List and Tuple

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
```

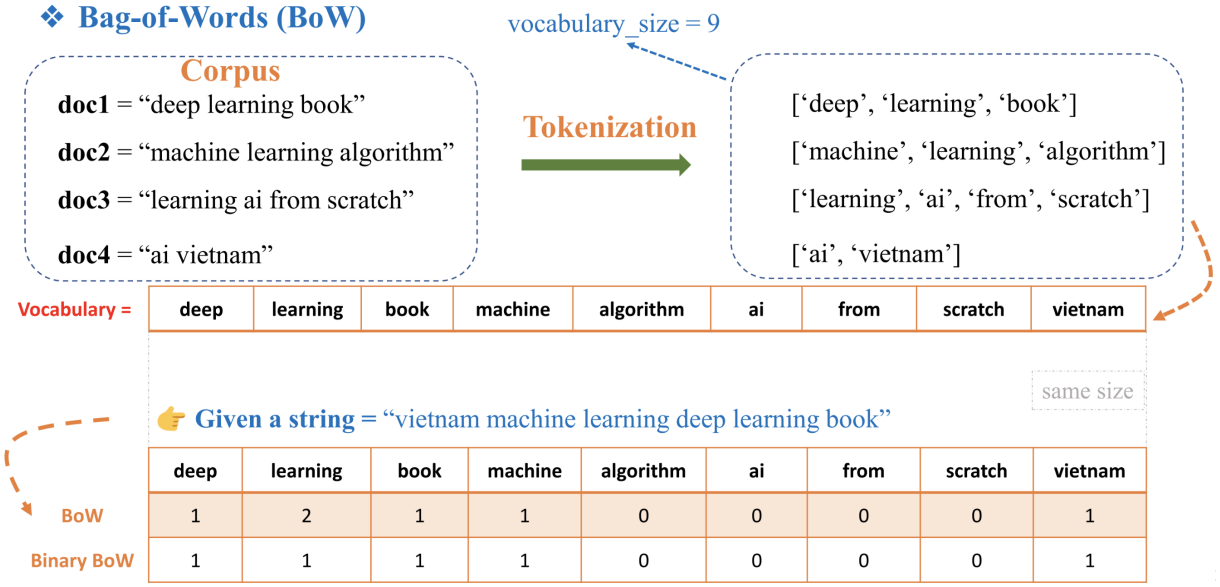
Output:
- **Câu 2:** Result_vector1=(5,6), Result_vector2=(9,18)
- **Câu 3:** $\sqrt{10}$=3.1622776601683795.
- **Câu 4:** index=(1, 0).

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
```

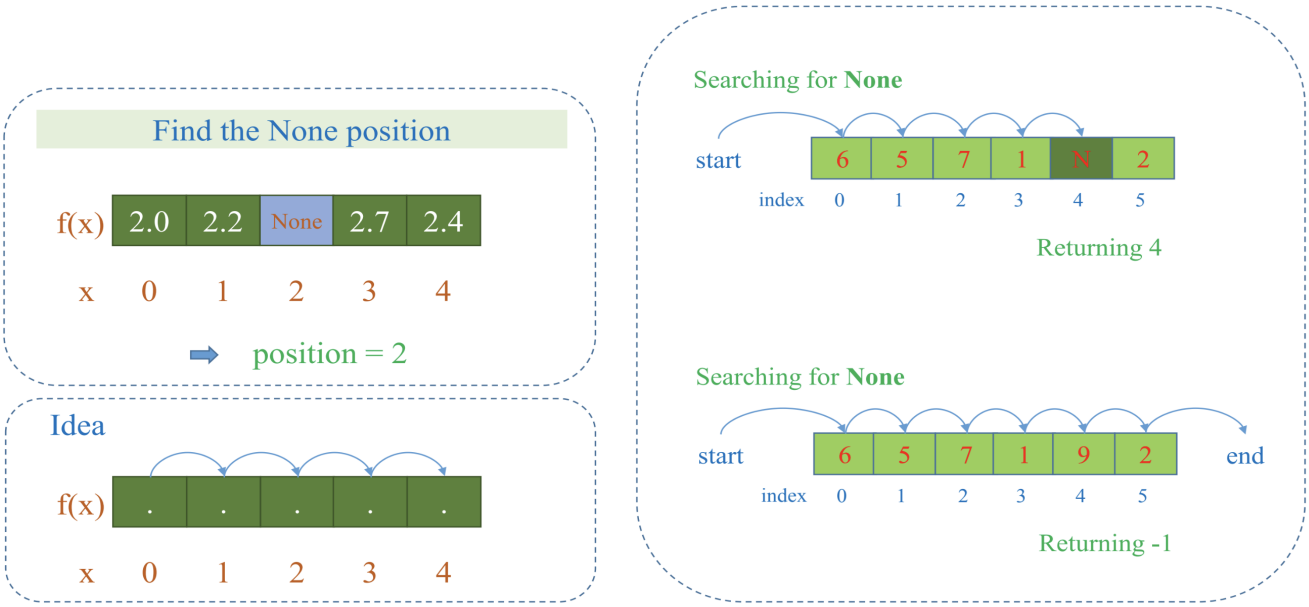
Output:
- **Tôi thích AI thích Toán:** [1, 1, 1, 0, 0, 2, 0]
- **Bag-of-Words:** [AI, Toán, Tôi, môn, nhạc, thích, âm]

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
```

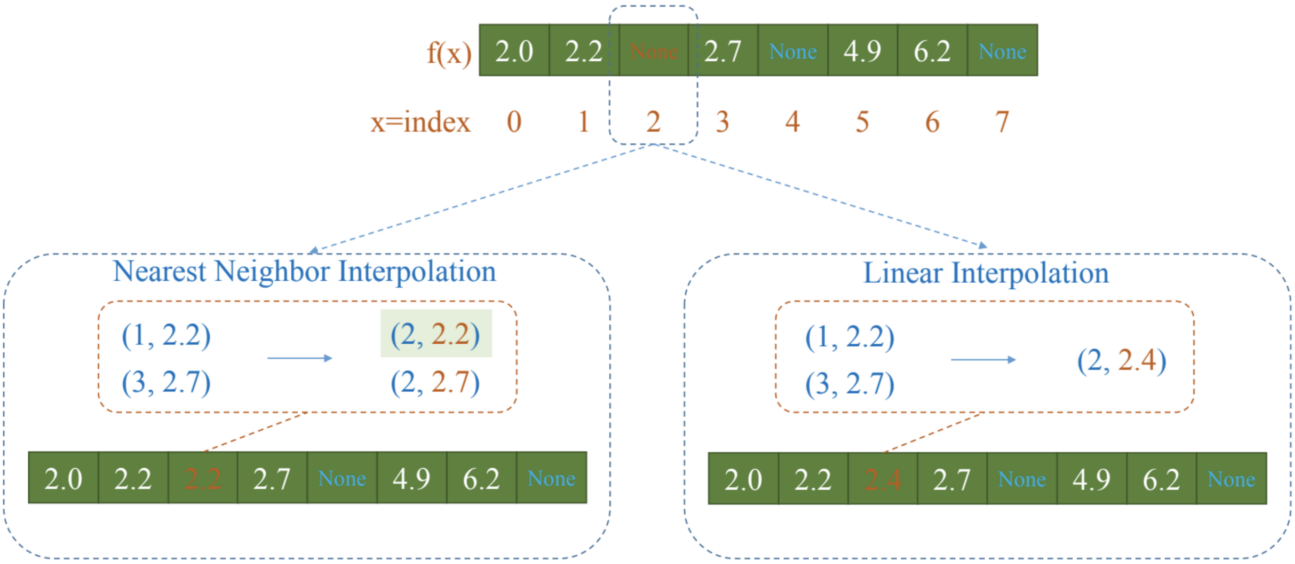
Output:
- Vị trí None đầu tiên: 2 - Danh sách vị trí có giá trị None: [2, 4, 6]

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]

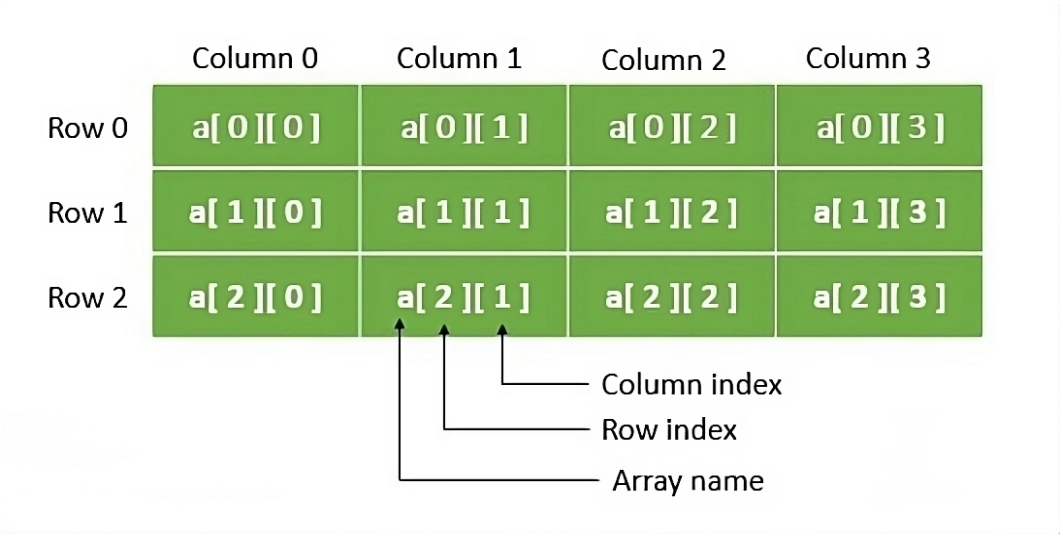
2D List

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]]

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]]

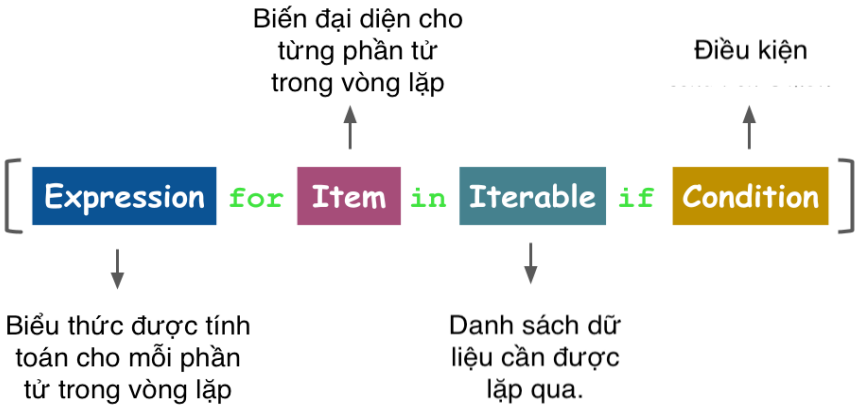
List Comprehension

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']

Mathemetic with List

Hoang-Nguyen Vu

1 Chuẩn hóa dữ liệu (Normalization)

Mô tả: Trong Machine Learning, dữ liệu có thể có phạm vi giá trị rất lớn, làm cho mô hình khó học được quy luật chung. Vì vậy, cần chuẩn hóa dữ liệu về khoảng $[0, 1]$ theo công thức:

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Điều này giúp dữ liệu đồng nhất, tránh hiện tượng một số giá trị quá lớn lấn át các giá trị nhỏ hơn trong mô hình học máy.

```python
def normalize(lst_data):
    ## Your code here

test_cases = [
    [11, 18, 24, 30, 36],
    [50, 100, 150, 200, 250],
    [3, 5, 7, 9, 11]
]

for i, test in enumerate(test_cases):
    result = normalize(test)
    print(f"Test {i+1}: {result}")
```

Output:
Test 1: [0.0, 0.28, 0.52, 0.76, 1.0]
Test 2: [0.0, 0.25, 0.5, 0.75, 1.0]
Test 3: [0.0, 0.25, 0.5, 0.75, 1.0]

2 Trung bình động (Moving Average)

Mô tả: Trung bình động làm mượt dữ liệu, giúp phát hiện xu hướng rõ ràng hơn. Công thức:

$$MA_t = \frac{1}{k} \sum_{i=t-k+1}^{t} X_i$$

```python
def moving_average(lst, k):
    # Your code here

test_cases = [
    ([1, 2, 3, 4, 5, 6, 7, 8, 9], 3),
    ([10, 20, 30, 40, 50, 60, 70], 4),
    ([5, 10, 15, 20, 25], 2)
```

```
]

for i, (test, k) in enumerate(test_cases):
    result = moving_average(test, k)
    print(f"Test {i+1}: {result}")
```

Output:
Test 1: [2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
Test 2: [25.0, 35.0, 45.0, 55.0]
Test 3: [7.5, 12.5, 17.5, 22.5]

3 Tích vô hướng của hai vector

Mô tả: Tích vô hướng dùng để đo độ tương đồng giữa hai vector:

$$v_1 \cdot v_2 = \sum_{i=1}^{n} v_1[i] \times v_2[i]$$

```
def dot_product(v1, v2):
    ## Your code here ##

test_cases = [
    ([1, 2, 3], [4, 5, 6]),
    ([2, 4, 6], [1, 3, 5]),
    ([0, 1, 2], [3, 4, 5])
]

for i, (v1, v2) in enumerate(test_cases):
    result = dot_product(v1, v2)
    print(f"Test {i+1}: {result}")
```

Output:
Test 1: 32
Test 2: 44
Test 3: 14

4 Mô phỏng Perceptron

Mô tả: Perceptron thực hiện tổng trọng số và áp dụng hàm kích hoạt ReLU:

$$y = \sum w_i \cdot x_i + bias$$

$$\text{Output} = ReLU(y)$$

```
def perceptron_relu(weights, inputs, bias):
    ## Your code here ##

test_cases = [
```

```
    ([0.5, -0.6, 0.8], [1.0, 0.0, 1.0], -0.3),
    ([0.2, 0.5, -0.4], [1.0, 2.0, -1.0], 0.1),
    ([1.0, -1.0, 0.5], [0.5, 1.0, -0.5], -0.2)
]

for i, (weights, inputs, bias) in enumerate(test_cases):
    result = perceptron_relu(weights, inputs, bias)
    print(f"Test {i+1}: {result}")
```

Output:
Test 1: 1.0
Test 2: 1.7000000000000002
Test 3: 0.0