In [5]:
```
a = 20
print("Inititalized var: ",  a)
del a
b = 22.5
c = 'ram'
print("Initialized var b: ", b)
print("Initialized var c: ", c)
del b
del c
```

```
Inititalized var:  20
Initialized var b:  22.5
Initialized var c:  ram
```

In [74]:
```
a=21
b="niket"
c=22.5
print("a", type(a))
print("b", type(b))
print("c", type(c))
```

```
a <class 'int'>
b <class 'str'>
c <class 'float'>
```

In [72]:
```
RollNumber = 398
Name = "Niket"
Branch = "IT"
print("RollNumber " , RollNumber)
print("Name " , Name)
print("Branch " , Branch)
```

```
RollNumber  398
Name  Niket
Branch  IT
```

In [9]:
```
a=5
b=10
print(a+b)
print(a*b)
print(a-b)
print(a/b)
```

```
15
50
-5
0.5
```

In [23]:
```
num1=float(input("enter the first number: "))
num2=float(input("enter the second number: "))
num3=float(input("enter the third number: "))
sum = num1 + num2 + num3
print(f" the sum of {num1} ,{num2}, {num3} is: {sum} ")
```

```
the sum of 33.67 ,21.86, 22.0 is: 77.53
```

In [25]:
```
r=float(input("enter the radius: "))
area =3.14*r*r
c= 2*3.14*r
print(f" area = {area}, c= {c} ")
```

```
area = 28.259999999999998, c= 18.84
```

In [36]:
```
P = float(input("Enter the principal amount (P): "))
T = float(input("Enter the time in years (T): "))
R = float(input("Enter the annual interest rate (R) in percentage: "))

A = P * (1 + R / 100) ** T
ci = A-P
print(f"{ci}")
```

```
0.6833999999999989
```

In [54]:
```
a = float(input(" enter the 1st number to be swap"))
b = float(input(" enter the 2nd number to be swap"))
a,b = b,a
print("after swapping:")
print("num1", a )
print("num2" ,b )
```

```
after swapping:
num1 56.0
num2 35.0
```

In [56]:
```
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

print("Is num1 equal to num2? ", num1 == num2)
print("Is num1 not equal to num2? ", num1 != num2)
print("Is num1 greater than num2? ", num1 > num2)
print("Is num1 less than num2? ", num1 < num2)
print("Is num1 greater than or equal to num2? ", num1 >= num2)
print("Is num1 less than or equal to num2? ", num1 <= num2)
```

```
Is num1 equal to num2?  False
Is num1 not equal to num2?  True
Is num1 greater than num2?  False
Is num1 less than num2?  True
Is num1 greater than or equal to num2?  False
Is num1 less than or equal to num2?  True
```

In [62]:
```
paisa = int(input("Enter the amount in paisa: "))

rupees = paisa // 100
remaining_paisa = paisa % 100

print(f"{paisa} paisa = {rupees} Rupee and {remaining_paisa} paisa")
```

```
2344 paisa = 23 Rupee and 44 paisa
```

In [68]:
```
seconds = int(input("Enter the number of seconds: "))

hours = seconds // 3600
minutes = (seconds % 3600) // 60
remaining_seconds = seconds % 60

print(f"{seconds} seconds = {hours} Hour, {minutes} Minute and {remaining_second
```

```
3856 seconds = 1 Hour, 4 Minute and 16 Second
```

In [70]:
```
meters = int(input("Enter the quantity in meters: "))

kilometers = meters // 1000
```

```
remaining_meters = meters % 1000

print(f"{meters} meter = {kilometers} Km and {remaining_meters} meter")
```

```
1500 meter = 1 Km and 500 meter
```

In [84]:
```
amount = int(input("Enter the amount to be withdrawn in hundreds: "))

hundred_notes = amount//100
fifty_notes = (amount%100)//50
remaining=amount%100
ten_notes = (remaining%50)//10
print(f"Number of 100 denomination notes: {hundred_notes}")
print(f"Number of 50 denomination notes: {fifty_notes}")
print(f"Number of 10 denomination notes: {ten_notes}")
basic_salary = float(input("Enter Ramesh's basic salary: "))
dearness_allowance = (40 / 100) * basic_salary
house_rent_allowance = (20 / 100) * basic_salary
gross_salary = basic_salary + dearness_allowance + house_rent_allowance #gross
print(f"Ramesh's gross salary is: {gross_salary}")
```

```
Number of 100 denomination notes: 7
Number of 50 denomination notes: 1
Number of 10 denomination notes: 3
Ramesh's gross salary is: 848.0
```

In [ ]:
```

```

In [60]:

```
# Question

def find_max(my_list1):
    return max(my_list1)

def find_min(my_list1):
    return min(my_list1)

def find_sum(my_list1):
    return sum(my_list1)

def find_length(my_list1):
    return length(my_list1)

def join_list(my_list1,my_list2):
    return my_list1 + my_list2

def repeat_element(my_list1,element,n):
    return my_list1 + [element]*n

def check_value(my_list1,value):
    return value in my_list1

def convert_to_list(obj):
    return list(obj)

def sort_list(my_list1):
    return sorted(my_list1)

my_list1 = [20,34,56,12,23,98,67,56,22,21]


print("max_value: ", find_max(my_list1))

print("min_value: ", find_min(my_list1))

print("sum: ", sum(my_list1))

print("length of my_list1: ", len(my_list1))

my_list2 = [35,21,12,87,67,56,99,25,65]

print("join_list: ", my_list1 + my_list2)

#print("Repeat_element: ", (my_list1))
print("Repeat element 10 five times:", repeat_element(my_list1, 10, 5))

print("is 22 present in the list: ", check_value(my_list1,22))

tuple_data = (1, 2, 3)
set_data = {4, 5, 6}
string_data = "hello"
print("Converted tuple to list:", convert_to_list(tuple_data))
print("Converted set to list:", convert_to_list(set_data))
print("Converted string to list:", convert_to_list(string_data))


print("Sorted list:",sort_list(my_list1))
```

```
max_value:  98
min_value:  12
sum:  409
length of my_list1:  10
join_list:  [20, 34, 56, 12, 23, 98, 67, 56, 22, 21, 35, 21, 12, 87, 67, 56, 99,
25, 65]
Repeat element 10 five times: [20, 34, 56, 12, 23, 98, 67, 56, 22, 21, 10, 10, 1
0, 10, 10]
is 22 present in the list:  True
Converted tuple to list: [1, 2, 3]
Converted set to list: [4, 5, 6]
Converted string to list: ['h', 'e', 'l', 'l', 'o']
Sorted list: [12, 20, 21, 22, 23, 34, 56, 56, 67, 98]
```

In [98]:
```python
# Q1

my_list = [10,20,37,75,45,44,22,10,20,22]
print("Original list:", my_list)

length = len(my_list)
print("Length of the list:", length)

duplicates = [item for item in set(my_list) if my_list.count(item) > 1]
print("Duplicate elements in the list:", duplicates)

my_list.reverse()
print("Reversed list:", my_list)
```

```
Original list: [10, 20, 37, 75, 45, 44, 22, 10, 20, 22]
Length of the list: 10
Duplicate elements in the list: [10, 20, 22]
Reversed list: [22, 20, 10, 22, 44, 45, 75, 37, 20, 10]
```

In [94]:
```python
# Q2

list1 = ['iron', 'super', 'Hanu']
list2 = ['man', 'man', 'man']

concatenated_list = [a + b for a, b in zip(list1, list2)]
print("Concatenated list index-wise:", concatenated_list)
```

```
Concatenated list index-wise: ['ironman', 'superman', 'Hanuman']
```

In [92]:
```python
# Q3

def square_elements(lst):
    return [x ** 2 for x in lst]

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

squared_list = square_elements(lst)
print("List with squared elements:", squared_list)
```

```
List with squared elements: [1, 4, 9, 16, 25]
```

In [90]:
```python
#4

def extend_nested_list(nested_lst, sublist):
    nested_lst.extend(sublist)
    return nested_lst
```

```
nested_lst = [[1, 2], [3, 4], [5, 6]]
sublist_to_add = [7, 8]

extended_list = extend_nested_list(nested_lst, sublist_to_add)
print("Extended nested list:", extended_list)
```

```
Extended nested list: [[1, 2], [3, 4], [5, 6], 7, 8]
```

In [86]:
```
# Q5

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

item_to_remove = 5

filtered_list = [x for x in my_list if x != item_to_remove]
print("List after removing all occurrences of", item_to_remove, ":",filtered_lis
```

```
List after removing all occurrences of 5 : [1, 3, 2, 4, 6]
```

In [ ]:

In [9]:
```python
# 1.wapp Maximum and Minimum K elements in Tuple(Find 1st maximum and 2nd maximu

def find_max_min_elements(tup):
    if len(tup) < 2:
        return "Tuple must have at least two elements"
    sorted_tup = sorted(tup)
    max1 = sorted_tup[-1]
    max2 = sorted_tup[-2]
    min1 = sorted_tup[0]
    min2 = sorted_tup[1]
    return {
        "1st_max": max1,
        "2nd_max": max2,
        "1st_min": min1,
        "2nd_min": min2
    }
tup = (15, 25, 50, 36, 9, 80, 78)
result = find_max_min_elements(tup)
print(result)
```

```
{'1st_max': 80, '2nd_max': 78, '1st_min': 9, '2nd_min': 15}
```

In [17]:
```python
# 2.wapp Row-wise element Addition in Tuple Matrix.

def row_wise_addition(matrix):
    row_sums = [sum(row) for row in matrix]
    return row_sums
matrix = (
    (5, 7, 10),
    (6, 8, 3),
    (1, 2, 9)
)

result = row_wise_addition(matrix)
print(result)
```

```
[22, 17, 12]
```

In [29]:
```python
# 3.Python Elements frequency in Tuple.

from collections import Counter

def element_frequency(tup):
    return Counter(tup)

tup = (1,2,2,3,4,3,5,5,4,4,1)
freq = element_frequency(tup)
print("element frequency:", freq)
```

```
element frequency: Counter({4: 3, 1: 2, 2: 2, 3: 2, 5: 2})
```

In [67]:
```python
# 4.wapp All pair combinations of 2 tuples

import itertools

tup1 = (1,2,3,4,5)
tup2 = (6,7,8,9,10)
```

```
pairs = list(itertools.product(tup1, tup2))
print(pairs)
```

```
[(1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (2, 6), (2, 7), (2, 8), (2, 9), (2, 1
0), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (4, 6), (4, 7), (4, 8), (4, 9), (4,
10), (5, 6), (5, 7), (5, 8), (5, 9), (5, 10)]
```

In [119…

```
# 5.wapp  Test if tuple is distinct.true or false.

def is_distinct(t):
    return len(t) == len(set(t))

tuple1 = (1,2,3,4)
tuple2 = (1,2,5,6,2)

print(is_distinct(tuple1))
print(is_distinct(tuple2))
```

```
True
False
```

In [79]:

```
# 6.Write a Python script to sort (ascending and descending) a dictionary by val


my_dict = {'yash': 11, 'ram': 7, 'shayam': 8,'isha' : 10}

print(sorted(my_dict.values()))
print(sorted(my_dict.values(),reverse = True))
```

```
[7, 8, 10, 11]
[11, 10, 8, 7]
```

In [91]:

```
# 7.Write a Python program to get the maximum and minimum values of a dictionary


my_dict = {'apple': 10, 'banana': 5, 'orange': 8, 'grape': 2}

max_item = max(my_dict.values())
min_item = min(my_dict.values())

print(f"Item with maximum value: {max_item}")
print(f"Item with minimum value: {min_item}")
```

```
Item with maximum value: 10
Item with minimum value: 2
```

In [111…

```
# 8.Write a Python program to create a dictionary of keys x, y, and z where each
#Access the fifth value of each key from the dictionary.(Take value from keyboar

x = list(range(11, 21))
y = list(range(21, 31))
z = list(range(31, 41))

dictionary = {'x': x, 'y': y, 'z': z}

dict_2 = {}
for key, values in dictionary.items():
    dict_2[key] = values[4]

print("Fifth value of each key:", dict_2)
```

```
Fifth value of each key: {'x': 15, 'y': 25, 'z': 35}
```

In [115…

```python
# 9.wapp Program to create grade calculator in Python.

def calculate_grade():

    try:
        marks1 = float(input("Enter marks for Subject 1: "))
        marks2 = float(input("Enter marks for Subject 2: "))
        marks3 = float(input("Enter marks for Subject 3: "))
        marks4 = float(input("Enter marks for Subject 4: "))
        marks5 = float(input("Enter marks for Subject 5: "))

        average_marks = (marks1 + marks2 + marks3 + marks4 + marks5) / 5

        if average_marks >= 90:
            grade = 'O'
        elif average_marks >= 80:
            grade = 'E'
        elif average_marks >= 70:
            grade = 'A'
        elif average_marks >= 60:
            grade = 'B'
        else:
            grade = 'F'

        print(f"Your average marks are: {average_marks:.2f}")
        print(f"Your grade is: {grade}")

    except ValueError:
        print("Invalid input! Please enter numeric values for marks.")

calculate_grade()
```

```
Your average marks are: 77.60
Your grade is: A
```

In [117…

```python
# 10.wapp Key with maximum unique values.

def key_with_max_unique_values(d):

    max_unique_count = 0
    max_unique_key = None

    for key, values in d.items():

        unique_values = set(values)
        unique_count = len(unique_values)


        if unique_count > max_unique_count:
            max_unique_count = unique_count
            max_unique_key = key

    return max_unique_key, max_unique_count

my_dict = {
    'a': [1, 2, 2, 3],
    'b': [4, 5, 5, 6, 7],
    'c': [8, 8, 9, 9],
}
```

```
key, count = key_with_max_unique_values(my_dict)
print(f"The key with the maximum unique values is '{key}' with {count} unique va
```

```
The key with the maximum unique values is 'b' with 4 unique values.
```

In [ ]:

In [1]:
```python
#Q1

divisible_by_4 = []

for num in range(1, 21):
    if num % 4 == 0:
        divisible_by_4.append(num)

print(divisible_by_4)
```

```
[4, 8, 12, 16, 20]
```

In [2]:
```python
# Q2

def is_positive(num):
    return num > 0
numbers = [10, -5, 3, -2, 7, -8, 0, 15, -1]
positive_numbers = list(filter(is_positive, numbers))

print(positive_numbers)
```

```
[10, 3, 7, 15]
```

In [6]:
```python
# Q3

def to_lowercase(s):
    return s.lower()
strings = ["HELLO", "WORLD", "NAMASKAR🙏"]

lowercase_strings = list(map(to_lowercase, strings))

print(lowercase_strings)
```

```
['hello', 'world', 'namaskar🙏']
```

In [9]:
```python
# Q4

from functools import reduce

def find_max(x, y):
    return x if x > y else y
input_numbers = input("Enter numbers separated by spaces: ")
numbers = list(map(int, input_numbers.split()))
largest_number = reduce(find_max, numbers)

print("The largest number in the list is:", largest_number)
```

```
The largest number in the list is: 56
```

In [10]:
```python
# Q5

import math

def circle_properties(radius):
    area = math.pi * radius ** 2
    circumference = 2 * math.pi * radius
    return area, circumference

radius = float(input("Enter the radius of the circle: "))
```

```
area, circumference = circle_properties(radius)
print(f"The area of the circle is: {area}")
print(f"The circumference of the circle is: {circumference}")
```

```
The area of the circle is: 78.53981633974483
The circumference of the circle is: 31.41592653589793
```

In [16]:
```
# Q6

def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))
target = int(input("Enter the number to search for: "))

numbers.sort()
result = binary_search(numbers, target)

print(f"Number {target} found at index {result}" if result != -1 else f"Number {
```

```
Number 11 found at index 0
```

In [11]:
```
# Q7

def sequential_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1
numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))
target = int(input("Enter the number to search for: "))
result = sequential_search(numbers, target)

if result != -1:
    print(f"Number {target} found at index {result}.")
else:
    print(f"Number {target} not found in the list.")
```

```
Number 23 found at index 1.
```

In [13]:
```
# Q8

def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
```

```
    return -1

numbers = list(map(int, input("Enter sorted numbers separated by spaces: ").spli
target = int(input("Enter the number to search for: "))

result = binary_search(numbers, target)

if result != -1:
    print(f"Number {target} found at index {result}.")
else:
    print(f"Number {target} not found in the list.")
```

```
Number 45 found at index 2.
```

In [17]:

```
# Q9

import math

def jump_search(arr, target):
    n = len(arr)
    step = int(math.sqrt(n))
    prev = 0

    while arr[min(step, n) - 1] < target:
        prev = step
        step += int(math.sqrt(n))
        if prev >= n:
            return -1

    for i in range(prev, min(step, n)):
        if arr[i] == target:
            return i

    return -1

numbers = list(map(int, input("Enter sorted numbers: ").split()))
target = int(input("Enter target: "))

result = jump_search(numbers, target)
print(f"Number {target} found at index {result}" if result != -1 else f"Number {
```

```
Number 56 found at index 2
```

In [19]:

```
# Q10

def interpolation_search(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high and target >= arr[low] and target <= arr[high]:
        pos = low + ((target - arr[low]) * (high - low)) // (arr[high] - arr[low
        if arr[pos] == target:
            return pos
        if arr[pos] > target:
            high = pos - 1
        else:
            low = pos + 1
    return -1
numbers = list(map(int, input("Enter sorted numbers separated by spaces: ").spli
target = int(input("Enter the number to search for: "))
result = interpolation_search(numbers, target)
```

```
if result != -1:
    print(f"Number {target} found at index {result}.")
else:
    print(f"Number {target} not found in the list.")
```

```
Number 11 found at index 0.
```

In [ ]:

In [1]:
```python
#Q1:- Write a Python program to sort a list of elements using the bubble sort al

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break

sequence = [2, 23, 10, 1, 78, 45, 65]
print("Unsorted list:", sequence)

bubble_sort(sequence)
print("Sorted list:", sequence)
```

```
Unsorted list: [2, 23, 10, 1, 78, 45, 65]
Sorted list: [1, 2, 10, 23, 45, 65, 78]
```

In [2]:
```python
#Q2

def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and arr[left] > arr[largest]:
        largest = left

    if right < n and arr[right] > arr[largest]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

arr = [12, 11, 13, 5, 6, 7]
heap_sort(arr)
print("Sorted array:", arr)
```

```
Sorted array: [5, 6, 7, 11, 12, 13]
```

In [5]:
```python
#Q3

def quick_sort(arr):
    if len(arr) <= 1:
```

```
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

sequence = [2, 23, 10, 1, 78, 45, 65]
print("Unsorted list:", sequence)

sorted_sequence = quick_sort(sequence)
print("Sorted list:", sorted_sequence)
```

```
Unsorted list: [2, 23, 10, 1, 78, 45, 65]
Sorted list: [1, 2, 10, 23, 45, 65, 78]
```

In [9]:

```
#Q4

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

sequence = [2, 23, 10, 1, 78, 45, 65]
print("Unsorted list:", sequence)

merge_sort(sequence)
print("Sorted list:", sequence)
```

```
Unsorted list: [2, 23, 10, 1, 78, 45, 65]
Sorted list: [1, 2, 10, 23, 45, 65, 78]
```

In [11]:

```
#Q5

def insertion_sort(arr):
    for i in range(1, len(arr)):
```

```
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

sequence = [2, 23, 10, 1, 78, 45, 65]
print("Unsorted list:", sequence)

insertion_sort(sequence)
print("Sorted list:", sequence)
```

```
Unsorted list: [2, 23, 10, 1, 78, 45, 65]
Sorted list: [1, 2, 10, 23, 45, 65, 78]
```

In [13]:

```
#Q6

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]

sequence = [2, 23, 10, 1, 78, 45, 65]
print("Unsorted list:", sequence)

selection_sort(sequence)
print("Sorted list:", sequence)
```

```
Unsorted list: [2, 23, 10, 1, 78, 45, 65]
Sorted list: [1, 2, 10, 23, 45, 65, 78]
```

In [ ]:

In [1]:
```python
#Q1

import re

def extract_email_info(email):
    match = re.match(r'([^@]+)@([^@]+\.[^@]+)', email)
    if match:
        username = match.group(1)
        domain = match.group(2)
        return (username, domain)
    else:
        return "Invalid email address"

email = input("Enter an email address: ")
result = extract_email_info(email)
print(f"Extracted info: {result}")
```

```
Extracted info: ('niks', 'gmail.com')
```

In [3]:
```python
#Q2

def filter_positive_numbers(numbers):
    positive_numbers = tuple(num for num in numbers if num > 0)
    return positive_numbers

numbers = [5, -3, 7, -1, 0, 9, -2]
result = filter_positive_numbers(numbers)
print(f"Original list: {numbers}")
print(f"Positive numbers tuple: {result}")
```

```
Original list: [5, -3, 7, -1, 0, 9, -2]
Positive numbers tuple: (5, 7, 9)
```

In [8]:
```python
#Q3

def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def generate_prime_numbers(limit):
    return {num for num in range(2, limit + 1) if is_prime(num)}

def generate_odd_numbers(limit):
    return {num for num in range(1, limit + 1, 2)}

limit = 30

prime_numbers = generate_prime_numbers(limit)
odd_numbers = generate_odd_numbers(limit)

union_set = prime_numbers | odd_numbers
intersection_set = prime_numbers & odd_numbers
difference_set = prime_numbers - odd_numbers

print(f"Prime Numbers: {prime_numbers}")
```

```
print(f"Odd Numbers: {odd_numbers}")
print(f"Union of Prime and Odd Numbers: {union_set}")
print(f"Intersection of Prime and Odd Numbers: {intersection_set}")
print(f"Difference of Prime and Odd Numbers (Prime - Odd): {difference_set}")
```

```
Prime Numbers: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
Odd Numbers: {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29}
Union of Prime and Odd Numbers: {1, 2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 2
5, 27, 29}
Intersection of Prime and Odd Numbers: {3, 5, 7, 11, 13, 17, 19, 23, 29}
Difference of Prime and Odd Numbers (Prime - Odd): {2}
```

In [10]:
```
#Q4

brics_countries = ["Brazil", "Russia", "India", "China", "South Africa"]

brics_set = set(brics_countries)

sorted_brics = sorted(brics_set)

print("Sorted BRICS Countries:", sorted_brics)
```

```
Sorted BRICS Countries: ['Brazil', 'China', 'India', 'Russia', 'South Africa']
```

In [12]:
```
#Q5

setn1 = set([1, 1, 2, 3, 4, 5])
setn2 = set([1, 5, 6, 7, 8, 9])

r1 = setn1 ^ setn2

print("Symmetric Difference:", r1)
```

```
Symmetric Difference: {2, 3, 4, 6, 7, 8, 9}
```

In [16]:
```
#Q6

import random
setn1 = set(random.sample(range(1, 20), 10))
setn2 = set(random.sample(range(1, 20), 10))

missing_in_set2 = setn1 - setn2

missing_in_set1 = setn2 - setn1

print(f"Set 1: {setn1}")
print(f"Set 2: {setn2}")
print(f"Numbers in Set 1 but not in Set 2: {missing_in_set2}")
print(f"Numbers in Set 2 but not in Set 1: {missing_in_set1}")
```

```
Set 1: {1, 2, 7, 8, 9, 10, 13, 15, 17, 18}
Set 2: {4, 5, 6, 8, 9, 11, 12, 13, 14, 15}
Numbers in Set 1 but not in Set 2: {1, 2, 7, 10, 17, 18}
Numbers in Set 2 but not in Set 1: {4, 5, 6, 11, 12, 14}
```

In [ ]:
```

```

In [2]:

```
def fcfs(processes):
    processes.sort(key=lambda x: x[1])
    time = 0
    for p in processes:
        time = max(time, p[1]) + p[2]
        p.append(time)
        p.append(time - p[1])
        p.append(p[-1] - p[2])
    return processes


def sjf(processes):
    processes.sort(key=lambda x: (x[1], x[2]))
    time, completed = 0, []
    while processes:
        available = [p for p in processes if p[1] <= time]
        if available:
            p = min(available, key=lambda x: x[2])
            time += p[2]
            p.append(time)
            p.append(time - p[1])
            p.append(p[-1] - p[2])
            completed.append(p)
            processes.remove(p)
        else:
            time += 1
    return completed


def ljf(processes):
    processes.sort(key=lambda x: (x[1], -x[2]))
    time, completed = 0, []
    while processes:
        available = [p for p in processes if p[1] <= time]
        if available:
            p = max(available, key=lambda x: x[2])
            time += p[2]
            p.append(time)
            p.append(time - p[1])
            p.append(p[-1] - p[2])
            completed.append(p)
            processes.remove(p)
        else:
            time += 1
    return completed


def priority_scheduling(processes):
    processes.sort(key=lambda x: (x[1], x[3]))
    time = 0
    for p in processes:
        time = max(time, p[1]) + p[2]
        p.append(time)
        p.append(time - p[1])
        p.append(p[-1] - p[2])
    return processes
```

```python
def round_robin(processes, quantum):
    queue, time, completed = processes[:], 0, []
    while queue:
        p = queue.pop(0)
        if p[2] > quantum:
            time += quantum
            p[2] -= quantum
            queue.append(p)
        else:
            time += p[2]
            p.append(time)
            p.append(time - p[1])
            p.append(p[-1] - p[3])
            completed.append(p)
    return completed


def srtf(processes):
    time, completed, remaining = 0, [], processes[:]
    while remaining:
        available = [p for p in remaining if p[1] <= time]
        if available:
            p = min(available, key=lambda x: x[2])
            time += 1
            p[2] -= 1
            if p[2] == 0:
                p.append(time)
                p.append(time - p[1])
                p.append(p[-1] - p[3])
                completed.append(p)
                remaining.remove(p)
        else:
            time += 1
    return completed


def lrtf(processes):
    time, completed, remaining = 0, [], processes[:]
    while remaining:
        available = [p for p in remaining if p[1] <= time]
        if available:
            p = max(available, key=lambda x: x[2])
            time += 1
            p[2] -= 1
            if p[2] == 0:
                p.append(time)
                p.append(time - p[1])
                p.append(p[-1] - p[3])
                completed.append(p)
                remaining.remove(p)
        else:
            time += 1
    return completed


def print_results(algo_name, processes):
    print(f"\n{algo_name} Results:")
    print("ID | Completion | Turnaround | Waiting")
    for p in processes:
        print(f"{p[0]} | {p[-3]} | {p[-2]} | {p[-1]}")
```

```
    avg_wt = sum(p[-1] for p in processes) / len(processes)
    avg_tat = sum(p[-2] for p in processes) / len(processes)
    print(f"Average Waiting Time: {avg_wt:.2f}")
    print(f"Average Turnaround Time: {avg_tat:.2f}\n")


if __name__ == "__main__":
    n = int(input("Enter number of processes: "))
    processes = []
    for i in range(n):
        at = int(input(f"Enter Arrival Time for Process {i+1}: "))
        bt = int(input(f"Enter Burst Time for Process {i+1}: "))
        pr = int(input(f"Enter Priority for Process {i+1} (if not applicable, en
        processes.append([i+1, at, bt, pr])

    quantum = int(input("Enter Time Quantum for Round Robin: "))

    algorithms = {
        "FCFS": fcfs,
        "SJF": sjf,
        "LJF": ljf,
        "Priority Scheduling": priority_scheduling,
        "Round Robin": lambda p: round_robin(p, quantum),
        "SRTF": srtf,
        "LRTF": lrtf
    }

    for name, algo_func in algorithms.items():
        result = algo_func([p[:] for p in processes])
        print_results(name, result)
```

FCFS Results:
ID | Completion | Turnaround | Waiting
1 | 5 | 5 | 0
2 | 8 | 7 | 4
3 | 16 | 14 | 6
Average Waiting Time: 3.33
Average Turnaround Time: 8.67

SJF Results:
ID | Completion | Turnaround | Waiting
1 | 5 | 5 | 0
2 | 8 | 7 | 4
3 | 16 | 14 | 6
Average Waiting Time: 3.33
Average Turnaround Time: 8.67

LJF Results:
ID | Completion | Turnaround | Waiting
1 | 5 | 5 | 0
3 | 13 | 11 | 3
2 | 16 | 15 | 12
Average Waiting Time: 5.00
Average Turnaround Time: 10.33

Priority Scheduling Results:
ID | Completion | Turnaround | Waiting
1 | 5 | 5 | 0
2 | 8 | 7 | 4
3 | 16 | 14 | 6
Average Waiting Time: 3.33
Average Turnaround Time: 8.67

Round Robin Results:
ID | Completion | Turnaround | Waiting
2 | 9 | 8 | 6
1 | 12 | 12 | 11
3 | 16 | 14 | 11
Average Waiting Time: 9.33
Average Turnaround Time: 11.33

SRTF Results:
ID | Completion | Turnaround | Waiting
2 | 4 | 3 | 1
1 | 8 | 8 | 7
3 | 16 | 14 | 11
Average Waiting Time: 6.33
Average Turnaround Time: 8.33

LRTF Results:
ID | Completion | Turnaround | Waiting
1 | 14 | 14 | 13
2 | 15 | 14 | 12
3 | 16 | 14 | 11
Average Waiting Time: 12.00

```
Average Turnaround Time: 14.00
```

In [ ]: