**Group 6**

Câu 1: Viết chương trình Perl hoặc Python tính tổng các phần tử của danh sách

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

**Output**: tong = 129.

**Thuật toán:**

Bước 1: Nhập danh sách các số nguyên lst.

Bước 2: Khởi tạo biến tong = 0.

Bước 3: Lặp qua từng phần tử x trong lst:

Cộng x vào tong.

Bước 4: Xuất kết quả: tong.

**Code:**

# Cách không dùng các hàm có sẵn

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

sum = 0

for x in lst:

    sum += x

print("Tổng là:", sum)

# dùng hàm Deg

def tinh\_tong(lst):

    tong = 0

    for x in lst:

        tong += x

    return tong

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

tong = tinh\_tong(lst)

print("Tổng là:", tong)

#Cách dùng hàm có sẵn(Sum)

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

tong = sum(lst)

print("Tổng là:", tong)

# dùng hàm Deg

def tinh\_tong(lst):

    return sum(lst)

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

tong = tinh\_tong(lst)

print("Tổng là:", tong)

**KeyWords:**

|  |  |
| --- | --- |
| **Tên** | **Vai trò** |
| sum() | Hàm tích hợp sẵn dùng để tính tổng (chỉ dùng trong cách có hàm có sẵn) |
| lst | Danh sách (list) chứa các số nguyên |
| tong hoặc sum | Biến dùng để lưu tổng |

Câu 2: Viết chương trình Perl hoặc Python đếm số lượng các số hạng dương và tổng của các số hạng dương.

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

**Output**: Số lượng số dương = 6.

Tổng các số dương = 146.

**Thuật toán:**Bước 1: Khởi tạo biến:

* dem = 0 (đếm số dương)
* tong = 0 (tổng số dương)

Bước 2: Duyệt từng phần tử x trong danh sách lst:

* Nếu x > 0:
  + Tăng dem lên 1.
  + Cộng x vào tong.

Bước 3: Trả về hoặc in ra kết quả:

* dem: số lượng số hạng dương
* tong: tổng các số hạng dương

**Code:**  
# Cách không dùng các hàm có sẵn

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

so\_luong\_duong = 0

tong\_duong = 0

for x in lst:

    if x > 0:

        so\_luong\_duong += 1

        tong\_duong += x

print("Số lượng số hạng dương là:", so\_luong\_duong)

print("Tổng các số hạng dương là:", tong\_duong)

# Cách không dùng các hàm có sẵn(hàm Deg)

def thong\_ke\_duong(lst):

    dem = 0

    tong = 0

    for x in lst:

        if x > 0:

            dem += 1

            tong += x

    return dem, tong

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

so\_luong\_duong, tong\_duong = thong\_ke\_duong(lst)

print("Số lượng số hạng dương là:", so\_luong\_duong)

print("Tổng các số hạng dương là:", tong\_duong)

#Cách dùng hàm có sẵn(Sum)

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

so\_luong\_duong = len([x for x in lst if x > 0])

tong\_duong = sum([x for x in lst if x > 0])

print("Số lượng số hạng dương là:", so\_luong\_duong)

print("Tổng các số hạng dương là:", tong\_duong)

#Cách dùng hàm có sẵn(Sum)hàm Deg

def thong\_ke\_duong(lst):

    so\_duong = [x for x in lst if x > 0]

    return len(so\_duong), sum(so\_duong)

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

so\_luong\_duong, tong\_duong = thong\_ke\_duong(lst)

print("Số lượng số hạng dương là:", so\_luong\_duong)

print("Tổng các số hạng dương là:", tong\_duong)

**KeyWords:**

|  |  |
| --- | --- |
| **Từ khóa / Hàm** | **Vai trò** |
| def | Định nghĩa hàm |
| for, in | Duyệt qua danh sách |
| if | Kiểm tra điều kiện (x > 0) |
| return | Trả kết quả từ hàm |
| print() | In kết quả ra màn hình |
| len() | Đếm số phần tử |
| sum() | Tính tổng |
| [] | Tạo danh sách |
| += | Cộng dồn giá trị |

Câu 3: Viết chương trình Perl hoặc Python tính trung bình cộng của cả danh sách, trung bình cộng các phần tử dương trong danh sách.

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

**Output**: Trung bình cộng toàn danh sách là: 17.875

Trung bình cộng các số dương là: 24.333333333333332

**Thuật toán:**

Bước 1: Khởi tạo tong\_all = 0, tong\_duong = 0, dem\_duong = 0.

Bước 2 :Duyệt từng phần tử x trong danh sách lst:

* + Cộng x vào tong\_all.
  + Nếu x > 0 thì:
  + Cộng x vào tong\_duong.
  + Tăng dem\_duong lên 1.

Bước 3:Tính trung bình cộng toàn bộ: trung\_binh\_all = tong\_all / n

* + Tính trung bình cộng các số dương:
  + Nếu dem\_duong > 0, tính trung\_binh\_duong = tong\_duong / dem\_duong.
  + Ngược lại, trung bình dương bằng 0.

Bước 4:In kết quả.

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

# Trung bình cộng của cả danh sách

tb\_cong = sum(lst) / len(lst)

print("Trung bình cộng toàn bộ danh sách là:", tb\_cong)

# Trung bình cộng các phần tử dương

duong = [x for x in lst if x > 0]

tb\_cong\_duong = sum(duong) / len(duong)

print("Trung bình cộng các phần tử dương là:", tb\_cong\_duong)

def tinh\_trung\_binh(lst):

return sum(lst) / len(lst)

def tinh\_trung\_binh\_duong(lst):

duong = [x for x in lst if x > 0]

return sum(duong) / len(duong)

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

tb = tinh\_trung\_binh(lst)

print("Trung bình cộng toàn bộ danh sách là:", tb)

tb\_duong = tinh\_trung\_binh\_duong(lst)

print("Trung bình cộng các phần tử dương là:", tb\_duong)

câu 4:

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

for i, x in enumerate(lst):

if x < 0:

print("Vị trí phần tử âm đầu tiên là:", i)

break ;

def:  
def vi\_tri\_am\_dau(lst):

for i, x in enumerate(lst):

if x < 0:

return i

return -1 # nếu không có phần tử âm

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

vt = vi\_tri\_am\_dau(lst)

print("Vị trí phần tử âm đầu tiên là:", vt)

Bài 5: Viết chương trình Perl hoặc Python tìm vị trí của phần tử dương cuối cùng trong danh sách

**A. Xác định input / output**

**Input:** Một danh sách các số nguyên. Ví dụ:[1, -3 , 5 , -2 , 0 , 7 , -9]

**Output:** Vị trí (chỉ số) của phần tử dương cuối cùng (tính từ đầu danh sách, chỉ số bắt đầu từ 0). Nếu không có phần tử dương thì trả về -1

**B. Thuật toán**

Duyệt danh sách từ đầu đến cuối

Mỗi khi gặp số dương, lưu lại chỉ số của nó

Sau khi duyệt xong, in ra chỉ số cuối cùng đã lưu (nếu không có thì trả về -1)

**C. Code**

**Cách 1.1: Không sử dụng hàm có sẵn (viết thẳng trong chương trình, không dùng def)**

lst = [1, -3, 5, -2, 0, 7, -9]

index = -1

for i in range(len(lst)):

if lst[i] > 0:

index = i

print("Vị trí phần tử dương cuối cùng:", index)

Cách 1.2: Sử dụng hàm tự định nghĩa def

def find\_last\_positive(lst):

index = -1

for i in range(len(lst)):

if lst[i] > 0:

index = i

return index

lst = [1, -3, 5, -2, 0, 7, -9]

result = find\_last\_positive(lst)

print("Vị trí phần tử dương cuối cùng:", result)

**Cách 2.1: Sử dụng phương thức/hàm có sẵn của Python**

lst = [1, -3, 5, -2, 0, 7, -9]

index = -1

for i, value in enumerate(lst):

if value > 0:

index = i

print("Vị trí phần tử dương cuối cùng:", index)

**Cách 2.2: Kết hợp phương thức/hàm có sẵn và def**

def find\_last\_positive(lst):

index = -1

for i, value in enumerate(lst):

if value > 0:

index = i

return index

lst = [1, -3, 5, -2, 0, 7, -9]

result = find\_last\_positive(lst)

print("Vị trí phần tử dương cuối cùng:", result)

**D. Keywords**

|  |  |
| --- | --- |
| Từ khóa/Phương thức | Giải thích |
| for | Vòng lặp dùng để duyệt qua các phần tử trong danh sách |
| range() | Tạo dãy số từ 0 đến len(lst)-1 |
| len() | Lấy độ dài danh sách |
| if | Câu điều kiện |
| > | Toán tử so sánh lớn hơn |
| enumerate() | Hàm trả về cả chỉ số (index) và giá trị (value) khi duyệt qua danh sách |
| def | Dùng để định nghĩa hàm |
| return | Trả về giá trị từ hàm |

Bài 6: Viết chương trình Perl hoặc Python tìm phần tử lớn nhất của danh sách và vị trí phần tử lớn nhất cuối cùng

**A. Xác định input / output**

**Input:** Một danh sách các số nguyên. Ví dụ: [1, 5, 3, 9, 7, 9, 2]

Output: Giá trị: 9; Vị trí cuối: 5 (vì phần tử 9 xuất hiện ở vị trí 3 và 5, vị trí cuối là 5)

B. Thuật toán

Duyệt danh sách từ đầu đến cuối

Lưu giá trị lớn nhất hiện tại và vị trí của nó

Nếu gặp phần tử bằng với max hiện tại thì cập nhật vị trí

Nếu gặp phần tử lớn hơn thì cập nhật cả max và vị trí

**C. Code**

**Cách 1.1: Không dùng hàm có sẵn (không dùng def)**

lst = [1, 5, 3, 9, 7, 9, 2]

max\_value = lst[0]

index = 0

for i in range(len(lst)):

if lst[i] > max\_value:

max\_value = lst[i]

index = i

elif lst[i] == max\_value:

index = i

print("Giá trị lớn nhất:", max\_value)

print("Vị trí cuối cùng:", index)

Cách 1.2: Dùng hàm tự định nghĩa def

def find\_max\_and\_index(lst):

max\_value = lst[0]

index = 0

for i in range(len(lst)):

if lst[i] > max\_value:

max\_value = lst[i]

index = i

elif lst[i] == max\_value:

index = i

return max\_value, index

lst = [1, 5, 3, 9, 7, 9, 2]

max\_value, index = find\_max\_and\_index(lst)

print("Giá trị lớn nhất:", max\_value)

print("Vị trí cuối cùng:", index)

**Cách 2.1: Dùng hàm/phương thức có sẵn (max() và enumerate())**

lst = [1, 5, 3, 9, 7, 9, 2]

max\_value = max(lst)

index = -1

for i, value in enumerate(lst):

if value == max\_value:

index = i

print("Giá trị lớn nhất:", max\_value)

print("Vị trí cuối cùng:", index)

**Cách 2.2: Kết hợp max(), enumerate() và hàm def**

def find\_max\_and\_index(lst):

max\_value = max(lst)

index = -1

for i, value in enumerate(lst):

if value == max\_value:

index = i

return max\_value, index

lst = [1, 5, 3, 9, 7, 9, 2]

max\_value, index = find\_max\_and\_index(lst)

print("Giá trị lớn nhất:", max\_value)

print("Vị trí cuối cùng:", index)

**D. Keywords**

|  |  |
| --- | --- |
| **Từ khóa/Phương thức** | **Giải thích** |
| for | Vòng lặp duyệt qua danh sách |
| range() | Tạo dãy số từ 0 đến len(lst)-1 |
| len() | Lấy độ dài danh sách |
| if, elif | Câu điều kiện |
| > | Toán tử so sánh lớn hơn |
| == | Toán tử so sánh bằng |
| enumerate() | Lấy index và value khi duyệt |
| max() | Hàm trả về phần tử lớn nhất trong danh sách |
| def | Định nghĩa hàm |
| return | Trả về giá trị từ hàm |

Bài 7: Viết chương trình Perl hoặc Python tìm phần tử lớn thứ nhì của danh sách và các vị trí của các phần tử đạt giá trị lớn nhì

**A. Xác định input / output**

**Input:** Một danh sách các số nguyên. Ví dụ: [1, 5, 9, 7, 9, 2, 5]

Output:Giá trị: 7; Vị trí: [3]

**B. Thuật toán**

Xác định phần tử lớn nhất

Duyệt lần thứ hai để tìm phần tử nhỏ hơn max nhưng là lớn nhất trong số còn lại (tức là lớn thứ nhì)

Lưu lại tất cả các vị trí có giá trị đó

**C. Code**

**Cách 1.1: Không dùng hàm def**

lst = [1, 5, 9, 7, 9, 2, 5]

max\_value = max(lst)

second\_max = None

# Tìm phần tử lớn thứ nhì

for value in lst:

if value != max\_value:

if (second\_max is None) or (value > second\_max):

second\_max = value

if second\_max is None:

print("Không có phần tử lớn thứ nhì")

else:

positions = []

for i in range(len(lst)):

if lst[i] == second\_max:

positions.append(i)

print("Giá trị lớn thứ nhì:", second\_max)

print("Vị trí các phần tử:", positions)

**Cách 1.2: Có dùng hàm def**

def find\_second\_max\_and\_positions(lst):

max\_value = max(lst)

second\_max = None

for value in lst:

if value != max\_value:

if (second\_max is None) or (value > second\_max):

second\_max = value

if second\_max is None:

return None, []

else:

positions = [i for i, value in enumerate(lst) if value == second\_max]

return second\_max, positions

lst = [1, 5, 9, 7, 9, 2, 5]

second\_max, positions = find\_second\_max\_and\_positions(lst)

if second\_max is None:

print("Không có phần tử lớn thứ nhì")

else:

print("Giá trị lớn thứ nhì:", second\_max)

print("Vị trí các phần tử:", positions)

**Cách 2.1: Dùng hàm/phương thức có sẵn (sắp xếp + enumerate)**

lst = [1, 5, 9, 7, 9, 2, 5]

# Loại trùng để tìm các giá trị duy nhất rồi sắp xếp giảm dần

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

if len(unique\_sorted) < 2:

print("Không có phần tử lớn thứ nhì")

else:

second\_max = unique\_sorted[1]

positions = [i for i, value in enumerate(lst) if value == second\_max]

print("Giá trị lớn thứ nhì:", second\_max)

print("Vị trí các phần tử:", positions)

**Cách 2.2: Dùng def + phương thức có sẵn**

def find\_second\_max\_and\_positions(lst):

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

if len(unique\_sorted) < 2:

return None, []

second\_max = unique\_sorted[1]

positions = [i for i, value in enumerate(lst) if value == second\_max]

return second\_max, positions

lst = [1, 5, 9, 7, 9, 2, 5]

second\_max, positions = find\_second\_max\_and\_positions(lst)

if second\_max is None:

print("Không có phần tử lớn thứ nhì")

else:

print("Giá trị lớn thứ nhì:", second\_max)

print("Vị trí các phần tử:", positions)

**D. Keywords**

|  |  |
| --- | --- |
| **Từ khóa/Phương thức** | **Giải thích** |
| for | Vòng lặp duyệt qua danh sách |
| range() | Tạo dãy số từ 0 đến len(lst)-1 |
| len() | Lấy độ dài danh sách |
| if, elif | Câu điều kiện |
| max() | Tìm phần tử lớn nhất |
| set() | Tập hợp, loại bỏ phần tử trùng |
| sorted() | Sắp xếp danh sách |
| reverse=True | Sắp xếp giảm dần |
| enumerate() | Lấy index và value khi duyệt |
| def | Định nghĩa hàm |
| return | Trả về giá trị từ hàm |
| append() | Thêm phần tử vào danh sách |

Bài 8: Viết chương trình Perl hoặc Python tính số lượng các số dương liên tiếp nhiều nhất

**A. Xác định input / output**

**Input:** Một danh sách các số nguyên. Ví dụ: [1, 2, -3, 4, 5, 6, -1, 2, 3, -4]

Output: 3 (chuỗi [4,5,6] có độ dài 3 là chuỗi số dương liên tiếp dài nhất)

**B. Thuật toán**

Duyệt danh sách từ đầu đến cuối

Dùng 2 biến:

current\_count: đếm số dương liên tiếp đang tính

max\_count: lưu số dương liên tiếp dài nhất từ trước đến nay

Nếu gặp số dương: tăng current\_count, cập nhật max\_count nếu cần

Nếu gặp số không dương (0 hoặc âm): reset current\_count về 0

**C. Code**

**Cách 1.1: Không dùng def**

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

max\_count = 0

current\_count = 0

for value in lst:

if value > 0:

current\_count += 1

if current\_count > max\_count:

max\_count = current\_count

else:

current\_count = 0

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

**Cách 1.2: Dùng hàm def**

def max\_consecutive\_positive(lst):

max\_count = 0

current\_count = 0

for value in lst:

if value > 0:

current\_count += 1

if current\_count > max\_count:

max\_count = current\_count

else:

current\_count = 0

return max\_count

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

result = max\_consecutive\_positive(lst)

print("Số lượng số dương liên tiếp nhiều nhất:", result)

**Cách 2.1: Dùng phương thức có sẵn (dùng groupby của itertools)**

from itertools import groupby

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

# Tách các nhóm liên tiếp (dương và không dương)

groups = groupby(lst, key=lambda x: x > 0)

# Lấy độ dài nhóm dương dài nhất

max\_count = max((len(list(g)) for k, g in groups if k), default=0)

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

**Cách 2.2: Dùng def + phương thức có sẵn**

from itertools import groupby

def max\_consecutive\_positive(lst):

groups = groupby(lst, key=lambda x: x > 0)

return max((len(list(g)) for k, g in groups if k), default=0)

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

result = max\_consecutive\_positive(lst)

print("Số lượng số dương liên tiếp nhiều nhất:", result)

**D. Keywords**

|  |  |
| --- | --- |
| **Từ khóa/Phương thức** | **Giải thích** |
| for | Vòng lặp duyệt qua danh sách |
| if, elif | Tạo dãy số từ 0 đến len(lst)-1 |
|  | Lấy độ dài danh sách |
|  | Câu điều kiện |

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

A. Xác định input / output

**Input:** 1, 2, -3, 4, 5, 6, -1, 2, 3

**Output:** 3

B. Thuật toán

Khởi tạo biến max\_sum và current\_sum là 0.

Khởi tạo biến count để đếm số lượng số dương liên tiếp.

Duyệt qua từng số trong danh sách:

Nếu số là dương, cộng vào current\_sum và tăng count.

Nếu số âm, so sánh current\_sum với max\_sum, cập nhật nếu cần, reset current\_sum và count.

Sau khi duyệt, kiểm tra một lần nữa để đảm bảo không bỏ sót.

C. Code

Cách 1.1: Không sử dụng hàm

numbers = [1, 2, -3, 4, 5, 6, -1, 2, 3]

max\_sum = 0

current\_sum = 0

count = 0

max\_count = 0

for num in numbers:

if num > 0:

current\_sum += num

count += 1

else:

if current\_sum > max\_sum:

max\_sum = current\_sum

max\_count = count

current\_sum = 0

count = 0

if current\_sum > max\_sum:

max\_count = count

print(max\_count)

Cách 1.2: Sử dụng hàm

numbers = [1, 2, -3, 4, 5, 6, -1, 2, 3]

max\_sum = 0

current\_sum = 0

count = 0

max\_count = 0

for num in numbers:

if num > 0:

current\_sum += num

count += 1

else:

if current\_sum > max\_sum:

max\_sum = current\_sum

max\_count = count

current\_sum = 0

count = 0

if current\_sum > max\_sum:

max\_count = count

print(max\_count)

Cách 2.1: Sử dụng phương thức/hàm có sẵn

from itertools import groupby

numbers = [1, 2, -3, 4, 5, 6, -1, 2, 3]

max\_count = max(len(list(group)) for key, group in groupby(numbers) if key > 0)

print(max\_count)

Cách 2.2: Kết hợp phương thức/hàm có sẵn và def

def count\_positive\_sequences(nums):

max\_sum = 0

current\_sum = 0

count = 0

max\_count = 0

for num in nums:

if num > 0:

current\_sum += num

count += 1

else:

if current\_sum > max\_sum:

max\_sum = current\_sum

max\_count = count

current\_sum = 0

count = 0

if current\_sum > max\_sum:

max\_count = count

return max\_count

numbers = [1, 2, -3, 4, 5, 6, -1, 2, 3]

print(count\_positive\_sequences(numbers))

D.Keywords

for: Vòng lặp dùng để duyệt qua các phần tử trong danh sách.

if: Câu điều kiện để kiểm tra một biểu thức.

>: Toán tử so sánh lớn hơn.

max(): Hàm trả về giá trị lớn nhất trong một iterable.

current\_sum: Biến lưu trữ tổng hiện tại của các số dương liên tiếp.

count: Biến đếm số lượng các số dương liên tiếp.

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

A. Xác định input / output

**Input:** 1, -2, 3, -4, -5, 6, -7, 8

**Output:** 8

B. Thuật toán

Khởi tạo biến max\_count và current\_count là 1.

Duyệt qua danh sách và kiểm tra dấu của các phần tử liên tiếp.

Nếu hai phần tử liên tiếp khác dấu, tăng current\_count.

Nếu cùng dấu, so sánh và cập nhật max\_count, reset current\_count.

Cuối cùng, so sánh một lần nữa.

C. Code

Cách 1.1: Không sử dụng hàm

numbers = [1, -2, 3, -4, -5, 6, -7, 8]

max\_count = 1

current\_count = 1

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

if (numbers[i] > 0 and numbers[i-1] < 0) or (numbers[i] < 0 and numbers[i-1] > 0):

current\_count += 1

else:

max\_count = max(max\_count, current\_count)

current\_count = 1

max\_count = max(max\_count, current\_count)

print(max\_count)

Cách 1.2: Sử dụng hàm

numbers = [1, -2, 3, -4, -5, 6, -7, 8]

max\_count = 1

current\_count = 1

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

if (numbers[i] > 0 and numbers[i-1] < 0) or (numbers[i] < 0 and numbers[i-1] > 0):

current\_count += 1

else:

max\_count = max(max\_count, current\_count)

current\_count = 1

max\_count = max(max\_count, current\_count)

print(max\_count)

Cách 2.1: Sử dụng phương thức/hàm có sẵn

from itertools import groupby

numbers = [1, -2, 3, -4, -5, 6, -7, 8]

max\_count = max(len(list(group)) for key, group in groupby(numbers, key=lambda x: x > 0))

print(max\_count)

Cách 2.2: Kết hợp phương thức/hàm có sẵn và def

def count\_alternating\_signs(nums):

max\_count = 1

current\_count = 1

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

if (nums[i] > 0 and nums[i-1] < 0) or (nums[i] < 0 and nums[i-1] > 0):

current\_count += 1

else:

max\_count = max(max\_count, current\_count)

current\_count = 1

max\_count = max(max\_count, current\_count)

return max\_count

numbers = [1, -2, 3, -4, -5, 6, -7, 8]

print(count\_alternating\_signs(numbers))

D.Keywords

for: Vòng lặp dùng để duyệt qua các phần tử trong danh sách.

if: Câu điều kiện để kiểm tra một biểu thức.

current\_count: Biến lưu trữ số lượng phần tử liên tiếp đan dấu hiện tại.

max(): Hàm trả về giá trị lớn nhất trong một iterable.

alternating: Kiểm tra xem hai phần tử có khác dấu hay không.

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

A. Xác định input / output

**Input:** 5, 3, 3, 2, 1, 4, 3, 2

**Output:** 5

B. Thuật toán

Khởi tạo biến max\_count và current\_count là 1.

Duyệt qua danh sách và kiểm tra xem phần tử có nhỏ hơn hoặc bằng phần tử trước đó không.

Nếu có, tăng current\_count.

Nếu không, so sánh và cập nhật max\_count, reset current\_count.

Cuối cùng, so sánh một lần nữa.

C. Code

Cách 1.1: Không sử dụng hàm

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

max\_count = 1

current\_count = 1

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

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

current\_count += 1

else:

max\_count = max(max\_count, current\_count)

current\_count = 1

max\_count = max(max\_count, current\_count)

print(max\_count)

Cách 1.2: Sử dụng hàm

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

max\_count = 1

current\_count = 1

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

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

current\_count += 1

else:

max\_count = max(max\_count, current\_count)

current\_count = 1

max\_count = max(max\_count, current\_count)

print(max\_count)

Cách 2.1: Sử dụng phương thức/hàm có sẵn

from itertools import groupby

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

max\_count = max(len(list(group)) for key, group in groupby(numbers, key=lambda x: x))

print(max\_count)

Cách 2.2: Kết hợp phương thức/hàm có sẵn và def

def count\_non\_increasing(nums):

max\_count = 1

current\_count = 1

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

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

current\_count += 1

else:

max\_count = max(max\_count, current\_count)

current\_count = 1

max\_count = max(max\_count, current\_count)

return max\_count

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

print(count\_non\_increasing(numbers))

D.Keywords

for: Vòng lặp dùng để duyệt qua các phần tử trong danh sách.

if: Câu điều kiện để kiểm tra một biểu thức.

current\_count: Biến lưu trữ số lượng phần tử không tăng hiện tại.

max(): Hàm trả về giá trị lớn nhất trong một iterable.

<=: Toán tử so sánh nhỏ hơn hoặc bằng.

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

A. Xác định input / output

**Input:** 1, 2, -3, 4, 5, 6, -1, 2

**Output:** 3

B. Thuật toán

Khởi tạo biến max\_length và start\_index là 0.

Duyệt qua danh sách để tìm đoạn con dương liên tiếp.

Lưu lại chỉ số bắt đầu khi gặp số dương đầu tiên.

So sánh độ dài đoạn con với max\_length, cập nhật nếu cần.

C. Code

Cách 1.1: Không sử dụng hàm

numbers = [1, 2, -3, 4, 5, 6, -1, 2]

max\_length = 0

current\_length = 0

start\_index = -1

temp\_start = -1

for i in range(len(numbers)):

if numbers[i] > 0:

if current\_length == 0:

temp\_start = i

current\_length += 1

else:

if current\_length > max\_length:

max\_length = current\_length

start\_index = temp\_start

current\_length = 0

if current\_length > max\_length:

start\_index = temp\_start

print(start\_index)

Cách 1.2: Sử dụng hàm

numbers = [1, 2, -3, 4, 5, 6, -1, 2]

max\_length = 0

current\_length = 0

start\_index = -1

temp\_start = -1

for i in range(len(numbers)):

if numbers[i] > 0:

if current\_length == 0:

temp\_start = i

current\_length += 1

else:

if current\_length > max\_length:

max\_length = current\_length

start\_index = temp\_start

current\_length = 0

if current\_length > max\_length:

start\_index = temp\_start

print(start\_index)

Cách 2.1: Sử dụng phương thức/hàm có sẵn

numbers = [1, 2, -3, 4, 5, 6, -1, 2]

max\_length = max(len(list(g)) for k, g in groupby(enumerate(numbers), key=lambda x: x[1] > 0) if k)

start\_index = max\_length[0] if max\_length else -1

print(start\_index)

Cách 2.2: Kết hợp phương thức/hàm có sẵn và def

def find\_start\_of\_longest\_positive\_sequence(nums):

max\_length = 0

current\_length = 0

start\_index = -1

temp\_start = -1

for i in range(len(nums)):

if nums[i] > 0:

if current\_length == 0:

temp\_start = i

current\_length += 1

else:

if current\_length > max\_length:

max\_length = current\_length

start\_index = temp\_start

current\_length = 0

if current\_length > max\_length:

start\_index = temp\_start

return start\_index

numbers = [1, 2, -3, 4, 5, 6, -1, 2]

print(find\_start\_of\_longest\_positive\_sequence(numbers))

D. Keywords

for: Vòng lặp dùng để duyệt qua các phần tử trong danh sách.

if: Câu điều kiện để kiểm tra một biểu thức.

current\_length: Biến lưu trữ độ dài của đoạn con dương hiện tại.

temp\_start: Biến lưu trữ chỉ số bắt đầu của đoạn con dương.

max(): Hàm trả về giá trị lớn nhất trong một iterable.

start\_index: Biến lưu trữ chỉ số bắt đầu của đoạn dương dài nhất

Bài 13: Viết chương trình Perl hoặc Python tìm đoạn con có các số hạng dương liên tiếp có tổng lớn nhất. (Nếu có nhiều đoạn con thoả mãn thì đưa ra màn hình: Số đoạn con thoả mãn và các đoạn con đó).

**A. Input và Output**

**Input:**

* Một danh sách các số nguyên, ví dụ:  
  lst = [1, -2, 3, 5, -1, 2, -3, 4, 5]

**Output:**

* Tổng lớn nhất của các đoạn con liên tiếp **chỉ gồm các số dương**
* Số đoạn con thỏa mãn tổng lớn nhất đó
* Các đoạn con thỏa mãn điều kiện đó  
  **Ví dụ Output:**

Tổng lớn nhất: 9

Số đoạn con thỏa mãn: 1

Các đoạn con:

[4, 5]

**B. Thuật toán – Trình bày từng bước**

1. **Gán danh sách lst.**
2. **Khởi tạo các biến:**
   * max\_sum = 0: tổng lớn nhất ban đầu.
   * current\_sum = 0: tổng đoạn con đang xét.
   * Mảng tạm temp để lưu đoạn con dương đang xét.
   * Mảng max\_sublists để lưu các đoạn con thỏa mãn.
   * Biến đếm count = 0 (số đoạn thỏa mãn).
3. **Duyệt từng phần tử trong danh sách lst:**
   * Nếu phần tử là số **dương**:
     + Cộng vào current\_sum.
     + Thêm vào temp.
   * Nếu phần tử là **số âm hoặc kết thúc danh sách**:
     + So sánh current\_sum với max\_sum:
       - Nếu lớn hơn max\_sum: cập nhật max\_sum, xóa danh sách cũ, lưu temp vào danh sách mới.
       - Nếu bằng max\_sum: thêm temp vào danh sách đoạn con thỏa mãn.
     + Reset current\_sum và temp.
4. **Kiểm tra lần cuối nếu danh sách kết thúc bằng đoạn dương.**
5. **In ra:**
   * Tổng lớn nhất (max\_sum)
   * Số lượng đoạn con thỏa mãn (count)
   * Các đoạn con.
6. **Cách 1.1 – Không dùng hàm/phương thức có sẵn**
7. lst = [1, -2, 3, 5, -1, 2, -3, 4, 5]
8. i = 0
9. n = 9  # số phần tử của lst, nhập tay vì không dùng len()
10. max\_sum = 0
11. current\_sum = 0
12. # mảng 2 chiều để lưu các đoạn con
13. # Vì không dùng append, giả sử tối đa có 10 đoạn, mỗi đoạn dài tối đa 10 phần tử
14. max\_sublists = [[0 for j in range(10)] for i in range(10)]
15. lengths = [0 for i in range(10)]
16. count = 0
17. temp = [0 for i in range(10)]
18. temp\_len = 0
19. while i < n:
20. if lst[i] > 0:
21. current\_sum = current\_sum + lst[i]
22. temp[temp\_len] = lst[i]
23. temp\_len = temp\_len + 1
24. else:
25. if current\_sum > max\_sum:
26. max\_sum = current\_sum
27. count = 1
28. j = 0
29. while j < temp\_len:
30. max\_sublists[0][j] = temp[j]
31. j = j + 1
32. lengths[0] = temp\_len
33. elif current\_sum == max\_sum and current\_sum != 0:
34. j = 0
35. while j < temp\_len:
36. max\_sublists[count][j] = temp[j]
37. j = j + 1
38. lengths[count] = temp\_len
39. count = count + 1
40. current\_sum = 0
41. temp\_len = 0
42. i = i + 1
43. # kiểm tra đoạn cuối nếu kết thúc bằng số dương
44. if current\_sum > max\_sum:
45. max\_sum = current\_sum
46. count = 1
47. j = 0
48. while j < temp\_len:
49. max\_sublists[0][j] = temp[j]
50. j = j + 1
51. lengths[0] = temp\_len
52. elif current\_sum == max\_sum and current\_sum != 0:
53. j = 0
54. while j < temp\_len:
55. max\_sublists[count][j] = temp[j]
56. j = j + 1
57. lengths[count] = temp\_len
58. count = count + 1
59. # in kết quả
60. print("Tổng lớn nhất:", max\_sum)
61. print("Số đoạn con thỏa mãn:", count)
62. i = 0
63. while i < count:
64. j = 0
65. print("Đoạn", i + 1, ":", end=" ")
66. while j < lengths[i]:
67. print(max\_sublists[i][j], end=" ")
68. j = j + 1
69. print()
70. i = i + 1

Cách 1.2 – Viết lại như hàm, vẫn **không dùng hàm có sẵn**

def tim\_doan\_con(lst, n):

    i = 0

    max\_sum = 0

    current\_sum = 0

    max\_sublists = [[0 for j in range(10)] for i in range(10)]

    lengths = [0 for i in range(10)]

    count = 0

    temp = [0 for i in range(10)]

    temp\_len = 0

    while i < n:

        if lst[i] > 0:

            current\_sum = current\_sum + lst[i]

            temp[temp\_len] = lst[i]

            temp\_len = temp\_len + 1

        else:

            if current\_sum > max\_sum:

                max\_sum = current\_sum

                count = 1

                j = 0

                while j < temp\_len:

                    max\_sublists[0][j] = temp[j]

                    j = j + 1

                lengths[0] = temp\_len

            elif current\_sum == max\_sum and current\_sum != 0:

                j = 0

                while j < temp\_len:

                    max\_sublists[count][j] = temp[j]

                    j = j + 1

                lengths[count] = temp\_len

                count = count + 1

            current\_sum = 0

            temp\_len = 0

        i = i + 1

    if current\_sum > max\_sum:

        max\_sum = current\_sum

        count = 1

        j = 0

        while j < temp\_len:

            max\_sublists[0][j] = temp[j]

            j = j + 1

        lengths[0] = temp\_len

    elif current\_sum == max\_sum and current\_sum != 0:

        j = 0

        while j < temp\_len:

            max\_sublists[count][j] = temp[j]

            j = j + 1

        lengths[count] = temp\_len

        count = count + 1

    return max\_sum, count, max\_sublists, lengths

# Gọi hàm

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

n = 9

tong, dem, doans, chieudai = tim\_doan\_con(lst, n)

print("Tổng lớn nhất:", tong)

print("Số đoạn con thỏa mãn:", dem)

i = 0

while i < dem:

    j = 0

    print("Đoạn", i + 1, ":", end=" ")

    while j < chieudai[i]:

        print(doans[i][j], end=" ")

        j = j + 1

    print()

    i = i + 1

Cách 2.1 – Dùng hàm/phương thức có sẵn

from itertools import groupby

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

# Tách các đoạn con dương liên tiếp

subsequences = [list(g) for k, g in groupby(lst, key=lambda x: x > 0) if k]

# Tính tổng từng đoạn

sums = [sum(sub) for sub in subsequences]

# Tìm tổng lớn nhất

max\_sum = max(sums)

# Lọc các đoạn con có tổng bằng max\_sum

result = [subsequences[i] for i in range(len(sums)) if sums[i] == max\_sum]

print("Tổng lớn nhất:", max\_sum)

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

for r in result:

    print(r)

Cách 2.2 – Viết thành hàm, dùng phương thức có sẵn

from itertools import groupby

def tim\_doan\_con\_dung\_itertools(lst):

    subsequences = [list(g) for k, g in groupby(lst, key=lambda x: x > 0) if k]

    tong = [sum(sub) for sub in subsequences]

    max\_sum = max(tong)

    ket\_qua = [subsequences[i] for i in range(len(tong)) if tong[i] == max\_sum]

    return max\_sum, ket\_qua

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

tong, ds = tim\_doan\_con\_dung\_itertools(lst)

print("Tổng lớn nhất:", tong)

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

for doan in ds:

    print(doan)

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

|  |  |
| --- | --- |
| **Từ khóa** | **Giải thích** |
| current\_sum | Biến lưu tổng tạm thời của đoạn con đang xét |
| max\_sum | Biến lưu tổng lớn nhất đã tìm được |
| temp | Mảng tạm lưu các số dương liên tiếp hiện tại |
| max\_sublists | Mảng 2 chiều lưu các đoạn con có tổng lớn nhất |
| if, else, while | Câu lệnh điều kiện và vòng lặp |
| def | Khai báo hàm |
| [x for x in lst if ...] | List comprehension – tạo danh sách theo điều kiện (chỉ dùng ở cách 2) |
| groupby() | Nhóm các phần tử liên tiếp theo điều kiện (có trong cách 2.1, 2.2) |

Bài 14: Viết chương trình P Perl hoặc Python đếm số lượng các phần tử bằng giá trị X nhập từ bàn phím.

**Input và Output**

🔹 **Input:**

* Một danh sách số nguyên, ví dụ:  
  lst = [12, -14, 11, 91, -3, 10, 7, 15]
* Một giá trị X do người dùng nhập từ bàn phím, ví dụ:  
  X = 10

🔹 **Output:**

* Số lượng phần tử trong danh sách bằng X.  
  **Ví dụ Output:**

css

Sao chépChỉnh sửa

Số lượng phần tử bằng 10 là: 1

**B. Thuật toán – Trình bày từng bước**

1. Gán danh sách lst.
2. Nhập giá trị X từ bàn phím.
3. Khởi tạo biến count = 0 để đếm số phần tử bằng X.
4. Duyệt từng phần tử trong lst:
   * Nếu phần tử bằng X, tăng count lên 1.
5. In ra giá trị của count.

Cách 1.1 – Không dùng hàm/phương thức có sẵn

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

# Nhập giá trị X từ bàn phím (dùng thủ công không dùng input())

X = 10  # ví dụ người dùng nhập 10

count = 0

i = 0

while i < 8:  # vì không dùng len(), số phần tử viết tay

    if lst[i] == X:

        count = count + 1

    i = i + 1

print("Số lượng phần tử bằng", X, "là:", count)

Cách 1.2 – Viết dưới dạng hàm, không dùng hàm có sẵn

def dem\_phan\_tu\_bang\_X(lst, n, X):

    count = 0

    i = 0

    while i < n:

        if lst[i] == X:

            count = count + 1

        i = i + 1

    return count

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

n = 8  # số phần tử

X = 10

so\_luong = dem\_phan\_tu\_bang\_X(lst, n, X)

print("Số lượng phần tử bằng", X, "là:", so\_luong)

Cách 2.1 – Dùng hàm/phương thức có sẵn

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

X = int(input("Nhập giá trị X: "))

count = lst.count(X)

print("Số lượng phần tử bằng", X, "là:", count)

Cách 2.2 – Dạng hàm, dùng phương thức có sẵn

def dem\_so\_luong(lst, X):

    return lst.count(X)

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

X = int(input("Nhập giá trị X: "))

print("Số lượng phần tử bằng", X, "là:", dem\_so\_luong(lst, X))

## **Bài 14 – Đếm số lượng phần tử bằng X**

|  |  |
| --- | --- |
| **Từ khóa** | **Giải thích** |
| input() | Nhập dữ liệu từ bàn phím (có ở cách 2) |
| int() | Ép kiểu chuỗi sang số nguyên (có ở cách 2) |
| count += 1 | Tăng biến đếm lên 1 mỗi khi tìm thấy phần tử thỏa mãn |
| lst.count(X) | Phương thức có sẵn để đếm số lần X xuất hiện trong danh sách (chỉ dùng ở cách 2) |
| for, while | Duyệt danh sách tuần tự |
| if | Câu lệnh điều kiện để kiểm tra i == X |

Bài 15: Viết chương trình Perl hoặc Python chuyển các phần tử dương của danh sách lên đầu danh sách và in danh sách ra màn hình.

**A. Input và Output**

**Input:**

* Một danh sách các số nguyên, ví dụ:  
  lst = [3, -1, 0, 4, -2, 5]

**Output:**

* Danh sách mới trong đó **các phần tử dương được chuyển lên đầu**, các phần tử khác giữ nguyên thứ tự phía sau.

**Ví dụ Output:**

less

Sao chépChỉnh sửa

Danh sách sau khi chuyển: [3, 4, 5, -1, 0, -2]

**B. Thuật toán – Trình bày từng bước**

1. Gán danh sách lst.
2. Khởi tạo hai danh sách phụ:
   * duong chứa các phần tử > 0
   * con\_lai chứa các phần tử ≤ 0
3. Duyệt từng phần tử trong lst:
   * Nếu > 0 → đưa vào duong
   * Ngược lại → đưa vào con\_lai
4. Nối hai danh sách duong + con\_lai thành danh sách mới.
5. In ra kết quả.

Cách 1.1 – Không dùng hàm/phương thức có sẵn

lst = [3, -1, 0, 4, -2, 5]

n = 6  # số phần tử

duong = [0 for i in range(n)]

con\_lai = [0 for i in range(n)]

k1 = 0  # số phần tử trong duong

k2 = 0  # số phần tử trong con\_lai

i = 0

while i < n:

    if lst[i] > 0:

        duong[k1] = lst[i]

        k1 = k1 + 1

    else:

        con\_lai[k2] = lst[i]

        k2 = k2 + 1

    i = i + 1

# Gộp hai danh sách

ket\_qua = [0 for i in range(n)]

i = 0

while i < k1:

    ket\_qua[i] = duong[i]

    i = i + 1

j = 0

while j < k2:

    ket\_qua[i + j] = con\_lai[j]

    j = j + 1

# In kết quả

print("Danh sách sau khi chuyển:", end=" ")

i = 0

while i < n:

    print(ket\_qua[i], end=" ")

    i = i + 1

Cách 1.2 – Dạng hàm, không dùng hàm/phương thức có sẵn

def chuyen\_phan\_tu\_duong(lst, n):

    duong = [0 for i in range(n)]

    con\_lai = [0 for i in range(n)]

    k1 = 0

    k2 = 0

    i = 0

    while i < n:

        if lst[i] > 0:

            duong[k1] = lst[i]

            k1 = k1 + 1

        else:

            con\_lai[k2] = lst[i]

            k2 = k2 + 1

        i = i + 1

    ket\_qua = [0 for i in range(n)]

    i = 0

    while i < k1:

        ket\_qua[i] = duong[i]

        i = i + 1

    j = 0

    while j < k2:

        ket\_qua[i + j] = con\_lai[j]

        j = j + 1

    return ket\_qua

lst = [3, -1, 0, 4, -2, 5]

n = 6

ket\_qua = chuyen\_phan\_tu\_duong(lst, n)

print("Danh sách sau khi chuyển:", end=" ")

i = 0

while i < n:

    print(ket\_qua[i], end=" ")

    i = i + 1

Cách 2.1 – Dùng hàm/phương thức có sẵn

lst = [3, -1, 0, 4, -2, 5]

duong = [x for x in lst if x > 0]

con\_lai = [x for x in lst if x <= 0]

ket\_qua = duong + con\_lai

print("Danh sách sau khi chuyển:", ket\_qua)

Cách 2.2 – Viết dạng hàm, dùng phương thức có sẵn

def chuyen\_phan\_tu\_duong(lst):

    duong = [x for x in lst if x > 0]

    con\_lai = [x for x in lst if x <= 0]

    return duong + con\_lai

lst = [3, -1, 0, 4, -2, 5]

ket\_qua = chuyen\_phan\_tu\_duong(lst)

print("Danh sách sau khi chuyển:", ket\_qua)

## **Bài 15 – Đưa phần tử dương lên đầu danh sách**

|  |  |
| --- | --- |
| **Từ khóa** | **Giải thích** |
| lst[i] > 0 | Kiểm tra số dương |
| duong, con\_lai | Danh sách phụ chứa số dương và còn lại |
| ket\_qua = duong + con\_lai | Ghép hai danh sách lại với nhau (chỉ dùng ở cách 2) |
| [x for x in lst if ...] | Tạo danh sách bằng điều kiện lọc (chỉ dùng ở cách 2) |
| while, if | Duyệt danh sách và kiểm tra điều kiện |
| def | Định nghĩa hàm xử lý dữ liệu |

Bài 16: Viết chương trình Perl hoặc Python tìm số phần tử là số nguyên tố của danh sách và vị trí của nó trong danh sách.

## **A. Input và Output**

**Input:**

* Một danh sách các số nguyên, ví dụ:  
  lst = [2, 4, 5, 6, 7, 9, 11, 12]

**Output:**

* Tổng số phần tử là **số nguyên tố**
* Vị trí của các số nguyên tố trong danh sách (tính từ chỉ số 0)

**Ví dụ Output:**

bash

Sao chépChỉnh sửa

Có 4 số nguyên tố

Vị trí: 0 2 4 6

## 🔶 **B. Thuật toán – Trình bày từng bước**

1. Gán danh sách lst.
2. Viết hàm kiểm tra số nguyên tố (chỉ dùng chia từ 2 đến căn bậc hai của số).
3. Duyệt từng phần tử trong danh sách lst:
   * Nếu là số nguyên tố:
     + Tăng biến đếm.
     + Lưu lại vị trí của phần tử đó.
4. In ra số lượng số nguyên tố và các vị trí.

**Cách 1.1 – Không dùng hàm/phương thức có sẵn**

lst = [2, 4, 5, 6, 7, 9, 11, 12]

n = 8  # số phần tử

count = 0

index\_list = [0 for i in range(n)]

i = 0

while i < n:

    num = lst[i]

    if num < 2:

        i = i + 1

        continue

    j = 2

    la\_nguyen\_to = 1

    while j \* j <= num:

        if num % j == 0:

            la\_nguyen\_to = 0

            break

        j = j + 1

    if la\_nguyen\_to == 1:

        index\_list[count] = i

        count = count + 1

    i = i + 1

# In kết quả

print("Có", count, "số nguyên tố")

print("Vị trí:", end=" ")

i = 0

while i < count:

    print(index\_list[i], end=" ")

    i = i + 1

Cách 1.2 – Viết hàm, không dùng hàm/phương thức có sẵn

def la\_nguyen\_to(n):

    if n < 2:

        return 0

    i = 2

    while i \* i <= n:

        if n % i == 0:

            return 0

        i = i + 1

    return 1

def tim\_so\_nguyen\_to(lst, n):

    index\_list = [0 for i in range(n)]

    count = 0

    i = 0

    while i < n:

        if la\_nguyen\_to(lst[i]) == 1:

            index\_list[count] = i

            count = count + 1

        i = i + 1

    return count, index\_list

lst = [2, 4, 5, 6, 7, 9, 11, 12]

n = 8

so\_luong, vi\_tri = tim\_so\_nguyen\_to(lst, n)

print("Có", so\_luong, "số nguyên tố")

print("Vị trí:", end=" ")

i = 0

while i < so\_luong:

    print(vi\_tri[i], end=" ")

    i = i + 1

**Cách 2.1 – Dùng hàm/phương thức có sẵn**

import math

lst = [2, 4, 5, 6, 7, 9, 11, 12]

index\_list = []

count = 0

for i in range(len(lst)):

    num = lst[i]

    if num < 2:

        continue

    la\_nguyen\_to = True

    for j in range(2, int(math.sqrt(num)) + 1):

        if num % j == 0:

            la\_nguyen\_to = False

            break

    if la\_nguyen\_to:

        count += 1

        index\_list.append(i)

print("Có", count, "số nguyên tố")

print("Vị trí:", index\_list)

**Cách 2.2 – Viết thành hàm, dùng hàm/phương thức có sẵn**

import math

def is\_prime(n):

    if n < 2:

        return False

    for i in range(2, int(math.sqrt(n)) + 1):

        if n % i == 0:

            return False

    return True

def tim\_vi\_tri\_so\_nguyen\_to(lst):

    vi\_tri = []

    for i in range(len(lst)):

        if is\_prime(lst[i]):

            vi\_tri.append(i)

    return len(vi\_tri), vi\_tri

lst = [2, 4, 5, 6, 7, 9, 11, 12]

so\_luong, danh\_sach\_vi\_tri = tim\_vi\_tri\_so\_nguyen\_to(lst)

print("Có", so\_luong, "số nguyên tố")

print("Vị trí:", danh\_sach\_vi\_tri)

## **Bài 16 – Tìm số nguyên tố và vị trí**

|  |  |
| --- | --- |
| **Từ khóa** | **Giải thích** |
| số nguyên tố | Số lớn hơn 1, chỉ chia hết cho 1 và chính nó |
| i \* i <= n | Tối ưu kiểm tra số nguyên tố đến căn bậc hai của n |
| la\_nguyen\_to | Biến cờ đánh dấu xem một số có phải nguyên tố không |
| continue | Bỏ qua bước còn lại của vòng lặp nếu điều kiện đúng |
| math.sqrt() | Tính căn bậc hai (chỉ dùng ở cách 2) |
| range(len(lst)) | Duyệt chỉ số các phần tử trong danh sách |
| append() | Thêm phần tử vào danh sách (chỉ dùng ở cách 2) |
| def | Định nghĩa hàm kiểm tra số nguyên tố và xử lý |

**CÂU 17**

# C/ CODE

# 1.1 Không sử dụng hàm, thư viện có sẵn (không def)

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

m = int(input("Nhập số m cần chèn: "))

lst = [m] + lst

print("Danh sách sau khi chèn vào đầu:", lst)

lst = lst + [m]

print("Danh sách sau khi chèn vào cuối:", lst)

if lst and lst[3] is not None:  # Kiểm tra vị trí thứ 4

    lst = lst[:4] + [m] + lst[4:]

    print("Danh sách sau khi chèn vào vị trí thứ 5:", lst)

else:

    print("Danh sách quá ngắn để chèn vào vị trí thứ 5")

# 1.2 Không sử dụng hàm, thư viện có sẵn (có def)

def chen\_dau(lst, m):

    return [m] + lst

def chen\_cuoi(lst, m):

    return lst + [m]

def chen\_vi\_tri\_5(lst, m):

    if len(lst) >= 4:

        return lst[:4] + [m] + lst[4:]

    else:

        return lst

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

m = int(input("Nhập số m cần chèn: "))

lst = chen\_dau(lst, m)

print("Danh sách sau khi chèn vào đầu:", lst)

lst = chen\_cuoi(lst, m)

print("Danh sách sau khi chèn vào cuối:", lst)

lst = chen\_vi\_tri\_5(lst, m)

print("Danh sách sau khi chèn vào vị trí thứ 5:", lst)

# 2.1 Sử dụng hàm, thư viện có sẵn (không def)

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

m = int(input("Nhập số m cần chèn: "))

lst.insert(0, m)

print("Danh sách sau khi chèn vào đầu:", lst)

lst.append(m)

print("Danh sách sau khi chèn vào cuối:", lst)

if len(lst) >= 4:

    lst.insert(4, m)

    print("Danh sách sau khi chèn vào vị trí thứ 5:", lst)

else:

    print("Danh sách quá ngắn để chèn vào vị trí thứ 5")

# 2.2 Sử dụng hàm, thư viện có sẵn (có def)

def chen\_dau(lst, m):

    lst.insert(0, m)

    return lst

def chen\_cuoi(lst, m):

    lst.append(m)

    return lst

def chen\_vi\_tri\_5(lst, m):

    if len(lst) >= 4:

        lst.insert(4, m)

    return lst

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

m = int(input("Nhập số m cần chèn: "))

lst = chen\_dau(lst, m)

print("Danh sách sau khi chèn vào đầu:", lst)

lst = chen\_cuoi(lst, m)

print("Danh sách sau khi chèn vào cuối:", lst)

lst = chen\_vi\_tri\_5(lst, m)

print("Danh sách sau khi chèn vào vị trí thứ 5:", lst)

**CÂU 18**

# C/ CODE

# 1.1 Không sử dụng hàm, thư viện có sẵn (không def)

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

list\_to\_insert = [1, 2, 3]

new\_list = list\_to\_insert + lst

print("Chèn vào đầu:", new\_list)

new\_list = lst + list\_to\_insert

print("Chèn vào cuối:", new\_list)

if lst and lst[3] is not None:

    new\_list = lst[:4] + list\_to\_insert + lst[4:]

    print("Chèn vào vị trí thứ 5:", new\_list)

else:

    print("Danh sách quá ngắn để chèn vào vị trí thứ 5")

# 1.2 Không sử dụng hàm, thư viện có sẵn (có def)

def insert\_at\_beginning(main\_list, insert\_list):

    return insert\_list + main\_list

def insert\_at\_end(main\_list, insert\_list):

    return main\_list + insert\_list

def insert\_at\_position\_5(main\_list, insert\_list):

    if main\_list and main\_list[3] is not None:

        return main\_list[:4] + insert\_list + main\_list[4:]

    return main\_list

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

insert\_list = [1, 2, 3]

print("Chèn vào đầu:", insert\_at\_beginning(lst, insert\_list))

print("Chèn vào cuối:", insert\_at\_end(lst, insert\_list))

print("Chèn vào vị trí thứ 5:", insert\_at\_position\_5(lst, insert\_list))

# 2.1 Sử dụng hàm, thư viện có sẵn (không def)

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

insert\_list = [1, 2, 3]

lst[:0] = insert\_list

print("Chèn vào đầu:", lst)

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

lst.extend(insert\_list)

print("Chèn vào cuối:", lst)

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

if len(lst) >= 4:

    lst[4:4] = insert\_list

    print("Chèn vào vị trí thứ 5:", lst)

else:

    print("Danh sách quá ngắn để chèn vào vị trí thứ 5")

# 2.2 Sử dụng hàm, thư viện có sẵn (có def)

def insert\_at\_beginning(main\_list, insert\_list):

    main\_list[:0] = insert\_list

    return main\_list

def insert\_at\_end(main\_list, insert\_list):

    main\_list.extend(insert\_list)

    return main\_list

def insert\_at\_position\_5(main\_list, insert\_list):

    if len(main\_list) >= 4:

        main\_list[4:4] = insert\_list

    return main\_list

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

insert\_list = [1, 2, 3]

import copy

print("Chèn vào đầu:", insert\_at\_beginning(copy.deepcopy(lst), insert\_list))

print("Chèn vào cuối:", insert\_at\_end(copy.deepcopy(lst), insert\_list))

print("Chèn vào vị trí thứ 5:", insert\_at\_position\_5(copy.deepcopy(lst), insert\_list))

**CÂU 19**

# C/ CODE

# 1.1 Không sử dụng hàm, thư viện có sẵn (không def)

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

k = int(input("Nhập vị trí k cần xóa (0-based index): "))

if lst:

    if k >= 0 and k < (lst[::1].count(0) + len(lst)):

        lst = lst[:k] + lst[k+1:]

        print("Danh sách sau khi xóa:", lst)

    else:

        print("Vị trí không hợp lệ!")

else:

    print("Danh sách rỗng!")

# 1.2 Không sử dụng hàm, thư viện có sẵn (có def)

def xoa\_phan\_tu(danh\_sach, vi\_tri):

    if danh\_sach:

        for index in range(len(danh\_sach)):

            if index == vi\_tri:

                return danh\_sach[:vi\_tri] + danh\_sach[vi\_tri + 1:]

    return danh\_sach

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

k = int(input("Nhập vị trí k cần xóa (0-based index): "))

print("Danh sách sau khi xóa:", xoa\_phan\_tu(lst, k))

# 2.1 Sử dụng hàm, thư viện có sẵn (không def)

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

k = int(input("Nhập vị trí k cần xóa (0-based index): "))

if 0 <= k < len(lst):

    lst.pop(k)

    print("Danh sách sau khi xóa:", lst)

else:

    print("Vị trí không hợp lệ!")

# 2.2 Sử dụng hàm, thư viện có sẵn (có def)

def xoa\_phan\_tu(danh\_sach, vi\_tri):

    if 0 <= vi\_tri < len(danh\_sach):

        danh\_sach.pop(vi\_tri)

    return danh\_sach

lst = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

k = int(input("Nhập vị trí k cần xóa (0-based index): "))

print("Danh sách sau khi xóa:", xoa\_phan\_tu(lst.copy(), k))

**CÂU 20**

# C/ CODE

# 1.1 Không sử dụng hàm, thư viện có sẵn (không def)

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

n = 0

for \_ in lst:

    n += 1

for i in range(n-1):

    for j in range(n-i-1):

        if lst[j] > lst[j+1]:

            lst[j], lst[j+1] = lst[j+1], lst[j]

print("Tăng dần:", lst)

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

for i in range(n-1):

    for j in range(n-i-1):

        if lst[j] < lst[j+1]:

            lst[j], lst[j+1] = lst[j+1], lst[j]

print("Giảm dần:", lst)

# 1.2 Không sử dụng hàm, thư viện có sẵn (có def)

def bubble\_sort\_ascending(arr):

    n = 0

    for \_ in arr:

        n += 1

    for i in range(n-1):

        for j in range(n-i-1):

            if arr[j] > arr[j+1]:

                arr[j], arr[j+1] = arr[j+1], arr[j]

    return arr

def bubble\_sort\_descending(arr):

    n = 0

    for \_ in arr:

        n += 1

    for i in range(n-1):

        for j in range(n-i-1):

            if arr[j] < arr[j+1]:

                arr[j], arr[j+1] = arr[j+1], arr[j]

    return arr

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

print("Tăng dần:", bubble\_sort\_ascending(lst.copy()))

print("Giảm dần:", bubble\_sort\_descending(lst.copy()))

# 2.1 Sử dụng hàm, thư viện có sẵn (không def)

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

lst\_asc = sorted(lst)

print("Tăng dần:", lst\_asc)

lst\_desc = sorted(lst, reverse=True)

print("Giảm dần:", lst\_desc)

# 2.2 Sử dụng hàm, thư viện có sẵn (có def)

def sap\_xep\_tang(arr):

    return sorted(arr)

def sap\_xep\_giam(arr):

    return sorted(arr, reverse=True)

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

print("Tăng dần:", sap\_xep\_tang(lst))

print("Giảm dần:", sap\_xep\_giam(lst))