In [8]:

```
# Experiment-3: Write a program to create a set of user-defined funct
# mathematical operations (addition, subtraction, multiplication, and
# ways of passing parameters to functions.

# Function for adding 2 numbers
def add(x, y):
    return x + y

# Function to subtract 2 numbers
def subtract(x, y):
    return x - y

# Function to multiply 2 numbers (multiply by 1 by default)
def multiply(x, y = 1):
    return x * y

# Function to divide 2 numbers (divide by 1 by default)
def divide(x, y = 1):
    if y == 0:
        raise ZeroDivisionError("Division by zero is not allowed.")
    return x / y

# Explore different ways of passing parameters:

# 1. Positional arguments:
addRes = add(5, 3)  # Pass arguments in the order defined in the func
divRes = divide(5, 3)
print("5 + 3 =", addRes)
print("5 / 3 =", divRes)

# 2. Keyword arguments:
result = subtract(y=10, x=20)  # Pass arguments by name
print("20 - 10 =", result)

# 3. Default arguments:
result = multiply(10)
print("10 * 1 =", result)

# 4. Variable-length arguments:
def sum_all(*args):
    """Sums all the arguments passed to the function."""
    total = 0
    for num in args:
        total += num
    return total

result = sum_all(1, 2, 3, 4, 5)
print("Sum of 1, 2, 3, 4, 5 =", result)
```

```
5 + 3 = 8
5 / 3 = 1.6666666666666667
20 - 10 = 10
10 * 1 = 10
Sum of 1, 2, 3, 4, 5 = 15
```

In [19]:

```
# Experiment-4: Create simple lambda functions for basic operations l
# multiplication, and division.
# Use lambda functions with built-in functions like filter() to filte
# Use lambda functions with built-in functions like map() to perform
# of a list.
# Use lambda functions with the sorted() function to customize sortin

# Lambda Function for addition, subtraction, multiplication and divis
add = lambda x, y: x + y
subtract = lambda x, y: x - y
multiply = lambda x, y: x * y
divide = lambda x, y: x / y if y else "Division by zero!"

# Examples:
print("Lambda Functions for basic operations")
result = add(5, 3)
print("5 + 3 =", result)
result = subtract(10, 2)
print("10 - 2 =", result)
result = multiply(4, 3)
print("4 * 3 =", result)
result = divide(10, 2)
print("10 / 2 =", result)
result = divide(10, 0)
print("10 / 0 =", result)

# Filtering
# List of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Printing the original list
print("\nOriginal List:", end = " ")
for num in numbers:
    print(num, end = " ")

# Filtering with filter()
# Filters the list of even numbers from the given list
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print("\nFiltering (even numbers only):", end = " ")
for num in even_numbers:
    print(num, end = " ")
```

```
# Mapping with map()
# Generates squares of all the elements in numbers and maps it into a
squared_numbers = list(map(lambda x: x * x, numbers))
print("\nMapping (Squared Numbers):", end = " ")
for num in squared_numbers:
    print(num, end = " ")

# Sorting with sort()
print("\n\nSorting:")
# Created a list of names
names = ["Alice", "Bob", "Charlie", "David", "Emily"]
print("Original List:", end = " ")
for name in names:
    print(name, end = " ")

# Sorts the names by length
sorted_by_length = sorted(names, key=lambda x: len(x))
print("\nSorted By Length:", end = " ")
for name in sorted_by_length:
    print(name, end = " ")

# Sorts the names by last letter
sorted_by_last_letter = sorted(names, key=lambda x: x[-1])
print("\nSorted By Last Letter:", end = " ")
for name in sorted_by_last_letter:
    print(name, end = " ")
```

```
Lambda Functions for basic operations
5 + 3 = 8
10 - 2 = 8
4 * 3 = 12
10 / 2 = 5.0
10 / 0 = Division by zero!

Original List: 1 2 3 4 5 6 7 8 9 10
Filtering (even numbers only): 2 4 6 8 10
Mapping (Squared Numbers): 1 4 9 16 25 36 49 64 81 100

Sorting:
Original List: Alice Bob Charlie David Emily
Sorted By Length: Bob Alice David Emily Charlie
Sorted By Last Letter: Bob David Alice Charlie Emily
```

In [24]:

```
# Experiment-5: Create a Python module by defining a few functions and
# module created in step 1 into another Python script and use its func

# Importing Module
from my_utils import *

name = "Aritra"
print(greet(name))

print(f"Circle area with radius 5: {calculate_area('circle', 5)}")
print(f"Rectangle area with length 4 and width 3: {calculate_area('rec
try:
  print(f"Triangle area with base 4 and height 3: {calculate_area('tri
except ValueError as e:
  print(e)  # Output: Unsupported shape: square

print(f"Value of PI from the module: {PI}")  # Output: 3.14159
```

```
Hello, John!
Circle area with radius 5: 78.53975
Rectangle area with length 4 and width 3: 12
Triangle area with base 4 and height 3: 6.0
Value of PI from the module: 3.14159
```

In [27]:

```
# Experiment-6: Define a base class, e.g., Vehicle, with attributes s
# Implement a constructor to initialize these attributes.
# Create instances of the Vehicle class, representing different vehic
# Display the information for each vehicle using appropriate methods.
# Overload the + operator to combine the make and model attributes of
# Display the result of this operator overloading operation.
# Create a derived class, e.g., Car, that inherits from the Vehicle c
# Add specific attributes to the Car class, such as num_doors and fue
# Instantiate objects of both the Vehicle and Car classes.
# Display the information for each object, showcasing the inheritance

# Defining base class Vehicle
class Vehicle:
    # Constructor to initialize all attributes
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    # Displaying Information
    def display_info(self):
        print(f"Make: {self.make}")
        print(f"Model: {self.model}")
        print(f"Year: {self.year}")

    # Overloading the '+' operator
    def __add__(self, other):
        return f"{self.make} {self.model} + {other.make} {other.model

# Defining derived class
class Car(Vehicle):
    # Constructor to initialize all attributes
    def __init__(self, make, model, year, num_doors, fuel_type):
        super().__init__(make, model, year)
        self.num_doors = num_doors
        self.fuel_type = fuel_type

    # Displaying information
    def display_info(self):
        super().display_info()
        print(f"Number of Doors: {self.num_doors}")
```

```python
        print(f"Fuel Type: {self.fuel_type}")

# Create Vehicle instances
vehicle1 = Vehicle("Toyota", "Camry", 2020)
vehicle2 = Vehicle("Honda", "Civic", 2022)

# Create Car instances
car1 = Car("Ford", "Mustang", 2023, 2, "Gasoline")
car2 = Car("Tesla", "Model S", 2021, 4, "Electric")

# Display information
vehicle1.display_info()
print()
vehicle2.display_info()
print()
car1.display_info()
print()
car2.display_info()
print()

# Operator overloading example
combined_vehicle = vehicle1 + car1
print(f"\nCombined vehicle info: {combined_vehicle}")
```

```
Make: Toyota
Model: Camry
Year: 2020

Make: Honda
Model: Civic
Year: 2022

Make: Ford
Model: Mustang
Year: 2023
Number of Doors: 2
Fuel Type: Gasoline

Make: Tesla
Model: Model S
Year: 2021
Number of Doors: 4
Fuel Type: Electric


Combined vehicle info: Toyota Camry + Ford Mustang
```