
# Factorial of a number using recursion

def recur_factorial(n):
 if n == 1:
 return n
 else:
 return n*recur_factorial(n-1)

num = int(input("Enter the number: "))

# check if the number is negative
if num < 0:
 print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
 print("The factorial of 0 is 1")
else:
 print("The factorial of", num, "is", recur_factorial(num))
```

```
Enter the number: 7
The factorial of 7 is 5040
```

Program 29

Write a Python Program to calculate your Body Mass Index.

Body Mass Index (BMI) is a measure of body fat based on an individual's weight and height. It is commonly used as a screening tool to categorize individuals into different weight status categories, such as underweight, normal weight, overweight, and obesity.

The BMI is calculated using the following formula:

$$\text{BMI} = \frac{\text{Weight (kg)}}{\text{Height (m)}^2}$$

Alternatively, in the imperial system:

$$BMI = \frac{\text{Weight (lb)}}{\text{Height (in)}^2} \times 703$$

BMI provides a general indication of body fatness but does not directly measure body fat or distribution. It is widely used in public health and clinical settings as a quick and simple tool to assess potential health risks associated with weight. Different BMI ranges are associated with different health categories, but it's important to note that BMI has limitations and does not account for factors such as muscle mass or distribution of fat.

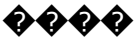
In [12]:

```
def bodymassindex(height, weight):
    return round((weight / height**2),2)

h = float(input("Enter your height in meters: "))
w = float(input("Enter your weight in kg: "))

print("Welcome to the BMI calculator.")

bmi = bodymassindex(h, w)
print("Your BMI is: ", bmi)

if bmi <= 18.5:
    print("You are underweight.")
elif 18.5 < bmi <= 24.9:
    print("Your weight is normal.")
elif 25 < bmi <= 29.29:
    print("You are overweight.")
else:
    print("You are obese.")
```

```
Enter your height in meters: 1.8
Enter your weight in kg: 70
Welcome to the BMI calculator.
Your BMI is: 21.6
Your weight is normal.
```

Program 30

Write a Python Program to calculate the natural logarithm of any number.



The **natural logarithm**, often denoted as , is a mathematical function that represents the
logarithm to the base , where is the mathematical constant approximately equal to
2.71828 . In other words, for a positive number , the natural logarithm of is the exponent
that satisfies the equation .

Mathematically, the natural logarithm is expressed as:

$$\ln(x)$$

It is commonly used in various branches of mathematics, especially in calculus and mathematical analysis, as well as in fields such as physics, economics, and engineering. The natural logarithm has properties that make it particularly useful in situations involving exponential growth or decay.

In [13]:

```
import math

num = float(input("Enter a number: "))

if num <= 0:
    print("Please enter a positive number.")
else:
    # Calculate the natural logarithm (base e) of the number
    result = math.log(num)
    print(f"The natural logarithm of {num} is: {result}")
```

```
Enter a number: 1.4
The natural logarithm of 1.4 is: 0.3364722366212129
```

Program 31

Write a Python Program for cube sum of first n natural numbers?

In [14]:

```
def cube_sum_of_natural_numbers(n):
    if n <= 0:
        return 0
    else:
        total = sum([i**3 for i in range(1, n + 1)])
        return total

# Input the number of natural numbers
n = int(input("Enter the value of n: "))
```

```
if n <= 0:
    print("Please enter a positive integer.")
else:
    result = cube_sum_of_natural_numbers(n)
    print(f"The cube sum of the first {n} natural numbers is: {result}")
```

```
Enter the value of n: 7
The cube sum of the first 7 natural numbers is: 784
```

Program 32

Write a Python Program to find sum of array.

In Python, an **array** is a data structure used to store a collection of elements, each identified by an index or a key. Unlike some other programming languages, Python does not have a built-in array type. Instead, the most commonly used array-like data structure is the list.

A list in Python is a dynamic array, meaning it can change in size during runtime. Elements in a list can be of different data types, and you can perform various operations such as adding, removing, or modifying elements. Lists are defined using square brackets `[]` and can be indexed and sliced to access specific elements or sublists.

Example of a simple list in Python:

```
my_list = [1, 2, 3, 4, 5]
```


This list can be accessed and manipulated using various built-in functions and methods in Python.

In [15]:
```
# Finding Sum of Array Using sum()
arr = [1,2,3]
ans = sum(arr)
print('Sum of the array is ', ans)
```

```
Sum of the array is 6
```

In [16]:
```
# Function to find the sum of elements in an array
def sum_of_array(arr):
    total = 0 # Initialize a variable to store the sum
    for element in arr:
        total += element # Add each element to the total
    return total

# Example usage:
array = [1, 2, 3]
result = sum_of_array(array)
print("Sum of the array:", result)
```

```
Sum of the array: 6
```

Program 33

Write a Python Program to find largest element in an array.

In [18]:
```
def find_largest_element(arr):
    if not arr:
        return "Array is empty"

    # Initialize the first element as the largest
    largest_element = arr[0]

    # Iterate through the array to find the largest element
    for element in arr:
        if element > largest_element:
            largest_element = element

    return largest_element

# Example usage:
my_array = [10, 20, 30, 99]
result = find_largest_element(my_array)
print(f"The largest element in the array is: {result}")
```

```
The largest element in the array is: 99
```


Program 34

Write a Python Program for array rotation.

In [19]:
```
def rotate_array(arr, d):
    n = len(arr)

    # Check if 'd' is valid, it should be within the range of array len
    if d < 0 or d >= n:
        return "Invalid rotation value"

    # Create a new array to store the rotated elements.
    rotated_arr = [0] * n

    # Perform the rotation.
    for i in range(n):
        rotated_arr[i] = arr[(i + d) % n]

    return rotated_arr

# Input array
arr = [1, 2, 3, 4, 5]

# Number of positions to rotate
d = 2

# Call the rotate_array function
```

```
result = rotate_array(arr, d)

# Print the rotated array
print("Original Array:", arr)
print("Rotated Array:", result)
```

```
Original Array: [1, 2, 3, 4, 5]
Rotated Array: [3, 4, 5, 1, 2]
```

Program 35

Write a Python Program to Split the array and add the first part to the end?

In [20]:

```
def split_and_add(arr, k):
    if k <= 0 or k >= len(arr):
        return arr

    # Split the array into two parts
    first_part = arr[:k]
    second_part = arr[k:]

    # Add the first part to the end of the second part
    result = second_part + first_part

    return result

# Test the function
arr = [1, 2, 3, 4, 5]
k = 3
result = split_and_add(arr, k)
print("Original Array:", arr)
print("Array after splitting and adding:", result)
```

```
Original Array: [1, 2, 3, 4, 5]
Array after splitting and adding: [4, 5, 1, 2, 3]
```

Program 36

Write a Python Program to check if given array is Monotonic. A monotonic

array is one that is entirely non-increasing or non-decreasing.

```
In [21]:
def is_monotonic(arr):
    increasing = decreasing = True

    for i in range(1, len(arr)):
        if arr[i] > arr[i - 1]:
            decreasing = False
        elif arr[i] < arr[i - 1]:
            increasing = False

    return increasing or decreasing

# Test the function
arr1 = [1, 2, 2, 3] # Monotonic (non-decreasing)
arr2 = [3, 2, 1] # Monotonic (non-increasing)
arr3 = [1, 3, 2, 4] # Not monotonic

print("arr1 is monotonic:", is_monotonic(arr1))
print("arr2 is monotonic:", is_monotonic(arr2))
print("arr3 is monotonic:", is_monotonic(arr3))
```

```
arr1 is monotonic: True
arr2 is monotonic: True
arr3 is monotonic: False
```


Program 37

Write a Python Program to Add Two Matrices.

```
In [1]:
# Function to add two matrices
def add_matrices(mat1, mat2):
    # Check if the matrices have the same dimensions
    if len(mat1) != len(mat2) or len(mat1[0]) != len(mat2[0]):
        return "Matrices must have the same dimensions for addition"

    # Initialize an empty result matrix with the same dimensions
    result = []

    for i in range(len(mat1)):
        row = []
        for j in range(len(mat1[0])):
            row.append(mat1[i][j] + mat2[i][j])
        result.append(row)

    return result

# Input matrices
matrix1 = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

```
matrix2 = [
 [9, 8, 7],
 [6, 5, 4],
 [3, 2, 1]
]

# Call the add_matrices function
result_matrix = add_matrices(matrix1, matrix2)

# Display the result
if isinstance(result_matrix, str):
 print(result_matrix)
else:
 print("Sum of matrices:")
 for row in result_matrix:
 print(row)
```

```
Sum of matrices:
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]
```

Program 38

Write a Python Program to Multiply Two Matrices.

In [2]:

```
# Function to multiply two matrices
def multiply_matrices(mat1, mat2):
 # Determine the dimensions of the input matrices
 rows1 = len(mat1)
 cols1 = len(mat1[0])
 rows2 = len(mat2)
 cols2 = len(mat2[0])

 # Check if multiplication is possible
 if cols1 != rows2:
 return "Matrix multiplication is not possible. Number of column
 # Initialize the result matrix with zeros
 result = [[0 for _ in range(cols2)] for _ in range(rows1)]

 # Perform matrix multiplication
 for i in range(rows1):
 for j in range(cols2):
 for k in range(cols1):
 result[i][j] += mat1[i][k] * mat2[k][j]

 return result

# Example matrices
matrix1 = [[1, 2, 3],
 [4, 5, 6]]

matrix2 = [[7, 8],
 [9, 10],
 [11, 12]]

# Multiply the matrices
```

```
result_matrix = multiply_matrices(matrix1, matrix2)

# Display the result
if isinstance(result_matrix, str):
    print(result_matrix)
else:
    print("Result of matrix multiplication:")
    for row in result_matrix:
        print(row)
```

```
Result of matrix multiplication:
[58, 64]
[139, 154]
```

Program 39

Write a Python Program to Transpose a Matrix.

In [3]:

```
# Function to transpose a matrix
def transpose_matrix(matrix):
    rows, cols = len(matrix), len(matrix[0])
    # Create an empty matrix to store the transposed data
    result = [[0 for _ in range(rows)] for _ in range(cols)]

    for i in range(rows):
        for j in range(cols):
            result[j][i] = matrix[i][j]

    return result

# Input matrix
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]

# Transpose the matrix
transposed_matrix = transpose_matrix(matrix)

# Print the transposed matrix
for row in transposed_matrix:
    print(row)
```

```
[1, 4]
[2, 5]
[3, 6]
```

Program 40

Write a Python Program to Sort Words in Alphabetic Order.

In [4]:

```
# Program to sort alphabetically the words form a string provided by th

my_str = input("Enter a string: ")

# breakdown the string into a list of words
words = [word.capitalize() for word in my_str.split()]

# sort the list
words.sort()

# display the sorted words

print("The sorted words are:")
for word in words:
 print(word)
```

```
Enter a string: suresh ramesh vibhuti gulgule raji ram shyam ajay
The sorted words are:
Ajay
Gulgule
Raji
Ram
Ramesh
Shyam
Suresh
Vibhuti
```

Program 41

Write a Python Program to Remove Punctuation From a String.

In [5]:

```
# define punctuation
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

# To take input from the user
my_str = input("Enter a string: ")

# remove punctuation from the string
no_punct = ""
for char in my_str:
    if char not in punctuations:
        no_punct = no_punct + char

# display the unpunctuated string
print(no_punct)
```

```
Enter a string: Hello!!!, he said ---and went
Hello he said and went
```

Program 42

In [ ]: In [ ]:

```
1
```

```
1
```

Program 43

Write a Python program to check if the given number is a Disarium Number.

A **Disarium number** is a number that is equal to the sum of its digits each raised to the power of its respective position. For example, 89 is a Disarium number because $^{1}9^{2}$ 8 + = 8 + 81 = 89.

In [1]:

```
def is_disarium(number):
    # Convert the number to a string to iterate over its digits
    num_str = str(number)

    # Calculate the sum of digits raised to their respective positions
    digit_sum = sum(int(i) ** (index + 1) for index, i in enumerate(num

    # Check if the sum is equal to the original number
    return digit_sum == number

# Input a number from the user
try:
    num = int(input("Enter a number: "))


    # Check if it's a Disarium number
    if is_disarium(num):
        print(f"{num} is a Disarium number.")
    else:
        print(f"{num} is not a Disarium number.")
```

```
except ValueError:                          print("Invalid input. Please enter a
21                                         valid number.")
```

```
Enter a number: 89
89 is a Disarium number.
```

Program 44

Write a Python program to print all disarium numbers between 1 to 100.

In [2]:

```
1                                                                    6
def is_disarium(num):                                                disarium_numbers = [num for num in
2                                                                    range(1, 101) if is_disarium(num)] 7
 num_str = str(num)                                                  8
3                                                                    print("Disarium numbers between 1 and
 digit_sum = sum(int(i) ** (index + 1) forfor num in disarium_numbers:
index, i in enumerate(num 4                                          10
 return num == digit_sum                                              print(num, end=" | ")
5
```

```
Disarium numbers between 1 and 100:
1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 89 |
```

Program 45

Write a Python program to check if the given number is Happy Number.

Happy Number: A Happy Number is a positive integer that, when you repeatedly replace the number by the sum of the squares of its digits and continue the process, eventually reaches 1. If the process never reaches 1 but instead loops endlessly in a cycle, the number is not a Happy Number.

For example:

19 is a Happy Number because:

$^{2}9^{2}$

1 + = 82

$^{2}2^{2}$

8 + = 68

$^{2}8^{2}$

6 + = 100

$^{2}0^{2}0^{2}$

1 + + = 1

The process reaches 1, so 19 is a Happy Number.

```
In [3]:
def is_happy_number(num):
    seen = set() # To store previously seen numbers
    while num != 1 and num not in seen:
        seen.add(num)
        num = sum(int(i) ** 2 for i in str(num))

    return num == 1

# Test the function with a number
num = int(input("Enter a number: "))
if is_happy_number(num):
    print(f"{num} is a Happy Number")
else:
    print(f"{num} is not a Happy Number")
```

```
Enter a number: 23
23 is a Happy Number
```


Program 46

Write a Python program to print all happy numbers between 1 and 100.

```
In [4]:
def is_happy_number(num):
    seen = set()
    while num != 1 and num not in seen:
        seen.add(num)
        num = sum(int(i) ** 2 for i in str(num))

    return num == 1

happy_numbers = []
for num in range(1, 101):
    if is_happy_number(num):
        happy_numbers.append(num)

print("Happy Numbers between 1 and 100:")
print(happy_numbers)
```

```
Happy Numbers between 1 and 100:
[1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
```

Program 47

Write a Python program to determine whether the given number is a Harshad Number.

A **Harshad number** (or Niven number) is an integer that is divisible by the sum of its digits. In other words, a number is considered a Harshad number if it can be evenly divided by the sum of its own digits.

For example:

$$1 + 8 = 9$$

18 is a Harshad number because , and 18 is divisible by 9

$$4 + 2 = 6$$

42 is not a Harshad number because , and 42 is not divisible by 6.

In [5]:

```
def is_harshad_number(num):
    # Calculate the sum of the digits of the number
    digit_sum = sum(int(i) for i in str(num))

    # Check if the number is divisible by the sum of its digits
    return num % digit_sum == 0

# Input a number
num = int(input("Enter a number: "))

# Check if it's a Harshad Number
if is_harshad_number(num):
    print(f"{num} is a Harshad Number.")
else:
    print(f"{num} is not a Harshad Number.")
```

```
Enter a number: 18
18 is a Harshad Number.
```

Program 48

Write a Python program to print all pronic numbers between 1 and 100.

A pronic number, also known as an oblong number or rectangular number, is a type of figurate number that represents a rectangle. It is the product of two consecutive integers, n and (n + 1). Mathematically, a pronic number can be expressed as:

$$P_n = n * (n + 1)$$

For example, the first few pronic numbers are:

$$P_1 = 1 * (1 + 1) = 2$$
$$P_2 = 2 * (2 + 1) = 6$$
$$P_3 = 3 * (3 + 1) = 12$$
$$P_4 = 4 * (4 + 1) = 20$$

In [6]:

```
def is_pronic_number(num):
    for n in range(1, int(num**0.5) + 1):
        if n * (n + 1) == num:
            return True
    return False

print("Pronic numbers between 1 and 100 are:")
for i in range(1, 101):
    if is_pronic_number(i):
        print(i, end=" | ")
```

```
Pronic numbers between 1 and 100 are: 2 | 6 | 12 | 20 | 30 | 42 | 56 | 72 | 90 |
```

Program 49

Write a Python program to find sum of elements in list

In [7]:

```
# Sample list of numbers
numbers = [10, 20, 30, 40, 50]

# Initialize a variable to store the sum
sum_of_numbers = 0

# Iterate through the list and accumulate the sum
for i in numbers:
    sum_of_numbers += i

# Print the sum
print("Sum of elements in the list:", sum_of_numbers)
```

```
Sum of elements in the list: 150
```

Program 50

Write a Python program to Multiply all numbers in the list.

In [8]:

```
# Sample list of numbers
```

```
numbers = [10, 20, 30, 40, 50]
3
4
# Initialize a variable to store the
product
5
product_of_numbers = 1
6
7
# Iterate through the list and
accumulate the product 8
for i in numbers:
9
 product_of_numbers *= i
10
11
# Print the product
12
print("Product of elements in the
list:", product_of_numbers)
```

```
Product of elements in the list: 12000000
```

Program 51

Write a Python program to find smallest number in a list.

```
In [9]:
1
# Sample list of numbers
2
numbers = [30, 10, -45, 5, 20]
3
4
# Initialize a variable to store the
minimum value, initially set to th 5
minimum = numbers[0]
6
7
# Iterate through the list and update the
minimum value if a smaller nu 8
for i in numbers:
9
 if i < minimum:
10
 minimum = i
11
12
# Print the minimum value
13
print("The smallest number in the list
is:", minimum)
```

```
The smallest number in the list is: -45
```


Program 52

Write a Python program to find largest number in a list.

```
In [10]:
1
# Sample list of numbers
2
numbers = [30, 10, -45, 5, 20]
3
4
# Initialize a variable to store the
minimum value, initially set to th 5
minimum = numbers[0]
6
7
# Iterate through the list and update the
minimum value if a smaller nu 8
for i in numbers:
9
 if i > minimum:
10
 minimum = i
11
12
# Print the minimum value
13
print("The largest number in the list
is:", minimum)
```

```
The largest number in the list is: 30
```

Program 53

Write a Python program to find second largest number in a list.

In [11]:

```
# Sample list of numbers
numbers = [30, 10, 45, 5, 20]

# Sort the list in descending order
numbers.sort(reverse=True)

# Check if there are at least two elements in the list
if len(numbers) >= 2:
    second_largest = numbers[1]
    print("The second largest number in the list is:", second_largest)
else:
    print("The list does not contain a second largest number.")
```

```
The second largest number in the list is: 30
```

Program 54

Write a Python program to find N largest elements from a list.

In [12]:

```
def find_n_largest_elements(lst, n):
    # Sort the list in descending order
    sorted_lst = sorted(lst, reverse=True)

    # Get the first N elements
    largest_elements = sorted_lst[:n]

    return largest_elements

# Sample list of numbers
numbers = [30, 10, 45, 5, 20, 50, 15, 3, 345, 54, 67, 87, 98, 100, 34,  12]
# Number of largest elements to find
N = int(input("N = " ))

# Find the N largest elements from the list
result = find_n_largest_elements(numbers, N)

# Print the N largest elements
print(f"The {N} largest elements in the list are:", result)
```

```
N = 3
The 3 largest elements in the list are: [345, 100, 98]
```

Program 55

Write a Python program to print even numbers in a list.

In [13]:
```
# Sample list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using a list comprehension to filter even numbers
even_numbers = [num for num in numbers if num % 2 == 0]

# Print the even numbers
print("Even numbers in the list:", even_numbers)
```

```
Even numbers in the list: [2, 4, 6, 8, 10]
```

Program 56

Write a Python program to print odd numbers in a List.

In [14]:
```
# Sample list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using a list comprehension to filter even numbers
even_numbers = [num for num in numbers if num % 2 != 0]

# Print the even numbers
print("Odd numbers in the list:", even_numbers)
```

```
Odd numbers in the list: [1, 3, 5, 7, 9]
```


Program 57

Write a Python program to Remove empty List from List.

In [15]:
```
# Sample list containing lists
list_of_lists = [[1, 2, 3], [], [4, 5], [], [6, 7, 8], []]

# Using a list comprehension to remove empty lists
filtered_list = [i for i in list_of_lists if i]

# Print the filtered list
print("List after removing empty lists:", filtered_list)
```

```
List after removing empty lists: [[1, 2, 3], [4, 5], [6, 7, 8]]
```

Program 58

Write a Python program to Cloning or Copying a list.

In [16]:
```
# 1. Using Using the Slice Operator
original_list = [1, 2, 3, 4, 5]
cloned_list = original_list[:]
print(cloned_list)
```

```
[1, 2, 3, 4, 5]
```

In [17]:
```
# 2. Using the list() constructor
original_list = [1, 2, 3, 4, 5]
cloned_list = list(original_list)
print(cloned_list)
```

```
[1, 2, 3, 4, 5]
```

In [18]:
```
# 3. Using List Comprehension
original_list = [1, 2, 3, 4, 5]
cloned_list = [item for item in original_list]
print(cloned_list)
```

```
[1, 2, 3, 4, 5]
```

Program 59

Write a Python program to Count occurrences of an element in a list.

In [19]:
```
def count_occurrences(l, element):
    count = l.count(element)
    return count

# Example usage:
my_list = [1, 2, 3, 4, 2, 5, 2, 3, 4, 6, 5]
element_to_count = 2

occurrences = count_occurrences(my_list, element_to_count)
print(f"The element {element_to_count} appears {occurrences} times in t
```

```
The element 2 appears 3 times in the list.
```

Program 60

Write a Python program to find words which are greater than given length k.

In [20]:
```
def find_words(words, k):
```

```
    # Create an empty list to store words greater than k
    result = []

    # Iterate through each word in the list
    for i in words:
        # Check if the length of the i is greater than k
        if len(i) > k:
            # If yes, append it to the result list
            result.append(i)

    return result

# Example usage
word_list = ["apple", "banana", "cherry", "date", "elderberry", "dragon
k = 5
long_words = find_words(word_list, k)

print(f"Words longer than {k} characters: {long_words}")
```

```
Words longer than 5 characters: ['banana', 'cherry', 'elderberry', 'dragon fruit']
```

Program 61

Write a Python program for removing ﷺ character from a string.

In [21]:

```
def remove_char(input_str, i):
    # Check if i is a valid index
    if i < 0 or i >= len(input_str):
        print(f"Invalid index {i}. The string remains unchanged.")
        return input_str

    # Remove the i-th character using slicing
    result_str = input_str[:i] + input_str[i + 1:]

    return result_str

# Input string
input_str = "Hello, wWorld!"
i = 7 # Index of the character to remove

# Remove the i-th character
new_str = remove_char(input_str, i)

print(f"Original String: {input_str}")
print(f"String after removing {i}th character : {new_str}")
```

```
Original String: Hello, wWorld!
```

```
String after removing 7th character : Hello, World!
```

Program 62

Write a Python program to split and join a string.

```
In [22]:
# Split a string into a list of words
input_str = "Python program to split and join a string"
word_list = input_str.split() # By default, split on whitespace

# Join the list of words into a string
separator = " " # specify the separator between words
output_str = separator.join(word_list)

# Print the results
print("Original String:", input_str)
print("List of split Words:", word_list)
print("Joined String:", output_str)
```

```
Original String: Python program to split and join a string List of split Words: ['Python', 'program', 'to', 'split', 'and', 'join', 'a', 'string']
Joined String: Python program to split and join a string
```

Program 63

Write a Python program to check if a given string is binary string or not.


```
In [23]:
def is_binary_str(input_str):
    # Iterate through each character in the input string
    for i in input_str:
        # Check if the i is not '0' or '1'
        if i not in '01':
            return False # If any character is not '0' or '1', it's no
    return True # If all characters are '0' or '1', it's a binary stri

# Input string to check
input_str = "1001110"

# Check if the input string is a binary string
if is_binary_str(input_str):
    print(f"'{input_str}' is a binary string.")
else:
    print(f"'{input_str}' is not a binary string.")
```

```
'1001110' is a binary string.
```

Program 64

Write a Python program to find uncommon words from two Strings.

In [24]:
```
def uncommon_words(str1, str2):
    # Split the strings into words and convert them to sets
    words1 = set(str1.split())
    words2 = set(str2.split())

    # Find uncommon words by taking the set difference
    uncommon_words_set = words1.symmetric_difference(words2)

    # Convert the set of uncommon words back to a list
    uncommon_words_list = list(uncommon_words_set)
    return uncommon_words_list

# Input two strings
string1 = "This is the first string"
string2 = "This is the second string"

# Find uncommon words between the two strings
uncommon = uncommon_words(string1, string2)

# Print the uncommon words
print("Uncommon words:", uncommon)
```

```
Uncommon words: ['second', 'first']
```

Program 65

Write a Python program to find all duplicate characters in string.

In [25]:
```
def find_duplicates(input_str):
    # Create an empty dictionary to store character counts
    char_count = {}

    # Initialize a list to store duplicate characters
    duplicates = []

    # Iterate through each character in the input string
    for i in input_str:
        # If the character is already in the dictionary, increment its
        if i in char_count:
            char_count[i] += 1
        else:
            char_count[i] = 1

    # Iterate through the dictionary and add characters with count > 1
```

```
    for i, count in char_count.items():
        if count > 1:
            duplicates.append(i)

    return duplicates

# Input a string
input_string = "piyush sharma"

# Find duplicate characters in the string
duplicate_chars = find_duplicates(input_string)

# Print the list of duplicate characters
print("Duplicate characters:", duplicate_chars)
```

```
Duplicate characters: ['s', 'h', 'a']
```

Program 66

Write a Python Program to check if a string contains any special character.

In [26]:

```
import re

def check_special_char(in_str):
    # Define a regular expression pattern to match special characters
    pattern = r'[!@#$%^&*()_+{}\[\]:;<>,.?~\\\/\'"\-=]'

    # Use re.search to find a match in the input string
    if re.search(pattern, in_str):
        return True
    else:
        return False

# Input a string
input_string = str(input("Enter a string: "))

# Check if the string contains any special characters
```

```
contains_special =
check_special_char(input_string)
18
19
# Print the result
20
if contains_special:
21
 print("The string contains special
characters.")
22
else:
23
 print("The string does not contain
special characters.")
```

```
Enter a string: "Hello, World!"
The string contains special characters.
```

Program 67

Write a Python program to Extract Unique dictionary values.

```
In [27]:
1
# Sample dictionary
2
my_dict = {
3
 'a': 10,
4
 'b': 20,
5
 'c': 10,
6
 'd': 30,
7
 'e': 20
8
}
9
10
# Initialize an empty set to store
unique values
11
uni_val = set()
12
13
# Iterate through the values of the
dictionary
14
for i in my_dict.values():
15
 # Add each value to the set
16
 uni_val.add(i)
17
18
# Convert the set of unique values
back to a list (if needed) 19
unique_values_list = list(uni_val)
20
21
# Print the unique values
22
print("Unique values in the
dictionary:", unique_values_list)
```

```
Unique values in the dictionary: [10, 20, 30]
```


Program 68

Write a Python program to find the sum of all items in a dictionary.

```
In [28]:
1
# Sample dictionary
2
my_dict = {
3
 'a': 10,
4
 'b': 20,
5
 'c': 30,
6
 'd': 40,
7
 'e': 50
8
}
```

```
9
10
# Initialize a variable to store the sum
11
total_sum = 0
12
13
# Iterate through the values of the
dictionary and add them to the tota 14
for i in my_dict.values():
15
 total_sum += i
16
17
# Print the sum of all items in the
dictionary
18
print("Sum of all items in the
dictionary:", total_sum)
```

```
Sum of all items in the dictionary: 150
```

Program 69

Write a Python program to Merging two Dictionaries.

```
In [29]:
1
# 1. Using the update() method:
2
3
dict1 = {'a': 1, 'b': 2}
4
dict2 = {'c': 3, 'd': 4}
5
6
dict1.update(dict2)
7
8
# The merged dictionary is now
in dict1
9
print("Merged Dictionary (using
update()):", dict1)
```

```
Merged Dictionary (using update()): {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

```
In [30]:
1
# 2. Using dictionary unpacking
2
3
dict1 = {'a': 1, 'b': 2}
4
dict2 = {'c': 3, 'd': 4}
5
6
# Merge dict2 into dict1 using dictionary
unpacking
7
merged_dict = {**dict1, **dict2}
8
9
# The merged dictionary is now in
merged_dict
10
print("Merged Dictionary (using
dictionary unpacking):", merged_dict)
```

```
Merged Dictionary (using dictionary unpacking): {'a': 1, 'b': 2, 'c': 3,
 'd': 4}
```


Program 70

Write a Python program to convert key-values list to flat dictionary.

```
In [31]:
1
key_values_list = [('a', 1), ('b', 2),
('c', 3), ('d', 4)] 2
3
# Initialize an empty dictionary
4
flat_dict = {}
5
6
# Iterate through the list and add
key-value pairs to the dictionary 7
for key, value in key_values_list:
8
 flat_dict[key] = value
9
```

```
10
# Print the resulting flat dictionary
```

```
11
print("Flat Dictionary:", flat_dict)
```

```
Flat Dictionary: {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

Program 71

Write a Python program to insertion at the beginning in OrderedDict.

```
In [32]:
1
from collections import OrderedDict
2
3
# Create an OrderedDict
4
ordered_dict = OrderedDict([('b', 2),
('c', 3), ('d', 4)]) 5
6
# Item to insert at the beginning
7
new_item = ('a', 1)
8
9
```

```
# Create a new OrderedDict with the new
item as the first element 10
new_ordered_dict =
OrderedDict([new_item])
11
12
# Merge the new OrderedDict with the
original OrderedDict 13
new_ordered_dict.update(ordered_dict)
14
15
# Print the updated OrderedDict
16
print("Updated OrderedDict:",
new_ordered_dict)
```

```
Updated OrderedDict: OrderedDict([('a', 1), ('b', 2), ('c', 3), ('d', 4)])
```

Program 72

Write a Python program to check order of character in string using OrderedDict().


```
In [33]:
1
from collections import OrderedDict
2
3
def check_order(string, reference):
4
 # Create OrderedDicts for both strings
5
 string_dict =
OrderedDict.fromkeys(string)
6
 reference_dict =
OrderedDict.fromkeys(reference)
7
```

```
8
 # Check if the OrderedDict for the string
matches the OrderedDict f 9
 return string_dict == reference_dict
10
11
# Input strings
12
input_string = "hello world"
13
reference_string = "helo wrd"
14
15
# Check if the order of characters in
input_string matches reference_st 16
```

```
if check_order(input_string, reference_string):
 print("The order of characters in the input string matches the refe
else:
 print("The order of characters in the input string does not match t
```

```
The order of characters in the input string matches the reference string.
```

Program 73

Write a Python program to sort Python Dictionaries by Key or Value.

In [34]:

```
# Sort by Keys:

sample_dict = {'apple': 3, 'banana': 1, 'cherry': 2, 'date': 4}

sorted_dict_by_keys = dict(sorted(sample_dict.items()))

print("Sorted by keys:")
for key, value in sorted_dict_by_keys.items():
 print(f"{key}: {value}")
```

```
Sorted by keys:
apple: 3
banana: 1
cherry: 2
date: 4
```


In [35]:

```
# Sort by values

sample_dict = {'apple': 3, 'banana': 1, 'cherry': 2, 'date': 4}

sorted_dict_by_values = dict(sorted(sample_dict.items(), key=lambda ite

print("Sorted by values:")
for key, value in sorted_dict_by_values.items():
 print(f"{key}: {value}")
```

```
Sorted by values:
banana: 1
cherry: 2
apple: 3
date: 4
```

Program 74

Write a program that calculates and prints the value according to the given formula:

$$Q = \text{Square root of} \frac{2 * C * D}{H}$$

Following are the fixed values of C and H:

C is 50. H is 30.

D is the variable whose values should be input to your program in a comma separated sequence.

Example

Let us assume the following comma separated input sequence is given to the program:

100,150,180

The output of the program should be:

18,22,24

In [36]:

```
import math

# Fixed values
C = 50
H = 30

# Function to calculate Q
def calculate_Q(D):
    return int(math.sqrt((2 * C * D) / H))

# Input comma-separated sequence of D values
input_sequence = input("Enter comma-separated values of D: ")
D_values = input_sequence.split(',')
```

```
15
# Calculate and print Q for each D
value
16
result = [calculate_Q(int(D)) for D
in D_values]
17
print(','.join(map(str, result)))
```

```
Enter comma-separated values of D: 100,150,180
18,22,24
```

Program 75

Write a program which takes 2 digits, X,Y as input and generates a 2-dimensional array. The element value in the i-th row and j-th column of the array should be i*j.

Note: i=0,1.., X-1; j=0,1,¡Y-1.

Example

Suppose the following inputs are given to the program:

3,5

Then, the output of the program should be:

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

In [37]:

```
1
# Input two digits, X and Y
2
X, Y = map(int, input("Enter two
digits (X, Y): ").split(',')) 3
4
# Initialize a 2D array filled with
zeros
5
array = [[0 for j in range(Y)] for i
in range(X)]
6
7
# Fill the array with values i * j
8
for i in range(X):
9
 for j in range(Y):
10
 array[i][j] = i * j
11
12
# Print the 2D array
13
for row in array:
14
 print(row)
```

```
Enter two digits (X, Y): 3,5
[0, 0, 0, 0, 0]
[0, 1, 2, 3, 4]
[0, 2, 4, 6, 8]
```


Program 76

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

Suppose the following input is supplied to the program:

without,hello,bag,world

Then, the output should be:

bag,hello,without,world

In [38]:

```
# Accept input from the user
input_sequence = input("Enter a comma-separated sequence of words: ")

# Split the input into a list of words
words = input_sequence.split(',')

# Sort the words alphabetically
sorted_words = sorted(words)

# Join the sorted words into a comma-separated sequence
sorted_sequence = ','.join(sorted_words)

# Print the sorted sequence
print("Sorted words:", sorted_sequence)
```

```
Enter a comma-separated sequence of words: without, hello, bag, world
Sorted words: bag, hello, world,without
```

Program 77

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

Suppose the following input is supplied to the program:

hello world and practice makes perfect and hello world again

Then, the output should be:

again and hello makes perfect practice world

In [39]:

```
# Accept input from the user
input_sequence = input("Enter a sequence of whitespace-separated words: ")

# Split the input into words and convert it into a set to remove duplic
words = set(input_sequence.split())

# Sort the unique words alphanumerically
sorted_words = sorted(words)

# Join the sorted words into a string with whitespace separation
result = ' '.join(sorted_words)

# Print the result
print("Result:", result)
```

```
Enter a sequence of whitespace-separated words: hello world and practice m
akes perfect and hello world again
Result: again and hello makes perfect practice world
```

Program 79

Write a program that accepts a sentence and calculate the number of letters and digits. Suppose the following input is supplied to the program:

hello world! 123

Then, the output should be:

LETTERS 10

DIGITS 3

```
In [40]:
# Accept input from the user
sentence = input("Enter a sentence: ")

# Initialize counters for letters and digits
letter_count = 0
digit_count = 0

# Iterate through each character in the sentence
for char in sentence:
    if char.isalpha():
        letter_count += 1
    elif char.isdigit():
        digit_count += 1

# Print the results
print("LETTERS", letter_count)
print("DIGITS", digit_count)
```

```
Enter a sentence: hello world! 123
LETTERS 10
DIGITS 3
```


Program 80

A website requires the users to input username and password to register. Write a program to check the validity of password input by users. Following are the criteria for checking the password:

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

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

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

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

4. Minimum length of transaction password: 6

5. Maximum length of transaction password: 12

Your program should accept a sequence of comma separated passwords and will check them according to the above criteria. Passwords that match the criteria are to be printed, each separated by a comma.

Example

If the following passwords are given as input to the program:

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

Then, the output of the program should be:

ABd1234@1

In [41]:

```
import re

# Function to check if a password is valid
def is_valid_password(password):
    # Check the length of the password
    if 6 <= len(password) <= 12:
        # Check if the password matches all the criteria using regular expressions
        if re.match(r"^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[$#@])", password):
            return True
    return False

# Accept input from the user as comma-separated passwords
passwords = input("Enter passwords separated by commas: ").split(',')

# Initialize a list to store valid
```

```
passwords
16
valid_passwords = []
17
18
# Iterate through the passwords and check their validity 19
for psw in passwords:
20
 if is_valid_password(psw):
21
 valid_passwords.append(psw)
22
23
# Print the valid passwords separated by commas
24
print(','.join(valid_passwords))
```

```
Enter passwords separated by commas: ABd1234@1,a F1#,2w3E*,2We3345
 ABd1234@1
```

Program 81

Define a class with a generator which can iterate the numbers, which are divisible by 7, between a given range 0 and n.

```
In [42]:
1
class DivisibleBySeven:
2
 def __init__(self,n):
3
 self.n = n
4
5
 def generate_divisible_by_seven(self): 6
 for num in range(self.n + 1): 7
 if num%7 == 0:
8
 yield num
```



```
In [43]:
1
n = int(input("Enter your desired range: "))
2
3
divisible_by_seven_generator = DivisibleBySeven(n).generate_divisible_b 4
5
for num in divisible_by_seven_generator:
6
 print(num)
```

```
Enter your desired range: 50
0
7
14
21
28
```

```
35
42
49
```

Program 82

Write a program to compute the frequency of the words from the input. The output should output after sorting the key alphanumerically. Suppose the following input is supplied to the program:

New to Python or choosing between Python 2 and Python 3? Read Python 2 or Python 3.

Then, the output should be:

2:2

3.:1

3?:1

New:1

Python:5

Read:1

and:1

between:1

choosing:1

or:2

to:1

In [44]:

```
input_sentence = input("Enter a sentence: ")

# Split the sentence into words
words = input_sentence.split()

# Create a dictionary to store word frequencies
word_freq = {}

# Count word frequencies
for word in words:
    # Remove punctuation (., ?) from the word
    word = word.strip('.,?')
```

```
 # Convert the word to lowercase to ensure case-insensitive counting
 word = word.lower()

 # Update the word frequency in the dictionary
 if word in word_freq:
 word_freq[word] += 1
 else:
 word_freq[word] = 1

# Sort the words alphanumerically
sorted_words = sorted(word_freq.items())

# Print the word frequencies
for word, frequency in sorted_words:
 print(f"{word}:{frequency}")
```

```
Enter a sentence: New to Python or choosing between Python 2 and Python 3? Read Python 2 or Python 3.
2:2
3:2
and:1
between:1
choosing:1
new:1
or:2
python:5
read:1
to:1
```

Program 83

Define a class Person and its two child classes: Male and Female. All classes have a method "getGender" which can print "Male" for Male class and "Female" for Female class.

In [45]:

```
class Person:
    def getGender(self):
        return "Unknown"

class Male(Person):
    def getGender(self):
        return "Male"

class Female(Person):
    def getGender(self):
        return "Female"

person = Person()
male = Male()
female = Female()

print(person.getGender())
print(male.getGender())
print(female.getGender())
```

In [46]:

```
Unknown
Male
Female
```

Program 84

Please write a program to generate all sentences where subject is in ["I", "You"] and verb is in ["Play", "Love"] and the object is in ["Hockey","Football"].

In [47]:

```
subjects = ["I", "You"]
verbs = ["Play", "Love"]
objects = ["Hockey", "Football"]

sentences = []

for sub in subjects:
    for vrb in verbs:
        for obj in objects:
            sentence = f"{sub} {vrb} {obj}."
            sentences.append(sentence)

for sentence in sentences:
    print(sentence)
```

```
I Play Hockey.
I Play Football.
I Love Hockey.
I Love Football.
You Play Hockey.
You Play Football.
You Love Hockey.
You Love Football.
```

```
Love Football.
```


Program 85

Please write a program to compress and decompress the string "hello world!hello world!hello world!hello world!".

In [48]:

```
import zlib

string = "hello world!hello world!hello world!hello world!"

# Compress the string
compressed_string = zlib.compress(string.encode())

# Decompress the string
decompressed_string = zlib.decompress(compressed_string).decode()

print("Original String:", string)
print("Compressed String:", compressed_string)
print("Decompressed String:", decompressed_string)
```

```
Original String: hello world!hello world!hello world!hello world!
Compressed String: b'x\x9c\xcbH\xcd\xc9\xc9W(\xcf/\xcaIQ\xcc \x82\r\x00\xbd[\x11\xf5'
Decompressed String: hello world!hello world!hello world!hello world!
```

Program 86

Please write a binary search function which searches an item in a sorted list. The function should return the index of element to be searched in the list.

In [49]:

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == target:
            return mid # Element found, return its index
        elif arr[mid] < target:
            left = mid + 1 # Target is in the right half
        else:
            right = mid - 1 # Target is in the left half
    return -1 # Element not found in the list

# Example usage:
sorted_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
target_element = 4

result = binary_search(sorted_list, target_element)
if result != -1:
    print(f"Element {target_element} found at index {result}")
else:
    print(f"Element {target_element} not found in the list")
```

```
Element 4 found at index 3
```

Program 87

Please write a program using generator to print the numbers which can be divisible by 5 and 7 between 0 and n in comma separated form while n is input by console.

Example:

If the following n is given as input to the program:

100

Then, the output of the program should be:

0,35,70

In [50]:
```
def divisible_by_5_and_7(n):
 for num in range(n + 1):
 if num % 5 == 0 and num % 7 == 0:
 yield num
```

In [51]:
```
try:
 n = int(input("Enter a value for n: "))
 result = divisible_by_5_and_7(n)
 print(','.join(map(str, result)))
except ValueError:
 print("Invalid input. Please enter a valid integer for n.")
```

```
Enter a value for n: 100
0,35,70
```


Program 88

Please write a program using generator to print the even numbers between 0 and n in comma separated form while n is input by console.

Example:

If the following n is given as input to the program:

10

Then, the output of the program should be:

0,2,4,6,8,10

In [52]:
```
def even_numbers(n):
 for num in range(n + 1):
 if num % 2 == 0:
 yield num
```

In [53]:
```
try:
```

```
2
 n = int(input("Enter a value for n:
"))
3
 result = even_numbers(n)
4
```

```
 print(','.join(map(str, result)))
5
except ValueError:
6
 print("Invalid input. Please enter a
valid integer for n.")
```

```
Enter a value for n: 10
0,2,4,6,8,10
```

Program 89

The Fibonacci Sequence is computed based on the following formula:

f(n)=0 if n=0

f(n)=1 if n=1

f(n)=f(n-1)+f(n-2) if n>1

Please write a program using list comprehension to print the Fibonacci Sequence in comma separated form with a given n input by console.

Example:

If the following n is given as input to the program:

8

Then, the output of the program should be:

0,1,1,2,3,5,8,13

In [55]:

```
1
def fibonacci(n):
2
 sequence = [0, 1] # Initializing the
sequence with the first two F 3
 [sequence.append(sequence[-1] +
sequence[-2]) for _ in range(2, n)] 4
 return sequence
```

In [56]:

```
1
try:
2
 n = int(input("Enter a value for n: "))
3
 result = fibonacci(n)
4
 print(','.join(map(str, result)))
5
except ValueError:
6
 print("Invalid input. Please enter a
valid integer for n.")
```

```
Enter a value for n: 8
0,1,1,2,3,5,8,13
```

Program 90

Assuming that we have some email addresses in the "username@companyname.com (mailto:username@companyname.com)" format, please write program to print the user name of a given email address. Both user names and company names are composed of letters only.

Example:

If the following email address is given as input to the program:

john@google.com (mailto:john@google.com)

Then, the output of the program should be:

john

In [57]:

```
def extract_username(email):
    # Split the email address at '@' to separate the username and domai
    parts = email.split('@')

    # Check if the email address has the expected format
    if len(parts) == 2:
        return parts[0] # The username is the first part
    else:
        return "Invalid email format"
```

In [58]:

```
try:
    email = input("Enter an email address: ")
    username = extract_username(email)
    print(username)
except ValueError:
    print("Invalid input. Please enter a valid email address.")
```

```
Enter an email address: john@google.com
john
```


Program 91

Define a class named Shape and its subclass Square. The Square class has an init function which takes a length as argument. Both classes have an area function which can print the area of the shape where Shape's area is 0 by default.

In [59]:

```
class Shape:
    def __init__(self):
        pass # Default constructor, no need to initialize anything

    def area(self):
        return 0 # Shape's area is 0 by default
```

In [60]:

```
class Square(Shape):
    def __init__(self, length):
        super().__init__() # Call the constructor of the parent class
        self.length = length

    def area(self):
        return self.length ** 2 # Calculate the area of the square


# Create instances of the classes
shape = Shape()
square = Square(float(input("Enter the shape of the square: "))) 
# Calculate and print the areas
print("Shape's area by default:", shape.area())
print("Area of the square:", square.area())
```

```
Enter the shape of the square: 5
Shape's area by default: 0
Area of the square: 25.0
```

Program 92

Write a function that stutters a word as if someone is struggling to read it. The first two letters are repeated twice with an ellipsis ... and space after each, and then the word is pronounced with a question mark ?.

Examples

stutter("incredible") → "in... in... incredible?"

stutter("enthusiastic") → "en... en... enthusiastic?"

stutter("outstanding") → "ou... ou... outstanding?"

Hint :- Assume all input is in lower case and at least two characters long.

In [61]:
```
def stutter(word):
 if len(word) < 2:
 return "Word must be at least two characters long."
```

In [62]:
```
 stuttered_word = f"{word[:2]}... {word[:2]}... {word}?"
 return stuttered_word


# Test cases
print(stutter("incredible"))
print(stutter("enthusiastic"))
print(stutter("outstanding"))
```

```
in... in... incredible?
en... en... enthusiastic?
ou... ou... outstanding?
```

Program 93

Create a function that takes an angle in radians and returns the corresponding angle in degrees rounded to one decimal place.

Examples

radians_to_degrees(1) → 57.3

radians_to_degrees(20) → 1145.9

radians_to_degrees(50) → 2864.8

In [63]:
```
import math

def radians_to_degrees(radians):
    degrees = radians * (180 / math.pi)
    return round(degrees, 1)
```

In [64]:
```
# Test cases
print(radians_to_degrees(1))
print(radians_to_degrees(20))
print(radians_to_degrees(50))
```

```
57.3
1145.9
2864.8
```

Program 94

In this challenge, establish if a given integer num is a Curzon number. If 1 plus 2 elevated to num is exactly divisible by 1 plus 2 multiplied by num, then num is a Curzon number.

Given a non-negative integer num, implement a function that returns True if num is a Curzon number, or False otherwise.

Examples

is_curzon(5) → True

```
# 2 ** 5 + 1 = 33

# 2 * 5 + 1 = 11

# 33 is a multiple of 11
```

is_curzon(10) → False

```
# 2 ** 10 + 1 = 1025

# 2 * 10 + 1 = 21

# 1025 is not a multiple of 21
```

is_curzon(14) → True

```
# 2 ** 14 + 1 = 16385

# 2 * 14 + 1 = 29

# 16385 is a multiple of 29
```

Curzon Number:

It is defined based on a specific mathematical relationship involving powers of 2. An integer 'n' is considered a Curzon number if it satisfies the following condition:

If (2^n + 1) is divisible by (2n + 1), then 'n' is a Curzon number.

For example:

If n = 5: 2^5 + 1 = 33, and 2*5 + 1 = 11. Since 33 is divisible by 11 (33 % 11 = 0), 5 is a Curzon number.
If n = 10: 2^10 + 1 = 1025, and 2*10 + 1 = 21. 1025 is not divisible by 21, so 10 is not a Curzon number.

Curzon numbers are a specific **subset of integers** with this unique mathematical property.

In [65]:
```
1
2
3
4
```

```
True
```

In [66]:
```
1
2
3
4
```

```
def is_curzon(num):
    numerator = 2 ** num + 1
    denominator = 2 * num + 1
    return numerator % denominator == 0


# Test cases
print(is_curzon(5))
print(is_curzon(10))
print(is_curzon(14))
```

```
False
True
```

Program 95

Given the side length x find the area of a hexagon.

Examples

area_of_hexagon(1) → 2.6

area_of_hexagon(2) → 10.4

area_of_hexagon(3) → 23.4

In [67]:
```
1
2
3
4
5
```

In [68]:
```
2.6 10.4 23.4
```

```
import math

def area_of_hexagon(x):
    area = (3 * math.sqrt(3) * x**2) / 2
    return round(area, 1)

# Example usage:
print(area_of_hexagon(1))
print(area_of_hexagon(2))
print(area_of_hexagon(3))
```

Program 96

Create a function that returns a base-2 (binary) representation of a base-10 (decimal) string number. To convert is simple: ((2) means base-2 and (10) means base-10) 010101001(2) = 1 + 8 + 32 + 128.

Going from right to left, the value of the most right bit is 1, now from that every bit to the left will be x2 the value, value of an 8 bit binary numbers are (256, 128, 64, 32, 16, 8, 4, 2, 1).

Examples

binary(1) → "1"

```
# 1*1 = 1
```

binary(5) → "101"

```
# 1*1 + 1*4 = 5
```

binary(10) → 1010

```
# 1*2 + 1*8 = 10
```


In [69]:
```
while decimal > 0:
    remainder = decimal % 2
    binary_str = str(remainder) + binary_str
    decimal = decimal // 2
return binary_str if binary_str else "0"
```

In [70]:
```
1
def binary(decimal):
    binary_str = ""
```

```
1
print(binary(1))
print(binary(5))
print(binary(10))
```

```
101
1010
```

Program 97

Create a function that takes three arguments a, b, c and returns the sum of the numbers that are evenly divided by c from the range a, b inclusive.

Examples

evenly_divisible(1, 10, 20) → 0

```
# No number between 1 and 10 can be evenly divided by 20.
```

evenly_divisible(1, 10, 2) → 30

```
# 2 + 4 + 6 + 8 + 10 = 30
```

evenly_divisible(1, 10, 3) → 18

```
# 3 + 6 + 9 = 18
```

In [71]:

```
In [72]: 0
```

```
1
def evenly_divisible(a,
b, c): 2
 total = 0
3
 for num in range(a, b +
1): 4
 if num % c == 0:
5
 total += num
6
 return total


1
print(evenly_divisible(1
, 10, 20)) 2
print(evenly_divisible(1
, 10, 2)) 3
print(evenly_divisible(1
, 10, 3))
```

```
30
18
```

Program 98

Create a function that returns True if a given inequality expression is correct and False otherwise.

Examples

correct_signs("3 < 7 < 11") → True

correct_signs("13 > 44 > 33 <1") → False

correct_signs("1 < 2 < 6 < 9 > 3") → True

```
In [74]: In [75]:
```

```
1
2
3
4
5
```

```
1
2
3

True
def
correct_signs(expr
ession):
 try:
 return
eval(expression)
except:
 return False


print(correct_sign
s("3 < 7 < 11"))
print(correct_sign
s("13 > 44 > 33 <
1"))
print(correct_sign
s("1 < 2 < 6 < 9 >
3"))
```

```
False
True
```

Program 99

Create a function that replaces all the vowels in a string with a specified character.

Examples

replace_vowels("the aardvark", "#") → "th# ##rdv#rk"

replace_vowels("minnie mouse", "?") → "m?nn?? m??s?"

replace_vowels("shakespeare", "*") → "sh*k*spr*"**

In [76]:

```
def replace_vowels(string, char):
    vowels = "AEIOUaeiou" # List of vowels to be replaced
    for vowel in vowels:
        string = string.replace(vowel, char) # Replace each vowel with
    return string
```

In [77]:

```
print(replace_vowels("the aardvark", "#"))
print(replace_vowels("minnie mouse", "?"))
print(replace_vowels("shakespeare", "*"))
```

```
th# ##rdv#rk
m?nn?? m??s?
sh*k*sp**r*
```

Program 100

Write a function that calculates the factorial of a number recursively.

Examples

factorial(5) → 120

factorial(3) → 6

factorial(1) → 1

factorial(0) → 1

In [78]:

```
def factorial(n):
    if n == 0:
        return 1 # Base case: factorial of 0 is 1
    else:
        return n * factorial(n - 1) # Recursive case: n! = n * (n-1)!
```

In [79]:

```
print(factorial(5))
print(factorial(3))
print(factorial(1))
print(factorial(0))
```

```
120
6
1
1
```

Program 101

Hamming distance is the number of characters that differ between two strings.

To illustrate:

String1: "abcbba"

String2: "abcbda"

Hamming Distance: 1 - "b" vs. "d" is the only difference.

Create a function that computes the hamming distance between two strings.

Examples

hamming_distance("abcde", "bcdef") → 5

hamming_distance("abcde", "abcde") → 0

hamming_distance("strong", "strung") → 1

In [80]:

In [81]:

```
def hamming_distance(str1, str2):
    # Check if the strings have the same length
    if len(str1) != len(str2):
        raise ValueError("Input strings must have the same length")

    # Initialize a counter to keep track of differences
    distance = 0

    # Iterate through the characters of both strings
    for i in range(len(str1)):
        if str1[i] != str2[i]:
            distance += 1 # Increment the counter for differences
```

```
14
 return distance


1
print(hamming_distance("abcde",
"bcdef"))
2
print(hamming_distance("abcde",
"abcde"))
3
print(hamming_distance("strong",
"strung"))
```

```
5
0
1
```

Program 102

Create a function that takes a list of non-negative integers and strings and return a new list without the strings.

Examples

filter_list([1, 2, "a", "b"]) → [1, 2]

filter_list([1, "a", "b", 0, 15]) → [1, 0, 15]

filter_list([1, 2, "aasf", "1", "123", 123]) → [1, 2, 123]

In [82]:

```
1
def filter_list(lst):
2
 # Initialize an empty list to store
non-string elements 3
 result = []
4
 # Iterate through the elements in the
input list
6
 for element in lst:
7
 # Check if the element is a non-negative
integer (not a string) 8
 if isinstance(element, int) and element
>= 0:
9
 result.append(element)
10

11
 return result
```

In [83]:

```
1
filter_list([1, 2, "a", "b"])
```

Out[83]: [1, 2]

In [84]:

```
filter_list([1, "a",
"b", 0, 15])
```

```
1
```

Out[84]: [1, 0, 15]

In [85]:

```
filter_list([1, 2, "aasf",
"1", "123", 123])
```

```
1
```

Out[85]: [1, 2, 123]

Program 103

The "Reverser" takes a string as input and returns that string in reverse order, with the opposite case.

Examples

reverse("Hello World") → "DLROw OLLEh"

reverse("ReVeRsE") → "eSrEvEr"

reverse("Radar") → "RADAr"

In [86]:
```
def reverse(input_str):
    # Reverse the string and swap the case of characters
    reversed_str = input_str[::-1].swapcase()

    return reversed_str
```

In [87]:
```
reverse("Hello World")
```

Out[87]: 'DLROw OLLEh'

In [88]:
```
reverse("ReVeRsE")
```

Out[88]: 'eSrEvEr'

In [89]:
```
reverse("Radar")
```

Out[89]: 'RADAr'

Program 104

You can assign variables from lists like this:

lst = [1, 2, 3, 4, 5, 6]

first = lst[0]

middle = lst[1:-1]

last = lst[-1]

print(first) → outputs 1

print(middle) → outputs [2, 3, 4, 5]

print(last) → outputs 6

With Python 3, you can assign variables from lists in a much more succinct way. Create variables first, middle and last from the given list using destructuring assignment (check the Resources tab for some examples), where:

first → 1

middle → [2, 3, 4, 5]

last → 6

Your task is to unpack the list `writeyourcodehere` into three variables, being `fi t iddl d l t ith iddl b i thi i b t th fi t d`

In [90]:
```
# Unpack the list into variables 4
first, *middle, last = writeyourcodehere
```

In [91]: Out[91]: 1
```
1
first
```

In [92]:
```
1
writeyourcodehere = [1, 2, 3, 4, 5, 6] 2
3
```

Out[92]: [2, 3, 4, 5]
```
1
middle
```

Out[93 last

In]: 6

[93]:
```
1
```

Program 105

Write a function that calculates the factorial of a number recursively.

Examples

factorial(5) → 120

factorial(3) → 6

factorial(1) → 1

factorial(0) → 1

In [94]: In [95]:
```
1
def factorial(n):
2
 if n == 0:
3
 return 1 # Base case: factorial of 0 is 1
4
 else:
5
 return n * factorial(n - 1) # Recursive case: n! = n * (n-1)!
```

1

```
factorial(5)
```

Out[95]: 120


```
1       factoria
```

In [96]:

```
1 1 1
```

Out[96]:

```
                l(1)
```

6

```
factoria
```

In [97]:

```
                factoria
```

Out[97]:

```
l(3)
```

1

```
                l(0)
```

In [98]:

Out[98]:

Program 106

Write a function that moves all elements of one type to the end of the list.

Examples

move_to_end([1, 3, 2, 4, 4, 1], 1) → [3, 2, 4, 4, 1, 1]

Move all the 1s to the end of the array.

move_to_end([7, 8, 9, 1, 2, 3, 4], 9) → [7, 8, 1, 2, 3, 4, 9]

move_to_end(["a", "a", "a", "b"], "a") → ["b", "a", "a", "a"]

In [99]:

```
1
def move_to_end(lst, element):
2
 # Initialize a count for the
specified element 3
 count = lst.count(element)
4
```

In [100]:

```
5
 # Remove all occurrences of the
element from the list 6
 lst = [x for x in lst if x !=
element]
7

8
 # Append the element to the end of
the list count times 9
 lst.extend([element] * count)
10

11
 return lst
```

```
1
move_to_end([1, 3, 2, 4, 4, 1], 1)
```

```
Out[100]: [3, 2, 4, 4, 1, 1]
                          move_to_end([7, 8, 9, 1,
                          2, 3, 4], 9)
In [101]:
1

Out[101]: [7, 8, 1, 2, 3, 4, 9]
                           move_to_end(["a", "a", "a",
                           "b"], "a")
In [102]:
1

Out[102]: ['b', 'a', 'a', 'a']
```

Program 107

Question1

Create a function that takes a string and returns a string in which each character is repeated once.

Examples

double_char("String") → "SSttrriinngg"

double_char("Hello World!") → "HHeellllloo WWoorrlldd!!"

double char("1234! ") → "11223344!! "

```
                          3
In [103]: In [104]:
                          4
                           for char in
                          input_str: 5
                           doubled_str += char *
                          2 6

                          7
                           return doubled_str
1
def
double_char(input_str) 1
: 2                       double_char("String")
 doubled_str = ""

Out[104]: 'SSttrriinngg'
                        double_char("Hello
                        World!")
In [105]:
1

Out[105]: 'HHeellllloo WWoorrlldd!!'
            1             "1234!_ ")
            double_char(
In [106]:

Out[106]: '11223344!!__ '
```

Program 108

Create a function that reverses a boolean value and returns the string "boolean expected" if another variable type is given.

Examples

reverse(True) → False

reverse(False) → True

reverse(0) → "boolean expected"

reverse(None) → "boolean expected"

In [107]:
```
1
def reverse(value):
2
 if isinstance(value,
bool): 3
 return not value
4
 else:
5
 return "boolean
expected"
```

In [108]:
```
1
reverse(True)
```

Out[108]: False

In [109]:
```
1
reverse(Fa
lse)
```

Out[109]: True

In [110]:
```
1
reverse(
0)
```

Out[110]: 'boolean expected'

In [111]:
```
reverse(N
one)
```

Out[111]: 'boolean expected'

Program 109

Create a function that returns the thickness (in meters) of a piece of paper after folding it n number of times. The paper starts off with a thickness of 0.5mm.

Examples

num_layers(1) → "0.001m"

- Paper folded once is 1mm (equal to 0.001m)

num_layers(4) → "0.008m"

- Paper folded 4 times is 8mm (equal to 0.008m)

num_layers(21) → "1048.576m"

- Paper folded 21 times is 1048576mm (equal to 1048.576m)

In [112]:
```
initial_thickness_mm = 0.5 # Initial thickness in millimeters
final_thickness_mm = initial_thickness_mm * (2 ** n)
final_thickness_m = final_thickness_mm / 1000 # Convert millimeter
return f"{final_thickness_m:.3f}m"
```

In [113]:
```
num_layers(1)
```

Out[113]: '0.001m'

In [114]:
```
num_layer(4)
```

Out[114]: '0.008m'

In [115]:
```
num_layer(21)
```

Out[115]: '1048.576m'

Program 110

Create a function that takes a single string as argument and returns an ordered list containing the indices of all capital letters in the string.

Examples

index_of_caps("eDaBiT") → [1, 3, 5]

index_of_caps("eQuINoX") → [1, 3, 4, 6]

index_of_caps("determine") → []

index_of_caps("STRIKE") → [0, 1, 2, 3, 4, 5]

index_of_caps("sUn") → [1]

In [116]:
```
# Use list comprehension to find indices of capital letters
return [i for i, char in enumerate(word) if char.isupper()]
```

In [117]:
```
index_of_caps("eDaBiT")
```

Out[117]: [1, 3, 5]

In [118]:
```
index_of_caps("eQuINoX")
```

Out[118]: [1, 3, 4, 6]

```
                    index_of_caps("deter
                    mine")
In [119]:
1

Out[119]: []
                  index_of_caps("STRI
                  KE")
In [120]:
1

Out[120]: [0, 1, 2, 3, 4, 5]
            1           ps("sUn")
            index_of_ca
In [121]:

Out[121]: [1]
```

Program 111

Using list comprehensions, create a function that finds all even numbers from 1 to the given number.

Examples

find_even_nums(8) → [2, 4, 6, 8]

find_even_nums(4) → [2, 4]

find_even_nums(2) → [2]

```
In [123]:
1
def find_even_nums(num):
2
    # Use a list comprehension to generate
even numbers from 1 to num 3
    return [x for x in range(1, num + 1) if x
% 2 == 0]
```



```
            1           nums(8)
In [124]:   find_even_

Out[124]: [2, 4, 6, 8]
            1           nums(4)
            find_even_
In [125]:

Out[125]: [2, 4]
            1           nums(2)
            find_even_
In [126]:

Out[126]: [2]
```

Program 112

Create a function that takes a list of strings and integers, and filters out the list so that it returns a list of integers only.

Examples

filter_list([1, 2, 3, "a", "b", 4]) → [1, 2, 3, 4]

filter_list(["A", 0, "Edabit", 1729, "Python", 1729]) → [0, 1729]

filter_list(["Nothing", "here"]) → []

In [127]:
```
def filter_list(lst):
```

In [128]:
```
    # Use a list comprehension to filter out integers
    return [x for x in lst if isinstance(x, int)]
```

In [129]:
```
filter_list([1, 2, 3, "a", "b", 4])
```

Out[128]: [1, 2, 3, 4]

In [130]:
```
filter_list(["A", 0, "Edabit", 1729, "Python", 1729])
```

Out[129]: [0, 1729, 1729]

In [131]:
```
filter_list(["A", 0, "Edabit", 1729, "Python", 1729])
```

Out[130]: [0, 1729, 1729]

In [132]:
```
filter_list(["Nothing", "here"])
```

Out[131]: []

Program 113

Given a list of numbers, create a function which returns the list but with each element's index in the list added to itself. This means you add 0 to the number at index 0, add 1 to the number at index 1, etc...

Examples

add_indexes([0, 0, 0, 0, 0]) → [0, 1, 2, 3, 4]

add_indexes([1, 2, 3, 4, 5]) → [1, 3, 5, 7, 9]

add_indexes([5, 4, 3, 2, 1]) → [5, 5, 5, 5, 5]

In [132]:
```
def add_indexes(lst):
```

In [133]:
```
    # Use list comprehension to add index to each element
    return [i + val for i, val in enumerate(lst)]
```

```
add_indexes([0, 0, 0, 0, 0])
```

Out[133]: [0, 1, 2, 3, 4]

```
add_indexes([1, 2, 3,
4, 5])
```

In [134]:
```
1
```

Out[134]: [1, 3, 5, 7, 9]

```
add_indexes([5, 4, 3,
2, 1])
```

In [135]:
```
1
```

Out[135]: [5, 5, 5, 5, 5]

Program 114

Create a function that takes the height and radius of a cone as arguments and returns the volume of the cone rounded to the nearest hundredth. See the resources tab for the formula.

Examples

cone_volume(3, 2) → 12.57

cone_volume(15, 6) → 565.49

cone_volume(18, 0) → 0

In [136]:
```
1
import math
2
3
```

In [137]:
```
def cone_volume(height, radius):
4
 if radius == 0:
5
 return 0
6
 volume = (1/3) * math.pi *
(radius**2) * height 7
 return round(volume, 2)
```

```
1
cone_volume(3, 2)
```

Out[137]: 12.57

```
cone_volume(15,
6)
```

In [138]:
```
1
```

Out[138]: 565.49

Out[139]: 0 cone_volume
(18, 0)

In [139]:
```
1
```

Program 115

This Triangular Number Sequence is generated from a pattern of dots that form a triangle. The first 5 numbers of the sequence, or dots, are:

1, 3, 6, 10, 15

This means that the first triangle has just one dot, the second one has three dots, the third one has 6 dots and so on.

Write a function that gives the number of dots with its corresponding triangle number of the sequence.

Examples

triangle(1) → 1

triangle(6) → 21

triangle(215) → 23220

In [140]:
```
def triangle(n):
    if n < 1:
        return 0
    return n * (n + 1)
    // 2
```

In [141]:
```
triangle(1)
```

Out[141]: 1

In [142]:
```
triangle(6)
```

Out[142]: 21

In [143]:
```
triangle(215)
```

Out[143]: 23220

Program 116

Create a function that takes a list of numbers between 1 and 10 (excluding one number) and returns the missing number.

Examples

missing_num([1, 2, 3, 4, 6, 7, 8, 9, 10]) → 5

missing_num([7, 2, 3, 6, 5, 9, 1, 4, 8]) → 10

missing_num([10, 5, 1, 2, 4, 6, 8, 3, 9]) → 7

In [144]:
```
def missing_num(lst):
    total_sum = sum(range(1, 11)) # Sum of numbers from 1 to 10
    given_sum = sum(lst) # Sum of the given list of numbers
    missing = total_sum - given_sum
    return missing
```

In [145]:
```
missing_num([1, 2, 3, 4, 6, 7, 8, 9, 10])
```
Out[145]: 5

In [146]:
```
missing_num([7, 2, 3, 6, 5, 9, 1, 4, 8])
```
Out[146]: 10

In [147]:
```
missing_num([10, 5, 1, 2, 4, 6, 8, 3, 9])
```
Out[147]: 7

Program 117

Write a function that takes a list and a number as arguments. Add the number to the end of the list, then remove the first element of the list. The function should then return the updated list.

Examples

next_in_line([5, 6, 7, 8, 9], 1) → [6, 7, 8, 9, 1]

next_in_line([7, 6, 3, 23, 17], 10) → [6, 3, 23, 17, 10]

next_in_line([1, 10, 20, 42], 6) → [10, 20, 42, 6]

next_in_line([], 6) → "No list has been selected"

In [148]:
```
def next_in_line(lst, num):
    if lst:
        lst.pop(0) # Remove the first element
        lst.append(num) # Add the number to the end
        return lst
    else:
        return "No list has been selected"

next_in_line([5, 6, 7, 8, 9], 1)
```

Out[149]: [6, 7, 8, 9, 1]

In [150]:
```
next_in_line([7, 6, 3, 23, 17], 10)
```

Out[150]: [6, 3, 23, 17, 10]

In [151]:
```
next_in_line([1, 10, 20, 42 ], 6)
```

Out[151]: [10, 20, 42, 6]

In [152]:
```
next_in_line([], 6)
```

Out[152]: 'No list has been selected'

Program 118

Create the function that takes a list of dictionaries and returns the sum of people's budgets.

Examples

get_budgets([

{ 'name': 'John', 'age': 21, 'budget': 23000 },

{ 'name': 'Steve', 'age': 32, 'budget': 40000 },

{ 'name': 'Martin', 'age': 16, 'budget': 2700 }

]) → 65700

get_budgets([

{ 'name': 'John', 'age': 21, 'budget': 29000 },

{ 'name': 'Steve', 'age': 32, 'budget': 32000 },

{ 'name': 'Martin', 'age': 16, 'budget': 1600 }

]) → 62600

In [153]:

```
1
def get_budgets(lst):
2
 total_budget = sum(person['budget']
for person in lst) 3
 return total_budget
4
5
```

In [154]:

```
# Test cases
6
budgets1 = [
7
 {'name': 'John', 'age': 21,
'budget': 23000}, 8
 {'name': 'Steve', 'age': 32,
'budget': 40000}, 9
 {'name': 'Martin', 'age': 16,
'budget': 2700} 10
]
11
12
budgets2 = [
13
 {'name': 'John', 'age': 21,
'budget': 29000}, 14
 {'name': 'Steve', 'age': 32,
'budget': 32000}, 15
 {'name': 'Martin', 'age': 16,
'budget': 1600} 16
]


1
get_budgets(budgets1)
```

Out[154]: 65700

```
  1           budgets2)
  get_budgets(
```

In [155]:

Out[155]: 62600

Program 119

Create a function that takes a string and returns a string with its letters in alphabetical order.

Examples

alphabet_soup("hello") → "ehllo"

alphabet_soup("edabit") → "abdeit"

alphabet_soup("hacker") → "acehkr"

alphabet_soup("geek") → "eegk"

alphabet_soup("javascript") → "aacijprstv"

In [156]:
```
def alphabet_soup(txt):
    return ''.join(sorted(txt))
```

In [157]:
```
alphabet_soup("hello")
```

Out[157]: 'ehllo'

In [158]:
```
alphabet_soup("edabit")
```

Out[158]: 'abdeit'

In [159]:
```
alphabet_soup("hacker")
```

Out[159]: 'acehkr'

In [160]:
```
alphabet_soup("geek")
```

Out[160]: 'eegk'

In [161]:
```
alphabet_soup("javascript")
```

Out[161]: 'aacijprstv'

Program 120

Suppose that you invest $10,000 for 10 years at an interest rate of 6% compounded monthly. What will be the value of your investment at the end of the 10 year period?

Create a function that accepts the principal p, the term in years t, the interest rate r, and the number of compounding periods per year n. The function returns the value at the end of term rounded to the nearest cent.

For the example:

compound_interest(10000, 10, 0.06, 12) → 18193.97

Note that the interest rate is given as a decimal and n=12 because with monthly compounding there are 12 periods per year. Compounding can also be done

annually, quarterly, weekly, or daily.

Examples

compound_interest(100, 1, 0.05, 1) → 105.0

compound_interest(3500, 15, 0.1, 4) → 15399.26

compound_interest(100000, 20, 0.15, 365) → 2007316.26

In [162]:
```
def compound_interest(p, t, r, n):
    # Calculate the compound interest
    # using the formula
    a = p * (1 + (r / n)) ** (n * t)
    # Round the result to the nearest cent
    return round(a, 2)
```

In [163]:
```
compound_interest(10000, 10, 0.06, 12)
```

Out[163]: 18193.97

In [164]:
```
compound_interest(100, 1, 0.05, 1)
```

Out[164]: 105.0

In [165]:
```
compound_interest(3500, 15, 0.1, 4)
```

Out[165]: 15399.26

In [166]:
```
compound_interest(100000, 20, 0.15, 365)
```

Out[166]: 2007316.26

Program 121

Write a function that takes a list of elements and returns only the integers.

Examples

return_only_integer([9, 2, "space", "car", "lion", 16]) → [9, 2, 16]

return_only_integer(["hello", 81, "basketball", 123, "fox"]) → [81, 123]

return_only_integer([10, "121", 56, 20, "car", 3, "lion"]) → [10, 56, 20,3]

return_only_integer(["String", True, 3.3, 1]) → [1]

In [167]:
```
def return_only_integer(lst):
    # Use list comprehension to filter out integers
    return [x for x in lst if isinstance(x, int) and not isinstance(x,
```

In [168]:
```
return_only_integer([9, 2, "space", "car", "lion", 16])
```

Out[168]: [9, 2, 16]

In [169]:
```
return_only_integer(["hello", 81, "basketball", 123, "fox"])
```

Out[169]: [81, 123]

In [170]:
```
return_only_integer([10, "121", 56, 20, "car", 3, "lion"])
```

Out[170]: [10, 56, 20, 3]

In [171]:
```
return_only_integer(["String", True, 3.3, 1])
```

Out[171]: [1]

Program 122

Create a function that takes three parameters where:

- x is the start of the range (inclusive).

- y is the end of the range (inclusive).

- n is the divisor to be checked against.

Return an ordered list with numbers in the range that are divisible by the third parameter n.

Return an empty list if there are no numbers that are divisible by n.

Examples

list_operation(1, 10, 3) → [3, 6, 9]

list_operation(7, 9, 2) → [8]

list_operation(15, 20, 7) → []

In [172]:
```
def list_operation(x, y, n):
    # Use list comprehension to generate the list of numbers divisible
    return [num for num in range(x, y + 1) if num % n == 0]
```

In [173]:
```
list_operation(1, 10, 3)
```

```
list_operation(1, 10, 3)
```

```
1
```

Out[173]: [3, 6, 9]

```
list_operation(7,
9, 2)
```

In [174]:

```
1
```

Out[174]: [8]

```
list_operation(15,
20, 7)
```

In [175]:

```
1
```

Out[175]: []

Program 123

Create a function that takes in two lists and returns True if the second list follows the first list by one element, and False otherwise. In other words, determine if the second list is the first list shifted to the right by 1.

Examples

simon_says([1, 2], [5, 1]) → True

simon_says([1, 2], [5, 5]) → False

simon_says([1, 2, 3, 4, 5], [0, 1, 2, 3, 4]) → True

simon_says([1, 2, 3, 4, 5], [5, 5, 1, 2, 3]) → False

Notes:

- Both input lists will be of the same length, and will have a minimum length of 2.

- The values of the 0-indexed element in the second list and the n-1th indexed element in the first list do not matter.

In [176]:

```
2
 # Check if the second list is the first
list shifted to the right b 3
 return list1[:-1] == list2[1:]
1
def simon_says(list1, list2):
```