Section 1 - List Creation and Accessing

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - List là một kiểu dữ liệu cơ bản trong Python, được sử dụng để lưu trữ tập hợp các phần tử có thứ tự. List có thể chứa bất kỳ kiểu dữ liệu nào, bao gồm số nguyên, chuỗi, số thập phân, danh sách con, v.v.
   - **Đặc điểm của List:**
     - **Có thứ tự:** Các phần tử trong list được sắp xếp theo thứ tự nhất định.
     - **Có thể thay đổi:** List có thể được thêm, xóa, thay đổi phần tử sau khi được tạo.
     - **Có thể chứa nhiều kiểu dữ liệu:** List có thể chứa bất kỳ kiểu dữ liệu nào, không nhất thiết phải đồng nhất.
     - **Có thể truy cập bằng chỉ mục:** Mỗi phần tử trong list có một chỉ mục bắt đầu từ 0.

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

   - **Câu 1:** Tạo mới 1 List gồm các số từ 1 đến 10.
   - **Câu 2:** In ra kết quả 5 phần tử đầu tiên có trong List trên
   - **Câu 3:** In ra kết quả phần tử có **giá trị** không chia hết cho 2 (dùng kết hợp với vòng lặp for)
   - **Câu 4:** In ra kết quả tổng các giá trị trong list (dùng kết hợp với vòng lặp for)

```
lst_data = [''' Your code here ''']
# Your code here
```

Đáp án:
- **Câu 2:** 1, 2, 3, 4, 5
- **Câu 3:** 1, 3, 5, 7, 9
- **Câu 4:** 55

Section 2 - List CRUD

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - Ở bài tập trước, chúng ta đã làm quen với cách tạo và truy xuất các phần tử có trong List. Phần này, chúng ta sẽ làm quen với thêm, xóa, sửa các phần tử trong List.

Chức năng	Câu lệnh
Tạo mới danh sách	`list = [1, 'a', 3]` `or list(tuple)`
Thêm phần tử vào cuối	`list.append(value)`
Thêm phần tử vào vị trí index	`list.insert(index_position, value)`
Xóa phần tử đầu tiên của giá trị cần xóa	`list.remove(value)`
Xóa theo chỉ số index	`list.pop(index_position)`
Cập nhật phần tử	`list[index_position] = new_value`

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

```python
lst_data = []
# Your code here
```

Output:
- **Câu 1:** [2, 4, 6, 8, 10, 12]
- **Câu 2:** [2, 4, 8, 10]
- **Câu 3:** [2, 4, 8, 6, 7, 8, 10, 1, 2, 3]
- **Câu 4:** [0, 0, 0, 0, 7, 0, 0, 1, 0, 3]

Section 3 - List and Branching

Trung-Trực Trần

1. **Giới thiệu số Armstrong:**

   - Số Armstrong (còn được gọi là số Narcissistic) là một số nguyên dương có giá trị bằng tổng lập phương của các chữ số cấu thành nó.

$$X = \sum_{i=1}^{d} {X_i}^3 \tag{1}$$

     – X là số Armstrong.
     – X_i là các phần tử trong X.
     – d là số lượng chữ số của X.

     VD:

     – Số 370 là một số Armstrong vì: $3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370$
     – 153 là một số Armstrong vì: $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$ (Vì 153 là số có 3 chữ số nên n=3)

   - **Đặc điểm của số Armstrong:**
     – Số Armstrong luôn lớn hơn hoặc bằng 18.
     – Số lượng các số Armstrong là vô hạn.

   - **Ý nghĩa của số Armstrong:**
     – Số Armstrong được coi là những con số đặc biệt vì chúng có mối quan hệ độc đáo giữa các chữ số cấu thành nó.
     – Số Armstrong được sử dụng trong nhiều lĩnh vực khác nhau như toán học, khoa học máy tính và mật mã học.

2. **Yêu cầu và ý tưởng: Cho một list hãy trả về các số armstrong bên trong list đó.**

   - **Bước 1:** Viết hàm xét số Armstrong, hàm trả về 1 nếu phần tử đang xét là số Armstrong, ngược lại trả về 0.
   - **Bước 2:** Chạy vòng lặp qua tất cả các phần tử của list.
   - **Bước 3:** Xét điều kiện nếu phần từ là số Armtrong thì lưu vào 1 list khác.
   - **Bước 4:** In ra list kết quả chứa các số Armstrong đã xét.

```
test_case_1 = [130, 270, 153, 407, 177, 371, 1000, 1634, 370]
result = []
# Your code here
```

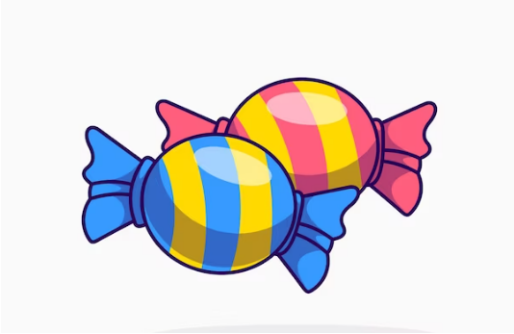
Output: Các số Armstrong có trong list là: [153, 407, 371, 1634, 370]

Section 4 - List and Branching

Trung-Trực Trần

1. **Bài toán tìm có nhiều kẹo nhất:**

   - Có n người, Bạn được cho một list với n phần tử (với n_i là số lượng kẹo của người đó) và một số extra_candies.



   Viết hàm trả về một boolean list với chiều dài n, với phần tử thứ n_i là true nếu số kẹo của người n_i cộng với extra_candies sẽ thành người có nhiều kẹo nhất, ngược lại trả về false.

2. **Ví dụ: Cho một list và số extra_candies trả về boolean list.**

   - **Input: candies = [2,3,5,1,3], extraCandies = 3:** Giả sử bạn cho tất cả số kẹo bổ sung (extraCandies) cho các người như sau:
     - người 1, họ sẽ có $2 + 3 = 5$ viên kẹo, đây là số kẹo lớn nhất trong 5 người.
     - người 2, họ sẽ có $3 + 3 = 6$ viên kẹo, đây là số kẹo lớn nhất trong 5 người.
     - người 3, họ sẽ có $5 + 3 = 8$ viên kẹo, đây là số kẹo lớn nhất trong 5 người.
     - người 4, họ sẽ có $1 + 3 = 4$ viên kẹo, đây không phải là số kẹo lớn nhất trong 5 người.
     - người 5, họ sẽ có $3 + 3 = 6$ viên kẹo, đây là số kẹo lớn nhất trong 5 người.
     - **Output: [true,true,true,false,true]**

```
test_case_1 = [2, 3, 5, 1, 3]   extraCandies1 = 3
test_case_2 = [4, 2, 1, 1, 2]   extraCandies2 = 1
test_case_3 = [12, 1, 12]       extraCandies3 = 10
result = []
# Your code here
```

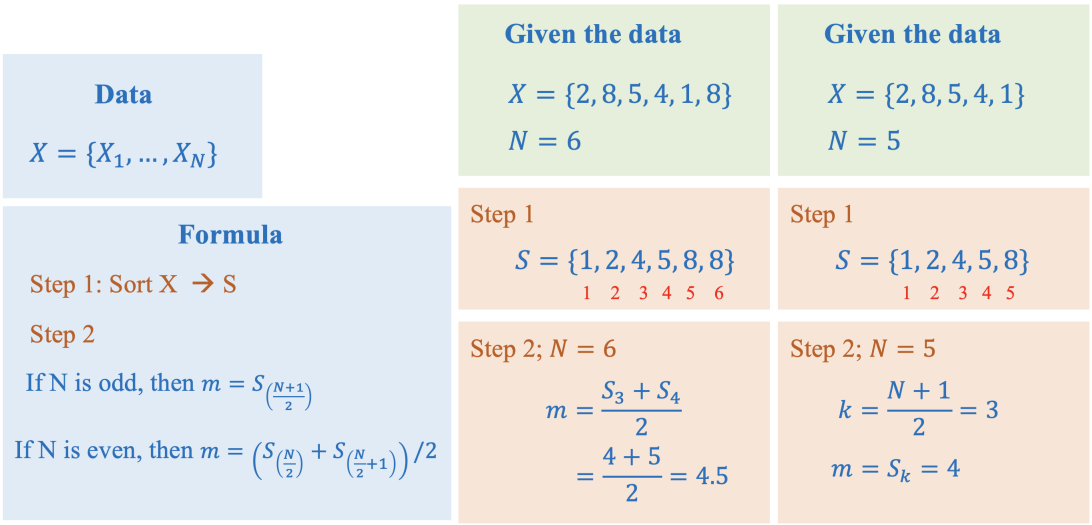
Output: [true,true,true,false,true]
Output: [true,false,false,false,false]
Output: [true,false,true]

Section 5 - Computing median for a list of numbers

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - **Median**, hay còn gọi là số trung vị, là một đại lượng thống kê quan trọng dùng để mô tả vị trí trung tâm của một tập dữ liệu. Nó là giá trị chia tập dữ liệu thành hai phần bằng nhau, với một nửa có giá trị lớn hơn hoặc bằng median và một nửa có giá trị nhỏ hơn hoặc bằng median. Cách tính giá trị Median như sau:
     - **Bước 1:** Sắp xếp dữ liệu theo thứ tự từ nhỏ đến lớn.
     - **Bước 2:** Xác định vị trí trung tâm của tập dữ liệu.
     - **Bước 3:** Giá trị tại vị trí trung tâm là median.



2. **Bài tập:**

   - **Câu 1:** Tạo mới một List có tên là *lst_data*, gồm các số từ 1 đến 10.
   - **Câu 2:** Tính giá trị **trung vị** từ *lst_data* vừa tạo. (Không sử dụng numpy)
   - **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_data = []
# Your code here
```

Output:
- **Câu 1:** [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- **Câu 2:** Median : 5.5
- **Câu 3:** [9, 7, 5, 3, 1]

Section 6 - Computing mean for a list of numbers

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - **Mean**, hay còn gọi là trung bình cộng, là một đại lượng thống kê quan trọng dùng để mô tả vị trí trung tâm của một tập dữ liệu. Nó được tính bằng cách cộng tất cả các giá trị trong tập dữ liệu và chia cho số lượng giá trị. Cách tính giá trị Mean như sau:
     - **Bước 1:** Cộng tất cả các giá trị trong tập dữ liệu.
     - **Bước 2:** Chia tổng đó cho số lượng giá trị.

Data

$X = \{X_1, \dots, X_N\}$

Given the data

$X = \{2, 8, 5, 4, 1, 4\}$

$N = 6$

Formula

$$m = \frac{1}{N}\sum_{i=1}^{N} X_i$$

$$m = \frac{1}{N}\sum_{i-1}^{N} X_i = \frac{1}{6}(2+8+5+4+1+4)$$
$$= \frac{24}{6} = 4$$

2. **Bài tập:**

   - **Câu 1:** Tạo mới một List có tên là *lst_data*, gồm các số từ 1 đến 10.
   - **Câu 2:** Tính giá trị **trung bình** cho các số lẻ và số chẵn từ *lst_data* vừa tạo. (Không sử dụng numpy)
   - **Câu 3:** Tính giá trị **trung bình** và **trung vị** cho tất cả dữ liệu trong *lst_data* và cho nhận xét.

```
lst_data = []
# Your code here
```

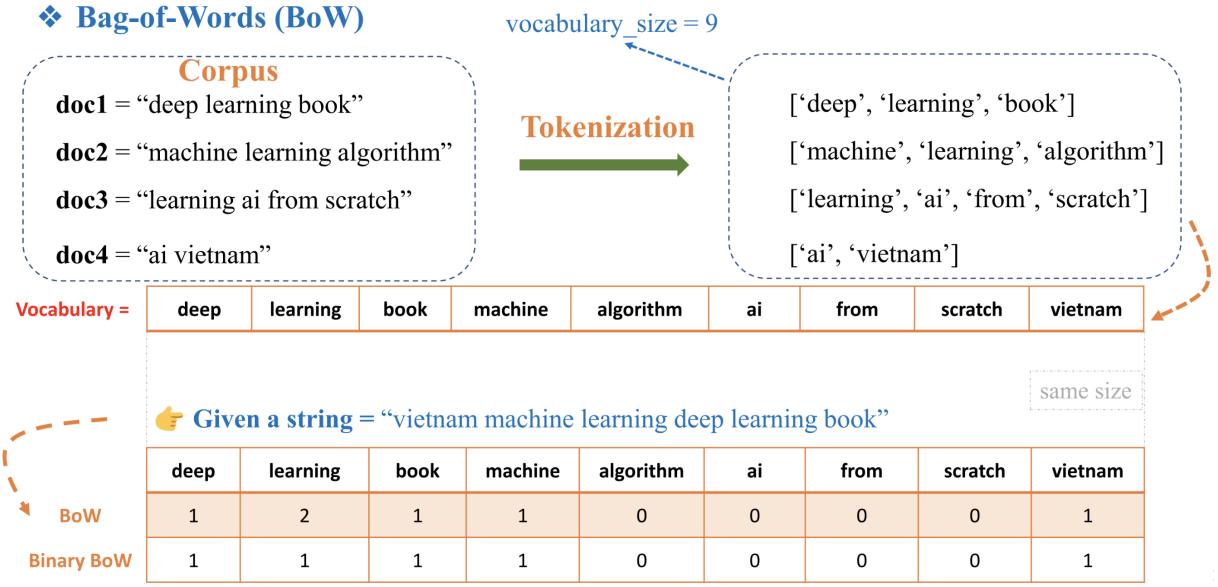
Output:
- **Câu 1:** [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- **Câu 2:** Mean lẻ: 5.00 - Mean chẵn: 6.00
- **Câu 3:** Mean = Median = 5.5

Section 7 - Bag of Words (NLP)

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - **Bag of Words** là một thuật toán hỗ trợ xử lý ngôn ngữ tự nhiên và mục đích của BoW là phân loại text hay văn bản. Ý tưởng của BoW là phân tích và phân nhóm dựa theo "Bag of Words" (corpus). Với test data mới, tiến hành tìm ra số lần từng từ của test data xuất hiện trong "Bag". Cách thức thực hiện như sau:
     - **Bước 1:** Chia nhỏ văn bản thành các từ riêng lẻ.
     - **Bước 2:** Tạo một tập hợp các từ xuất hiện trong văn bản. Tập hợp này không có phần tử trùng nhau.
     - **Bước 3:** Biểu diễn văn bản input ở dạng vector: Mỗi câu (mỗi input) được biểu diễn bằng một vector, với mỗi phần tử trong vector thể hiện số lần xuất hiện của từ đó trong input.



2. **Bài tập:** Tạo Bag-Of-Word cho tập dataset sau: *corpus* = ["Tôi thích môn Toán", "Tôi thích AI", "Tôi thích âm nhạc"]. Sau đó tạo list có tên *vector* để lưu vector sau khi thực hiện bước Tokenization đoạn văn bản sau: **Tôi thích AI thích Toán**, biết Bag-Of-Word được sắp theo thứ tự tăng dần

```
corpus = [''' Your Code Here ''']
# Your code here
```

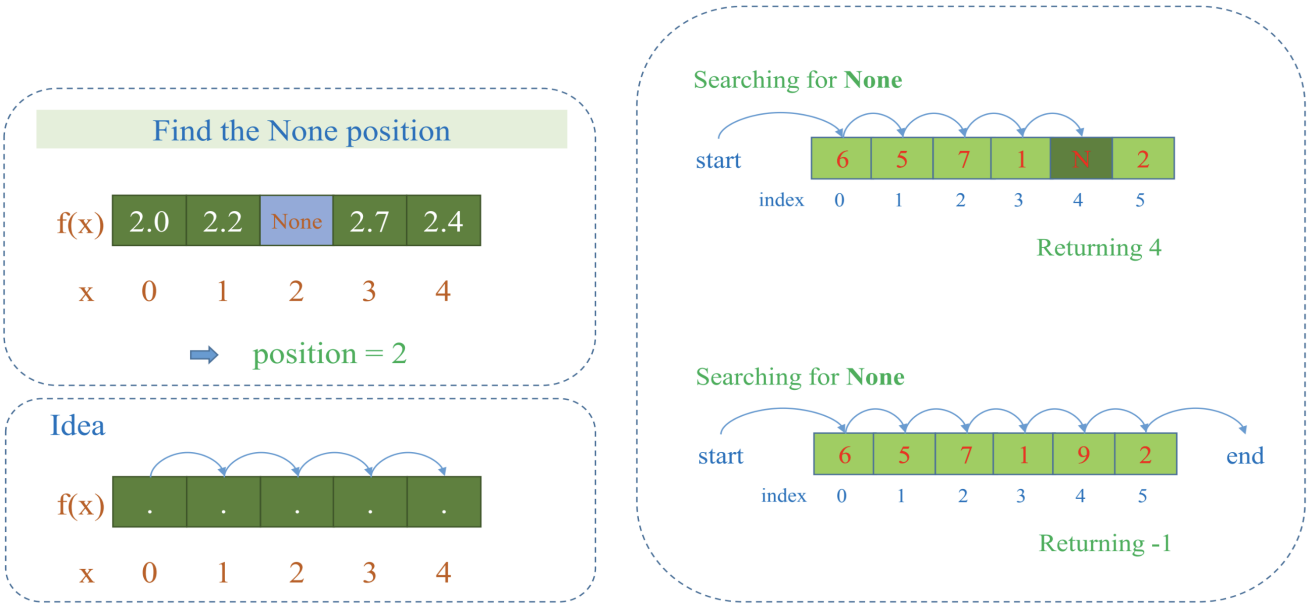
Output:
- **Tôi thích AI thích Toán:** [1, 1, 1, 0, 0, 2, 0]
- **Bag-of-Words:** [AI, Toán, Tôi, môn, nhạc, thích, âm]

Section 8 - Algorithms on List

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - **Tìm kiếm (Search)** là thao tác cơ bản thường được sử dụng trong lập trình Python để xác định vị trí hoặc sự tồn tại của một phần tử cần tìm trong list. Ở bài tập này, chúng ta sẽ làm quen với tìm kiếm các giá trị None (Trạng thái không có giá trị. Nó khác với các giá trị 0, False, hoặc "", vì những giá trị này thể hiện một ý nghĩa cụ thể) có trong List:
     - **Bước 1:** Duyệt qua từng phần tử trong list và so sánh với giá trị None.
     - **Bước 2:** Nếu tồn tại giá trị None, thì trả về vị trí hiện tại của giá trị None và kết thúc vòng lặp.



2. **Bài tập:**

   - Tạo List có tên là lst_data = [1, 1.1, None, 1.4, None, 1.5, None, 2.0]. Sau đó, áp dụng phương pháp tìm kiếm để tìm vị trí có giá trị None có trong lst_data theo 2 cách: tìm vị trí None đầu tiên, và tìm tất cả vị trí có giá trị None

```
lst_data = [''' Your Code Here ''']
# Your code here
```

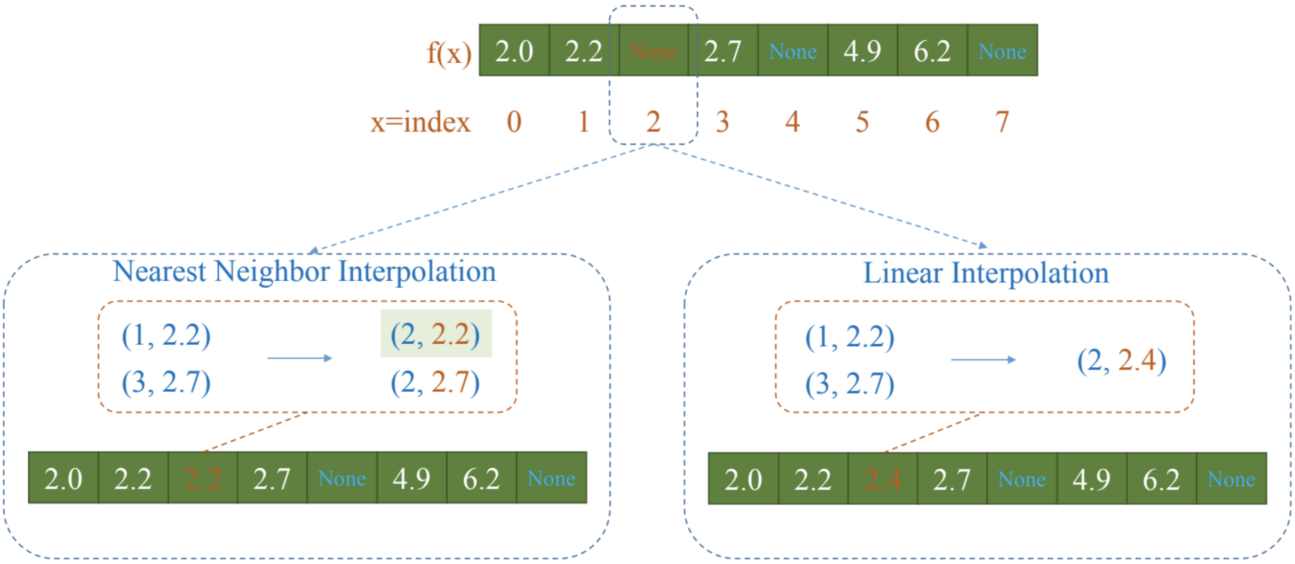
Output:
- Vị trí None đầu tiên: 2 - Danh sách vị trí có giá trị None: [2, 4, 6]

Section 9 - Interpolation for List (Time-series)

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - **Nội suy (Interpolate)** là kỹ thuật dự đoán giá trị tại một điểm chưa biết dựa trên các giá trị đã biết tại các điểm lân cận. Nó được sử dụng trong nhiều lĩnh vực như khoa học, kỹ thuật, kinh tế,... Có nhiều phương pháp nội suy khác nhau, trong đó 2 phương pháp đơn giản và dễ tiếp cận nhất là: **Láng giềng gần nhất (Nearest Neighbor)** và **Nội suy tuyến tính.** Ở bài tập này, chúng ta sẽ tiếp cận với phương pháp Nearest Neighbor (NN). Cách thức hoạt động của phương pháp NN như sau:
     - **Bước 1:** Xác định điểm dữ liệu lân cận nhất với điểm cần dự đoán.
     - **Bước 2:** Gán giá trị của điểm dữ liệu lân cận nhất cho điểm cần dự đoán.



2. **Bài tập:**

   - Tạo List có tên là lst_data = [1, 1.1, None, 1.4, None, 1.5, None, 2.0]. Sau đó, áp dụng phương pháp nội suy **Nearest Neighbor** để gắn giá trị None có trong lst_data

```
lst_data = [''' Your Code Here ''']
# Your code here
```

Output:
- [1, 1.1, 1.1, 1.4, 1.4, 1.5, 1.5, 2.0]

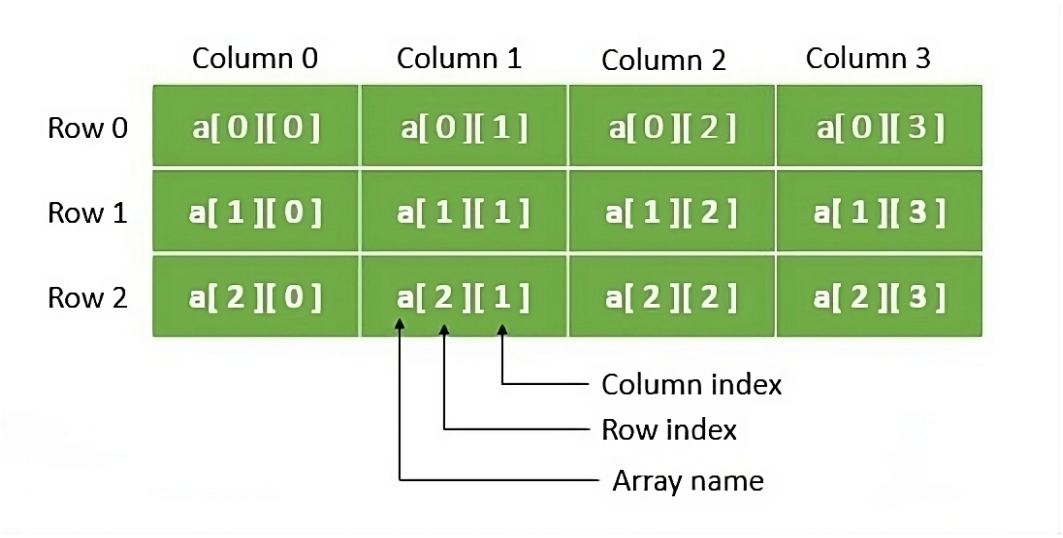
Section 10 - 2D List

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - **2D List** hay còn gọi là list hai chiều, là một cấu trúc dữ liệu trong Python cho phép lưu trữ dữ liệu dạng bảng. Nó bao gồm các list con, mỗi list con là một hàng trong bảng. Các ứng dụng của 2D List được sử dụng như: Lưu trữ dữ liệu ở dạng bảng, tạo ma trận, biểu diễn đồ thị, xử lý dữ liệu ảnh, ... Để khởi tạo 1 List hai chiều trong python, ta có thể sử dụng đoạn code sau đây:

```
list_2d = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
]
```



2. **Bài tập:**

   - Tạo List 2D có tên là *lst_data* có dạng 3 x 3, gồm các số từ 1 đến 9, ứng với các vị trí trong List. Sau đó tạo list khác có tên *lst_sub_data* là 2D List để lưu giá trị tại vị trí index thứ 0 và thứ 2 của *lst_data* (Chỉ sử dụng For). In ra màn hình kết quả của *lst_sub_data*

```
lst_data = [''' Your Code Here ''']
# Your code here
```

Output: [[1, 3], [4, 6], [7, 9]]

Section 11 - Matrix representation using List

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - **Ma trận** là một công cụ toán học hữu ích để biểu diễn và thao tác với dữ liệu. Nó được cấu tạo bởi các hàng và cột, chứa các giá trị số được sắp xếp theo thứ tự. Ma trận có thể được sử dụng để biểu diễn nhiều loại dữ liệu khác nhau, chẳng hạn như: dữ liệu hình ảnh, dữ liệu ngôn ngữ, v.v... Dưới đây là một số phép toán cơ bản trong ma trận:
     - **Cộng/Trừ ma trận:** Hai ma trận có cùng kích thước có thể được cộng/trừ với nhau để tạo ra một ma trận mới
     - **Tích vô hướng ma trận:** Dot product của hai ma trận A và B có kích thước tương thích (m x n và n x p) là ma trận C có kích thước m x p

$$\text{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ và B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$A \pm B = \begin{bmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \text{ x } \begin{bmatrix} w \\ z \end{bmatrix} = \begin{bmatrix} a.w + b.z \\ c.w + d.z \\ e.w + f.z \end{bmatrix}$$

2. **Bài tập:** Cho 2 ma trận sau: $\text{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ và $\text{B} = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \\ 1 & 0 & 1 \end{bmatrix}$. Sử dụng Python, không dùng thư viện numpy

Câu 1. Hãy tính tổng và hiệu 2 ma trận A + B và A - B

Câu 2. Hãy tính dot product 2 ma trận A và B

```
mat_a = [''' Your Code Here ''']
mat_b = [''' Your Code Here ''']
# Your code here
```

Output:
- Tổng: [[3, 6, 9], [5, 8, 11], [8, 8, 10]]
- Hiệu: [[-1, -2, -3], [3, 2, 1], [6, 8, 8]]
- Dot Product: [[7, 10, 19], [19, 31, 55], [31, 52, 91]]

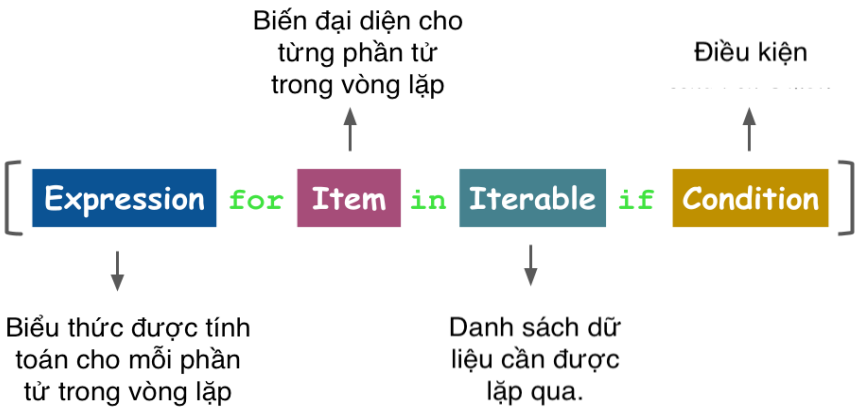
Section 12 - List Comprehension

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - **List comprehension** là một cú pháp ngắn gọn và mạnh mẽ trong Python để tạo ra một danh sách mới từ một danh sách hoặc tập hợp dữ liệu có sẵn. Cú pháp này giúp bạn tiết kiệm thời gian và viết code ngắn gọn hơn so với việc sử dụng vòng lặp for truyền thống.
     - **Ưu điểm:**
       + **Giảm thiểu số lượng code:** giúp code ngắn gọn và dễ đọc hơn so với sử dụng vòng lặp for truyền thống.
       + **Dễ sử dụng:** có cú pháp đơn giản và dễ học.
     - **Nhược điểm:**
       + **Khó đọc:** có thể khó đọc và khó hiểu hơn so với vòng lặp for truyền thống trong một số trường hợp.
       + **Hạn chế về chức năng:** không thể thực hiện một số thao tác mà vòng lặp for truyền thống có thể làm được

     Cấu trúc của List comprehension như sau:



2. **Bài tập:** Trong NLP, chúng ta cần loại bỏ 1 số từ không quan trọng (stopwords) ra khỏi câu để tránh gây nhiễu trong việc xử lý. Hãy loại bỏ các từ có trong *stop_words* = ["I", "love", "and", "to"] câu đầu vào **"I love AI and listen to music"**. Hãy áp dụng **List comprehension** và For truyền thống để thực hiện

```
stop_words = ["I", "love", "and", "to"]
input = "I love AI and listen to music"
# Your code here
```

Output: ['AI', 'listen', 'music']

Section 13 - List and Tuple

Trung-Trực Trần và Hoàng-Nguyên Vũ

1. **Giới thiệu:**

   - Tuple là một kiểu dữ liệu cơ bản trong Python, được sử dụng để lưu trữ tập hợp các phần tử có thứ tự. Tuple có thể chứa bất kỳ kiểu dữ liệu nào, bao gồm số nguyên, chuỗi, số thập phân, danh sách con, v.v.

2. **Mô tả:**

Bảng 1: Sự Giống Nhau và Khác Nhau giữa Tuple và List

Đặc Điểm	**Tuple**	**List**
Đặc Điểm Cơ Bản	Tập hợp các phần tử không thay đổi được, đặt trong cặp dấu ngoặc đơn.	Tập hợp các phần tử có thể thay đổi được, đặt trong cặp dấu ngoặc vuông.
Thay Đổi Dữ Liệu	Không thể thay đổi (immutable). Không thể thêm, xóa hoặc thay đổi các phần tử.	Có thể thay đổi (mutable). Có thể thêm, xóa và thay đổi các phần tử.
Sử Dụng	Thích hợp để bảo vệ dữ liệu.	Thích hợp cho các cấu trúc dữ liệu có thể thay đổi.
Hiệu Suất	Trong một số trường hợp, tuple có thể nhanh hơn	List có sự linh hoạt cao hơn.

3. **Bài tập: Khởi tạo Tuple và thao tác tìm kiếm, trích xuất thông tin trên Tuple đó.**

   - **Câu 1:** Tạo mới hai Tuple: $my\_tuple1 = (2,3)$, $my\_tuple2 = (3,6)$ mỗi Tuple có 2 phần tử đại diện cho một vector trong không gian 2D.
   - **Câu 2:** In ra kết quả của tổng và tích 2 vector trên.
   - **Câu 3:** In ra kết quả của khoảng cách của hai vector trên theo công thức.
     Biết $\text{distance}(P,Q) = \sqrt{(p_1-q_1)^2 + (p_2-q_2)^2 + \ldots + (p_n-q_n)^2}$
   - **Câu 4:** In ra vị trí của phần tử có giá trị là 3
     Sử dụng cú pháp: my_tuple.index(values) để trích xuất vị trí của giá trị cần tìm.

```
my_tuple1 = ()
my_tuple2 = ()
# Your code here
```

Output:
- **Câu 2:** Result_vector1=(5,6), Result_vector2=(9,18)
- **Câu 3:** $\sqrt{10}$=3.1622776601683795.
- **Câu 4:** index=(1, 0).