**Section 1**

Bài tập: Khởi tạo List và truy xuất các phần tử thông qua chỉ số (index)

# cau 1: tạo list có 10 phần tử từ 1 đến 10

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

# cau 2: in ra ket qua 5 phan tu dau tien

print('Câu 2:', end=" ")

for index in range(0, 6):

    print(lst\_data[index], end=" ")

# câu 3: in ra kết quả phần tử có giá trị không chia hết cho 2 (kết hợp for)

print('\nCâu 3: ', end=" ")

for x in lst\_data:

    if x % 2 != 0:

        print(x, end=" ")

# câu 4: In ra kết quả tổng các giá trị trong list (dùng kết hợp với vòng lặp for)

print('\nCâu 4: ', end="")

sum = 0

for x in lst\_data:

    sum = sum+x

print(sum)

**Section 2**

2. Bài tập: Khởi tạo List và thao tác thêm xóa sửa trên List

* Câu 1: Tạo mới một List có tên là *lst*\_*data*, gồm các số chẵn từ 1 đến 12. • Câu 2: Xóa tất cả các số chia hết cho 3 trong *lst*\_*data* vừa tạo
* Câu 3: Thêm vào cuối *lst*\_*data* các số từ 1 đến 3, và thêm vào vị trí *index* = 3 một chuỗi các số từ 6 đến 8
* Câu 4: Nếu các số trong list *lst*\_*data* chia hết cho 2 hoặc chia hết cho 5 thì cập nhật thành số 0
* # Bài tập: Khởi tạo List và thao tác thêm xóa sửa trên List
* # Câu 1: Tạo mới một List có tên là lst\_data, gồm các số chẵn từ 1 đến 12.
* lst\_data = [2, 4, 6, 8, 10, 12]
* # Câu 2: Xóa tất cả các số chia hết cho 3 trong lst\_data vừa tạo
* for x in lst\_data:
* if x % 3 == 0:
* lst\_data.remove(x)
* print(lst\_data)
* # Câu 3: Thêm vào cuối lst\_data các số từ 1 đến 3,
* # và thêm vào vị trí index = 3 một chuỗi các số từ 6 đến 8
* lst\_data.extend([1, 2, 3])
* for k in range(8, 5, -1):
* lst\_data.insert(3, k)
* print(lst\_data)
* # Câu 4: Nếu các số trong list lst\_data chia hết cho 2
* # hoặc chia hết cho 5 thì cập nhật thành số 0
* lenght = len(lst\_data)
* for index in range(lenght):
* if lst\_data[index] % 2 == 0 or lst\_data[index] % 5 == 0:
* lst\_data[index] = 0
* print(lst\_data)

Section 3

# Bước 1: Viết hàm xét số Armstrong, hàm trả về 1 nếu phần tử đang xét là số Armstrong,

# ngược lại trả về 0.

def checkArmstrong(x):

    t = x

    sum = 0

    while (t > 0):

        k = t % 10

        sum = sum+k\*\*3

        t = t//10

    if sum == x:

        return 1

    else:

        return 0

# Bước 2: Chạy vòng lặp qua tất cả các phần tử của list.

test\_case1 = [130, 270, 153, 407, 177, 371, 1000, 1634, 370]

result = []

# Bước 3: Xét điều kiện nếu phần từ là số Armtrong thì lưu vào 1 list khác.

for x in test\_case1:

    if checkArmstrong(x):

        result.append(x)

# Bước 4: In ra list kết quả chứa các số Armstrong đã xét.

print('Các số Armstrong có trong list là: ', result)

Section 4 - List and Branching

# Cho một list và số extra\_candies trả về boolean list.

def getBooleanList(lst\_data, extra\_candies):

    max\_candies = max(lst\_data)+extra\_candies

    result = []

    lenght = len(lst\_data)

    for index in range(lenght):

        result.append(lst\_data[index]+extra\_candies == max\_candies)

        # if lst\_data[index]+extra\_candies == max\_candies:

        #   result.append(True)

        # else:

        #    result.append(False)

    return result

# testcase

test\_case\_1 = [2, 3, 5, 1, 3]

extraCandies1 = 3

print(getBooleanList(test\_case\_1, extraCandies1))

test\_case\_2 = [4, 2, 1, 1, 2]

extraCandies2 = 1

print(getBooleanList(test\_case\_2, extraCandies2))

test\_case\_3 = [12, 1, 12]

extraCandies3 = 10

print(getBooleanList(test\_case\_3, extraCandies3))

Section 5 - Computing median for a list of

# Câu 1: Tạo mới một List có tên là lst\_data, gồm các số từ 1 đến 10.

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

print("Câu 1:", lst\_data)

# Câu 2: Tính giá trị trung vị từ lst\_data vừa tạo. (Không sử dụng numpy)

lst\_data.sort()

length = len(lst\_data)

k = length//2

if length % 2 == 0:

    m = (lst\_data[k-1]+lst\_data[k])/2

else:

    k = (length+1)//2

    m = lst\_data[k-1]

print('Câu 2. Median: ', m)

# Câu 3: Lọc các giá trị số lẻ trong lst\_data và lưu ra list mới có tên là:

# lst\_odd\_filter với thứ tự giảm dần

# ( Sử dụng phương thức reverse=True trong hàm sort/sorted).

lst\_odd\_filter = []

for x in lst\_data:

    if x % 2 != 0:

        lst\_odd\_filter.append(x)

lst\_odd\_filter.sort(reverse=True)

print('Câu 3:', lst\_odd\_filter)

**Section 6 - Computing mean for a list of numbers**

def getMedian(data):

    data.sort()

    length = len(data)

    k = length//2

    if length % 2 == 0:

        m = (data[k-1]+data[k])/2

    else:

        k = (length+1)//2

        m = data[k-1]

    return m

# Câu 1: Tạo mới một List có tên là lst\_data, gồm các số từ 1 đến 10.

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

print("Câu 1:", lst\_data)

# Câu 2: Tính giá trị trung bình cho các số lẻ và số chẵn từ lst\_data vừa tạo.

# (Không sử dụng numpy)

lst\_odd = [x for x in lst\_data if x % 2 != 0]

lst\_even = [x for x in lst\_data if x % 2 == 0]

meanEven = sum(lst\_even)/len(lst\_even)

meanOdd = sum(lst\_odd)/len(lst\_odd)

print(f'Câu 2: Mean lẻ:{meanOdd} - Mean chẵn: {meanEven}')

# Câu 3: Tính giá trị trung bình và trung vị cho tất cả

# dữ liệu trong lst\_data và cho nhận xét.

mean = sum(lst\_data)/len(lst\_data)

median = getMedian(lst\_data)

if (mean == median):

    print('Câu 3: Mean = Median =', mean)

else:

    print(f'Câu 3: Mean {mean} != Median ={median}')

Section 7 - Bag of Words (NLP)

def tokeNization(data):

    result = []

    rows = len(data)

    for id in range(rows):

        words = data[id].split(' ')

        result.extend(words)

    # loại bỏ từ trùng nhau

    vocabulary = []

    [vocabulary.append(x) for x in result if x not in vocabulary]

    vocabulary.sort()

    return vocabulary

def getVector(data, voca):

    result = []

    lst = data.split(' ')

    len\_voca = len(voca)

    for index in range(len\_voca):

        result.append(lst.count(voca[index]))

    return result

corpus = ['Tôi thích Toán', 'Tôi thích AI', 'Tôi thích âm nhạc']

lst\_vocabulary = tokeNization(corpus)

inputText = 'Tôi thích AI thích Toán'

print(f'{inputText}: {getVector(inputText, lst\_vocabulary)}')

print(f'Bag-of-Words:{lst\_vocabulary}')

Section 8 - Algorithms on List

# tìm vị trí None đầu tiên, nếu không thấy trả về -1

def search\_first(data, x):

    for a in data:

        if a == x:

            return data.index(a)

    return -1

# tìm tất cả vị trí None, nếu không thấy trả về list rỗng

def search\_all(data, x):

    result = []

    length = len(data)

    for index in range(length):

        if (data[index] == x):

            result.append(index)

    return result

# ham main

# tạo list data

lst\_data = [1, 1.1, None, 1.4, None, 1.5, None, 2.0]

print(f'Vị trí None đầu tiên: {search\_first(lst\_data, None)} - ', end=' ')

print(f'Danh sách vị trí có giá trị None: {search\_all(lst\_data, None)}')

Section 9 - Interpolation for List (Time-series) *Hoàng-Nguyên Vũ*

# phương pháp nội suy Nearest Neighbor

def NearestNeighbor(data, x):

    while (data.count(x) > 0):

        index = data.index(x)

        tmp = min(data[index-1], data[index+1])

        data[index] = tmp

    return data

lst\_data = [1, 1.1, None, 1.4, None, 1.5, None, 2.0]

# áp dụng phương pháp nội suy Nearest Neighbor để gắn giá trị None có trong lst\_data

print(NearestNeighbor(lst\_data, None))

Section 10 - 2D List

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

rows = len(lst\_data)

cols = len(lst\_data[0])

sub\_list = []

for idr in range(rows):

    sub\_row = []

    for idc in range(cols):

        if (idc == 0 or idc == 2):

            sub\_row.append(lst\_data[idr][idc])

    sub\_list.append(sub\_row)

print(sub\_list)

Section 11 - Matrix representation using List

# tính tổng 2 ma trận

def Tong2MaTran(a, b):

    result = []

    rows = len(a)

    cols = len(a[0])

    for idr in range(rows):

        row\_tmp = []

        for idc in range(cols):

            t = a[idr][idc]+b[idr][idc]

            row\_tmp.append(t)

        result.append(row\_tmp)

    return result

# hieu 2 ma tran

def Hieu2MaTran(a, b):

    result = []

    rows = len(a)

    cols = len(a[0])

    for idr in range(rows):

        row\_tmp = []

        for idc in range(cols):

            t = a[idr][idc]-b[idr][idc]

            row\_tmp.append(t)

        result.append(row\_tmp)

    return result

# Tích 2 ma trận

def Tich2MaTran(a, b):

    rows\_a = len(a)

    cols\_a = len(a[0])

    cols\_b = len(b[0])

    result = [[0 for \_ in range(cols\_b)] for \_ in range(rows\_a)]

    for idra in range(rows\_a):

        for idrb in range(cols\_b):

            for idc in range(cols\_a):

                result[idra][idrb] += a[idra][idc]\*b[idc][idrb]

    return result

mat\_a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

mat\_b = [[2, 4, 6], [1, 3, 5], [1, 0, 1]]

# tong 2 ma tran

print(f'Tổng: {Tong2MaTran(mat\_a, mat\_b)}')

print(f'Hiệu: {Hieu2MaTran(mat\_a, mat\_b)}')

print(f'Dot product: {Tich2MaTran(mat\_a, mat\_b)}')

Section 12 - List Comprehension

stop\_words = ["I", "love", "and", "to"]

input = "I love AI and listen to music"

data = input.split(' ')

# cách 1

result = [item for item in data if stop\_words.count(item) == 0]

print(result)

# cách 2

result = []

for item in data:

    if stop\_words.count(item) == 0:

        result.append(item)

print(result)

Section 13 - List and Tuple

import math

my\_tuple1 = (2, 3)

my\_tuple2 = (3, 6)

tong\_vector = (my\_tuple1[0]+my\_tuple2[0], my\_tuple1[1]+my\_tuple2[1])

tich\_vector = my\_tuple1[0]\*my\_tuple2[0]+my\_tuple1[1]\*my\_tuple2[1]

print(tong\_vector)

print(tich\_vector)

# khoảng cách của 2 vector

distance = math.sqrt((my\_tuple1[0]-my\_tuple2[0])\*\* 2 + (my\_tuple1[1]-my\_tuple2[1])\*\*2)

print('Khoảng cách: ', distance)

my\_tuple=my\_tuple1+my\_tuple2

print(my\_tuple.index(3))