**Bài 1: Tính tổng các phần tử của danh sách**

* **Đầu vào**: lst = [12, -14, 11, 91, -3, 10, 7, 15]
* **Output**: Tổng các phần tử của danh sách.

**Thuật toán**:

1. Khởi tạo total\_sum bằng 0.
2. Duyệt qua từng phần tử trong lst.
3. Cộng từng phần tử vào total\_sum.
4. Xuất kết quả total\_sum.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

total\_sum = sum(lst)

print("Bai1")

print("Tổng các phần tử:", total\_sum)

**Bài 2: Đếm số lượng và tính tổng các số hạng dương**

* **Đầu vào**: lst = [12, -14, 11, 91, -3, 10, 7, 15]
* **Output**: Số lượng và tổng các số hạng dương.

**Thuật toán**:

1. Dùng list comprehension để tạo danh sách các phần tử dương (positive\_elements).
2. Đếm và tính tổng của positive\_elements.
3. Xuất số lượng và tổng.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

positive\_elements = [x for x in lst if x > 0]

count\_positive = len(positive\_elements)

sum\_positive = sum(positive\_elements)

print("Bai2")

print("Số lượng các số hạng dương:", count\_positive)

print("Tổng của các số hạng dương:", sum\_positive)

**Bài 3: Tính trung bình cộng của cả danh sách và trung bình cộng các phần tử dương**

* **Đầu vào**: lst = [12, -14, 11, 91, -3, 10, 7, 15]
* **Output**: Trung bình cộng của danh sách và các phần tử dương.

**Thuật toán**:

1. Tính tổng và số lượng phần tử của lst để có trung bình cộng.
2. Tính trung bình cộng cho các phần tử dương nếu danh sách dương không trống.
3. Xuất average và positive\_average.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

average = sum(lst) / len(lst)

positive\_elements = [x for x in lst if x > 0]

positive\_average = sum(positive\_elements) / len(positive\_elements) if positive\_elements else 0

print("Bai3")

print("Trung bình cộng của cả danh sách:", average)

print("Trung bình cộng các phần tử dương:", positive\_average)

**Bài 4: Tìm vị trí của phần tử âm đầu tiên**

* **Đầu vào**: lst
* **Output**: Vị trí của phần tử âm đầu tiên.

**Thuật toán**:

1. Duyệt qua lst và tìm vị trí đầu tiên có phần tử âm.
2. Xuất vị trí đó.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

first\_negative\_index = next((i for i, x in enumerate(lst) if x < 0), None)

print("Bai4")

print("Vị trí phần tử âm đầu tiên:", first\_negative\_index)

**Bài 5: Tìm vị trí của phần tử dương cuối cùng**

* **Đầu vào**: lst
* **Output**: Vị trí của phần tử dương cuối cùng.

**Thuật toán**:

1. Duyệt ngược qua lst để tìm phần tử dương cuối cùng.
2. Xuất vị trí đó.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

last\_positive\_index = max(i for i, x in enumerate(lst) if x > 0)

print("Bai5")

print("Vị trí phần tử dương cuối cùng:", last\_positive\_index)

**Bài 6: Tìm phần tử lớn nhất và vị trí cuối cùng của nó**

* **Đầu vào**: lst
* **Output**: Phần tử lớn nhất và vị trí cuối cùng của nó.

**Thuật toán**:

1. Tìm giá trị lớn nhất trong lst.
2. Tìm vị trí cuối cùng của giá trị lớn nhất.
3. Xuất kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

max\_value = max(lst)

last\_max\_position = len(lst) - 1 - lst[::-1].index(max\_value)

print("Bai6")

print("Phần tử lớn nhất:", max\_value)

print("Vị trí phần tử lớn nhất:", last\_max\_position)

**Bài 7: Tìm phần tử lớn thứ nhì và các vị trí của nó**

* **Đầu vào:** lst
* **Output:** Phần tử lớn thứ nhì và các vị trí của nó.

**Thuật toán:**

1. Loại bỏ phần tử trùng lặp, sắp xếp giảm dần, lấy phần tử lớn thứ nhì.
2. Tìm các vị trí của phần tử lớn thứ nhì.
3. Xuất kết quả.

**Code:**

lst = [12, -14, 11, 91, -3, 10, 7, 15]

unique\_sorted\_lst = sorted(set(lst), reverse=True)

second\_largest = unique\_sorted\_lst[1] if len(unique\_sorted\_lst) > 1 else None

second\_largest\_positions = [i for i, x in enumerate(lst) if x == second\_largest]

print("Bai7")

print("Phần tử lớn thứ nhì:", second\_largest)

print("Vị trí của phần tử lớn thứ nhì:", second\_largest\_positions)

**Bài 8: Tính số lượng các số dương liên tiếp nhiều nhất**

* **Đầu vào**: lst
* **Output**: Số lượng các số dương liên tiếp nhiều nhất.

**Thuật toán**:

1. Duyệt lst, đếm chuỗi các số dương liên tiếp.
2. So sánh với chuỗi dài nhất.
3. Xuất kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

max\_positive\_streak = 0

current\_streak = 0

for num in lst:

if num > 0:

current\_streak += 1

max\_positive\_streak = max(max\_positive\_streak, current\_streak)

else:

current\_streak = 0

print("Bai8")

print("Số lượng các số dương liên tiếp nhiều nhất:", max\_positive\_streak)

**Bài 9: Tính số lượng các số dương liên tiếp có tổng lớn nhất**

* **Đầu vào**: lst
* **Output**: Số lượng các số dương liên tiếp có tổng lớn nhất.

**Thuật toán**:

1. Cộng dồn các số dương liên tiếp.
2. Tìm chuỗi có tổng lớn nhất.
3. Xuất kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

max\_positive\_sum = 0

count = 0

temp\_sum = 0

for num in lst:

if num > 0:

temp\_sum += num

count += 1

max\_positive\_sum = max(max\_positive\_sum, temp\_sum)

else:

temp\_sum = 0

count = 0

print("Bai9")

print("Số lượng các số dương liên tiếp có tổng lớn nhất:", count)

**Bài 10: Tính số lượng các phần tử liên tiếp đan dấu nhiều nhất**

* **Đầu vào**: lst
* **Output**: Số lượng các phần tử đan dấu nhiều nhất.

**Thuật toán**:

1. Kiểm tra đan dấu giữa các phần tử.
2. Tăng đếm khi có đan dấu.
3. Xuất kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

max\_alternating\_streak = 0

current\_streak = 1

for i in range(1, len(lst)):

if lst[i] \* lst[i-1] < 0:

current\_streak += 1

max\_alternating\_streak = max(max\_alternating\_streak, current\_streak)

else:

current\_streak = 1

print("Bai10")

print("Số lượng các phần tử liên tiếp đan dấu nhiều nhất:", max\_alternating\_streak)

**Bài 11: Tính số lượng các phần tử không tăng nhiều nhất**

* **Đầu vào**: lst = [12, -14, 11, 91, -3, 10, 7, 15]
* **Output**: Số lượng các phần tử không tăng nhiều nhất.

**Thuật toán**:

1. Khởi tạo non\_increasing\_count và max\_non\_increasing\_count là 1.
2. Duyệt qua lst:

Nếu phần tử hiện tại ≤ phần tử trước đó, tăng non\_increasing\_count.

Nếu không, đặt lại non\_increasing\_count về 1.

1. So sánh và cập nhật max\_non\_increasing\_count.
2. Xuất max\_non\_increasing\_count.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

non\_increasing\_count = 1

max\_non\_increasing\_count = 1

for i in range(1, len(lst)):

if lst[i] <= lst[i - 1]:

non\_increasing\_count += 1

max\_non\_increasing\_count = max(max\_non\_increasing\_count, non\_increasing\_count)

else:

non\_increasing\_count = 1

print("Bai11")

print("Số lượng các phần tử không tăng nhiều nhất:", max\_non\_increasing\_count)

**Bài 12: Tìm vị trí bắt đầu đoạn con dương liên tiếp nhiều phần tử nhất**

* **Đầu vào**: lst
* **Output**: Vị trí bắt đầu của đoạn con dương liên tiếp dài nhất.

**Thuật toán**:

1. Duyệt lst, đếm chuỗi dương liên tiếp và lưu vị trí bắt đầu tạm thời.
2. Cập nhật max\_streak và start\_position khi tìm thấy chuỗi dài nhất.
3. Xuất start\_position.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

max\_streak = 0

current\_streak = 0

start\_position = None

temp\_start = 0

for i, num in enumerate(lst):

if num > 0:

current\_streak += 1

if current\_streak > max\_streak:

max\_streak = current\_streak

start\_position = temp\_start

else:

current\_streak = 0

temp\_start = i + 1

print("Bai12")

print("Vị trí bắt đầu đoạn con dương liên tiếp dài nhất:", start\_position)

**Bài 13: Tìm đoạn con dương liên tiếp có tổng lớn nhất**

* **Đầu vào**: lst
* **Output**: Các đoạn con dương liên tiếp có tổng lớn nhất và số lượng đoạn con.

**Thuật toán**:

1. Duyệt lst, cộng các số dương vào current\_sum và lưu vào temp\_array.
2. Khi gặp số không dương:
   * Nếu current\_sum > max\_sum, cập nhật max\_sum và subarrays.
   * Nếu current\_sum == max\_sum, thêm temp\_array vào subarrays.
3. Xuất max\_sum, subarrays và số lượng đoạn con.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

max\_sum = 0

current\_sum = 0

subarrays = []

temp\_array = []

for num in lst:

if num > 0:

current\_sum += num

temp\_array.append(num)

else:

if current\_sum > max\_sum:

max\_sum = current\_sum

subarrays = [temp\_array[:]]

elif current\_sum == max\_sum:

subarrays.append(temp\_array[:])

current\_sum = 0

temp\_array = []

if temp\_array:

if current\_sum > max\_sum:

max\_sum = current\_sum

subarrays = [temp\_array]

elif current\_sum == max\_sum:

subarrays.append(temp\_array)

print("Bai13")

print("Số đoạn con thoả mãn:", len(subarrays))

print("Các đoạn con có tổng lớn nhất:", subarrays)

**Bài 14: Đếm số lượng các phần tử bằng giá trị X nhập từ bàn phím**

* **Đầu vào**: lst và X
* **Output**: Số lượng phần tử bằng X.

**Thuật toán**:

1. Nhập giá trị X.
2. Đếm số lượng phần tử bằng X trong lst.
3. Xuất kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

X = int(input("Nhập giá trị X: "))

count\_X = lst.count(X)

print("Bai14")

print("Số lượng các phần tử bằng X:", count\_X)

**Bài 15: Chuyển các phần tử dương lên đầu danh sách**

* **Đầu vào**: lst
* **Output**: Danh sách sau khi chuyển các phần tử dương lên đầu.

**Thuật toán**:

1. Tách các phần tử dương và không dương.
2. Ghép các phần tử dương trước và không dương sau.
3. Xuất danh sách kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

positive\_elements = [x for x in lst if x > 0]

negative\_elements = [x for x in lst if x <= 0]

reordered\_list = positive\_elements + negative\_elements

print("Bai15")

print("Danh sách sau khi chuyển các phần tử dương lên đầu:", reordered\_list)

**Bài 16: Tìm số phần tử là số nguyên tố và vị trí của chúng**

* **Đầu vào**: lst
* **Output**: Các số nguyên tố và vị trí của chúng.

**Thuật toán**:

1. Tạo hàm is\_prime để kiểm tra số nguyên tố.
2. Tìm các số nguyên tố và vị trí của chúng.
3. Xuất kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

def is\_prime(n):

if n < 2:

return False

for i in range(2, int(n \*\* 0.5) + 1):

if n % i == 0:

return False

return True

primes = [(num, i) for i, num in enumerate(lst) if is\_prime(num)]

print("Bai16")

print("Các số nguyên tố và vị trí của chúng:", primes)

**Bài 17: Chèn một số m vào đầu, cuối và vị trí thứ 5 của danh sách**

* **Đầu vào**: lst và m
* **Output**: Danh sách sau khi chèn m.

**Thuật toán**:

1. Nhập giá trị m.
2. Chèn m vào đầu, cuối, và vị trí thứ 5.
3. Xuất danh sách kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

m = int(input("Nhập một số m: "))

lst.insert(0, m)

lst.append(m)

if len(lst) >= 5:

lst.insert(4, m)

print("Bai17")

print("Danh sách sau khi chèn số m:", lst)

**Bài 18: Chèn danh sách [1, 2, 3] vào đầu, cuối và vị trí thứ 5 của danh sách**

* **Đầu vào**: lst
* **Output**: Danh sách sau khi chèn [1, 2, 3].

**Thuật toán**:

1. Chèn [1, 2, 3] vào đầu, cuối và vị trí thứ 5 của lst.
2. Xuất danh sách kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

to\_insert = [1, 2, 3]

lst = to\_insert + lst

lst += to\_insert

if len(lst) >= 5:

lst[5:5] = to\_insert

print("Bai18")

print("Danh sách sau khi chèn [1, 2, 3]:", lst)

**Bài 19: Xóa phần tử thứ k trong danh sách**

* **Đầu vào**: lst và k
* **Output**: Danh sách sau khi xóa phần tử thứ k.

**Thuật toán**:

1. Nhập k.
2. Kiểm tra k hợp lệ, xóa phần tử ở vị trí k.
3. Xuất danh sách kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

k = int(input("Nhập vị trí k cần xoá (bắt đầu từ 0): "))

if 0 <= k < len(lst):

lst.pop(k)

print("Bai19")

print("Danh sách sau khi xoá phần tử thứ k:", lst)

**Bài 20: Sắp xếp danh sách theo thứ tự tăng dần và giảm dần**

* **Đầu vào**: lst
* **Output**: Danh sách sắp xếp tăng dần và giảm dần.

**Thuật toán**:

1. Sắp xếp lst theo thứ tự tăng dần và giảm dần.
2. Xuất danh sách kết quả.

Code:

lst = [12, -14, 11, 91, -3, 10, 7, 15]

ascending\_lst = sorted(lst)

descending\_lst = sorted(lst, reverse=True)

print("Bai20")

print("Danh sách sắp xếp tăng dần:", ascending\_lst)

print("Danh sách sắp xếp giảm dần:", descending\_lst)