



AI
VIETNAM

AIO2024
WARM UP

40 day coding Journey

Basic Python - If-Else

Hoàng-Nguyễn Vũ và Trung-Trực Trần

1. Mô tả:

- Can Chi là một hệ thống tính toán giờ, ngày, tháng, năm âm lịch của người trung quốc cổ đại. Can Chi có 10 thiên can và 12 địa chi. Tên gọi 10 thiên can Canh, Tân, Nhâm, Quý, Giáp, Ất, Bính, Dinh, Mậu, Kỷ. Tên gọi 12 địa chi Thân, Dậu, Tuất, Hợi, Tý, Sửu, Dần, Mão, Thìn, Tỵ, Ngọ, Mùi.



2. Bài tập: Tính lịch Can Chi - làm quen với câu lệnh If/Else

- Dể tính được can chi, chúng ta dựa vào quy tắc sau đây:
 - Can:** lấy năm sinh chia cho 10 và lấy phần dư. Nếu phần dư bằng 0 tương ứng với Canh, 1 tương ứng với Tân, tiếp tục cho tới 9 tương ứng với Kỷ

Phần dư	0	1	2	3	4	5	6	7	8	9
Can	Canh	Tân	Nhâm	Quý	Giáp	Ất	Bính	Dinh	Mậu	Kỷ

- Chi:** lấy năm sinh chia cho 12 và lấy phần dư. Nếu phần dư bằng 0 tương ứng với Thân, 1 tương ứng với Dậu, tiếp tục cho tới 11 tương ứng với Mùi

Phần dư	0	1	2	3	4	5	6	7	8	9	10	11
Chi	Thân	Dậu	Tuất	Hợi	Tý	Sửu	Dần	Mẹo	Thìn	Tỵ	Ngọ	Mùi

```

1 def calculate_can_chi_calendar(year):
2     result = ''
3     ##### Your code here #####
4     return result

```

Ví dụ:

- Test case 1: calculate_can_chi_calendar(2024) → Giáp Thìn
- Test case 2: calculate_can_chi_calendar(2023) → Quý Mão
- Test case 3: calculate_can_chi_calendar(1997) → Dinh Sửu

ANACONDA VS CODE INSTALLATION AND USAGE GUIDE



Download



Note: select box add
Anaconda to your PATH



Run file installer

Next



install



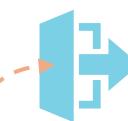
Agree

Next



Next

Just Me



Finish

add Anaconda to PATH...



Install

Working with environments

Display Conda version

conda --version

List environments with conda env list

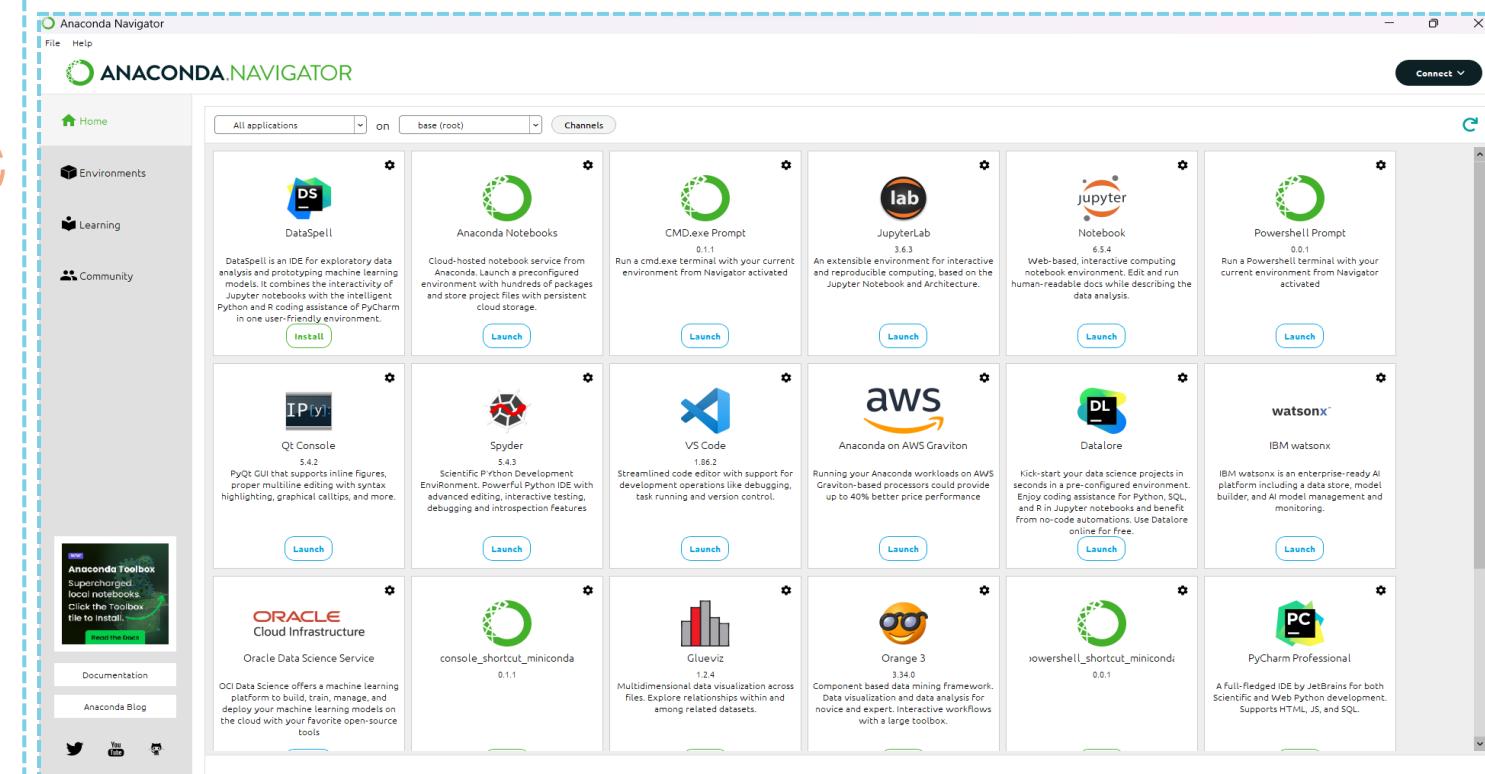
conda env list

Create an environment

conda create --name my_python_env python=3.8

Activate the environment

conda activate my_python_env





Method 1



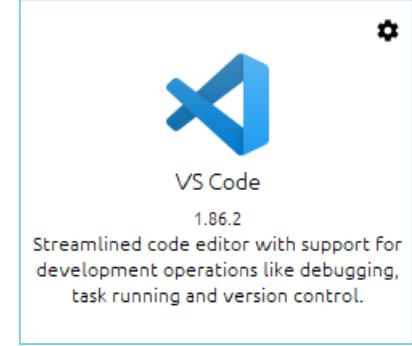
Install by default



Run the
installer

Next by
default

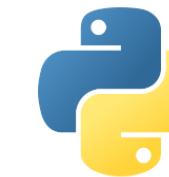
Finish



from ANACONDA
NAVIGATOR

Install

Important VS Code extensions



Python v2024.0.1

Microsoft microsoft.com | ⚡ 112,909,943 | ★★★★☆ (579)
IntelliSense (Pylance), Linting, Debugging (Python Debugger), code formatting, refactoring, unit tes...

[Disable](#) [Uninstall](#) [Switch to Pre-Release Version](#)

This extension is enabled globally.



Jupyter v2024.1.1

Microsoft microsoft.com | ⚡ 75,081,695 | ★★★★☆ (302)
Jupyter notebook support, interactive programming and computing that supports Intellisense, deb...

[Disable](#) [Uninstall](#) [Switch to Pre-Release Version](#)

This extension is enabled globally.

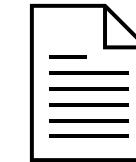
Select environments in VS code

Ctrl + Shift + p

Python: Select
Interpreter

Select
environment

Case for
jupyter file

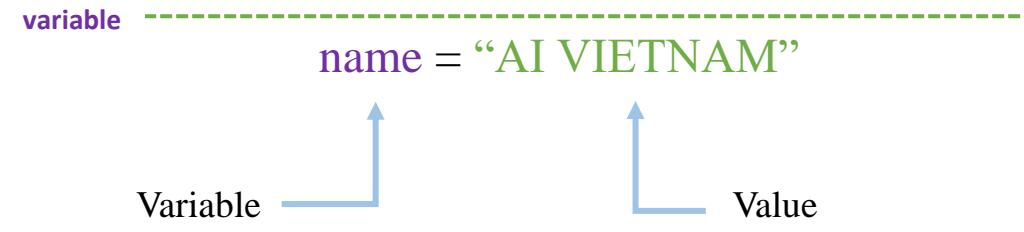
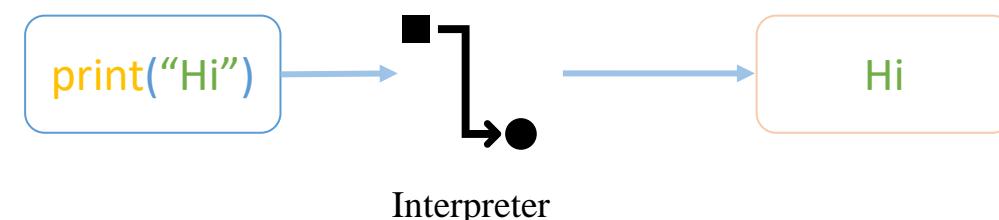


.ipynb

Click Select Kernel

Bài tập: Viết chương trình với yêu cầu dưới đây, mỗi yêu cầu bạn sẽ viết nó tại một cell trong file jupyter notebook.

- 1) Bạn hãy tạo file và đặt tên file bất kỳ nhưng tên phải được viết thường và mỗi từ cách nhau bởi dấu gạch dưới "_", ví dụ như simple_message.ipynb. Sau đó hãy lập trình in chuỗi "Hello world!" ra màn hình.
- 2) Gán một chuỗi "Have a nice day!" vào biến message. Sau đó sử dụng câu lệnh print để hiển thị biến này ra màn hình.
- 3) Bạn hãy tạo một chuỗi "Let's have Tet holidays!" và gán vào biến với tên 1_message. Sau đó sử dụng câu lệnh print để hiển thị biến này ra màn hình. Nếu kết quả hiển thị thông báo lỗi, hãy xác định lỗi này là gì và sửa như thế nào? Sau đó chạy lại chương trình.



5 Qui tắc đặt tên biến

Gồm chữ cái, số, dấu _
nhưng **không** được bắt
đầu bằng số.

Nên: message_1,
_message
Không: 1_message

Không khoảng trắng

Nên: fresh_apple
Không: fresh apple

Không trùng từ khóa
Python

Không: list, for, from,
if, is, False....

Phải rõ ràng

Nên: student_name
Không: s_n

Cẩn thận với ký tự l, o
với số 1, 0

10ve với love

simple print

```
1 # Câu 1
2 print("Hello World!")
```

Output: Hello World!

print variable

```
1 # Câu 2
2 message = "Have a nice day!"
3 print(message)
```

Output: Have a nice day!

Fix variable name

```
1 # Câu 3
2 1_message = "Let's have Tet holidays!"
3 print(1_message)
```

Output: error

```
1 # Câu 3
2 message_1 = "Let's have Tet holidays!"
3 print(message_1)
```

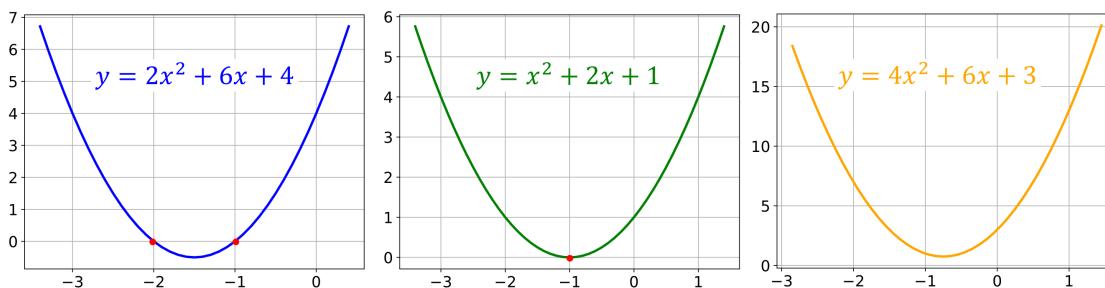
Output: Let's have Tet holidays!

Basic Python - If-Else

Trung-Trực Trần và Hoàng-Nguyễn Vũ

1. Bài tập:

- Cho một phương trình bậc 2 có dạng $y = ax^2 + bx + c = 0$ (a khác 0). Hãy hoàn thành function giải phương trình bậc 2 trên, biết $\Delta = b^2 - 4ac$.



2. Ý tưởng: Tính nghiệm phương trình bậc 2 - làm quen với câu lệnh If/Else

- Để tính được nghiệm, chúng ta dựa vào công thức sau đây:
 - Nếu Δ lớn hơn 0, phương trình có 2 nghiệm phân biệt là $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ và $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$
 - Nếu Δ bằng 0 thì phương trình có nghiệm kép $x_1 = x_2 = \frac{-b}{2a}$
 - Nếu Δ bé hơn 0 thì phương trình vô nghiệm

```

1 def quadratic_equation(a, b, c):
2     x1=0
3     x2=0
4
5     # Your code here #####
```

Ví dụ:

- test case 1:** với `quadratic_equation(a=2, b=6, c=4)`, kết quả gồm $x_1 = -1$ và $x_2 = -2$.
- test case 2:** với `quadratic_equation(a=1, b=2, c=1)`, kết quả gồm $x_1 = x_2 = -1$
- test case 3:** với `quadratic_equation(a=4, b=6, c=3)`, kết quả là phương trình vô nghiệm.

(*) **Mở rộng:** Trường hợp $a = 0$, hàm số sẽ thành phương trình bậc 1 có dạng: $y = bx + c = 0$, trường hợp này các bạn có thể mở rộng để bài toán tổng quát hơn.

Bài tập:

1. Tạo một biến name gán giá trị là tên một người, sau đó hiển thị ra màn hình một thông báo chứa tên đó sử dụng f-string. Ví dụ name = “Alice”, mình sẽ in ra màn hình nội dung là “Alice is a great teacher!”
2. Tạo một biến và gán giá trị “ms Taylor” cho nó, sau đó thực hiện in ra màn hình giá trị biến đó được viết hoa ký tự đầu tiên mỗi từ hoặc viết toàn bộ bằng chữ hoa, hoặc chữ thường.

Input: name = “ms Taylor”

Output:

Title case: Ms Taylor

Upper case: MS TAYLOR

Lower case: ms taylor

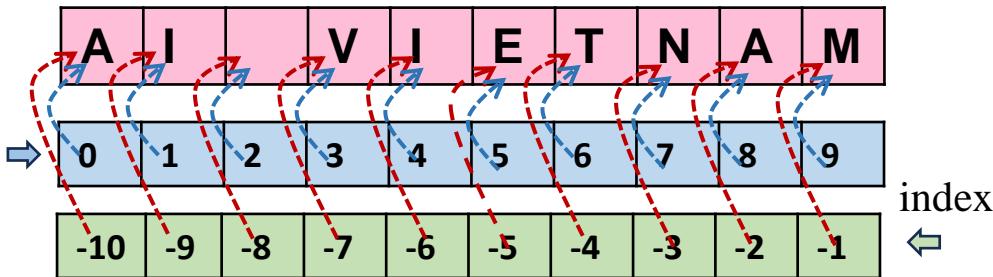
string

"AI VIETNAM"

'AI VIETNAM'

“ Multi
lines””

Length = 10



💡 A Python string is a sequence enclosed in quotes.

f-string

syntax: `f“{variable}”`

Add `f` before
a string

Embed variables
by their name

```
student_name = "Tom"
class_name = "AI VIETNAM"
message = f"{student_name} học lập trình
tại{class_name}"
print(message)
```

```
print(f"{student_name} học lập trình tại
{class_name}")
```

Changing case

`lower()`

tom holland

“tom Holland”

`upper()`

TOM HOLLAND

`title()`

Tom Holland

Common error

⭐ A syntax error often happens when you mix single and double quotes incorrectly.

`print('I'm a student')`

error

`print('I\'m a student')`

solution

Solution

```
1 # Câu 1
2 name = "Alice"
3 print(f"{name} is a great teacher!")
```

Output: Alice is a great teacher!

```
1 # Câu 2
2 # Define variables
3 name = "ms Taylor"
4
5 # Title case the name
6 title_name = name.title()
7
8 # Upper case the name
9 upper_name = name.upper()
10
```

```
11 # Lower case the name
12 lower_name = name.lower()
13
14 # Print the results
15 print(f"Title case: {title_name}")
16 print(f"Upper case: {upper_name}")
17 print(f"Lower case: {lower_name}")
```

Output: Title case: Ms Taylor
Upper case: MS TAYLOR
Lower case: ms taylor

AI VIETNAM
All-in-One Course

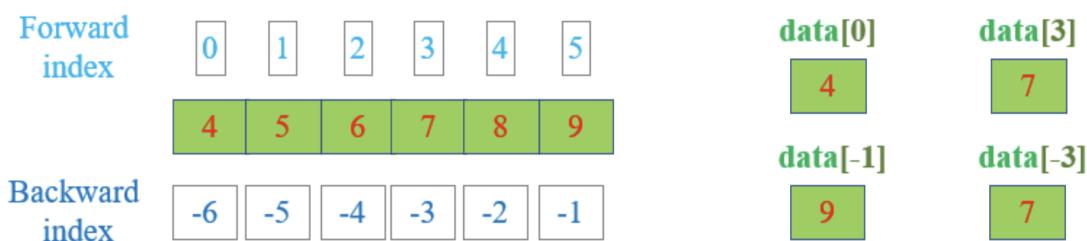
Basic Python - List

Hoàng-Nguyễn 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.

data = [4, 5, 6, 7, 8, 9]



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)

```

1     lst_data = [ ''' Your code here ''']
2     # Your code here
3

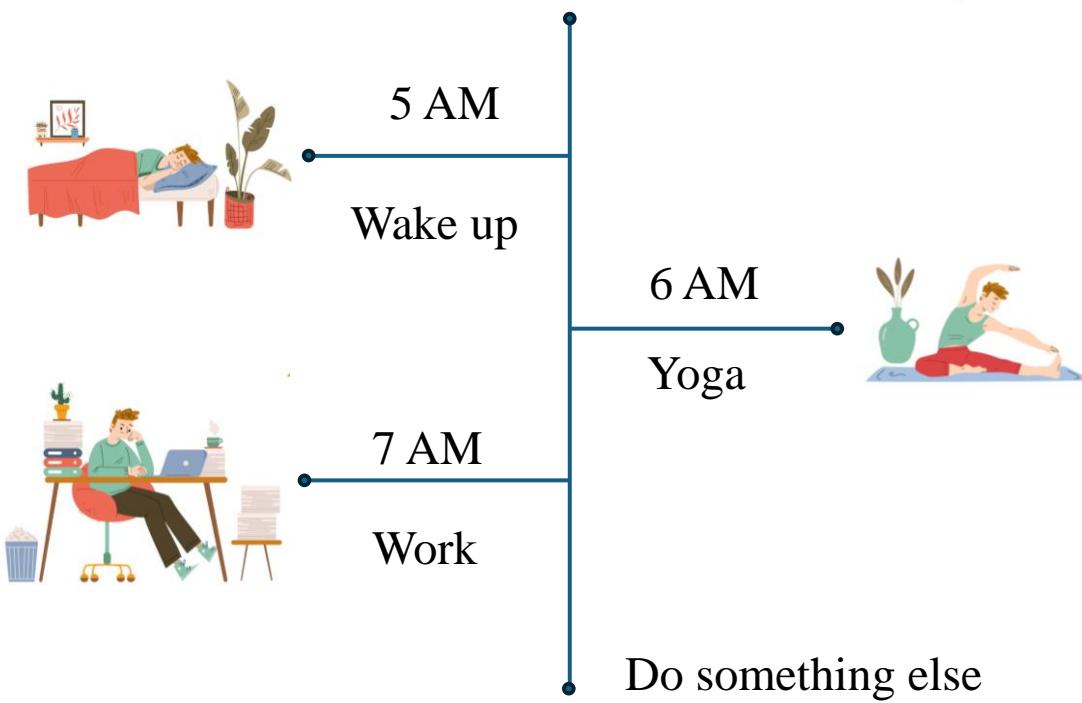
```

Dáp án:

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

SWITCH CASE

TOM MORNING ROUTINE



Viết chương trình nhập đầu vào là thời gian để xem hoạt động của TOM.

Input:

5 AM

6 AM

7 AM

Giá trị khác

Output:

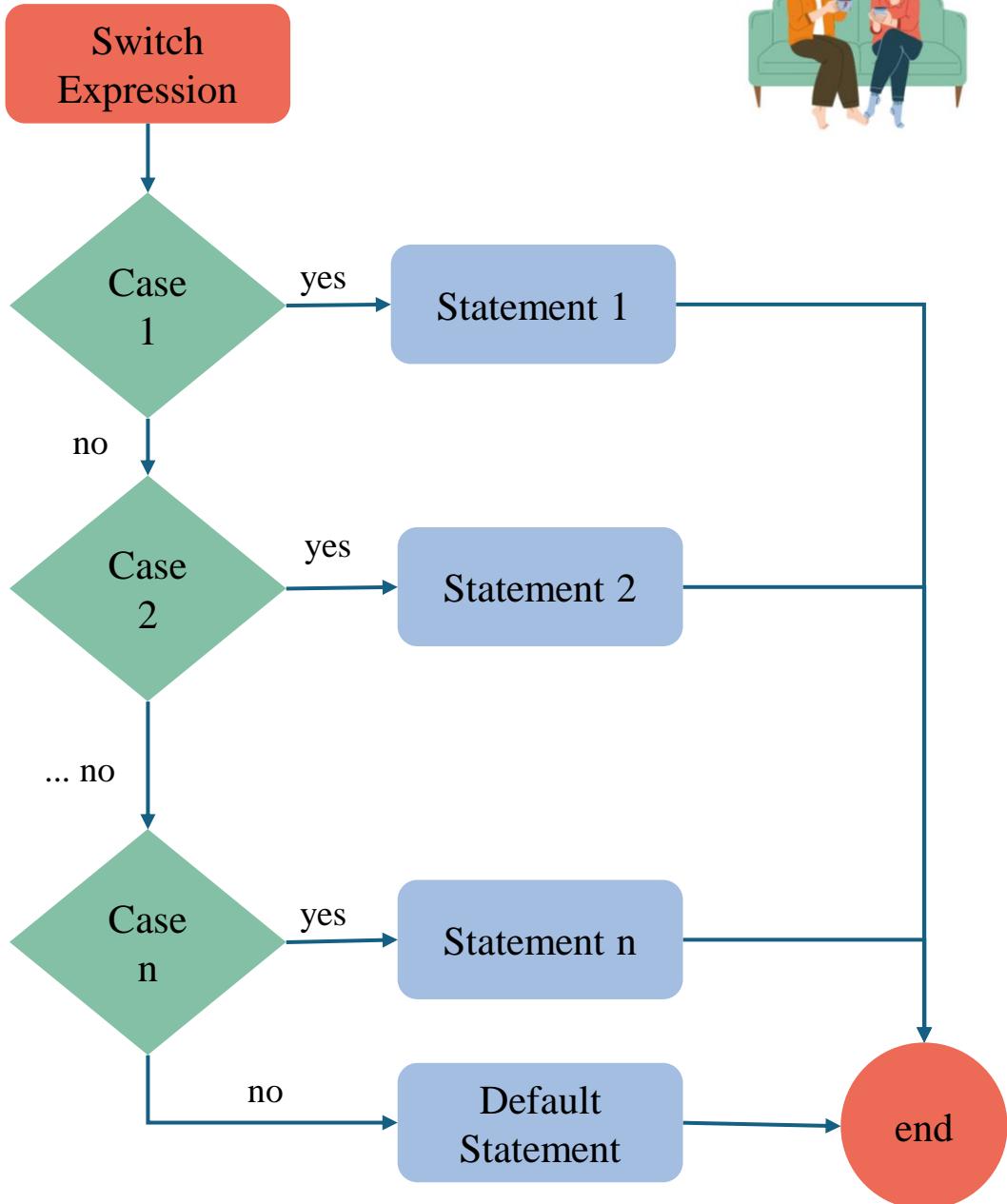
Wake up

Yoga

Work

Do something else

SWITCH CASE



Method 1: If elif else

```
if (condition):  
    statement  
elif (condition):  
    statement  
.  
.  
else:  
    statement
```

```
1 time = input("Input time: ")  
2 if time == "5 AM":  
3     print("Wake up")  
4 elif time == "6 AM":  
5     print("Yoga")  
6 elif time == "7 AM":  
7     print("Work")  
8 else:  
9     print("Do something else")
```

Method 2: Dictionary

```
dict.get(key, default=None)
```

```
1 todo = {"5 AM": "Wake up",  
2         "6 AM": "Yoga",  
3         "7 AM": "Work"}  
4 time = input("Input time: ")  
5 todo.get(time, "Do something else")
```

Method 3: Match case

```
match expression:  
case value1:  
    #do something  
case value2:  
    #do something  
...  
case valueN:  
    #do something  
case _:  
    #do something
```

⚠ Python>= 3.10

```
1 time = input("Input time: ")  
2 match time:  
3     case "5 AM":  
4         print("Wake up")  
5     case "6 AM":  
6         print("Yoga")  
7     case "7 AM":  
8         print("Work")  
9     case _:  
10        print("Do something else")
```

Basic Python - List

Hoàng-Nguyễn 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	<code>list = [1, 'a', 3] or list(tuple)</code>
Thêm phần tử vào cuối	<code>list.append(value)</code>
Thêm phần tử vào vị trí index	<code>list.insert(index_position, value)</code>
Xóa phần tử đầu tiên của giá trị cần xóa	<code>list.remove(value)</code>
Xóa theo chỉ số index	<code>list.pop(index_position)</code>
Cập nhật phần tử	<code>list[index_position] = new_value</code>

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

```

1
2     lst_data = []
3     # Your code here
4

```

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]

ENUMERATE

TOM' SHOPPING LIST



Cà rốt

Táo

Sữa

Giúp Tom đánh số thứ tự danh sách sản phẩm cần mua

**Enumerate****enumerate(iterable, start=0)**

Cà rốt

1 | Cà rốt

Táo

Enumerate
start=1

2 | Táo

Sữa

3 | Sữa

Method 1

```
1 shopping_list = ["Cà rốt", "Táo", "Sữa"]
2
3 print("Danh sách mua sắm:")
4 for index in range(len(shopping_list)):
5     print(f"{index+1}. {shopping_list[index]}")
```

Method 2

```
1 shopping_list = ["Cà rốt", "Táo", "Sữa"]
2
3 print("Danh sách mua sắm:")
4 for index, item in enumerate(shopping_list, start=1):
5     print(f"{index}. {item}")
```

Ưu Điểm

Ngắn gọn, dễ đọc

Nhược Điểm

Chỉ hỗ trợ vòng lặp từ trái sang phải

Tiết kiệm thời gian

Sử dụng nhiều bộ nhớ

Chỉ số linh hoạt

Dễ kết hợp với vòng lặp

			
row 1	Bơ	Pizza	Sữa
row 2			
row 3			
	column 1	column 2	column 3

? Hãy giúp Tom tìm vị trí của Cà rốt, Táo, Sữa?

```

1 food_list = [
2     ["Bơ", "Pizza", "Sữa"],
3     ["Xúc xích", "Táo", "Kem"],
4     ["Cà rốt", "Bánh dâu", "Cupcake"]
5 ]
6
7 search_items = ["Cà rốt", "Táo", "Sữa"]

```

Method 1

```

9 # Lặp qua từng hàng của danh sách thực phẩm
10 for i in range(len(food_list)):
11     # Lặp qua từng phần tử trong hàng đó
12     for j in range(len(food_list[i])):
13         # Kiểm tra xem phần tử có trong
14         # danh sách tìm kiếm không
15         if food_list[i][j] in search_items:
16             # In ra vị trí nếu phần tử được tìm thấy
17             print(f"{food_list[i][j]} được tìm "
18                  f"thấy ở hàng {i + 1} và cột {j + 1}.")

```

Method 2

```

9 # Lặp qua từng hàng của danh sách thực phẩm
10 # và lấy ra chỉ số của hàng đó
11 for i, row in enumerate(food_list, start=1):
12     # Lặp qua từng phần tử trong hàng
13     # và lấy ra chỉ số của phần tử đó
14     for j, item in enumerate(row, start=1):
15         # Kiểm tra xem phần tử có trong
16         # danh sách tìm kiếm không
17         if item in search_items:
18             # In ra vị trí nếu phần tử được tìm thấy
19             print(f"{item} được tìm thấy "
20                  f"ở hàng {i} và cột {j}.")

```

Basic Python - List

Hoàng-Nguyễn 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.

Data	Given the data	Given the data
$X = \{X_1, \dots, X_N\}$	$X = \{2, 8, 5, 4, 1, 8\}$ $N = 6$	$X = \{2, 8, 5, 4, 1\}$ $N = 5$
Formula Step 1: Sort $X \rightarrow S$ Step 2 If N is odd, then $m = S_{(\frac{N+1}{2})}$ If N is even, then $m = (S_{(\frac{N}{2})} + S_{(\frac{N+1}{2})})/2$	Step 1 $S = \{1, 2, 4, 5, 8, 8\}$ 1 2 3 4 5 6	Step 1 $S = \{1, 2, 4, 5, 8\}$ 1 2 3 4 5
	Step 2; $N = 6$ $m = \frac{S_3 + S_4}{2}$ $= \frac{4 + 5}{2} = 4.5$	Step 2; $N = 5$ $k = \frac{N + 1}{2} = 3$ $m = S_k = 4$

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 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).

```

1   lst_data = []
2   # Your code here
3

```

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]

Numpy Array và Pytorch/Tensorflow Tensor

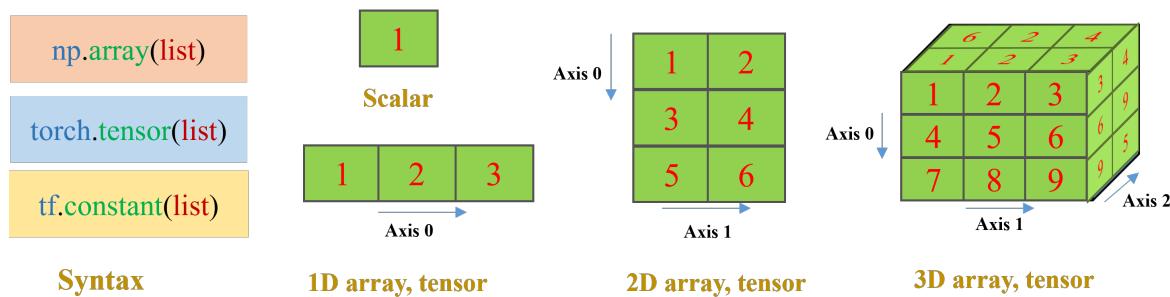
Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Array trong Numpy và Tensor trong các thư viện Pytorch, Tensorflow là những thành phần nền tảng cốt lõi cho việc tính toán của các thư viện này. Trong đó:

- **Array** là một cấu trúc dữ liệu đa chiều mạnh mẽ, giúp lưu trữ và thực hiện các phép toán toán học trên dữ liệu số. Array có thể là mảng 0 chiều (scalar), mảng một chiều (vector), mảng hai chiều (ma trận), hoặc nhiều chiều hơn.
- **Tensor trong Pytorch và Tensorflow** là một cấu trúc dữ liệu nhiều chiều, tương tự như array trong Numpy, nhưng được thiết kế để tương thích với học sâu và tính toán song song, đặc biệt là trên các thiết bị như GPU và TPU. Trong Pytorch và Tensorflow, tensors là đơn vị cơ bản để lưu trữ và xử lý dữ liệu.

Có nhiều cách để tạo tensor (hay array trong Numpy), ta có thể sử dụng cú pháp sau đây để khởi tạo chúng từ list Python.



Hình 1: Minh họa và cú pháp tạo Array và Tensor.

Khi làm việc với cấu trúc dữ liệu tensor, ta có thể kiểm tra thuộc tính của chúng thông qua một số phương thức sau:

- **shape** cho biết kích thước của mảng hoặc tensor, tức là số phần tử trong mỗi chiều của chúng.
- **dtype** cho biết kiểu dữ liệu của các phần tử trong mảng hoặc tensor.
- **type** cho biết kiểu của đối tượng mảng hoặc tensor.
- **device** chỉ có ở trong Pytorch và Tensorflow, và nó cho biết nơi lưu trữ tensor, có thể là CPU hoặc GPU.

2. Bài tập

Câu 1: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor từ danh sách 1 chiều?

Câu 2: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor từ danh sách 2 chiều. Sau đó thực hiện kiểm tra thuộc tính shape, dtype, type, device từ các array, tensor vừa tạo ?

3. Đáp án

Có nhiều cách khác nhau để ta tạo array hay tensor. Cách đầu tiên, ta tạo chúng từ list-kiểu dữ liệu quen thuộc trong Python.

```

1 import numpy as np
2 import torch
3 import tensorflow as tf
4
5 # Tạo một danh sách 1 chiều
6 list_1D = [1, 2, 3, 4, 5, 6, 7, 8, 9]
7
8 # Tạo một mảng 1 chiều từ danh sách
9 arr_1D = np.array(list_1D)
10 print("Mảng 1 chiều NumPy:\n",
11      arr_1D, type(arr_1D))
12
13 # Tạo một tensor PyTorch từ danh sách
14 tensor_1D_pt = torch.tensor(list_1D)
15 print("Tensor PyTorch 1 chiều:\n",
16      tensor_1D_pt)
17
18 # Tạo một tensor TensorFlow từ danh sách
19 tensor_1D_tf = tf.convert_to_tensor(
20             list_1D)
21 print("Tensor TensorFlow 1 chiều:\n",
22      tensor_1D_tf)

```

```

=====
Output =====
Mảng 1 chiều NumPy:
[1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>

Tensor PyTorch 1 chiều:
tensor([1, 2, 3, 4, 5, 6, 7, 8, 9])

Tensor TensorFlow 1 chiều:
tf.Tensor([1 2 3 4 5 6 7 8 9],
shape=(9,), dtype=int32)

=====
```

Ví dụ trên thực hiện tạo array, tensor từ một list Python. Để tạo array chúng ta dùng cú pháp `np.array(list_1D)`, với pytorch thì ta dùng `torch.tensor(list_1D)` và với Tensorflow ta dùng `tf.convert_to_tensor(list_1D)` hoặc `tf.constant(list_1D)`.

Đối với việc tạo array, tensor từ list 2D, chúng ta cũng thực hiện tương tự cách trên, Ví dụ Numpy:

```

1 # NumPy code
2 import numpy as np
3 # Tao một danh sách 2 chiều
4 list_2D = [[1, 2, 3],
5             [4, 5, 6],
6             [7, 8, 9]]
7 # Tạo một mảng 2 chiều
8 arr_2D = np.array(list_2D)
9 # Kiểm tra hình dạng, kiểu dữ liệu và loại
10 print("Hình dạng của mảng: ",
11       arr_2D.shape)
12 print("Kiểu dữ liệu của mảng: ",
13       arr_2D.dtype)
14 print("Loại của mảng: ",
15       type(arr_2D))
16 print("Mảng:\n", arr_2D)

```

```

=====
Output =====
Hình dạng của mảng: (3, 3)
Kiểu dữ liệu của mảng: int32
Loại của mảng: <class 'numpy.ndarray'>
Mảng:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

=====
```

Trong đoạn mã Numpy trên, chúng ta bắt đầu bằng việc tạo ra một danh sách 2D (`list_2D`) đại diện cho một ma trận có 3 hàng và 3 cột. Sau đó, chúng ta sử dụng Numpy để chuyển danh sách này thành một mảng 2 chiều (`arr_2D`). Dòng mã tiếp theo sử dụng các hàm tích hợp trong Numpy để in ra hình dạng (`shape`), kiểu dữ liệu (`dtype`), và kiểu của mảng (`type`). Cuối cùng, chúng ta in ra nội dung của mảng để kiểm tra kết quả

Ví dụ Pytorch:

```

1 # PyTorch code
2 import torch
3
4 # Tạo một danh sách 2 chiều
5 list_2D = [[1, 2, 3],
6             [4, 5, 6],
7             [7, 8, 9]]
8 # Tạo một tensor 2 chiều
9 tensor_2D_pt = torch.tensor(list_2D)
10
11 # Kiểm tra hình dạng, kiểu dữ liệu, loại,
12 # thiết bị lưu trữ của tensor
12 print("Hình dạng của tensor: ",
13       tensor_2D_pt.shape)
14 print("Kiểu dữ liệu của tensor: ",
15       tensor_2D_pt.dtype)
16 print("Loại của tensor: ",
17       type(tensor_2D_pt))
18 print("Thiết bị lưu trữ của tensor: ",
19       tensor_2D_pt.device)
20 print("Tensor:\n", tensor_2D_pt)

```

```

=====
Output
=====
Hình dạng của tensor: torch.Size([3, 3])
Kiểu dữ liệu của tensor: torch.int64
Loại của tensor: <class 'torch.Tensor'>
Thiết bị lưu trữ của tensor: cpu
Tensor:
tensor([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
=====
```

Trong đoạn mã Pytorch trên, chúng ta bắt đầu bằng cách tạo một danh sách 2D (**list_2D**) đại diện cho ma trận 3x3. Sau đó, chúng ta sử dụng Pytorch để chuyển đổi danh sách này thành tensor 2 chiều (**tensor_2D_pt**).

Dòng mã tiếp theo sử dụng các thuộc tính tích hợp trong Pytorch để in ra hình dạng (**shape**), kiểu dữ liệu (**dtype**), kiểu của tensor (**type**), và thiết bị lưu trữ của tensor (**device**). Cuối cùng, chúng ta in ra nội dung của tensor để kiểm tra kết quả.

Ví dụ Tensorflow:

```

1 import tensorflow as tf
2
3 # Tạo một danh sách 2 chiều
4 list_2D = [[1, 2, 3],
5             [4, 5, 6],
6             [7, 8, 9]]
7 # Tạo một tensor 2 chiều từ danh sách
8 tensor_2D_tf = tf.convert_to_tensor(
9                         list_2D)
10
11 # Kiểm tra hình dạng, kiểu dữ liệu, loại,
12 # và thiết bị mà tensor được lưu trữ
13 print("Hình dạng của tensor: ",
14       tensor_2D_tf.shape)
15 print("Kiểu dữ liệu của tensor: ",
16       tensor_2D_tf.dtype)
17 print("Loại của tensor: ",
18       type(tensor_2D_tf))
19 print("Thiết bị mà tensor được lưu trữ: ",
20       tensor_2D_tf.device)
21 print(tensor_2D_tf)

```

```

=====
Output
=====
Hình dạng của tensor: (3, 3)
Kiểu dữ liệu của tensor: <dtype: 'int32'>
Loại của tensor: <class 'tensorflow.python.framework.ops.EagerTensor'>
Thiết bị mà tensor được lưu trữ: /job:localhost/replica:0/task:0/device:CPU:0
tf.Tensor(
[[1 2 3]
 [4 5 6]
 [7 8 9]], shape=(3, 3), dtype=int32)
=====
```

Tương tự như Pytorch, trong đoạn mã Tensorflow trên, chúng ta bắt đầu bằng cách tạo một danh sách

2D (**list _ 2D**) đại diện cho ma trận 3x3. Sau đó, chúng ta sử dụng Tensorflow để chuyển đổi danh sách này thành một tensor 2 chiều (**tensor _ 2D _ tf**) bằng hàm **tf.convert _ to _ tensor()**.

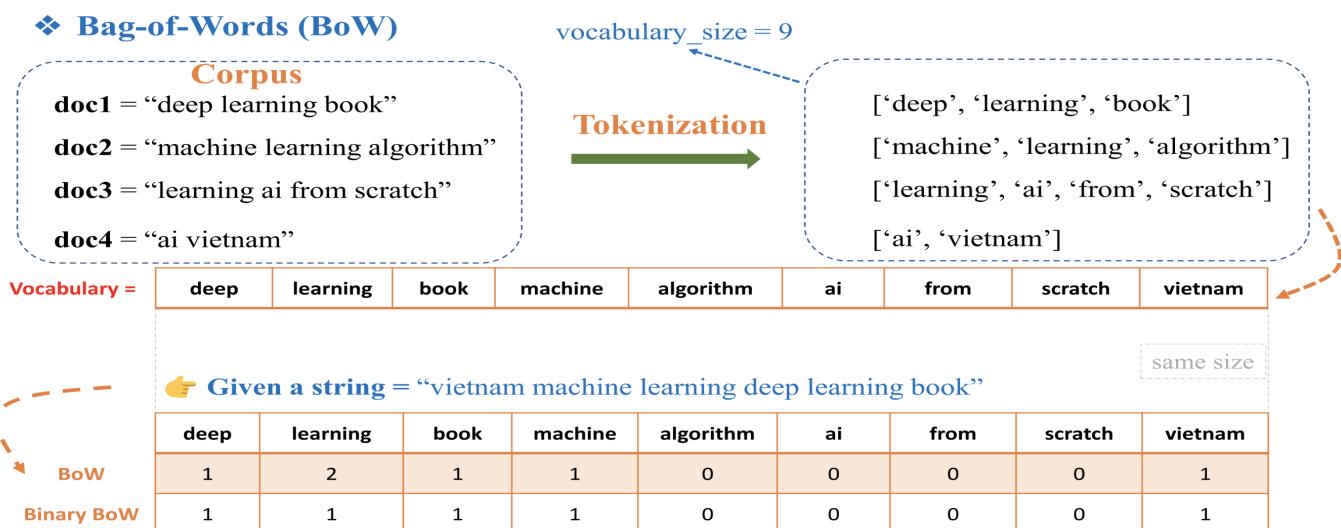
Dòng mã tiếp theo sử dụng các thuộc tính tích hợp trong Tensorflow để in ra hình dạng (**shape**), kiểu dữ liệu (**dtype**), kiểu của tensor (**type**), và thiết bị lưu trữ của tensor (**device**). Cuối cùng, chúng ta in ra nội dung của tensor để kiểm tra kết quả.

Basic Python - List

Hoàng-Nguyễn 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**

```

1  corpus = [## Your Code Here ##]
2  # Your code here
3

```

Output:

- **Tôi thích AI thích Toán:** [1, 1, 2, 0, 0, 0, 1]

Các Hàm Khởi Tạo Numpy Array và Pytorch/Tensorflow Tensor - Phần 1

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Khi lập trình với các thư viện Numpy, Pytorch, Tensorflow có một số cách để nhanh chóng tạo ra các array với giá trị, kích thước khác nhau. Trong bài tập này, chúng ta sẽ tìm hiểu các sử dụng 3 hàm **zeros**, **ones**, **full**.

a) **zeros**

zeros là hàm thực hiện chức năng tạo array, tensor toàn giá trị 0 với đầu vào là kích thước và kiểu dữ liệu ta muốn. Cả 3 thư viện đều sử dụng hàm trên, với cú pháp tương tự nhau.

np.zeros(shape)	zeros() function									
torch.zeros(shape)	<table border="1"> <tr> <td>0</td><td>1</td><td>2</td></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> </table>	0	1	2	0	0	0	1	0	0
0	1	2								
0	0	0								
1	0	0								
tf.zeros(shape)										

Hình 1: Minh họa và cú pháp sử dụng hàm zeros.

Nhìn chung, hàm **zeros** chủ yếu dùng để khởi tạo mảng hoặc tensor với các giá trị 0, thường được dùng trong quá trình xây dựng các mô hình máy học và thực hiện các phép toán số học.

b) **ones**

Tương tự với **zeros**, hàm **ones** tạo mảng chứa toàn số 1 với đầu vào là kích thước do người dùng chỉ định.

Syntax

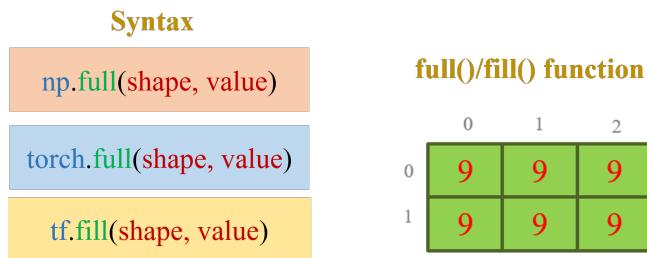
np.ones(shape)	ones() function									
torch.ones(shape)	<table border="1"> <tr> <td>0</td><td>1</td><td>2</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> </table>	0	1	2	0	1	1	1	1	1
0	1	2								
0	1	1								
1	1	1								
tf.ones(shape)										

Hình 2: Minh họa và cú pháp sử dụng hàm ones.

c) **full, fill**

Hàm **full** giúp chúng ta tạo array, tensor(Pytorch) với phần tử là giá trị chỉ định, đầu vào của hàm **full** gồm kích thước array và giá trị hằng số muốn tạo. Khác với hai thư viện trên, Tensorflow sử dụng hàm **fill**.

Hàm **full** với Numpy, Pytorch hay **fill** với Tensorflow giúp tạo array hoặc tensor với kích thước và giá trị được chỉ định, có ích khi ta cần khởi tạo các cấu trúc dữ liệu với giá trị đồng nhất.



Hình 3: Minh họa và cú pháp sử dụng hàm full/fill.

2. Bài tập

Câu 1: Hãy viết chương trình sử dụng hàm zeros tạo Numpy array, Tensorflow tensor, Pytorch tensor chỉ chứa giá trị là số 0 với kích thước (3, 4)?

Câu 2: Hãy viết chương trình sử dụng hàm ones tạo Numpy array, Tensorflow tensor, Pytorch tensor chỉ chứa giá trị là số 1 với kích thước (3, 4)?

Câu 3: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor chỉ chứa giá trị là số 5 với kích thước (3, 4)?

3. Đáp án

Câu 1: zeros

Chương trình sau tạo array với kích thước (3, 4), đối với thư viện Numpy sử dụng cú pháp **np.zeros**, bên trong hàm này chúng ta truyền vào kích thước array chúng ta mong muốn là (3, 4).

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo array toàn số 0
5 arr_zeros = np.zeros((3, 4))
6 print(arr_zeros)

```

```

=====
Output =====
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
=====
```

Tương tự như thư viện Numpy, Pytorch sử dụng cú pháp **torch.zeros**

```

1 #Pytorch code
2 import torch
3
4 # Tạo tensor toàn số 0
5 tensor_zeros = torch.zeros((3, 4))
6 print(tensor_zeros)

```

```

=====
Output =====
tensor([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
=====
```

Tương tự, Tensorflow sử dụng cú pháp **tf.zeros**

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor toàn số 0
5 tensor_zeros = tf.zeros((3, 4))
6 print(tensor_zeros)

```

```

=====
Output =====
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]], shape=(3, 4), dtype=
float32
=====
```

Câu 2: ones

Chương trình sau tạo array với kích thước (3, 4), đối với thư viện Numpy sử dụng cú pháp **np.ones**

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo array toàn số 1
5 arr_ones = np.ones((3, 4))
6 print(arr_ones)

```

```

=====
Output =====
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
=====
```

Với Pytorch và Tensorflow cú pháp thực hiện tương tự với cú pháp torch.ones, tf.ones

```

1 #PyTorch code
2 import torch
3
4 # Tạo tensor toàn số 1
5 tensor_ones = torch.ones((3, 4))
6 print(tensor_ones)

```

```

=====
Output =====
tensor([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
=====
```

```

1 #TensorFlow code
2 import tensorflow as tf
3
4
5 # Tạo tensor toàn số 1
6 tensor_ones = tf.ones((3, 4))
7 print(tensor_ones)

```

```

=====
Output =====
tf.Tensor(
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]], shape=(3, 4), dtype=
float32)
=====
```

Câu 3: full, fill

Chương trình sau tạo array với kích thước (3, 4), giá trị hằng số là 5. Đối với thư viện Numpy sử dụng cú pháp **np.full**

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo array toàn số 5
5 arr_full = np.full((3, 4), 5)
6 print(arr_full)

```

```

=====
Output =====
[[5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]]
=====
```

Tương tự như thư viện Numpy, Pytorch sử dụng cú pháp **torch.full**

```

1 #Pytorch code
2 import torch
3
4 # Tạo tensor toàn số 5
5 tensor_full = torch.full((3, 4), 5)
6 print(tensor_full)

```

```

=====
Output =====
tensor([[5, 5, 5, 5],
       [5, 5, 5, 5],
       [5, 5, 5, 5]])
=====
```

Khác với hai thư viện trên, Tensorflow sử dụng hàm **fill** để tạo một tensor với giá trị được chỉ định.

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor toàn số 5
5 tensor_full = tf.fill((3, 4), 5)
6 print(tensor_full)

```

```

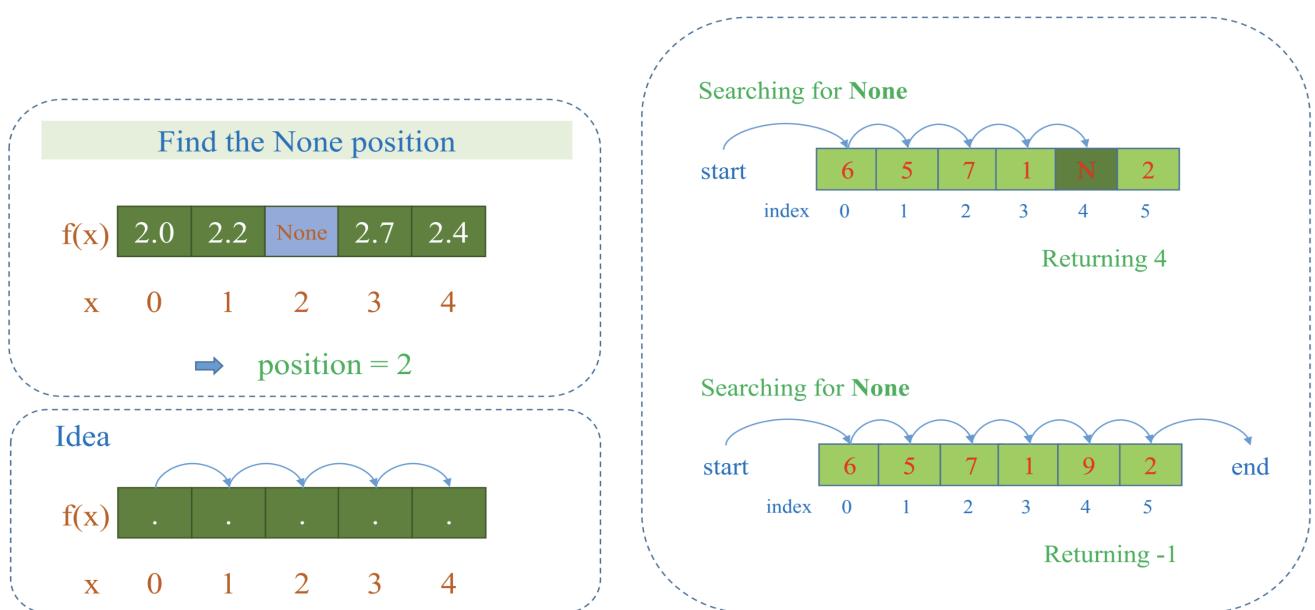
=====
Output =====
tf.Tensor(
[[5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]], shape=(3, 4), dtype=int32)
=====
```

Basic Python - List

Hoàng-Nguyễn 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`

```

1  lst_data = [## Your Code Here ##]
2  # Your code here
3

```

Output:

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

Các Phép Tính Numpy, Pytorch và Tensorflow Transpose và Summation

Dinh-Tiem Nguyen và Quang-Vinh Dinh

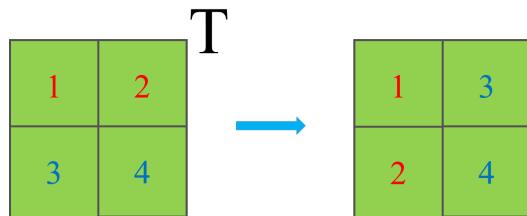
1. Mô tả

a) Chuyển vị - Transpose

Chuyển vị (Transpose) là một phép toán quan trọng trong đại số tuyến tính, nó thực hiện thay đổi vị trí của các hàng thành cột và ngược lại trong một ma trận. Kí hiệu của phép toán chuyển vị thường được biểu diễn bằng kí hiệu T hoặc \top . Nếu A là một ma trận, thì chuyển vị của A được kí hiệu là A^T hoặc A^\top .

Công Thức Chuyển Vị: Nếu A là một ma trận với các phần tử a_{ij} , thì chuyển vị của A , ký hiệu là A^T hay A^\top , có kích thước là số cột của A trở thành số hàng của A và ngược lại. Cụ thể, nếu A có kích thước $m \times n$, thì A^T có kích thước $n \times m$.

$$(A^T)_{ij} = A_{ji}$$



Hình 1: Minh họa transpose

Ví dụ:

Giả sử có ma trận A như sau:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Chuyển vị của A , ký hiệu là A^T , sẽ là:

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Một số tính chất của phép chuyển vị:

- Chuyển vị của một ma trận chuyển vị lại sẽ cho ra ma trận ban đầu: $(A^T)^T = A$.
- Chuyển vị của tổng hai ma trận bằng tổng chuyển vị của từng ma trận: $(A + B)^T = A^T + B^T$.
- Chuyển vị của tích hai ma trận bằng tích đảo ngược vị trí của chuyển vị từng ma trận: $(AB)^T = B^T A^T$.

Ví dụ:

```

1 # Numpy code
2 import numpy as np
3
4 #Tạo array 2d
5 arr_1 = np.array([[1, 2], [3, 4]])
6 # Chuyển vị array
7 # Cách 1: Sử dụng hàm np.transpose()
8 arr_transposed_1 = np.transpose(arr_1)
9 # Cách 2: Sử dụng toán tử T
10 arr_transposed_2 = arr_1.T
11 # In ra màn hình
12 print("array 1:\n", arr_1)
13 print("array 1 sau khi chuyển vị, cách 1:\n", arr_transposed_1)
14 print("array 1 sau khi chuyển vị, cách 2:\n", arr_transposed_2)

```

```

=====
Output =====
array 1:
[[1 2]
 [3 4]]

array 1 sau khi chuyển vị, cách 1:
[[1 3]
 [2 4]]

array 1 sau khi chuyển vị, cách 2:
[[1 3]
 [2 4]]

=====

```

Trong Numpy, chuyển vị của một mảng (array) có thể được thực hiện bằng cách sử dụng hàm `np.transpose()` hoặc toán tử chuyển vị `.T`. Chuyển vị thay đổi vị trí của các hàng thành cột và ngược lại trong array.

Trong Pytorch, chuyển vị của tensor có thể được thực hiện bằng cách sử dụng toán tử chuyển vị `.T`, hàm `torch.t()` hoặc `torch.transpose()`. Chuyển vị thay đổi vị trí của các hàng thành cột và ngược lại trong tensor. Cú pháp:

```

1 #cách 1:
2 torch.t(input)
3
4 #Cách 2
5 torch.transpose(input, dim0, dim1)

```

Kết quả trả về là một tensor mới là kết quả của phép chuyển vị. Trong đó:

- input: Tensor cần được chuyển vị.
- dim0, dim1: Các chiều được chọn để thực hiện chuyển vị.

Ví dụ:

```

1
2 #Pytorch code
3 import torch
4
5 #Tạo tensor 2d
6 tensor_1 = torch.tensor([[1, 2], [3, 4]])
7 # Chuyển vị tensor
8 # Cách 1: Sử dụng hàm torch.t()
9 tensor_transposed_1 = torch.t(tensor_1)
10 # Cách 2: Sử dụng hàm torch.transpose()
11 tensor_transposed_2 = torch.transpose(
12     tensor_1, 0, 1)
13 # In ra màn hình
14 print("tensor 1:\n", tensor_1)
15 print("tensor 1 sau khi chuyển vị, cách
1: \n", tensor_transposed_1)
16 print("tensor 1 sau khi chuyển vị, cách
2: \n", tensor_transposed_2)

```

```

=====
Output =====
tensor 1:
tensor([[1, 2],
       [3, 4]])

tensor 1 sau khi chuyển vị, cách 1:
tensor([[1, 3],
       [2, 4]])

tensor 1 sau khi chuyển vị, cách 2:
tensor([[1, 3],
       [2, 4]])

=====

```

Trong Tensorflow, không có toán tử chuyển vị **T**, mà chỉ dùng hàm **tf.transpose()**. Cách sử dụng cũng tương tự như Numpy và Pytorch:

Ví dụ:

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor 2d
5 tensor_1 = tf.constant([[1, 2], [3, 4]])
6 # Chuyển vị tensor
7 tensor_transposed = tf.transpose(tensor_1)
8
9 # In ra màn hình
10 print("tensor 1:\n", tensor_1)
11 print("tensor 1 sau khi chuyển vị:\n",
      tensor_transposed)

```

```

=====
Output =====
tensor 1:
tf.Tensor(
[[1 2]
 [3 4]], shape=(2, 2), dtype=int32)

tensor 1 sau khi chuyển vị:
tf.Tensor(
[[1 3]
 [2 4]], shape=(2, 2), dtype=int32)
=====
```

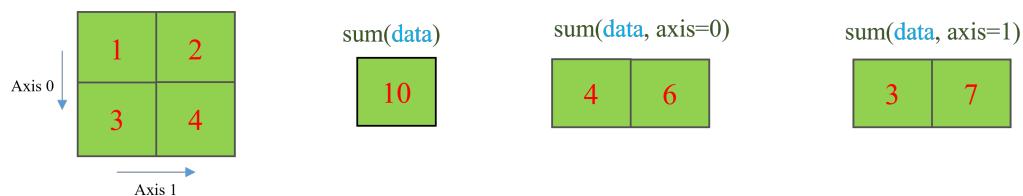
b) *Summation*

Trong thư viện Numpy, hàm **np.sum()** được sử dụng để tính tổng của các phần tử trong mảng (array). Hàm này có thể được áp dụng trên các mảng 1D, 2D hoặc có số chiều cao hơn. Cú pháp:

```
1 np.sum(a, axis=None, dtype=None, keepdims=False, initial=0, where=True)
```

Trong đó:

- a: Mảng đầu vào.
- axis: (Tùy chọn) Chiều hoặc các chiều trên đó tổng sẽ được thực hiện. Mặc định là None, tức là tổng của tất cả các phần tử trong mảng.
- dtype: (Tùy chọn) Kiểu dữ liệu của kết quả.
- keepdims: (Tùy chọn) Nếu là True, giữ chiều của mảng kết quả (nếu có) giống với chiều của mảng đầu vào.
- initial: (Tùy chọn) Giá trị khởi tạo cho tổng.
- where: (Tùy chọn) Một mảng Boolean chỉ định vị trí các phần tử được sử dụng trong phép toán.



Hình 2: Minh họa summation

Trong ví dụ sau, **np.sum()** được sử dụng để tính tổng của mảng array. Tham số sử dụng ở đây là mặc định khi tính tổng các phần tử trong toàn bộ array, sử dụng tham số axis để tính tổng theo cột hoặc hàng. Kết quả được in ra màn hình bao gồm tổng của tất cả các phần tử, tổng theo cột, và tổng theo hàng của mảng.

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo mảng 2D
5 arr = np.array([[1, 2, 3],
6                 [4, 5, 6]])
7
8 # Tính tổng của tất cả các phần tử trong mảng
9 total_sum = np.sum(arr)
10
11 # Tính tổng theo cột (theo chiều dọc)
12 column_sum = np.sum(arr, axis=0)
13
14 # Tính tổng theo hàng (theo chiều ngang)
15 row_sum = np.sum(arr, axis=1)
16
17 # In ra màn hình
18 print("Mảng:\n", arr)
19 print("Tổng của tất cả các phần tử trong mảng:\n", total_sum)
20 print("Tổng theo cột:\n", column_sum)
21 print("Tổng theo hàng:\n", row_sum)

```

```

===== Output =====
Mảng:
[[1 2 3]
 [4 5 6]]

Tổng của tất cả các phần tử trong mảng:
21

Tổng theo cột:
[5 7 9]

Tổng theo hàng:
[ 6 15]

=====

```

Trong Pytorch, chúng ta tính sum tương tự như trong Numpy, ví dụ:

```

1 #Pytorch code
2 import torch
3
4 # Tạo tensor 2d
5 tensor_1 = torch.tensor([[1, 2], [3, 4]])
6 # Tính tổng tensor
7 tensor_sum = torch.sum(tensor_1)
8 # Tính tổng tensor theo cột axis = 0
9 tensor_sum_axis_0 = torch.sum(tensor_1,
                               axis=0)
10 # Tính tổng tensor theo hàng axis = 1
11 tensor_sum_axis_1 = torch.sum(tensor_1,
                               axis=1)
12
13 # In ra màn hình
14 print("tensor:\n", tensor_1)
15 print("Tổng các phần tử trong tensor:\n",
      tensor_sum)
16 print("Tổng theo cột:\n",
      tensor_sum_axis_0)
17 print("Tổng theo hàng:\n",
      tensor_sum_axis_1)

```

```

===== Output =====
tensor:
tensor([[1, 2],
       [3, 4]])

Tổng các phần tử trong tensor:
tensor(10)

Tổng theo cột:
tensor([4, 6])

Tổng theo hàng:
tensor([3, 7])

=====

```

Ví dụ tính sum trong Tensorflow:

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor 2d
5 tensor_1 = tf.constant([[1, 2], [3, 4]])
6 # Tính tổng tensor
7 tensor_sum = tf.reduce_sum(tensor_1)
8 # Tính tổng tensor theo cột axis = 0
9 tensor_sum_axis_0 = tf.reduce_sum(tensor_1
10 , axis=0)
11 # Tính tổng tensor theo hàng axis = 1
12 tensor_sum_axis_1 = tf.reduce_sum(tensor_1
13 , axis=1)
14
15 # In ra màn hình
16 print("tensor:\n", tensor_1)
17 print("Tổng các phần tử trong tensor:\n",
18      tensor_sum)
19 print("Tổng theo cột:\n",
20      tensor_sum_axis_0)
21 print("Tổng theo hàng:\n",
22      tensor_sum_axis_1)

```

===== Output =====

tensor:
tf.Tensor(
[[1 2]
[3 4]], shape=(2, 2), dtype=int32)

Tổng các phần tử trong tensor:
tf.Tensor(10, shape=(), dtype=int32)

Tổng theo cột:
tf.Tensor([4 6], shape=(2,), dtype=int32)

Tổng theo hàng:
tf.Tensor([3 7], shape=(2,), dtype=int32)

=====

2. Bài tập

Câu 1: Viết chương trình tạo hai Numpy array, Pytorch tensor, Tensorflow tensor với các giá trị số nguyên ngẫu nhiên trong khoảng [-10, 10) với kích thước (3, 4). Sau đó chuyển vị array, tensor thứ 2 và thực hiện phép nhân matrix multiplication. Lưu ý: sử dụng seed=2024

Câu 2: Viết chương trình tạo một Numpy array, Pytorch tensor, Tensorflow tensor với các giá trị số nguyên ngẫu nhiên trong khoảng [-10, 10) với kích thước (3, 3). Sau đó hãy tính tổng của toàn bộ tensor, array, tiếp theo tính tổng theo chiều dọc, chiều ngang . Lưu ý: sử dụng seed=2024

3. Đáp án

Đáp án sẽ được gửi cho các bạn vào khoảng 8h tối trên group Code.

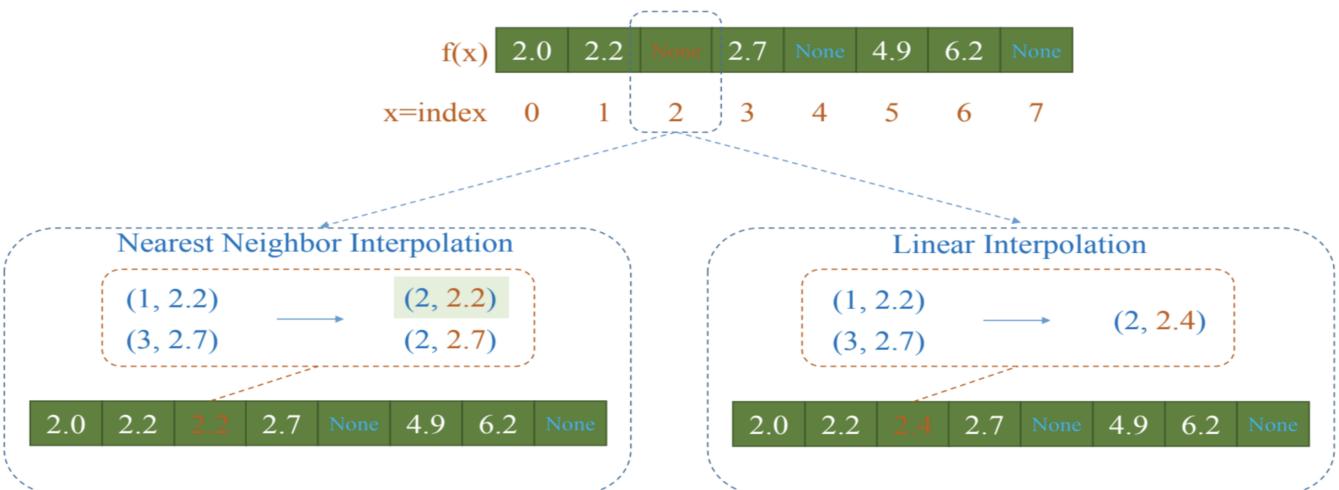
Basic Python - List

Hoàng-Nguyễn 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 $lst_data = [1, 1.1, \text{None}, 1.4, \text{None}, 1.5, \text{None}, 2.0]$. Sau đó, áp dụng phương pháp nội suy **Nearest Neighbor** để gán giá trị **None** có trong lst_data

```

1 lst_data = [## Your Code Here ##]
2   # Your code here
3

```

Output:

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

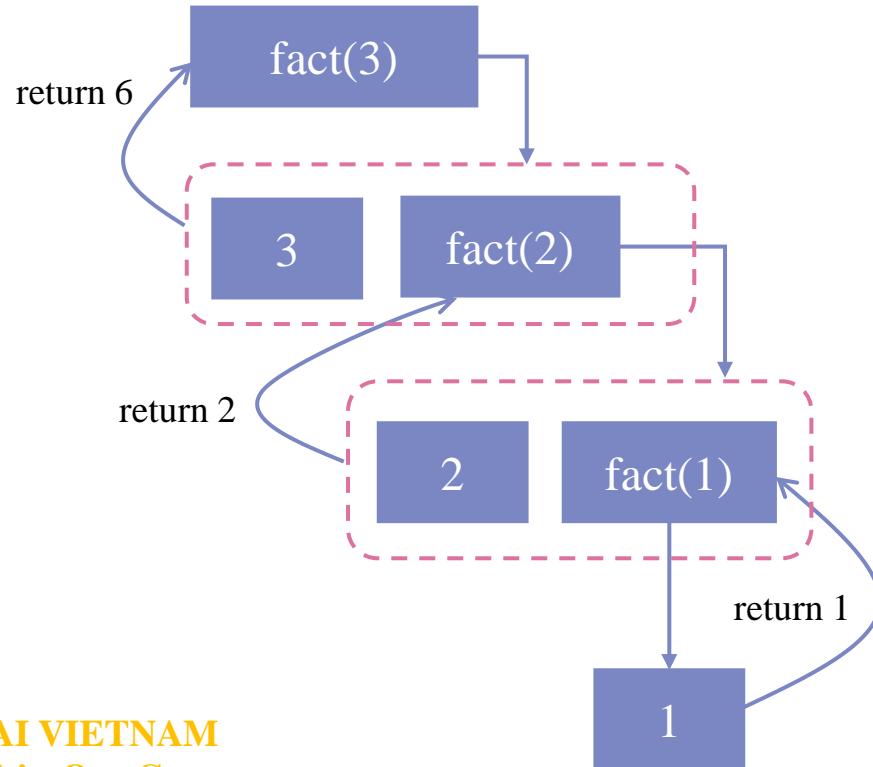
BUIDING A FACTORIAL APP WITH STREAMLIT



FACTORIAL

$$n! = n * (n-1) * (n-2) * \dots * 1$$

3! Phương pháp đệ quy



Factorial function

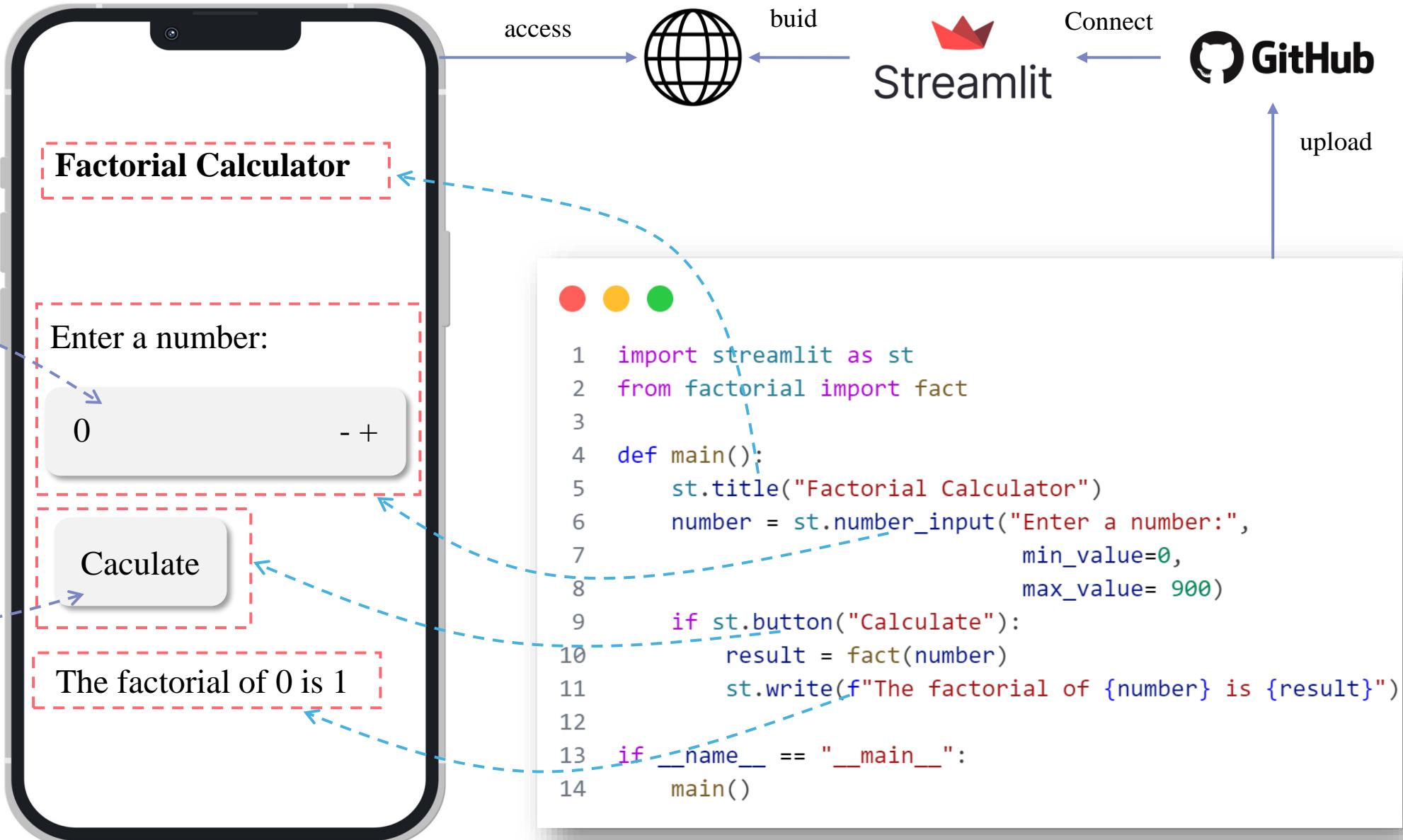


```
1 def fact(n):  
2     if n == 0 or n == 1:  
3         return 1  
4     else:  
5         return n * fact(n-1)
```



Streamlit

Syntax	Explain
pip install streamlit	Cài đặt streamlit
st.title(string)	Đặt tiêu đề cho ứng dụng
st.button(label)	Tạo ra một nút để người dùng có thể nhấn giúp thực hiện một hành động
st.write(*args, **kwargs)	Hiển thị văn bản hoặc giá trị của biến



Basic Python - List

Hoàng-Nguyễn 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:

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

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

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*

```
1 lst_data = [## Your Code Here ##]
2 # Your code here
3
```

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

Basic Python - List

Hoàng-Nguyễn 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 ($m \times n$ và $n \times p$) là ma trận C có kích thước $m \times p$

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

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

2. **Bài tập:** Cho 2 ma trận sau: $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ và $B = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \\ 1 & 0 & 1 \end{bmatrix}$. 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

```

1  mat_a = [## Your Code Here ##]
2  mat_b = [## Your Code Here ##]
3  # 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:** $[[7, 10, 19], [19, 31, 55], [31, 52, 91]]$

Basic Python - List Comprehension

Hoàng-Nguyễn 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.

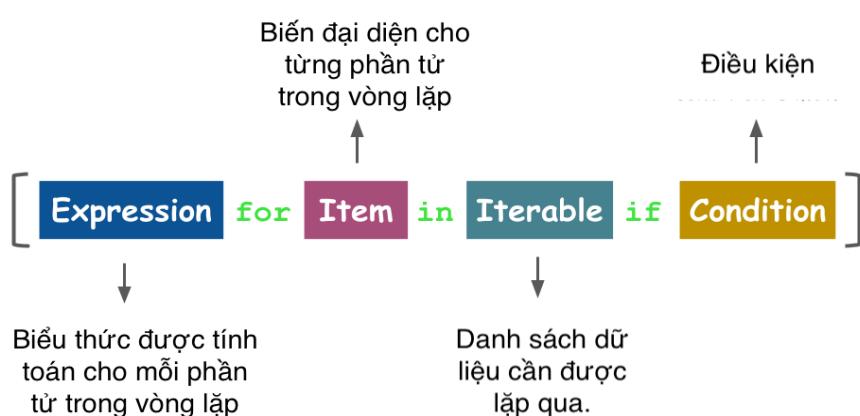
- + **Dễ sử dụng:** có cú pháp đơn giản và dễ học.

- 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

```

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

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

Basic Python - List - Tuple

Trung-Trực Trần và Hoàng-Nguyễn 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 $\text{distance}(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2}$
- **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.

```

1 my_tuple1 = ()
2 my_tuple2 = ()
3 # 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)`.

Basic Python - File

Hoàng-Nguyễn Vũ

1. Mô tả:

- **Đọc file** là thao tác lấy dữ liệu từ file vào chương trình Python để xử lý. **Ghi file** là thao tác lưu dữ liệu từ chương trình Python vào file. Có nhiều cách để đọc và ghi file trong Python:

- **Cách 1: Sử dụng hàm open()**: Hàm open() được sử dụng để mở file. Sau khi mở file, bạn có thể sử dụng các phương thức khác để đọc hoặc ghi dữ liệu vào file, cuối cùng ta cần phải đóng file lại sau khi thực thi xong các quá trình trên. Ví dụ:

```

1 # Read File
2 f = open("test.txt", "r")
3 data = f.read()
4 f.close()
5
6 print(data)
7
8 # Write File
9 f = open("test.txt", "w")
10 f.write("I Love AI Vietnam")
11 f.close()
```

- **Cách 2: Sử dụng câu lệnh with**: Câu lệnh with giúp bạn đảm bảo rằng file được đóng đúng cách sau khi sử dụng. Ví dụ:

```

1 # Read File
2 with open("test.txt", "r") as f:
3     data = f.read()
4 print(data)
5
6 # Write File
7 with open("test.txt", "w") as f:
8     f.write("I Love AI Vietnam")
```

2. Bài tập:

- **Câu 1:** Tạo List có tên *lst_data* gồm các số từ 1 đến 10, sau đó ghi toàn bộ list trên vào file có tên: *data.txt* với nội dung là 1 chuỗi số từ list trên nối với nhau bằng dấu -
- **Câu 2:** Đọc file *data.txt* vừa tạo ở câu 1 và lưu vào List mới có tên *lst_filter* gồm các số chia hết cho 3

Output:

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

Basic Python - File

Hoàng-Nguyễn Vũ

1. Mô tả:

Pandas là một thư viện mã nguồn mở được viết bằng Python, chuyên dụng cho việc phân tích dữ liệu và thao tác dữ liệu. Nó cung cấp một loạt các công cụ mạnh mẽ để giúp bạn:

- Đọc và ghi dữ liệu từ nhiều nguồn khác nhau, bao gồm tệp CSV, Excel, cơ sở dữ liệu SQL, v.v.
- Làm sạch và chuẩn bị dữ liệu cho phân tích.
- Thực hiện các phép toán thống kê trên dữ liệu.
- Tạo biểu đồ và hình ảnh để trực quan hóa dữ liệu.

2. Bài tập: Dữ liệu dataset McDonald

- **Phân tích dữ liệu thực đơn của McDonald** Hãy đọc file *menu.csv* có trong dataset phía trên và lưu vào biến *data* với thư viện Pandas. Sau đó sử dụng hàm *describe()*, *info()*, *head()*, *tail()* để thống kê tập dataset. **Lưu ý:** Lấy 5 phần tử đầu tiên và 10 phần tử cuối cùng khi dùng hàm *head* và *tail*

	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)
count	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000
mean	368.269231	127.096154	14.165385	21.815385	6.007692	29.965385
std	240.269886	127.875914	14.205998	21.885199	5.321873	26.639209
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	210.000000	20.000000	2.375000	3.750000	1.000000	4.750000
50%	340.000000	100.000000	11.000000	17.000000	5.000000	24.000000
75%	500.000000	200.000000	22.250000	35.000000	10.000000	48.000000
max	1880.000000	1060.000000	118.000000	182.000000	20.000000	102.000000

Hình 1: Kết quả của hàm *describe()*

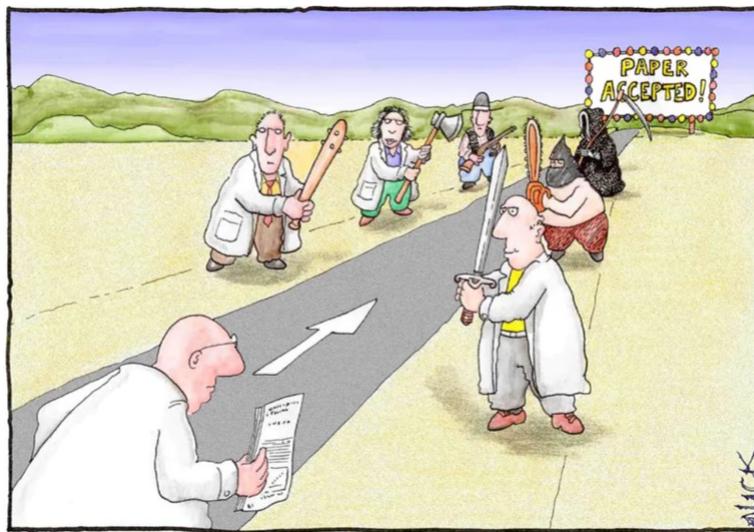
1 # Info			
<class 'pandas.core.frame.DataFrame'>			
RangeIndex: 260 entries, 0 to 259			
Data columns (total 24 columns):			
#	Column	Non-Null Count	Dtype
0	Category	260 non-null	object
1	Item	260 non-null	object
2	Serving Size	260 non-null	object
3	Calories	260 non-null	int64
4	Calories from Fat	260 non-null	int64
5	Total Fat	260 non-null	float64
6	Total Fat (% Daily Value)	260 non-null	int64

Hình 2: Kết quả của hàm *info()*

Basic Python - Work with Text Data

Hoàng-Nguyễn Vũ

1. Mô tả: Làm quen với thư viện newspaper3k và nlpk



- **Thư viện newspaper3k** là một thư viện Python mã nguồn mở giúp bạn trích xuất dữ liệu từ các bài báo trực tuyến. Thư viện này hỗ trợ nhiều trang web tin tức khác nhau, bao gồm: VnExpress, Tuổi Trẻ, Thanh Niên, Zing News, VTC News, ... Các tính năng nổi bật của thư viện bao gồm:
 - + **Trích xuất dữ liệu:** Newspaper3k có thể trích xuất nhiều loại dữ liệu từ các trang web báo chí, bao gồm tiêu đề, bài viết, tóm tắt, tác giả, ngày tháng, hình ảnh, video, v.v.
 - + **Hỗ trợ nhiều trang web:** Newspaper3k hỗ trợ hơn 100 trang web báo chí khác nhau, bao gồm cả các trang web tiếng Việt như VnExpress, Tuổi Trẻ, Thanh Niên, v.v.
 - + **Dễ sử dụng:** Newspaper3k cung cấp một API đơn giản để trích xuất dữ liệu từ các trang web báo chí.
 - + **Mã nguồn mở:** Newspaper3k là một thư viện mã nguồn mở, vì vậy bạn có thể sử dụng và sửa đổi nó miễn phí.
- **Thư viện nltk (Natural Language Toolkit)** là một thư viện mã nguồn mở được phát triển bởi Python. Nó cung cấp một bộ công cụ mạnh mẽ để xử lý ngôn ngữ tự nhiên (NLP) trong Python. Các tính năng chính của thư viện bao gồm:
 - + **Phân tích cú pháp:** NLTK có thể phân tích cú pháp của các câu tiếng Anh để xác định cấu trúc và thành phần của chúng.
 - + **Phân loại từ:** NLTK có thể xác định loại từ (danh từ, động từ, tính từ, v.v.) của các từ trong một câu.

- + **Gán nhãn ngữ nghĩa:** NLTK có thể gán nhãn ngữ nghĩa (tên riêng, địa điểm, tổ chức, v.v.) cho các từ trong một câu.
- + **Tóm tắt văn bản:** NLTK có thể tóm tắt các văn bản dài thành các văn bản ngắn hơn.
- + **Dịch máy:** NLTK có thể dịch văn bản từ ngôn ngữ này sang ngôn ngữ khác.

- **Cách cài đặt và sử dụng một số tính năng:**

- Để cài đặt thư viện newspaper3k, ta sẽ cài thông qua câu lệnh:

```
1 !pip install newspaper3k
2 !pip install nltk
```

- Cách sử dụng các tính năng chính của thư viện:

- + **Thư viện newspaper3k:**

```
1 from newspaper import Article
2
3 # Tạo một đối tượng Article từ URL của bài báo
4 article = Article('https://vnexpress.net/thoi-tiet-mien-bac
- -mien-trung-mien-nam-ngay-14-3-4518045.html')
5
6 # Tải bài báo
7 article.download()
8 article.parse()
9
10 # In bài báo
11 print(article.text)
12
13 # Lấy toàn bộ ảnh trong bài báo
14 print(article.images)
```

+ **Kết quả:** Tòa án quân sự Mỹ thông báo Ryan Mays, thủy thủ bị tố đốt tàu đổ bộ USS Bonhomme Richard, được trả án....

{Link đường dẫn ảnh của bài báo...}

- + **Thư viện nltk:**

```
1 import nltk
2 from nltk.tokenize import word_tokenize
3
4 nltk.download('punkt')
5
6 data = "Tôi thích học AI và Toán"
7 # Bước 1: Tokenization data
8 tokenization = word_tokenize(data)
9 # Bước 2: Gọi thư viện Pos tagging
10 result = nltk.pos_tag(tokenization)
11 print(result)
```

+ **Kết quả:** [('Tôi', 'NNP'), ('thích', 'NN'), ('học', 'NN'), ('AI', 'NNP'), ('và', 'NN'), ('Toán', 'NNP')]

2. Bài tập:

- **Câu 1:** Thực hiện đọc và tóm tắt bài báo tại đường dẫn sau: Bài báo VnExpress
- **Câu 2:** Thực hiện pos tagging bài báo trên với thư viện NLTK.

Kết quả:

+ **Câu 1:** Ngày 13/3, Cognition Labs, startup về công nghệ trí tuệ nhân tạo tại Mỹ, công bố kỹ sư phát triển phần mềm AI đầu tiên trên thế giới. Với Devin, các kỹ sư có thể tập trung vào những vấn đề thú vị hơn, các đội kỹ thuật có thể nỗ lực cho những mục tiêu tham vọng hơn", Cognition cho biết...

+ **Câu 2:** [('Kỹ', 'NNP'), ('sư', 'NN'), ('phần', 'NN'), ('mềm', 'NN'), ...]

Basic Python - File

Hoàng-Nguyễn Vũ

1. Mô tả:

- Ở bài tập trước, chúng ta đã làm quen với thao tác đọc ghi file, bài tập này sẽ giúp chúng ta hiểu rõ và áp dụng cơ bản trong việc phân tích dữ liệu có trong file:



2. Bài tập: Dữ liệu dataset

- **Câu 1:** Hãy đọc file *data.txt* có trong dataset phía trên và lưu vào biến *data* sau khi loại bỏ các ký tự \n và thay thế bằng khoảng trắng, đồng thời chuyển về chữ cái thường toàn bộ văn bản
- **Câu 2:** Phân tích văn bản trên và lưu vào biến có tên *distinct_words* các chữ cái duy nhất trong câu
- **Câu 3:** Đếm số lần từng chữ trong *distinct_words* xuất hiện trong văn bản. Và cho biết chữ nào xuất hiện **nhiều nhất** và **ít nhất**.

```
1 # Read File
2 with open("data.txt", "r") as f:
3     # Your Code Here
4
```

Output:

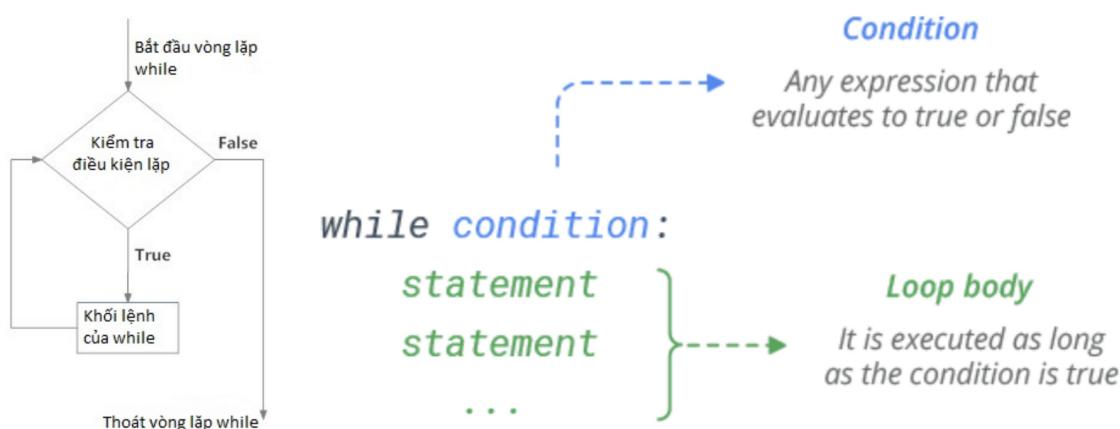
- **Câu 1:** he who conquers himself.....people get what they want
- **Câu 2:** 'a', 'again', 'and', ..., 'with', 'you', 'your'
- **Câu 3:**
 - + "thought" in data is 1
 - + "you" in data is 4
 - + "a" is most frequent word
 - + "makes" is the least common word

Basic Python - While-Loop

Hoàng-Nguyễn Vũ

1. Mô tả:

- Sự ngẫu nhiên là một đặc điểm cơ bản của các hiện tượng trong thực tế. Nó thể hiện qua việc kết quả của một thí nghiệm không thể dự đoán trước được một cách chính xác. **Ví dụ:** Chọn ngẫu nhiên hai số a và b sao cho tổng của a + b bằng 40, câu hỏi đặt ra rằng, chúng ta phải tạo ngẫu nhiên bao nhiêu lần để thỏa mãn điều kiện trên ?
- Cấu trúc vòng lặp While-Loop:** Vòng lặp while là một cấu trúc điều khiển trong Python cho phép thực thi một khối mã nhiều lần miễn là điều kiện cho trước vẫn còn đúng. Cấu trúc câu lệnh như sau:



2. Bài tập: về sự ngẫu nhiên - Dùng vòng lặp While

- Chọn ngẫu nhiên hai số a và b thuộc từ 1 đến 20 sao cho tổng của a + b bằng 40, câu hỏi đặt ra rằng, chúng ta phải tạo ngẫu nhiên bao nhiêu lần để thỏa mãn điều kiện trên ?

```

1 import random
2 def random_number_with_condition(total):
3     # Set a random value that is the same between devices
4     random.seed(0)
5     # Your code here
6

```

Ví dụ:

- Test case 1: `random_number_with_condition(40)` → 343 lần
- Test case 2: `random_number_with_condition(20)` → 32 lần
- Test case 3: `random_number_with_condition(35)` → 96 lần

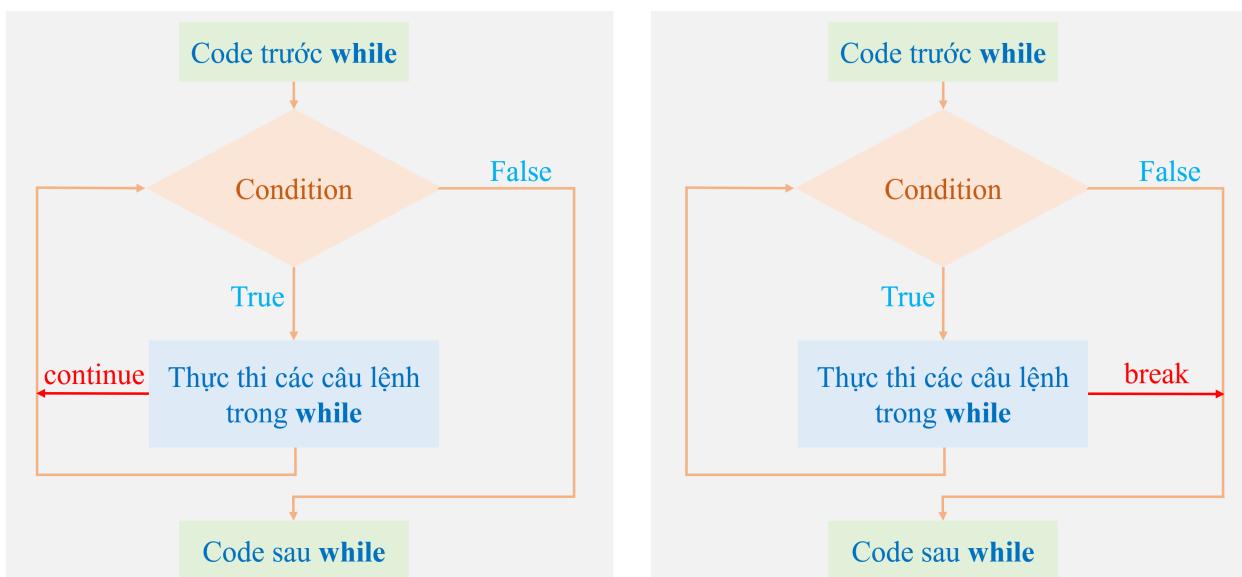
Basic Python

While Loop + continue and break keywords

Bảo-Sơn Trần

1. Mô tả:

- Trong Python, câu lệnh **while** được sử dụng để tạo một vòng lặp, trong đó các biểu thức được kiểm tra và nếu điều kiện đúng, khối mã lệnh bên trong vòng lặp sẽ được thực thi. Các lệnh **break** và **continue** được sử dụng để kiểm soát luồng của vòng lặp.



Hình 1: Vòng lặp while kết hợp với từ khóa continue và break.

- Câu lệnh **break** được sử dụng để thoát khỏi vòng lặp ngay lập tức.
- Câu lệnh **continue** được sử dụng để bỏ qua phần còn lại của vòng lặp và chuyển đến lần lặp tiếp theo.

2. Bài tập:

Cài đặt hàm `find_divisible_number(a)` tìm số nguyên dương nhỏ nhất lớn hơn 100 và chia hết cho số nguyên dương `a`

```

1 def find_divisible_number(a):
2     """
3         Find the smallest integer that is greater than 100 and divisible
4         by the integer a
5     """
6
7     #TODO: Your code here
  
```

Ví dụ:

- Test case 1: `find_divisible_number(5) → 105`
- Test case 2: `find_divisible_number(17) → 102`

Basic Python - While-Loop Exercise

Bảo-Sơn Trần

1. Mô tả:

- Phương pháp Newton (Newton's Method), còn được gọi là phương pháp Newton-Raphson, là một phương pháp số học để tìm gần đúng của các nghiệm của một hàm số thực. Cụ thể, nó thường được sử dụng để tìm gần đúng của các nghiệm của phương trình $f(x) = 0$.
- Ngoài ứng dụng trong tìm nghiệm của một hàm số, phương pháp Newton còn có ứng dụng trong máy học (Machine learning) trong việc tìm nghiệm của đạo hàm của hàm loss. Tuy nhiên đây là phương pháp không phổ biến bằng thuật toán gradient descent.
- Ở bài này, chúng ta sẽ dùng phương pháp Newton để tính căn bậc hai cho một số dương a . Chúng ta thực hiện các bước sau:
 - (a) Khởi tạo giá trị $x_0 = a$, $n = 0$ và cho trước giá trị ε (thực ra x_0 có thể nhận bất kỳ giá trị dương nào). Tiếp đó, ta sẽ đi xây dựng hàm $f(x) = x^2 - a$. Ở đây, ta xem x_n (ở bước hiện tại đang là x_0) chính là lời giải cho bài toán tính căn bậc hai của a .
 - (b) Cải thiện xấp xỉ x_n bằng xấp xỉ x_{n+1} theo công thức tổng quát như sau:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$
 - (c) So sánh xấp xỉ x_{n+1} và x_n . Nếu $|x_{n+1} - x_n| < \varepsilon$, ta sẽ dừng việc cải thiện xấp xỉ, và trả về kết quả căn bậc hai của a là x_{n+1} . Ngược lại thực hiện tiếp bước (b) với $n = n + 1$.

2. Bài tập:

Cài đặt hàm `find_squared_root(a)` tìm căn bậc hai cho một số a bất kì với $\varepsilon = 0.001$.

```

1
2     def find_squared_root(a):
3         """Find the squared root of number a"""
4         EPSILON = 0.001
5
6         #TODO: Your code here
7

```

Ví dụ:

- Test case 1: `find_squared_root(2)` → 1.4142135623746899
- Test case 2: `find_squared_root(3)` → 1.7320508100147276

Các Hàm Khởi Tạo Numpy Array và Pytorch/Tensorflow Tensor - Phần 2

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Trong bài tập này, chúng ta tiếp tục tìm hiểu các hàm có chức năng tạo array, tensor đặc biệt. Chúng ta sẽ tìm hiểu và sử dụng các hàm **arange**, **range**, **eye**, và **random**.

a) **arange**, **range**

Trong Numpy, Pytorch, hàm **arange** được sử dụng để tạo một mảng chứa dãy số có giá trị tăng dần với khoảng cách cố định. Trong Tensorflow chúng ta sẽ sử dụng hàm **range**. Cú pháp như sau:

Syntax	arange() / range() function
<code>np.arange(start=0, stop, step=1)</code>	
<code>torch.arange(start=0, stop, step=1)</code>	$\text{arr1} = \begin{array}{ c c c c c } \hline 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 2 & & \\ \hline \end{array}$
<code>tf.range(start=0, stop, step=1)</code>	$\text{arr2} = \begin{array}{ c c c } \hline 0 & 2 & 4 \\ \hline \end{array}$

Hình 1: Minh họa và cú pháp sử dụng hàm arange, range.

Hàm **arange** (hoặc **range** trong Tensorflow) rất hữu ích khi ta muốn tạo ra một dãy số tăng dần với các giá trị cách đều nhau.

b) **eye**

Hàm **eye** được sử dụng để tạo ma trận đơn vị, tức là một ma trận vuông có tất cả các phần tử trên đường chéo chính có giá trị 1, các phần tử còn lại là 0. Cú pháp sử dụng như sau:

Syntax	eye() function
<code>np.eye(N, M)</code>	
<code>torch.eye(N, M)</code>	
<code>tf.eye(N, M)</code>	$\begin{array}{ c c c } \hline 0 & 1 & 2 \\ \hline 1 & 1 & 0 \\ \hline 2 & 0 & 1 \\ \hline \end{array}$ <p>N: number of rows M: number of columns defaults to N</p>

Hình 2: Minh họa và cú pháp sử dụng hàm eye.

Hàm **eye** giúp tạo array hoặc tensor đơn vị, đặc biệt hữu ích trong nhiều ứng dụng toán học và xử lý dữ liệu.

c) **random**

Trong thư viện Numpy, Pytorch cung cấp hàm **rand** để tạo mảng với các giá trị ngẫu nhiên trong khoảng [0.0, 1.0] với kích cỡ (shape) được chỉ định. Ngoài ra có thể sử dụng hàm **randint** để tạo mảng với các số nguyên ngẫu nhiên. Đối với thư viện Tensorflow chúng ta sử dụng hàm **random.uniform**.

Syntax	rand() function	Syntax	randint() function
np.random.rand(shape)	0 1 2 0 0.574 0.682 0.704 2 0.806 0.844 0.799	np.random.randint(low, hight, shape) torch.randint(low, hight, shape) tf.random.uniform(shape, minval, maxval, dtype)	0 1 2 0 6 3 9 2 0 1 -5
torch.rand(shape)			
tf.random.uniform(shape, minval, maxval, dtype)			

Hình 3: Minh họa và cú pháp sử dụng hàm random.

Các hàm này giúp việc tạo dữ liệu ngẫu nhiên trở nên dễ dàng trong quá trình phát triển mô hình Machine Learning, Deep Learning.

2. Bài tập

Câu 1: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor trong khoảng [0, 10) với step=1?

Câu 2: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor là ma trận đơn vị với kích thước (3, 3).

Câu 3: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor ngẫu nhiên trong khoảng giá trị [0, 1] với kích thước (3, 4). Tiếp theo hãy tạo array, tensor với các giá trị là số nguyên trong khoảng [-10, 10]. Lưu ý trong bài tập này, các bạn sẽ sử dụng seed = 2024 để đảm bảo kết quả giống đáp án, cách sử dụng như sau:

```

1 # Numpy code
2 import numpy as np
3 np.random.seed(2024)
4
5 # Pytorch code
6 import torch
7 torch.manual_seed(2024)
8
9 # Tensorflow code
10 import tensorflow as tf
11 tf.random.set_seed(2024)

```

3. Đáp án

Câu 1: *arange, range*

Trong numpy chúng ta sử dụng hàm **np.arange**

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo array trong khoảng [0, 10),
5 # bước nhảy 1
6 arr_arange = np.arange(10)
7 print(arr_arange)

```

```

=====
Output =====
[0 1 2 3 4 5 6 7 8 9]
=====
```

Trong PyTorch, để tạo tensor chứa dãy số, ta sử dụng hàm **torch.arange**:

```
1 #Pytorch code
2 import torch
3
4 # Tạo tensor trong khoảng [0, 10],
5 # bước nhảy 1
6 tensor_arange = torch.arange(10)
7 print(tensor_arange)
```

```
===== Output =====
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
=====
```

Trong Tensorflow, ta có thể sử dụng hàm **tf.range** để thực hiện chức năng tương tự:

```
1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor trong khoảng [0, 10],
5 # bước nhảy 1
6 tensor_arange = tf.range(10)
7 print(tensor_arange)
```

```
===== Output =====
tf.Tensor([0 1 2 3 4 5 6 7 8 9], shape=(10,), dtype=int32)
=====
```

Câu 2: *eye*

Trong numpy chúng ta sử dụng hàm `np.eye` để tạo ma trận đơn vị, trong đó giá trị chúng ta truyền vào là số lượng hàng của ma trận đầu ra, số lượng cột mặc định bằng số lượng hàng, nếu muốn số lượng cột khác bạn cần truyền vào hàm `eye` số lượng cột chỉ định.

```
1 # Numpy code
2 import numpy as np
3
4 # Tạo array với đường chéo chính là 1,
5 # còn lại là 0
6 arr_eye = np.eye(3)
7 print(arr_eye)
```

```
===== Output =====
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
=====
```

Trong Pytorch, để tạo tensor đơn vị, ta sử dụng hàm **torch.eye**:

```
1 #Pytorch code
2 import torch
3
4 # Tạo tensor với đường chéo chính là 1,
5 # còn lại là 0
6 tensor_eye = torch.eye(3)
7 print(tensor_eye)
```

```
===== Output =====
tensor([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])
=====
```

Trong Tensorflow, hàm **eye** cũng được sử dụng để tạo constant tensor đơn vị:

```
1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor với đường chéo chính là 1,
5 # còn lại là 0
6 tensor_eye = tf.eye(3)
7 print(tensor_eye)
```

```
===== Output =====
tf.Tensor(
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]], shape=(3, 3), dtype=float32)
=====
```

Câu 3: *random*

Trong Numpy, chúng ta sẽ sử dụng hàm **random.rand** để tạo mảng với các giá trị ngẫu nhiên trong khoảng giá trị mặc định là [0, 1). Để tạo mảng với giá trị ngẫu nhiên là số nguyên chúng ta dùng **random.randint**, khi sử dụng hàm này ta cần truyền vào khoảng giá trị lấy ngẫu nhiên và kích thước đầu ra mong muốn.

```

1 # Numpy code
2 import numpy as np
3
4 # Đặt seed để đảm bảo kết quả ngẫu nhiên
5 # có thể tái tạo được
6 np.random.seed(2024)
7
8 # Tạo một mảng với các giá trị ngẫu nhiên
9 # trong khoảng [0, 1) với kích thước (3,
10 # 4)
11 arr_rand = np.random.rand(3, 4)
12 print("Mảng ngẫu nhiên trong khoảng
13     [0, 1):\n", arr_rand)
14
15 # Tạo một mảng với các giá trị ngẫu nhiên
16 # trong khoảng [-10, 10)
17 arr_randint = np.random.randint(-10, 10,
18                                 size=(3, 4))
19 print("Mảng ngẫu nhiên trong khoảng
20     [-10, 10):\n", arr_randint)

```

===== Output =====

```

Mảng ngẫu nhiên trong khoảng [0, 1):
[[0.58801452 0.69910875 0.18815196
  0.04380856]
 [0.20501895 0.10606287 0.72724014
  0.67940052]
 [0.4738457 0.44829582 0.01910695
  0.75259834]]

Mảng ngẫu nhiên trong khoảng [-10, 10):
[[ 5  1 -3  8]
 [-1 -4  0 -9]
 [-5  9 -6 -2]]

```

=====

Trong Pytorch, các hàm tương tự cũng có sẵn thông qua module torch. **torch.rand** tạo tensor với các giá trị ngẫu nhiên từ phân phối đều trong khoảng [0.0, 1.0], **torch.randint** dùng để tạo tensor với các số nguyên ngẫu nhiên trong một khoảng cụ thể.

```

1 # Pytorch code
2 import torch
3
4 # Thiết lập seed để đảm bảo kết quả ngẫu
5 # nhiên có thể tái tạo được
6 torch.manual_seed(2024)
7
8 # Tạo tensor với các giá trị ngẫu nhiên
9 # trong khoảng [0, 1) với kích thước (3,
10 # 4)
11 tensor_rand = torch.rand((3, 4))
12 print("Tensor ngẫu nhiên trong khoảng
13     [0, 1):\n", tensor_rand)
14
15 # Tạo tensor với các giá trị ngẫu nhiên
16 # trong khoảng [-10, 10)
17 tensor_randint = torch.randint(-10, 10,
18                                 size=(3, 4))
19 print("Tensor ngẫu nhiên trong khoảng
20     [-10, 10):\n", tensor_randint)

```

===== Output =====

```

Tensor ngẫu nhiên trong khoảng [0, 1):
tensor([[0.5317, 0.8313, 0.9718, 0.1193],
        [0.1669, 0.3495, 0.2150, 0.6201],
        [0.4849, 0.7492, 0.1521, 0.5625]])

Tensor ngẫu nhiên trong khoảng [-10, 10):
tensor([[ 1,  8,  0, -2],
        [-9, -10, -9,  0],
        [-3, -7, -4,  8]])

```

=====

Trong Tensorflow, ta cũng có các hàm tương tự là **tf.random.uniform** để tạo tensor với các giá trị ngẫu nhiên trong khoảng [0.0, 1.0], và sử dụng **tf.random.uniform** với **dtype = tf.dtypes.int32** để tạo tensor với các số nguyên ngẫu nhiên trong một khoảng cụ thể.

```
1 #TensorFlow code
2 import tensorflow as tf
3
4
5 # Thiết lập seed để đảm bảo kết quả ngẫu
6 # nhiên có thể tái tạo được
7 tf.random.set_seed(2024)
8
9 # Tạo tensor với các giá trị ngẫu nhiên
10 # trong khoảng [0, 1)
11 tensor_rand = tf.random.uniform((3, 4))
12 print("Tensor ngẫu nhiên trong khoảng
13     [0, 1):\n", tensor_rand)
14
15 # Tạo tensor với các giá trị ngẫu nhiên
16 # trong khoảng [-10, 10)
17 tensor_randint = tf.random.uniform((3, 4),
18                                   minval=-10, maxval=10,
19                                   dtype=tf.dtypes.int32)
20 print("Tensor ngẫu nhiên trong khoảng
21     [-10, 10):\n", tensor_randint)
```

```
=====
Tensor ngẫu nhiên trong khoảng [0, 1):
tf.Tensor(
[[0.90034294 0.19453335 0.36069036
 0.66361904]
[0.76605344 0.2159369 0.6261736
 0.07380784]
[0.22062695 0.934368 0.93327904
 0.69267046]],
shape=(3, 4), dtype=float32)

Tensor ngẫu nhiên trong khoảng [-10, 10):
tf.Tensor(
[[-3 -7 9 3]
 [2 -5 -3 -5]
 [4 -3 -3 5]],
shape=(3, 4), dtype=int32)
=====
```

Python cơ bản - Bài luyện tập

Anh-Khoa Nguyen và Quang-Vinh Dinh

Question	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Answer														
Question	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Answer														
Question	29	30	31	32	33									
Answer														

Question 1: Which is the output of the following program?

```
1 n = 0
2 for i in range(0, 1000, 100):
3     n += i
4 print(n)
```

- a) 900 (print out an integer 900)
- b) 4500 (print out an integer 4500)
- c) None (print out the keyword None)
- d) Raise an error (the program throws an error)
- e) None of the above

Question 2: Which is the output of the following program?

```
1 def a_function_helper(x):
2     if x > 0:
3         return 'pos'
4     else:
5         return 'neg'
6
7 def a_function(data):
8     res = [a_function_helper(x) for x in data]
9     return res
10 data = [1, 0, -1, 2]
11 print(a_function(data))
```

- a) ['pos', 'neg', 'pos', 'neg']
- b) ['pos', 'neg', 'neg', 'pos']
- c) ['neg', 'pos', 'neg', 'pos']
- d) ['pos', 'neg', 'pos', 'neg']
- e) None of the above

Question 3: Which is the output of the following program?

```
1 data = "Python Programming"
2 print(data.split()[0])
```

- a) Python
- b) Programming
- c) Python Programming
- d) Raise an error
- e) None of the above

Question 4: Which is the output of the following program?

```
1 "elephant" *1 > "ant" *100
```

- a) True
- b) False
- c) Raise an error
- d) None of the above

Question 5: Which is the output of the following program?

```
1 def my_function(signal):  
2     var = True  
3     while var:  
4         var = False  
5         for i in range(len(signal) - 1):  
6             if signal[i] > signal[i + 1]:  
7                 signal[i], signal[i + 1] = signal[i + 1], signal[i]  
8             var = True  
9  
10 my_signal = [1, 0, 0, 1, 1, 1]  
11 my_function(my_signal)  
12 print(my_signal)
```

- a) [0, 0, 0, 0, 0, 0]
- b) [1, 1, 1, 1, 0, 0]
- c) [0, 0, 1, 1, 1, 1]
- d) None
- e) None of the above

Question 6: Which is the output of the following program?

```
1 n = 0  
2 for i in range(10):  
3     n += i  
4     if n>0 and n%2 == 0:  
5         break  
6 print(n)
```

- a) 0
- b) 3
- c) Raise an error
- d) 6
- e) None of the above

Question 7: Which is the output of the following program?

```
1 def my_function(list_nums = [1, 2, 3, 4]):  
2     var = 0  
3     for i in list_nums:  
4         var += i  
5     return var/len(list_nums)  
6  
7 my_function()
```

- a) 2.5
- b) 1.5
- c) Raise an error

- d) A and C
- e) None of the above

Question 8: Which is the output of the following program?

```

1 space1 = "It's sunny here!"
2 space2 = "Is it hot here in the spring?"
3 space = space1 + space2
4 print(space[-7:-1])

```

- a) spring
- b) *SyntaxError*: invalid syntax
- c) here!
- d) spring?
- e) None of the above

Question 9: Which is the output of the following program?

```

1 data = [1, 2, 3, "I", "am", "AI"]
2 new_data = sorted(data)
3 print(new_data)

```

- a) ["I", "am", "AI", 1, 2, 3]
- b) ["AI", "am", "I", 3, 2, 1]
- c) ["1", "2", "3", "I", "am", "AI"]
- d) *TypeError*: '<' not supported between instances of 'str' and 'int'
- e) None of the above

Question 10: Which is the output of the following program?

```

1 my_string = "Duo, come here!"
2
3 my_bag_of_word = []
4 for element in my_string:
5     if element not in my_bag_of_word:
6         my_bag_of_word.append(element)
7 print(my_bag_of_word)

```

- a) ['Duo', ',', ' ', 'c', 'm', 'e', 'h', 'r']
- b) ['Duo, come here!']
- c) *IndentationError*
- d) ['D', 'u', 'o', ',', ' ', 'c', 'm', 'e', 'h', 'r', '!']
- e) None of the above

Question 11: Which is the output of the following program?

```

1 my_string = "It's raining cats and dogs"
2 results = my_string.count('s')
3
4 print(results)

```

- a) Raise an error
- b) 2
- c) 3
- d) ['It', 'raining', 'cat', 'and', 'dog']
- e) None of the above

Question 12: Which is the output of the following program?

```
1 def my_function(x):
2     for i in range(x):
3         for j in range(x):
4             if i == j:
5                 print("1 ", end=" ")
6             else:
7                 print("0 ", end=" ")
8         print()
9
10 my_function(3)
```

- a)
1 0 0
0 1 0
0 0 1
- b)
0 1 1
1 0 1
1 1 0
- c) [1, 0, 0], [0, 1, 0], [0, 0, 1]
- d) None
- e) None of the above

Question 13: Which is the output of the following program?

```
1 def my_function(y):
2     var = 1
3     while(y > 1):
4         var = var * y
5         y = y - 1
6     return var
7
8 print(my_function(3))
```

- a) 6
- b) 7
- c) 8
- d) Raise an error
- e) None of the above

Question 14: Which is the output of the following program?

```
1 even_numbers = [x for x in range(3, 12) if x % 3==0]
2 print(even_numbers)
```

- a) None
- b) [1, 2, 3, 4, 5]
- c) [3, 6, 9]
- d) [3, 6, 9, 12]
- e) None of the above

Question 15: Which is the output of the following program?

```
1 X = [[1, 1, 2],
2     [3, 51, 99],
3     [5, 81, 23]]
4
5 result = [[0,0,0],
6           [0,0,0],
7           [0,0,0]]
8
9 for i in range(len(X)):
10    for j in range(len(X[0])):
11        result[j][i] = X[i][j]
12
13 for r in result:
14     print(r)
```

- a) [1, 3, 5]
[1, 51, 81]
[2, 99, 23]
- b) [1, 3, 5] [1, 51, 81] [2, 99, 23]
- c) [1, 10, 10]
[1, 3, 5]
[2, 99, 23]
- d) [1, 0, 0]
[0, 1, 81]
[0, 0, 0]
- e) None of the above

Question 16: Which is the output of the following program?

```
1 def my_function(my_data):
2     rs = 0
3     for i in my_data:
4         rs += i
5     return rs
6
7 my_list = [2, 3, 4, 5]
8 print(my_function(my_data = my_list))
```

- a) 12
- b) 13
- c) 14
- d) 15
- e) None of the above

Question 17: Which is the output of the following program?

```
1 def my_function(my_data):
2     result = []
3     for element in data:
4         if element not in result:
5             result.append(element)
6     return result
7
8 data = [ {'friends': ['Tom', 'Mike', 'Taylor']}, 2002, 2099, 3000, 'moon']
9 print(my_function(data))
```

- a) ['friends': ['Tom', 'Mike', 'Taylor'], 2002, 2099, 3000, 'moon']
- b) ['Tom', 'Mike', 'Taylor', 2002, 2099, 3000, 'moon']
- c) [None, 2002, 2099, 3000, 'moon']
- d) Raise an error
- e) None of the above

Question 18: Which is the output of the following program?

```
1 def my_function(data, max, min):
2     result = []
3     for i in data:
4         if i < min:
5             result.append(min)
6         elif i > max:
7             result.append(max)
8         else:
9             result.append(i)
10    return result
11
12 my_list = [1, 2, 3, 4, 5, 6, 7]
13 max = 5
14 min = 2
15 print(my_function(max = max, min = min, data = my_list))
```

- a) [0, 0, 3, 4, 0, 0, 0]
- b) [1, 1, 3, 4, 5, 7, 7]
- c) [2, 2, 3, 4, 5, 5, 5]
- d) *TypeError: my_function() got multiple values for argument 'data'*
- e) None of the above

Question 19: Which is the output of the following program?

```
1 def my_function(x, y):
2     x.extend(y)
3     return x
4
5 list_num1 = [-1, -4, -9]
6 list_num2 = [2, 3, 5]
7 list_num3 = [0, 0, 0]
8
9 my_function(list_num1, my_function(list_num2, list_num3))
```

- a) [-1, -4, -9, 2, 3, 5, 0, 0, 0]
- b) [1, 4, 9, 2, 3, 5]
- c) [2, 3, 5, 0, 0, 0]
- d) *TypeError : my_function() got multiple values for argument 'my_function(list_num2, list_num3)'*
- e) None of the above

Question 20: Which is the output of the following program?

```
1 def check_the_number(N):
2     list_of_numbers = []
3     result = ""
4     for i in range(1, 10, 3):
5         list_of_numbers.append(i)
6     if N in list_of_numbers:
7         results = "True"
8     if N not in list_of_numbers:
9         results = "False"
10    return results
11
12 N = 0
13 N += 5
14 results = check_the_number(N)
15 print(results)
```

- a) True
- b) False
- c) Nothing
- d) *SyntaxError*: invalid syntax
- e) None of the above

Question 21: Which is the output of the following program?

```
1 data = "[ 'dog', 'cat', [1, 2]]"
2 print(type(data))

a) <class 'str' >
b) <class 'list' >
c) <class 'mydata' >
d) NameError: name 'type' is not defined
e) None of the above
```

Question 22: Which is the output of the following program?

```
1 def my_function(n):
2     x = n[0]
3     for i in n:
4         if i < x:
5             x = i
6     return x
7
8 my_list = [-7, 99, 100, 2, 0, -20]
9 my_function(my_list)
```

- a) None
- b) Raise an error
- c) -20
- d) 100
- e) None of the above

Question 23: Which is the output of the following program?

```
1 Ann = '@This tEst is Very eaSy @@'
2 David = "i love iT!"
3
4 txt = Ann.strip('@').capitalize() + David.title()
5 print(txt)
```

- a) This test is very easy I Love It!
- b) this test is very easy @@I Love It!
- c) This test is very easy@ I LOVE IT!
- d) *SyntaxError: invalid syntax*
- e) None of the above

Question 24: Which is the output of the following program?

```
1 def my_function(n):
2     x = n[0]
3     for i in n:
4         if i > x:
5             x = i
6     return x
7
8 my_list = [-200, -99, -100, -2004, 0, -20]
9 my_function(my_list)
```

- a) None
- b) Raise an error
- c) 0
- d) -200
- e) None of the above

Question 25: Which is the output of the following program?

```
1 def my_function(integers, number = -999):
2     return any([True if i == number else False for i in integers])
3
4 my_list = [1, 3, 6, 9, -100, 19, 2004, 2023]
5 my_function(my_list, 2004)
```

- a) -999
- b) 2004
- c) True
- d) False
- e) None of the above

Question 26: Which is the output of the following program?

```
1 my_exam = "I do my homework every evening!"
2 print(my_exam.strip('!').split())
```

- a) ['I', 'do', 'my', 'homework', 'every', 'evening!']
- b) ['I', 'do', 'my', 'homework', 'every', 'evening', '!']
- c) ['I', 'do', 'my', 'homework', 'every', 'evening']
- d) All A, B, C
- e) None of the above

Question 27: Which is the output of the following program?

```
1 def function_helper(x, data):
2     for i in data:
3         if x == i:
4             return 0
5
6     return 1
7
8 def a_function(data):
9     res = []
10    for i in data:
11        if function_helper(i, res):
12            res.append(i)
13
14    return res
15
16 lst = [1, 1, 2, 2, 3]
17 print(a_function(lst))
```

- a) [1, 2, 3]
- b) [3, 2, 1]
- c) [1, 1, 2, 2, 3]
- d) [3, 2, 2, 1, 1]
- e) None of the above

Question 28: Which is the output of the following program?

```
1 def my_function(signal1, signal2):
2     var = False
3     for s1 in signal1:
4         for s2 in signal2:
5             if s1 == s2:
6                 var = True
7             return var
8 print(my_function([0, 1, 1, 0], [100, 11, 110]))
```

- a) False
- b) True
- c) Raise an error
- d) None
- e) None of the above

Question 29: Which is the output of the following program?

```
1 def my_function(signal1, signal2):
2     var = False
3     for s1 in signal1:
4         for s2 in signal2:
5             if s1 == s2:
6                 var = True
7             return var
8 print(my_function([0, 1, 1, 9], [9, 9, 9]))
```

- a) False
- b) True
- c) Raise an error
- d) None
- e) None of the above

Question 30: Which is the output of the following program?

```
1 def my_function(data):
2     var = []
3     for i in data:
4         if i%2 == 0:
5             var.append(i)
6     return var
7
8 print(my_function([1, 2, -4, 5, 7, 23]))
```

- a) [1, 5]
- b) [1, 5, 7, 23]
- c) A and D
- d) [2, -4]
- e) None of the above

Question 31: Which is the output of the following program?

```
1 def square(num):
2     """
3     Returns the square of the number entered.
4     The number entered must be an integer
5     """
6     return num ** 2
7
8 print(square.__doc__)
```

- a) Returns the square of the number entered. The number entered must be an integer
- b) numnum
- c) Raise an error
- d) See you!
- e) None of the above

Question 32: Which is the output of the following program?

```
1 def a_function(x):
2     res = ""
3     for i in x:
4         res = i + res
5     return res
6
7 x = 'cat'
8 print(a_function(x))
```

- a) cat
- b) tac
- c) ca
- d) ta
- e) None of the above

Question 33: Which is the output of the following program?

```
1 my_list = [1, 1, 2, 3, 5]
2 odd = 1
3 even = 0
4
5 for number in my_list:
6     if number % 3 == 0:
7         even += number
8     else:
9         odd += number
10 print(f"odd:{odd}, even:{even}")
```

- a) odd:1, even:0
- b) odd:3, even:10
- c) odd:10, even:3
- d) *SyntaxError*: invalid syntax
- e) None of the above

Basic Python - Work with Image Data

Hoàng-Nguyễn Vũ

1. **Mô tả:** Computer Vision là một lĩnh vực khoa học máy tính tập trung vào việc tạo ra các hệ thống máy tính có thể hiểu và phân tích hình ảnh và video. Nó có nhiều ứng dụng trong các lĩnh vực như: Nhận diện đối tượng, xử lý ảnh Y Khoa, Robotic, v.v... Có rất nhiều thư viện trong Python để chúng ta xử lý dữ liệu ảnh, trong đó OpenCV là một trong những thư viện điển hình. Một số thao tác cơ bản khi xử lý dữ ảnh như:

- + **Đọc và hiển thị ảnh:**

```

1 img = cv2.imread("image_path",0)
2 # Using Matplotlib to display image
3 plt.imshow(img, cmap='gray')
4 plt.show()

```

- + **Tăng/Giảm độ sáng:**

```

1 # Read image
2 img = cv2.imread("image_path")
3 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
4
5 # Adjust brightness
6 new_img = img.astype(np.float32) + 50
7 new_img = np.clip(new_img, 0, 255)
8 new_img = new_img.astype(np.uint8)

```

- + **Chuyển đổi không gian màu:** OpenCV cung cấp nhiều hàm để chuyển đổi giữa các không gian màu khác nhau.

Hàm	Chuyển đổi	Mô tả
cv2.COLOR_BGR2GRAY	BGR sang ảnh xám	Chuyển đổi ảnh từ BGR sang ảnh xám.
cv2.COLOR_BGR2HSV	BGR sang HSV	Chuyển đổi ảnh từ BGR sang không gian màu HSV (Hue, Saturation, Value).
cv2.COLOR_BGR2RGB	BGR sang RGB	Chuyển đổi ảnh từ BGR sang không gian màu RGB (R - Red, G - Green, B - Blue).

Ví dụ:

```

1 # Read image
2 img = cv2.imread("image_path")
3 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

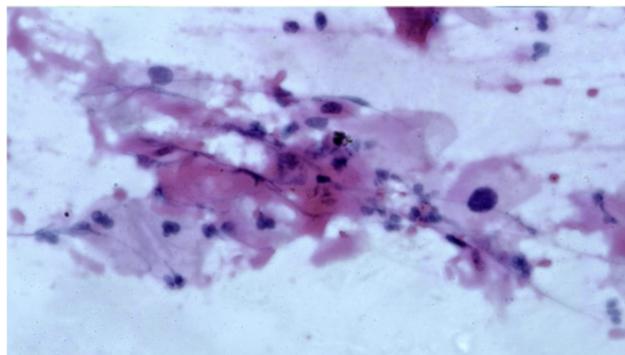
```

2. **Bài tập:** Dữ liệu ảnh Y Khoa - Nguồn: Kaggle

- **Câu 1:** Hãy đọc và hiển thị ảnh có tên 1.jpg trong tập dữ liệu trên

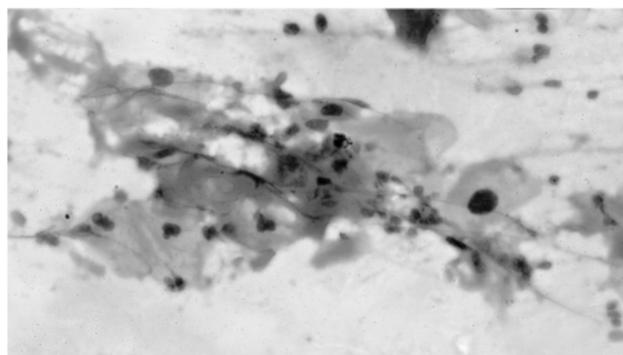
Kết quả:

<matplotlib.image.AxesImage at 0x798f508628c0>



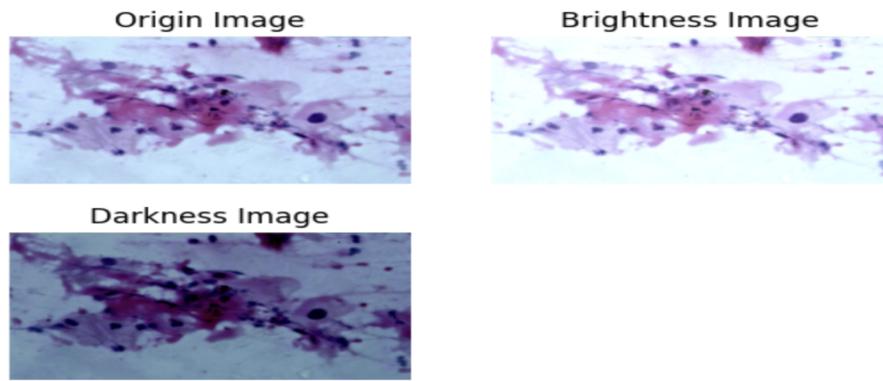
- **Câu 2:** Chuyển ảnh màu thành xám và hiển thị
Kết quả:

<matplotlib.image.AxesImage at 0x798f50868160>



- **Câu 3:** Tăng cường độ sáng ảnh màu lên 50 đơn vị, và giảm cường độ sáng xuống 80 đơn vị (Trên ảnh màu gốc ban đầu) và hiển thị

Kết quả:



Basic Python - Work with Text Data

Hoàng-Nguyễn Vũ

1. Mô tả: Làm quen với thư viện Underthesea

- **Underthesea** là một bộ công cụ mã nguồn mở hỗ trợ xử lý ngôn ngữ tự nhiên Tiếng Việt (NLP). Nó được phát triển bởi cộng đồng nghiên cứu NLP tại Việt Nam và được công bố lần đầu tiên vào năm 2017.

Bảng 1: Tính năng, ưu điểm của thư viện Underthesea

Tính năng	
Phân chia câu	Cắt một đoạn văn bản thành các câu riêng biệt.
Phân loại từ	Xác định loại từ (danh từ, động từ, tính từ, v.v.) của mỗi từ trong câu.
Gán thẻ POS	Gán thẻ cho mỗi từ với thông tin ngữ pháp (danh từ, động từ, tính từ, v.v.).
Nhận dạng thực thể tên riêng	Xác định các thực thể tên riêng (người, địa điểm, tổ chức, v.v.) trong văn bản.
Phân loại văn bản	Phân loại văn bản vào các chủ đề hoặc thể loại khác nhau.
Tóm tắt văn bản	Tạo tóm tắt ngắn gọn cho một đoạn văn bản dài.
Trích xuất quan điểm	Xác định các quan điểm và ý kiến trong văn bản.
Dịch máy	Dịch văn bản từ tiếng Việt sang tiếng Anh và ngược lại.
Ưu điểm	
Mã nguồn mở	Có thể sử dụng và sửa đổi miễn phí.
Dễ sử dụng	Cung cấp API đơn giản và dễ hiểu.
Hiệu quả	Đã được chứng minh hiệu quả trên nhiều tập dữ liệu tiếng Việt.
Cộng đồng	Được hỗ trợ bởi cộng đồng nghiên cứu NLP Việt Nam năng động.

• Cách cài đặt và sử dụng một số tính năng:

- Để cài đặt thư viện Underthesea, ta sẽ cài thông qua câu lệnh:

```
1 !pip install underthesea
```

- Các tính năng nổi bật trong thư viện Underthesea:

- + **Gán nhãn từ loại (POS tagging):**

```
1 from underthesea import pos_tag
2 pos_tag('Học sinh đang học toán')
```

+ Kết quả: [('Học sinh', 'N'), ('đang', 'R'), ('học', 'V'), ('toán', 'N')]

+ Phân loại văn bản (Text classification):

```
1 from underthesea import classify
2 classify('giá cổ phiếu đang có nhiều biến động trong thời gian qua')
```

+ Kết quả: ['kinh_doanh']

+ Phân tích cảm xúc (Sentiment Analysis):

```
1 from underthesea import sentiment
2 sentiment('Sản phẩm mình đặt về không như quảng cáo')
```

+ Kết quả: 'negative'

+ Phân đoạn câu văn (Sentence Segmentation):

```
1 from underthesea import sent_tokenize
2 text = 'Những đứa trẻ nghèo thế hệ tôi biết đọc biết viết, thành người bằng những cuốn giáo khoa đi mượn như thế. Cũng có những đứa nhà nghèo quá, không mượn đâu được bộ sách cho tử tế, môn được môn không, càng học càng đúp, cuối cùng bỏ dở giữa chừng.'
3 sent_tokenize(text)
```

+ Kết quả: ['Những đứa trẻ nghèo thế hệ tôi biết đọc biết viết, thành người bằng những cuốn giáo khoa đi mượn như thế.', 'Cũng có những đứa nhà nghèo quá, không mượn đâu được bộ sách cho tử tế, môn được môn không, càng học càng đúp, cuối cùng bỏ dở giữa chừng.]

+ Phân đoạn từ ngữ (Word Segmentation):

```
1 from underthesea import word_tokenize
2 sentence = 'Công trình của PGS Vân đã thay thế bạch kim trong pin nhiên liệu, giúp giảm giá thành mà pin có độ bền cao hơn.'
3 word_tokenize(sentence)
```

+ Kết quả: ['Công', 'trình', 'của', 'PGS', 'Vân', 'đã', 'thay', 'thế', 'bạch', 'kim', 'trong', 'pin', 'nhiên', 'liệu', ',', 'giúp', 'giảm', 'giá', 'thành', 'mà', 'pin', 'có', 'độ', 'bền', 'cao', 'hơn', ':']

2. Bài tập:

- Hãy thực hiện các task: POS Tagging, Text Classification, Sentiment Analysis, Sentence Segmentation, Word Segmentation đoạn văn bản sau:

Công cụ Suno AI nhanh chóng nhận được sự chú ý từ người dùng khi có thể tạo bài hát chỉ với vài câu lệnh. Phiên bản mới nhất V3 Alpha mới được giới thiệu cuối tháng 2, có bản miễn phí với 10 bài hát mỗi ngày.

Kết quả:

- + **POS Tagging:** [('Công cụ', 'N'), ('Suno', 'Np'), ('AI', 'P'), ...]
- + **Text Classification:** ['vi_tinh']
- + **Sentiment Analysis:** 'positive'
- + **Sentence Segmentation:** ['Công cụ Suno AI nhanh chóng nhận được sự chú ý từ người dùng khi có thể tạo bài hát chỉ với vài câu lệnh.', 'Phiên bản mới nhất V3 Alpha mới được giới thiệu cuối tháng 2, có bản miễn phí với 10 bài hát mỗi ngày.]
- + **Word Segmentation:** ['Công cụ', 'Suno', 'AI', 'nhanh chóng', ...]

Basic Python - Work with Text Data

Hoàng-Nguyễn Vũ

1. Mô tả: Làm quen với thư viện translate và googletrans

- **Thư viện Translate** là một thư viện mã nguồn mở được phát triển bởi Google. Nó cho phép bạn dịch văn bản, mã, tài liệu và trang web sang nhiều ngôn ngữ khác nhau. Thư viện này sử dụng công nghệ dịch máy của Google, được đào tạo trên một lượng lớn dữ liệu ngôn ngữ.
- **Googletrans** là một thư viện Python dựa trên thư viện Translate. Nó cung cấp một API đơn giản để dịch văn bản sang nhiều ngôn ngữ khác nhau. Googletrans cũng sử dụng công nghệ dịch máy của Google.
- **Điểm khác biệt chính giữa hai thư viện:**

Tính năng	Thư viện Translate	Googletrans
Ngôn ngữ hỗ trợ	Nhiều hơn	Ít hơn
Loại dữ liệu	Văn bản, mã, tài liệu, trang web	Văn bản
API	Phức tạp	Đơn giản
Dễ sử dụng	Khó hơn	Dễ hơn

• Cách cài đặt và sử dụng một số tính năng:

- Để cài đặt thư viện translate và Googletrans, ta sẽ cài thông qua câu lệnh:

```
1 !pip install translate
2 !pip install googletrans==4.0.0rc1
```

- Cách sử dụng dịch thuật trong thư viện translate và Googletrans:

+ **Thư viện Translate:**

```
1 from translate import Translator
2
3 translator= Translator(to_lang="vi")
4 translation = translator.translate("Don't cry because it's
      over, smile because it happened.")
5 print(translation)
```

+ **Kết quả:** Đừng khóc vì nó đã kết thúc, hãy mỉm cười vì nó đã xảy ra.

+ **Thư viện Googletrans:** Dịch từ tiếng Việt sang tiếng Pháp

```
1 from googletrans import Translator
2
3 # define a translate object
4 translate = Translator()
5
6 # Translate some text
```

```
7 result = translate.translate('Hôm nay thời tiết không tốt v  
à mưa nhiều', dest='fr')  
8  
9 print(result)  
10 print(result.text)
```

+ **Kết quả:** Translated(src=vi, dest=fr, text=Aujourd’hui, le temps n'est pas bon et pluvieux, pronunciation=None, extra_data="'confiden...")
Aujourd’hui, le temps n'est pas bon et pluvieux

2. Bài tập:

- Hãy thực hiện dịch câu sau với 2 ngôn ngữ: Anh, Nhật sử dụng cả 2 thư viện:

Công cụ Suno AI nhanh chóng nhận được sự chú ý từ người dùng khi có thể tạo bài hát chỉ với vài câu lệnh. Phiên bản mới nhất V3 Alpha mới được giới thiệu cuối tháng 2, có bản miễn phí với 10 bài hát mỗi ngày.

Kết quả:

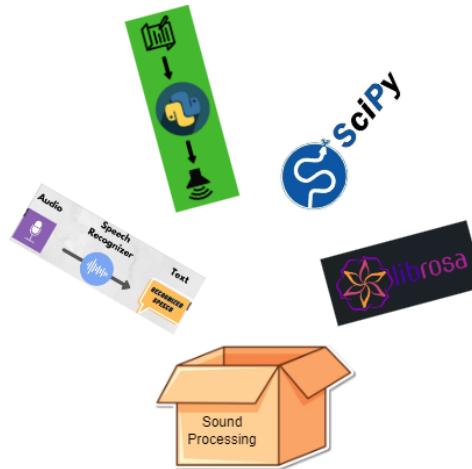
+ **Tiếng Anh:** Suno AI tools quickly get attention from users when they can create a song with just a few statements. The latest version of V3 Alpha was introduced at the end of February, with a free version with 10 songs a day.

+ **Tiếng Nhật:** Suno AIツールは、ユーザーがいくつかのステートメントで曲を作成できる場合、ユーザーからすぐに注目を集めます。V3 Alphaの最新バージョンは2月末に紹介され、1日10曲の無料バージョンがありました。

Thư viện xử lý âm thanh

Trung-Trực Trần

1. **Giới thiệu:** Trong Python, có một số thư viện phổ biến được sử dụng để xử lý âm thanh, các thư viện này giúp người dùng dễ dàng sử dụng để thao tác với file âm thanh chẳng hạn như đọc, ghi, cắt, nối và xử lý các file, chúng ta cùng tìm hiểu một số thư viện phổ biến hiện nay của Python trong dữ liệu này:



Librosa: Thư viện mạnh mẽ cho phân tích và xử lý âm thanh và âm nhạc.

- **librosa.load:** Hàm để đọc file âm thanh.
- **librosa.example:** Hàm để tải một ví dụ âm thanh từ bộ dữ liệu mẫu của Librosa.

Scipy: là một thư viện Python mạnh mẽ cho tính toán khoa học và kỹ thuật.

- **scipy.fft:** Hàm để thực hiện biến đổi Fourier.
- **scipy.signal.spectrogram:** Hàm để tạo biểu đồ phổ của tín hiệu.

SpeechRecognition: Thư viện để trích xuất thông tin từ dữ liệu âm thanh.

- **recognizer.record:** ghi âm từ nguồn âm thanh đã chọn và trả về dữ liệu âm thanh dưới dạng một đối tượng.
- **recognizer.recognize_google:** Phương thức này được sử dụng để nhận dạng văn bản từ dữ liệu âm thanh bằng cách sử dụng dịch vụ nhận dạng giọng nói của Google (Cần kết nối internet).

gtts: là một thư viện Python cho phép bạn tạo và phát lại giọng nói từ văn bản sử dụng dịch vụ Text-to-Speech của Google.

Cần các thuộc tính sau để sử dụng thư viện này:

- lang = 'vi' (Ngôn ngữ trả về).
- output_filename = 'record.mp3' (Loại file âm thanh đầu ra).
- content = "xin chào mọi người" (đoạn văn bản muốn chuyển thành voice).

2. Ví dụ cách sử dụng thư viện xử lý âm thanh.

Librosa

-Cài đặt thư viện:

```
1 !pip install librosa
2
```

-Sử dụng thư viện librosa để visualize âm thanh:

```
1 import librosa
2
3 filename = librosa.example('nutcracker')
4 # Load the audio as a waveform 'y'
5 # Store the sampling rate as 'sr'
6 audio, sr = librosa.load(filename)
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 time = np.linspace(0, audio.shape[0] / sr, num=audio.shape[0])
11 plt.plot(time, audio, color="blue")
12 plt.xlabel("Time (s)")
13 plt.ylabel("Amplitude (quantized)")
14 plt.title("Wav file visualization")
15 plt.axhline(y=0, color='r', linestyle='--')
16 plt.show()
17
```

Output: Biểu đồ cho thấy biên độ của sóng âm thanh theo thời gian.

speech recognition

-Cài đặt thư viện:

```
1 !pip install SpeechRecognition
2
```

-Sử dụng thư viện SpeechRecognition để chuyển âm thanh thành văn bản:

```
1 import speech_recognition as sr
2
3 r = sr.Recognizer()
4 audio_filename = 'data.wav'
5
6 my_audio = sr.AudioFile(audio_filename)
7 with my_audio as source:
8     audio = r.record(source)
9
10 print(type(audio))
11 your_speech = r.recognize_google(audio, language="vi-VN")
12 print("Audio transcription: ", your_speech)
13
```

Output: Hiện ra tận phía xa.

gtts

-Cài đặt thư viện:

```
1 !pip install gtts
2
```

-Sử dụng thư viện SpeechRecognition để chuyển âm thanh thành văn bản:

```
1 from gtts import gTTS
2 import librosa
3 import numpy as np
4
5 lang='vi'
6 output_filename = 'record.mp3'
7 content = "xin chào mọi người"
8 output = gTTS(content, lang=lang, slow=False) # text to speech
9
10 output.save(output_filename) # save google audio to a file
11 data, sr = librosa.load(output_filename) # load google audio
12 using librosa library
13
14 import IPython
15 IPython.display.display(IPython.display.Audio(np.transpose(data),
rate=sr)) # display audio
15
```

Output: Một đoạn âm thanh mp3 với nội dung là "Xin chào mọi người".

3. Bài tập:

- Hãy thực hiện các thư viện trên với các input âm thanh và văn bản khác nhau để có thể hiểu rõ hơn về cách sử dụng các thư viện này.

```
1 test_case_1 = Một file âm thanh tự do.
2 test_case_2 = "Tôi yêu AIO"
3 # Your code here
4
```

Output: Các đồ thị và đoạn văn bản tương ứng với file truyền vào.

Output: Một đoạn âm thanh mp3 với nội dung là "Tôi yêu AIO"

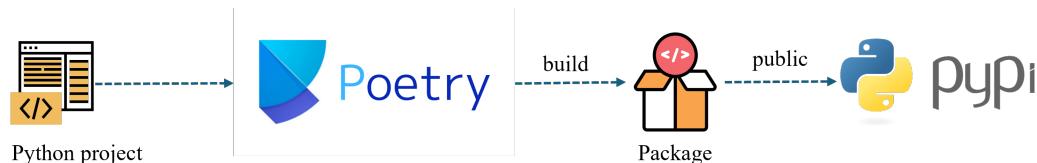
HƯỚNG DẪN TRIỂN KHAI VÀ ĐÓNG GÓI DỰ ÁN PYTHON VỚI POETRY

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1 Mở đầu

Làm thế nào để xây dựng các package, library như math, numpy, tensorflow, pytorch? Làm thế nào để tái sử dụng code trong dự án? Trong bài viết này sẽ hướng dẫn cách triển khai xây dựng một package đơn giản với Poetry và Anaconda sau đó phát hành nó trên nền tảng chia sẻ package nổi tiếng PyPI. Yêu cầu:

- Máy tính đã cài đặt python
- Đã biết cơ bản về cách tạo và quản lý môi trường trong python



Hình 1: Pipeline xây dựng package

2 Cài đặt thư viện Anaconda và Poetry

Poetry là một công cụ quản lý các package, library phụ thuộc trong môi trường riêng biệt để phát triển các dự án python, đặc biệt là nó cung cấp các chức năng thuận tiện cho việc xây dựng package python và phát hành lên nền tảng PyPI.

Có một số cách khác nhau để cài Poetry, trong hướng dẫn này sử dụng cách cài đặt thông qua pipx - một công cụ để cài đặt và quản lý các package, library python(không phải là các package, library như numpy, pandas mà bạn hay dùng để import khi viết code đâu, mà nó là các package có thể sử dụng từ dòng lệnh như Python, Poetry...) . Để cài đặt pipx, bạn xem hướng dẫn chi tiết tại [đây](#). Đối với hệ điều hành window, các bạn cài đặt qua lệnh sau:

```
1 py -m pip install --user pipx
```

Tiếp theo chúng ta sẽ cài đặt poetry với lệnh sau:

```
1 pipx install poetry
```

Sau khi cài đặt hoàn tất, ta có thể xác minh cài đặt thành công không qua lệnh sau:

```
1 poetry --version
```

Nếu bạn thấy kết quả tương tự như Poetry (version 1.8.2) thì tức là đã cài đặt thành công rồi đó.

Để cài đặt Anaconda, chúng ta có thể xem video hướng dẫn chi tiết tại đây: [40-Day Python Practice: Day 2 - Hướng dẫn tạo môi trường cho Python](#).

3 Thiết lập dự án

Để tạo một dự án mới với Poetry ta sẽ sử dụng lệnh poetry new, ví dụ chúng ta tạo dự án xây dựng package tính giai thừa có tên factorial-aivn, ta sẽ sử dụng lệnh sau:

```
1 poetry new factorial-aivn
```

Sau khi thực thi lệnh trên, ta sẽ nhận được một thư mục dự án có tên là factorial-aivn với cấu trúc sau:

```
1 |     pyproject.toml
2 |     README.md
3 |
4 |---factorial_aivn
5 |     __init__.py
6 |
7 |---tests
8 |     __init__.py
```

Trong thư mục factorial-aivn chứa các tệp và package được sử dụng với các mục đích khác nhau:

- factorial_aivn : Ta sẽ viết code để tạo package ở trong thư mục này
- tests : Ta sẽ viết code kiểm thử ở đây
- pyproject.toml : Là file chứa thông tin cấu hình package chúng ta đang xây dựng
- README.md : Là file mô tả package mà chúng ta đang xây dựng, hướng dẫn cài đặt... được viết bằng Markdown

Đó chính là cấu trúc tiêu chuẩn của một dự án xây dựng package với Poetry. Nếu ta có một dự án từ trước mà muốn đóng gói lại với Poetry thì cần phải tổ chức lại code theo cấu trúc này.

Vì file pyproject.toml chứa các thông tin quan trọng nên chúng ta sẽ cùng tìm hiểu nội dung của file này:

```
1 [tool.poetry]
2 name = "factorial-aivn"
3 version = "0.1.0"
4 description = ""
5 authors = ["NguyenDinhTiem <nguyendinhtiem1999@gmail.com>"]
6 readme = "README.md"
7 packages = [{include = "factorial_aivn"}]
8
9 [tool.poetry.dependencies]
10 python = "^3.12"
11
12
13 [build-system]
14 requires = ["poetry-core"]
15 build-backend = "poetry.core.masonry.api"
```

Phần đầu tiên trong file là [tool.poetry], trong phần này chứa thông tin:

- name : Tên của thư mục dự án
- version : Phiên bản mà chúng ta đang phát triển
- description : Mô tả ngắn về package
- authors : Thông tin về tác giả, nhóm phát triển package
- readme : Tệp để viết các mô tả, hướng dẫn chi tiết về package

- packages : Vị trí, tên của package, nơi mà chứa các file code của package.

Lưu ý: `packages = [{include = "factorial_aivn"}]` đây là thông tin vị trí của package, theo mặc định thì poetry sẽ lấy tên theo tool.poetry.name nhưng trong trường hợp dự án của chúng ta có tên thư mục dự án và tên package khác nhau, nên chúng ta cần chỉ định lại tên của package. Nếu không sẽ bị lỗi khi build package. Vậy nên lần tới phát triển các package khác thì nên đặt tên dự án đơn giản, không nên dùng dấu gạch "-" mặc dù trong tài liệu gốc của Poetry có sử dụng cách này.

Tiếp theo là phần [tool.poetry.dependencies] chứa thông tin về phiên bản python và các package, library tương thích với package chúng ta xây dựng. Ví dụ package chúng ta đang xây dựng cần sử dụng python, streamlit thì chúng sẽ được liệt kê tại đây.

Phần cuối của file là [build-system], phần này chứa một số thông tin về các phụ thuộc để thực hiện đóng gói package. Ta sẽ để các giá trị theo mặc định.

Theo mặc định phiên bản python Poetry sẽ sử dụng để tạo môi trường cho dự án là phiên bản python trên môi trường chung của máy tính. Ta có thể xem thông tin môi trường bằng cách điều hướng đến thư mục dự án vừa tạo và dùng lệnh sau:

```

1 ====== Input ======
2 cd factorial-aivn
3 poetry env info
4
5
6
7
8
9
10
11
12
13
14 ====== Output ======
Virtualenv
Python:      3.12.2
Implementation: CPython
Path:        NA
Executable:   NA

Base
Platform:    win32
OS:          nt
Python:      3.12.2
Path:        C:\Python312
Executable:  C:\Python312\python.exe
=====
```

Lúc này môi trường chưa được tạo, nhưng theo thiết lập mặc định nó đã chỉ định phiên bản python sử dụng là 3.12. Tuy nhiên thì chúng ta có thể chọn một phiên bản python cụ thể cho dự án bằng cách điều hướng đến vị trí dự án và sử dụng lệnh:

```
1 poetry env use /full/path/to/python
```

Trong đó /full/path/to/python là đường dẫn đến tệp thực thi python.exe của phiên bản python ta muốn sử dụng. Để có được file này, chúng ta cần cài đặt các phiên bản python khác nhau và có rất nhiều cách để thực hiện điều này. Tuy nhiên, chúng ta sẽ sử dụng Anaconda environment vì nó quá quen thuộc và dễ sử dụng.

Ta tạo môi trường trong Anaconda có tên là factorial_aivn_env với phiên bản python 3.10 với cú pháp dưới đây:

```
1 conda create -n factorial_aivn_env python=3.10
```

Tiếp theo chúng ta sẽ kích hoạt môi trường factorial_aivn_env vừa tạo bằng lệnh sau:

```
1 conda activate factorial_aivn_env
```

Để lấy đường dẫn của tệp thực thi python.exe trên window, tại môi trường đang được kích hoạt ta dùng lệnh:

```

1 ====== Input ======
2 where python
3
4
5
6
7
8
9
10
11
12
13
14 ====== Output ======
C:\Users\Tiem\anaconda3\envs\
    factorial_aivn_env\python.exe
C:\Python312\python.exe
C:\Users\Tiem\.pyenv\pyenv-win\shims\
    python
C:\Users\Tiem\.pyenv\pyenv-win\shims\
    python.bat
C:\Users\Tiem\anaconda3\python.exe
C:\Users\Tiem\AppData\Local\Programs\
    Python\Python311\python.exe
C:\Users\Tiem\AppData\Local\Microsoft\
    WindowsApps\python.exe
=====
```

Ta sẽ lấy đường dẫn đầu tiên có chứa tên môi trường, sau đó hãy thoát ra khỏi môi trường Anaconda hiện tại bằng cách sử dụng lệnh

```
1 conda deactivate
```

Tiếp theo ta sử dụng lệnh để tạo một môi trường mới với phiên bản python chúng ta đã tạo ra từ môi trường conda. Nhưng trước khi chạy lệnh này, ta cần sửa lại phiên bản python 3.12 trong file pyproject.toml thành 3.10.

```

1 ====== Input ======
2 poetry env use C:\Users\Tiem\anaconda3\
    envs\factorial_aivn_env\python.exe
3
4
5
6
7
8
9
10
11
12
13
14
15 ====== Output ======
Creating virtualenv factorial-aivn-
U849jFk2-py3.10 in C:\Users\Tiem\
AppData\Local\pypoetry\Cache\
virtualenvs
Using virtualenv: C:\Users\Tiem\AppData\
Local\pypoetry\Cache\virtualenvs\
factorial-aivn-U849jFk2-py3.10
=====
```

Sau khi thực hiện lệnh trên thì một môi trường có tên factorial-aivn-U849jFk2-py3.10 được tạo ra. Chúng ta có thể kích hoạt môi trường trong Poetry bằng lệnh sau:

```
1 poetry shell
```

Môi trường này hoạt động tương tự như môi trường trong anaconda. Nhưng điều thú vị là không như Anaconda có thể kích hoạt môi trường ở mọi nơi, môi trường trong Poetry chỉ có thể kích hoạt tại trong chính vị trí thư mục dự án.

Để cài đặt một package hay library cho dự án, ta sẽ sử dụng lệnh sau

```
1 poetry add name_package
```

Trong ví dụ này chúng ta sẽ cài đặt thư viện streamlit để tạo giao diện, để cài đặt chúng ta sử dụng lệnh:

```
1 poetry add streamlit
```

Khi chạy lệnh này, poetry sẽ tự động thêm tên package vừa cài đặt vào phần [tool.poetry.dependencies] trong file pyproject.toml. Ta cũng có thể cài nhiều package bằng cách thêm nó vào phần [tool.poetry.dependencies] sau đó sử dụng lệnh sau để cài đặt:

```
1 poetry install
```

4 Xây dựng và public package

4.1 Xây dựng package

Chúng ta sẽ xây dựng package tính gai thừa, ngoài ra ta sẽ xây dựng giao diện với streamlit nữa. Trong thư mục factorial_aivn chúng ta sẽ tạo file factorial.py, sau đó viết hàm tính gai thừa như sau.

```
1 def fact(n):
2     """
3         Tính gai thừa của một số nguyên dương n.
4
5     Tham số:
6     - n: Số nguyên dương.
7
8     Trả về:
9     - Gai thừa của n.
10    """
11    if n < 0:
12        raise ValueError("Factorial is not defined for negative numbers")
13    elif n == 0 or n == 1:
14        return 1
15    else:
16        return n * fact(n-1)
```

Ở đây chúng ta hiểu là mỗi file trong thư mục package được gọi là module nên ta có thể gọi nó trong các module khác. Ta sẽ tạo một file có tên là app.py để viết giao diện tính gai thừa với streamlit.

```
1 import streamlit as st
2 from factorial_aivn.factorial import fact
3
4 def main():
5     st.title("Factorial Calculator")
6     number = st.number_input("Enter a number:",
7                             min_value=0,
8                             max_value= 900)
9     if st.button("Calculate"):
10         result = fact(number)
11         st.write(f"The factorial of {number} is {result}")
12         st.balloons()
13
14 if __name__ == "__main__":
15     main()
```

Và chúng ta hoàn toàn có thể chạy code trong package này bằng cách chạy lệnh sau.

```
1 streamlit run factorial_aivn/app.py
```

Lưu ý: Có thể xuất hiện lỗi tại from factorial_aivn.factorial import fact, khi chạy chương trình tại máy cá nhân thì ta chỉ cần dùng from factorial import fact. Nhưng khi build package thì nhất định phải dùng đường dẫn đầy đủ là from factorial_aivn.factorial import fact

4.2 Xây dựng package kiểm thử

Kiểm thử là một phần quan trọng trong việc phát triển dự án, trong phần này chúng ta sẽ thực hiện kiểm thử package qua thư viện Pytest. Trước tiên chúng ta phải cài đặt pytest với lệnh sau:

```
1 poetry add pytest --group test
```

Trong package tests ta tạo file factorial_test.py với nội dung sau:

```
1 import pytest
2 from factorial_aivn.factorial import fact
3
4 def test_factorial_of_0():
5     assert fact(0) == 1
6 def test_factorial_of_1():
7     assert fact(1) == 1
8 def test_factorial_of_5():
9     assert fact(5) == 120
10 def test_factorial_of_10():
11    assert fact(10) == 3628800
12 def test_factorial_of_negative_number():
13    with pytest.raises(ValueError):
14        fact(-5)
15
16 if __name__ == "__main__":
17     pytest.main()
```

Mỗi hàm trên là một test case, nhằm mục đích kiểm tra xem hàm factorial trong package có hoạt động bình thường không. Chúng ta sử dụng câu lệnh assert để kiểm tra xem kết quả trả về từ hàm factorial có đúng như mong đợi không. Hàm test_factorial_of_negative_number kiểm tra xem hàm factorial có trả về một ngoại lệ ValueError khi được gọi với một số âm không. Cuối cùng, chúng ta gọi pytest.main() để chạy các testcase khi file được chạy như một chương trình độc lập.

Để chạy kiểm thử, chúng ta sử dụng lệnh sau:

```
1 poetry run pytest -v
```

Chúng ta sẽ nhận được thông tin sau khi thực hiện kiểm thử đã pass 100%, thật tuyệt vời:

```
1 ===== test session starts =====
2 platform win32 -- Python 3.10.14, pytest-8.1.1, pluggy-1.4.0 -- C:\Users\Tiem\AppData\Local\pypoetry\Cache\virtualenvs\factorial-aivn-U849jFk2-py3.10\Scripts\python.exe
3 cachedir: .pytest_cache
4 rootdir: D:\AIVIETNAM\Asisstant\Thang3\Day28\factorial-aivn
5 configfile: pyproject.toml
6 collected 5 items
7
8 tests/factorial_test.py::test_factorial_of_0 PASSED [ 20%]
9 tests/factorial_test.py::test_factorial_of_1 PASSED [ 40%]
10 tests/factorial_test.py::test_factorial_of_5 PASSED [ 60%]
11 tests/factorial_test.py::test_factorial_of_10 PASSED [ 80%]
12 tests/factorial_test.py::test_factorial_of_negative_number PASSED [100%]
13
14 ===== 5 passed in 0.02s =====
```

Lưu ý: Với test case tính giải thừa cho số lớn 1000 sẽ bị lỗi, do đây là hướng dẫn đơn giản nhằm mục đích chính là xây dựng package nên không quá khắt khe về việc bắt lỗi. Bạn đọc có thể cải tiến thuật toán và thêm nhiều test case khác.

4.3 Đóng gói và Xuất bản package

4.3.1 Đóng gói

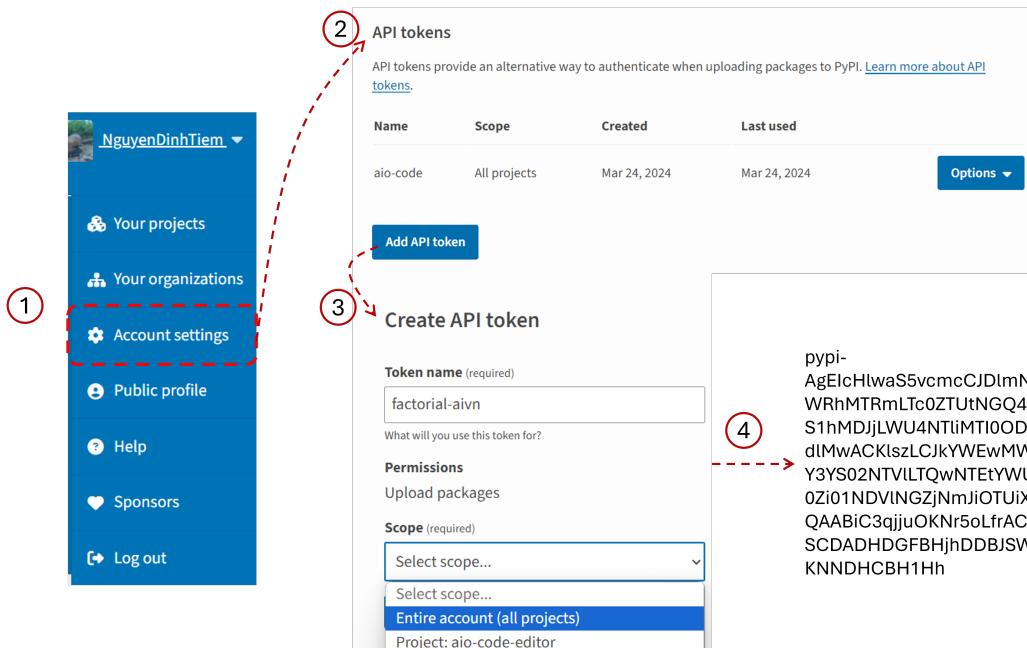
Để đóng gói package để cung cấp cho các nhóm phát triển khác trong dự án hoặc để đăng tải trực tuyến cho cộng đồng python có thể sử dụng. Trong Poetry chúng ta có thể thực hiện điều này rất dễ dàng. Đầu tiên chúng ta đóng gói lại dự án bằng lệnh sau:

```
1 ====== Input ======
2 poetry build
3
4
5
6
7 ====== Output ======
Building factorial-aivn (0.1.0)
- Building sdist
- Built factorial_aivn-0.1.0.tar.gz
- Building wheel
- Built factorial_aivn-0.1.0-py3-none-
any.whl
=====
```

Việc đóng gói này sẽ sử dụng các cấu hình mà chúng ta thiết lập trong file pyproject.toml. Vậy là quá trình đóng gói package của chúng ta đã thành công.

4.3.2 Xuất bản package

Tiếp theo, để đăng tải lên PyPI, ta sẽ truy cập vào trang web pypi.org và tiến hành đăng nhập, nếu chưa có tài khoản chúng ta sẽ cần phải đăng ký một tài khoản mới.



Hình 2: Tạo PyPI API token

Sau khi đăng nhập thành công, ta sẽ tạo một API token tại phần Account settings (1) hoặc tại link <https://pypi.org/manage/account/token/>. Sau đó tại phần API tokens chọn Add API tokens(2) phần

Scope chúng ta chọn Entire account (all projects) như hình trên(3).

Lưu ý: Trước khi tạo token PyPI sẽ yêu cầu chúng ta xác thực Two factor authentication (2FA). Chúng ta cần làm theo thông báo đó để bật xác thực 2FA, sau đó mới có thể tạo token được.

(4) Cuối cùng chúng ta sẽ nhấn vào button create token, một đoạn mã chứa thông tin về token xuất hiện, ta cần copy lại và lưu vào một tệp nào đó trên máy tính của chúng ta để có thể xem lại khi cần. Vì đoạn mã này chỉ xuất hiện một lần duy nhất khi chúng ta tạo token trên PyPI.

Tiếp theo ta sẽ thêm tài khoản PyPI vào Poetry với lệnh sau:

```
1 poetry config http-basic.foo <username> <password>
```

Trong đó username là tên đăng nhập, password là mật khẩu của tài khoản PyPI mà chúng ta đã tạo. Tiếp theo chúng ta thêm thông tin token của tài khoản bằng lệnh:

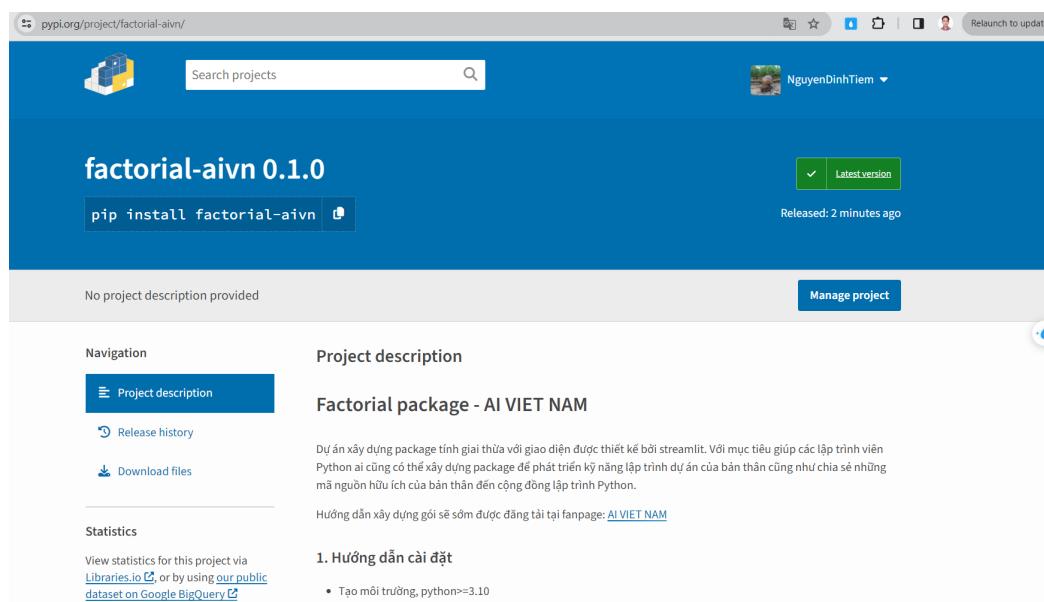
```
1 poetry config pypi-token.pypi <my-token>
```

Trong đó my-token là token ta đã tạo ở phía trên.

Cuối cùng chúng ta đẩy package của chúng ta lên nền tảng PyPI bằng lệnh:

<pre>1 ====== Input ====== 2 poetry publish 3 4 5 6 7 ======</pre>	<pre>===== Output ===== Publishing factorial-aivn (0.1.0) to PyPI - Uploading factorial_aivn-0.1.0-py3-none -any.whl 100% - Uploading factorial_aivn-0.1.0.tar.gz 100% =====</pre>
--	--

Vậy là package của chúng ta đã được đăng tải lên PyPI, ngay lúc này cả cộng đồng Python có thể sử dụng package của chúng ta thông qua cú pháp pip install, thật là tuyệt vời ông mặt trời phải không?



Hình 3: Package được đăng tải lên PyPI

Basic Python - Data Analysis with Visualization

Hoàng-Nguyễn Vũ

1. Mô tả: Làm quen với thư viện PygWalker



- **Thư viện PygWalker** là một thư viện Python mã nguồn mở giúp bạn dễ dàng chuyển đổi dữ liệu thành các ứng dụng phân tích trực quan. Thư viện này cung cấp một bộ công cụ mạnh mẽ để khám phá, tóm tắt và trực quan hóa dữ liệu của bạn, giúp bạn hiểu rõ hơn về dữ liệu và đưa ra quyết định sáng suốt hơn.
- **Điểm nổi bật của PygWalker:**
 - + **Tạo bảng điều khiển tương tác:** PygWalker cho phép bạn tạo các bảng điều khiển trực quan và dễ sử dụng để khám phá dữ liệu của bạn. Bạn có thể dễ dàng thêm và loại bỏ các biểu đồ, thay đổi bộ lọc và tương tác với dữ liệu theo thời gian thực.
 - + **Hỗ trợ nhiều loại biểu đồ:** PygWalker cung cấp nhiều loại biểu đồ khác nhau để trực quan hóa dữ liệu của bạn, bao gồm biểu đồ thanh, biểu đồ đường, biểu đồ phân tán, biểu đồ nhiệt, v.v.
 - + **Khả năng lọc và nhóm dữ liệu:** PygWalker cho phép bạn lọc dữ liệu theo các tiêu chí cụ thể và nhóm dữ liệu theo các trường khác nhau.
 - + **Tích hợp với Jupyter Notebook:** PygWalker có thể được sử dụng trong Jupyter Notebook, cho phép bạn kết hợp phân tích dữ liệu với mã Python khác.
 - + **Dễ sử dụng:** PygWalker có API đơn giản và dễ sử dụng, giúp bạn dễ dàng bắt đầu.
- **Ứng dụng của PygWalker trong việc trực quan hóa dữ liệu:**
 - + **Khoa học dữ liệu:** PygWalker có thể được sử dụng để khám phá và phân tích dữ liệu trong khoa học dữ liệu.
 - + **Học máy:** PygWalker có thể được sử dụng để chuẩn bị dữ liệu và đánh giá mô hình học máy.
 - + **Tài chính:** PygWalker có thể được sử dụng để phân tích dữ liệu tài chính và thị trường chứng khoán.
 - + **Tiếp thị:** PygWalker có thể được sử dụng để phân tích dữ liệu khách hàng và chiến dịch tiếp thị.

2. Cách cài đặt và sử dụng một số tính năng:

Để cài đặt thư viện PygWalker, chúng ta có thể cài trên Google Colab, hoặc ở máy cá nhân thông qua Jupyter Notebook. Cách cài đặt như sau:

1. Cài đặt thư viện PygWalker:

+ Để cài đặt thư PygWalker, chúng ta sử dụng câu lệnh sau ở Google Colab:

```
1 !pip install pygwalker
```

+ Để cài thư viện trên máy cá nhân và chạy với Jupyter Notebook, chúng ta sẽ chạy thông qua Terminal đối với hệ điều hành MacOS và CMD đối với hệ điều hành Windows thông qua lệnh sau:

```
1 pip install pandas
2 pip install pygwalker
```

```
[nguyen@Nguyen ~ % pip install pygwalker
Collecting pygwalker
  Downloading pygwalker-0.4.7-py3-none-any.whl.metadata (19 kB)
Collecting appdirs (from pygwalker)
  Downloading appdirs-1.4.4-py2.py3-none-any.whl.metadata (9.0 kB)
Requirement already satisfied: arrow in /Library/Frameworks/Python.f...
```

Hình 1: Cài đặt PygWalker

+ Sau khi cài xong chúng ta khởi động jupyter notebook tại thư mục chứa project của chúng ta để sử dụng, thông qua lệnh sau:

```
1 jupyter notebook
```

```
nguyen@Nguyen Demo PygWalker % jupyter notebook
[I 2024-03-25 21:18:57.415 ServerApp] Package notebook took
[I 2024-03-25 21:18:57.457 ServerApp] Package jupyter_lsp t
[W 2024-03-25 21:18:57.457 ServerApp] A `__jupyter_server_ex
ion was not found in jupyter_lsp. Instead, a `__jupyter_serv
unction was found and will be used for now. This function n
d in future releases of Jupyter Server.
```

Hình 2: Khởi chạy Jupyter Notebook

+ Để khởi động thư viện PygWalker trong Colab/Jupyter Notebook, trước tiên ta cần phải có dataset để thư viện có thể trực quan hóa dữ liệu. Sau khi chúng ta đã có data, để khởi tạo thư viện PygWalker như sau:

```
1 import pygwalker as pyg
2 import pandas as pd
3 # FILE_PATH: đường dẫn tới tập tin CSV
4 data = pd.read_csv(FILE_PATH)
5 pyg.walk(data)
```

Showing 1 to 100 of 199 results

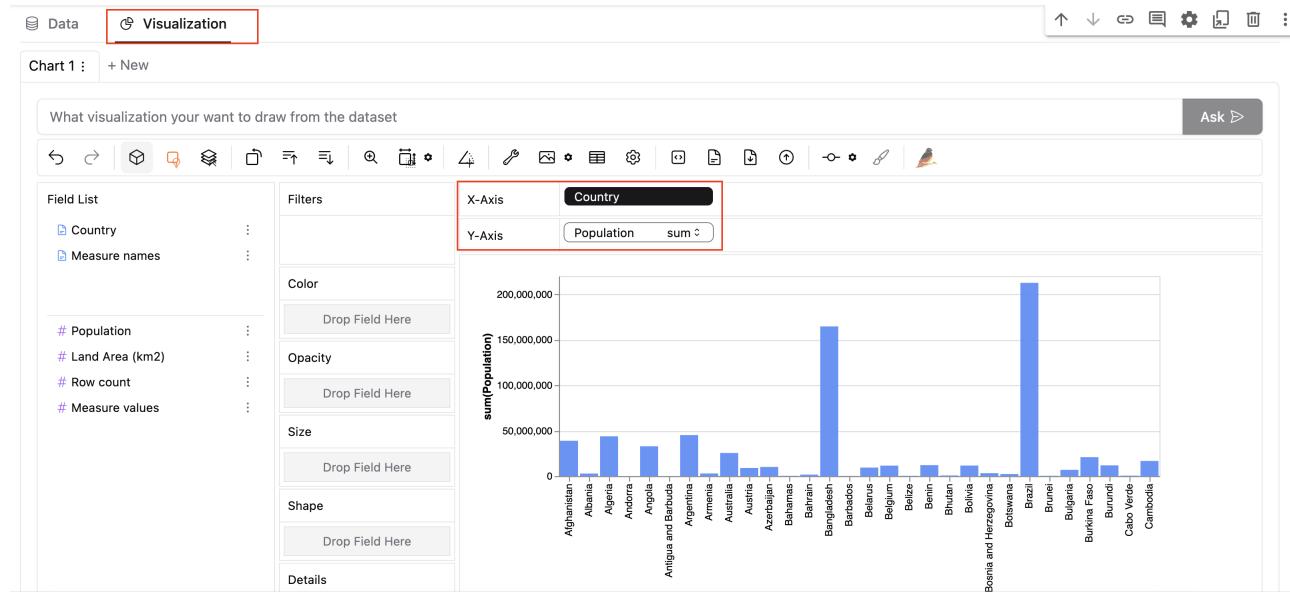
Country	Population	Land Area (km2)
199 unique values	199 unique values	700 1% 460 1% Other (195) 98%
Brunei	212,559,417	5,330,140
Brunei	437,479	5,270
Bulgaria	6,948,445	108,560
Burkina Faso	20,903,273	273,600
Burundi	11,890,784	25,680
Cabo Verde	555,987	4,030
Cambodia	16,718,965	176,520
Cameroon	26,545,863	472,710
Canada	37,742,154	9,093,510
Central African Republic	4,829,767	622,980
Chad	16,425,864	1,259,200
Chile	19,116,201	743,532
China	1,439,323,776	9,388,211
Colombia	50,882,891	1,109,500

Hình 3: Giao diện PygWalker

2. Sử dụng một số tính năng trực quan hóa:

- + **Tạo biểu đồ cột:** Tạo biểu đồ cột cho tập data mẫu trên để thể hiện dân số (Population) theo quốc gia (Country).

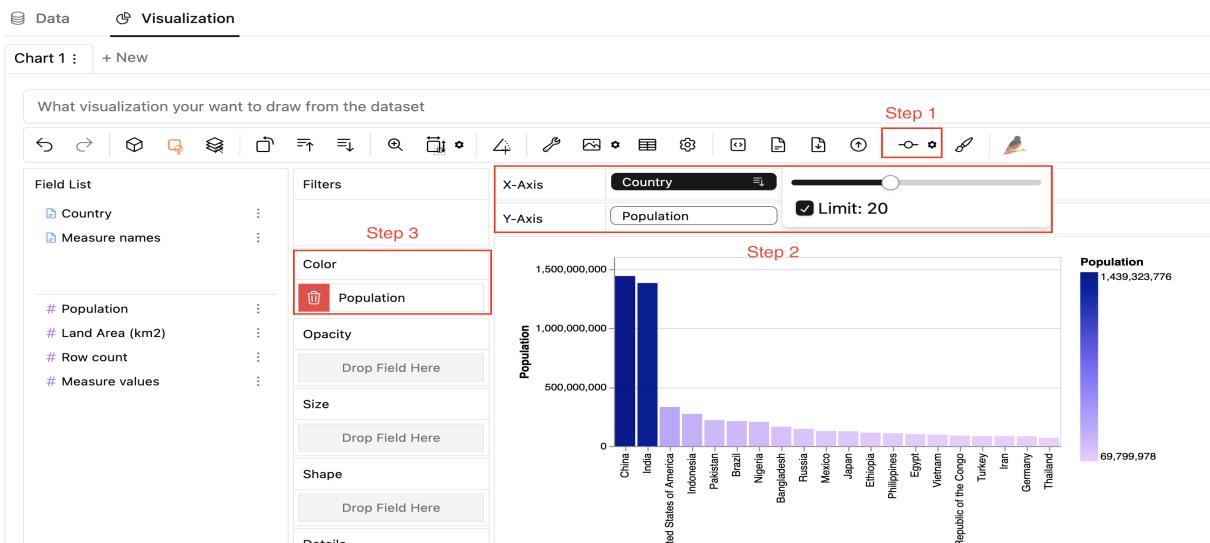
(*) Ta sẽ thực hiện kéo 2 cột: Country vào X-Axis và Population vào Y-Axis. Ứng với 2 thông số của biểu đồ cột mà chúng ta cần thực hiện trực quan hóa biểu đồ: Trục Ox (Trục ngang) thể hiện cho Quốc gia (Country) và Trục Oy (Trục dọc) thể hiện cho Dân số (Population)



Hình 4: Biểu đồ cột thể hiện dân số theo quốc gia

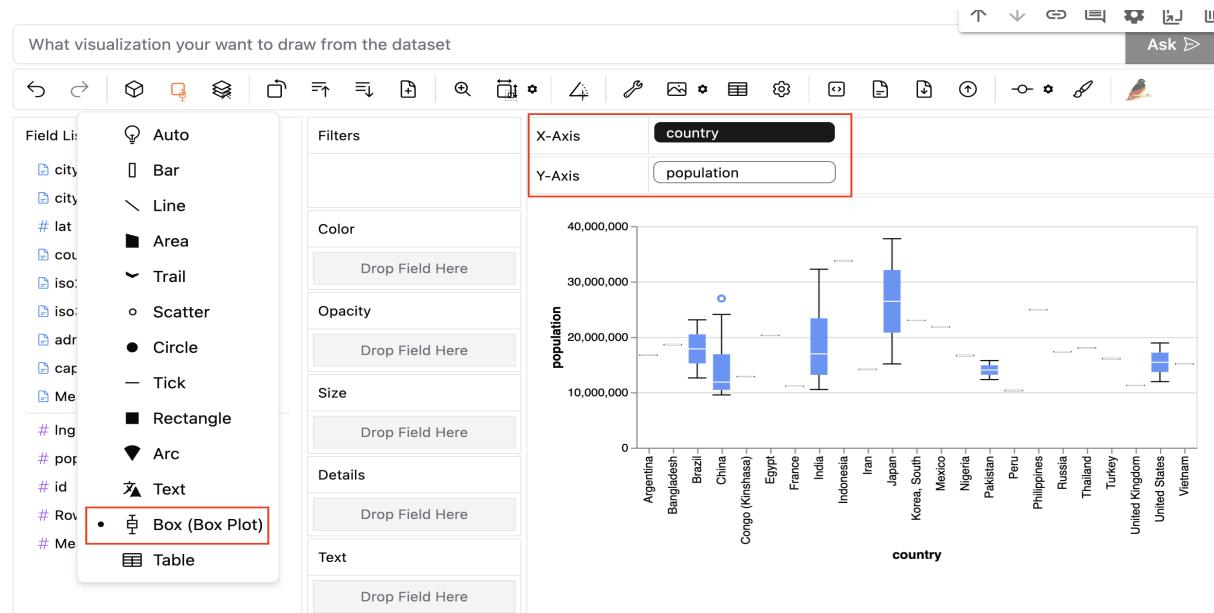
- + Lấy Top 20 Quốc Gia có giảm dần theo dân số, và tô màu theo độ lớn của dân số:

(*) Ta sẽ thực hiện kéo 2 cột: Chúng ta cũng thực hiện tương tự bài trên nhưng chúng sẽ thực hiện sắp xếp Population giảm dần và Limit 20 dòng. Đồng thời gắn Color là cột Population để thư viện hiện hiện tô màu theo Population.



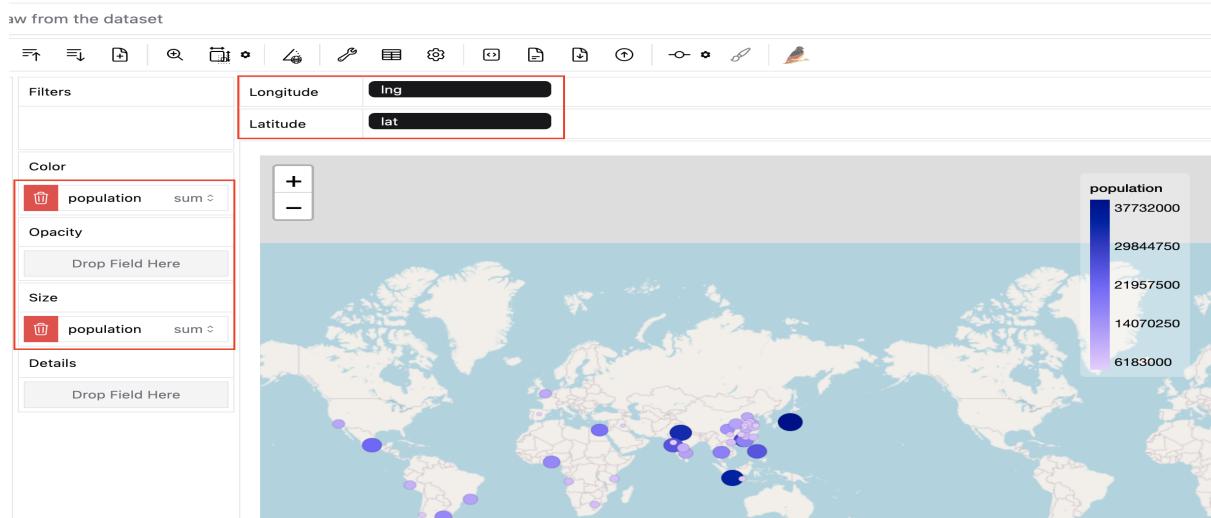
Hình 5: Biểu đồ cột thể hiện dân số theo quốc gia

- + Vẽ biểu đồ hộp thể hiện phân phối dữ liệu:



Hình 6: Biểu đồ hộp thể hiện dân số theo quốc gia

+ Vẽ bản đồ hộp thể hiện phân phối dữ liệu:

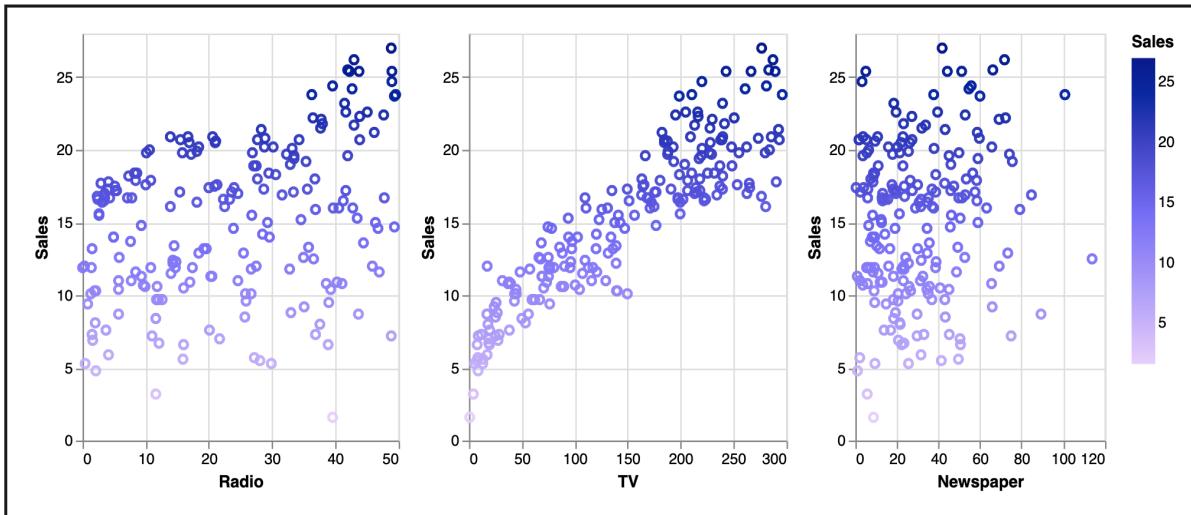


Hình 7: Bản đồ thể hiện dân số theo quốc gia

3. **Bài tập:** Hãy đọc dữ liệu ở file: advertising.csv và khởi chạy thư viện PygWalker sau đó thực hiện trực quan các biểu đồ sau đây:

- **Câu 1:** Vẽ biểu đồ phân phối dữ liệu cho 3 loại: TV, Radio, Paper và được Color theo độ lớn của giá bán (Sales).

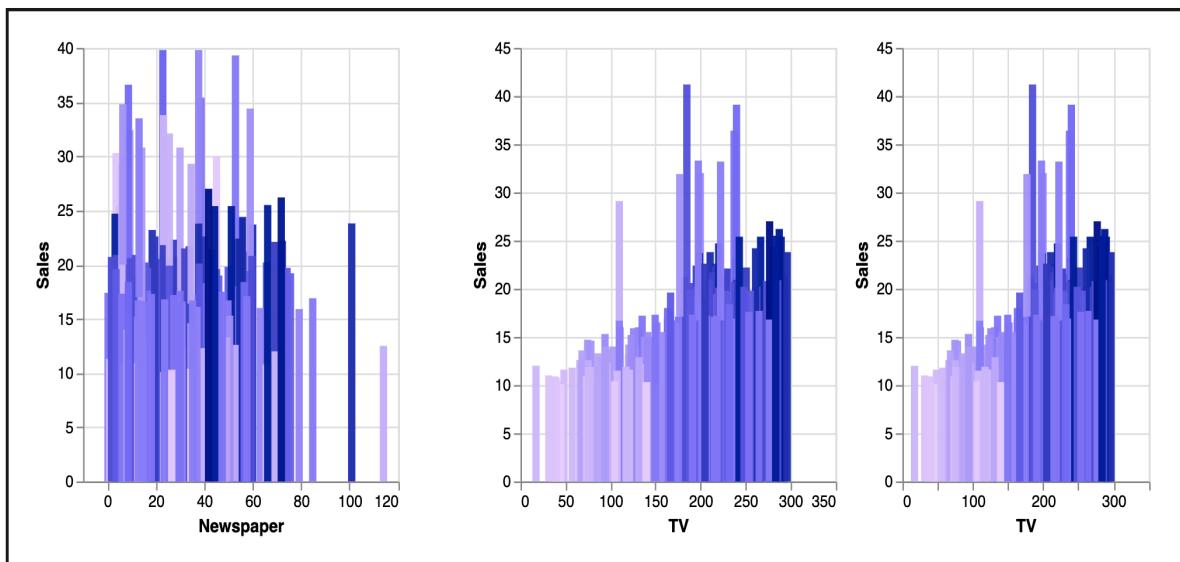
Kết Quả:



Hình 8: Biểu đồ phân phối dữ liệu cho 3 loại: TV, Radio, Paper

- **Câu 2:** Vẽ biểu đồ cột thể hiện doanh số bán (Sales) ≥ 10 của cả 3 loại TV, Radio và Newspaper.

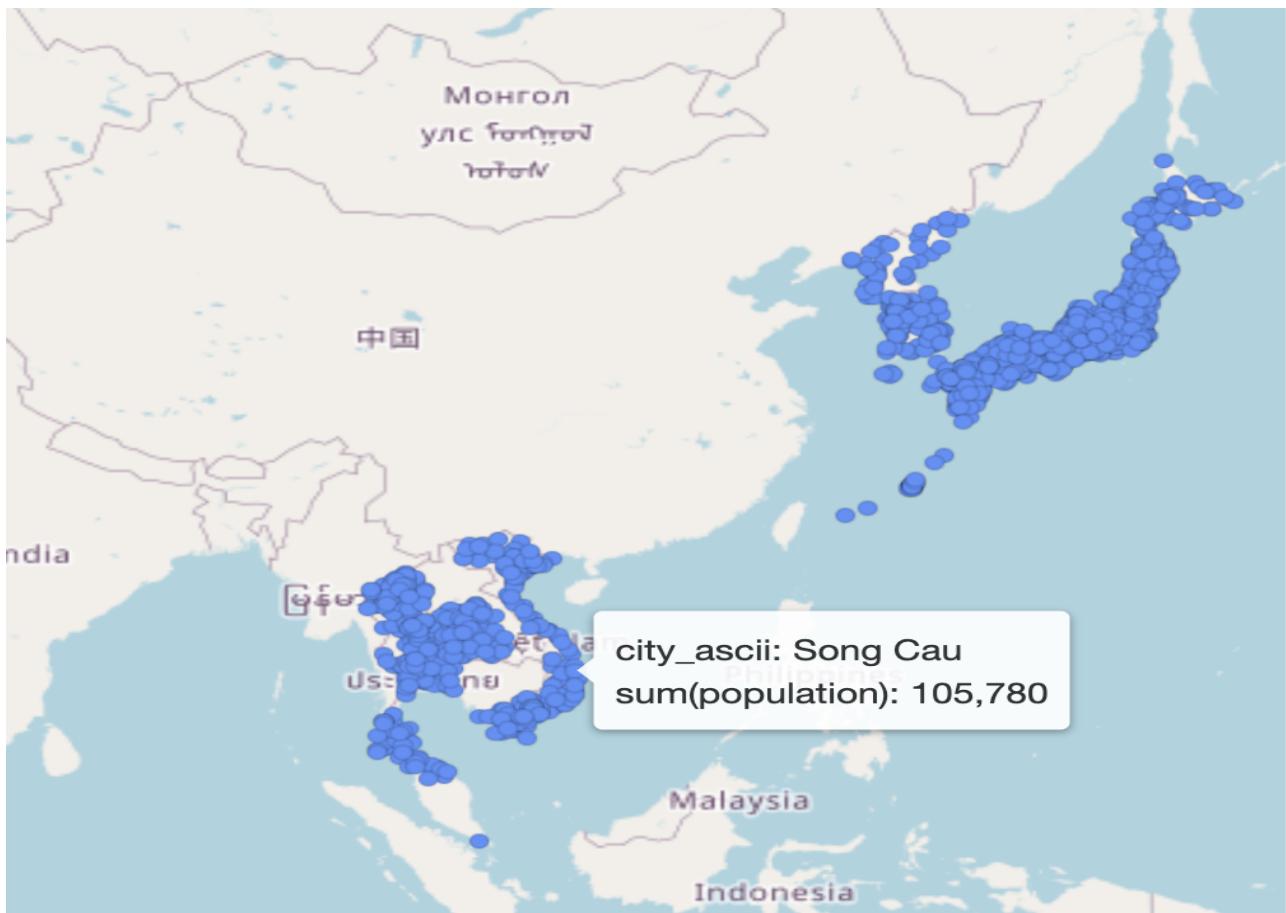
Kết quả:



Hình 9: Biểu đồ doanh số bán hàng trên 10 sản phẩm cho 3 loại: TV, Radio, Paper

- **Câu 3:** Hãy vẽ bản đồ biểu diễn phân bố dân số theo thành phố của các nước thuộc các nước: Việt Nam, Hàn Quốc, Nhật Bản, Singapore và Thái Lan dựa theo dữ liệu sau: Dữ liệu dân số thế giới

Kết quả:



Hình 10: Bản đồ thể hiện phân bố dân số theo thành phố của các nước thuộc các nước: Việt Nam, Hàn Quốc, Nhật Bản, Singapore và Thái Lan

- Hết -

Python cơ bản - Bài luyện tập

Dinh-Tiem Nguyen và Quang-Vinh Dinh

Question	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Answer														
Question	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Answer														
Question	29	30	31	32	33									
Answer														

Question 1: Which is the output of the following program?

```
1 n = 1
2 for i in range(0, 500, 100):
3     n = i
4 print(n)
```

- a) 900
- b) 400
- c) None
- d) Raise an Error

Question 2: Which is the output of the following program?

```
1 data = "I'm learning Python!"
2 print(data.split()[1])
```

- a) Python
- b) learning
- c) I'm
- d) Raise an Error

Question 3: Which is the output of the following program?

```
1 def check_the_number(N):
2     list_of_numbers = []
3     result = ""
4     for i in range(1, 5):
5         list_of_numbers.append(i)
6     if N in list_of_numbers:
7         results = "True"
8     if N not in list_of_numbers:
9         results = "False"
10    return results
11
12 N = 2
13 results = check_the_number(N)
14 print(results)
```

- a) True
- b) False
- c) None
- d) Raise an Error

Question 4: Which is the output of the following program?

```
1 data = "my list : [ 1, 2, [3, 4]]"
2 print(type(data))
```

- a) <class 'str'>
- b) <class 'list'>
- c) <class 'mydata'>
- d) Raise an Error

Question 5: Which is the output of the following program?

```
1 "Eiffel" > "Apple"
```

- a) True
- b) False
- c) Raise an Error
- d) None

Question 6: Which is the output of the following program?

```
1 n = 0
2 for i in range(5):
3     n += i
4     if n>0 and n%3 == 0:
5         break
6 print(n)
```

- a) 0
- b) 3
- c) Raise an Error
- d) 6

Question 7: Which is the output of the following program?

```
1 space1 = "Cherry Blossom After Winter"
2 space2 = "Flowers are blooming on the hillsides, which signals the coming of spring"
3 space = space1 + space2
4 print(space[-6:])
```

- a) spring
- b) Raise an Error
- c) Winter
- d) A and C

Question 8: Which is the output of the following program?

```
1 my_list = [0, 1, 1, 2, 1]
2 odd = 1
3 even = 0
4
5 for number in my_list:
6     if number % 2 == 0:
7         odd += number
8     else:
9         even += number
10 print(f"odd:{odd}, even:{even}")
```

- a) odd:1, even:0
- b) odd:3, even:10
- c) odd:3, even:3
- d) Raise an Error

Question 9: Which is the output of the following program?

```

1 weather = '@the drizzle in spring makes the air more humid. @@'
2 me = "i love iT!"
3
4 txt = weather.strip('@').capitalize() + me.title()
5 print(txt)

```

- a) The drizzle in spring makes the air more humid. I Love It!
- b) The drizzle in spring makes the air more humid. i love it!
- c) the drizzle in spring makes the air more humid. @@ I Love It!
- d) Raise an Error

Question 10: Which is the output of the following program?

```

1 data = [1, 9, 3, -3]
2 data.sort()
3 print(data)

```

- a) [-3, 1, 3, 9]
- b) [1, 2, 3, 4]
- c) [9, 3, 1, -3]
- d) Raise an Error

Question 11: Which is the output of the following program?

```

1 my_exam = "Under the drizzle, the flower field seems covering with the glitter water
           drops!"
2 print(my_exam.strip('!').split())

```

- a) ['Under', 'the', 'drizzle', 'the', 'flower', 'field', 'seems', 'covering', 'with', 'the', 'glitter', 'water', 'drops']
- b) Raise an Error
- c) ['Under', 'the', 'drizzle', 'the', 'flower', 'field', 'seems', 'covering', 'with', 'the', 'glitter', 'water', 'drops', '!']
- d) a, c

Question 12: Which is the output of the following program?

```

1 my_string = "Peach blossoms bloom in spring"
2
3 my_bag_of_word = []
4 for element in my_string:
5     if element not in my_bag_of_word:
6         my_bag_of_word.append(element)
7 print(my_bag_of_word)

```

- a) ['P', 'e', 'a', 'c', 'h']
- b) ['']
- c) Raise a Error
- d) ['P', 'e', 'a', 'c', 'h', ' ', 'b', 'l', 'o', 's', 'm', 'i', 'n', 'p', 'r', 'g']

Question 13: Which is the output of the following program?

```

1 my_string = "My name is Tom"
2 results = my_string.count('m')
3
4 print(results)

```

- a) Raise an Error
- b) 2
- c) 3
- d) 4

Question 14: Which is the output of the following program?

```

1 even_numbers = [x for x in range(1, 5) if x % 2==0]
2 print(even_numbers)

```

- a) Raise an Error
- b) [1, 2, 3, 4, 5]
- c) [2, 4]
- d) [3, 5]

Question 15: Which is the output of the following program?

```

1 X = [[1, 1],
2     [2, 2]]
3
4 result = [[0,0],
5           [0,0]]
6
7 for i in range(len(X)):
8     for j in range(len(X[0])):
9         result[j][i] = X[i][j]
10
11 for r in result:
12     print(r)

```

- a)
[1, 2]
[1, 2]
- b) [1, 2, 3] [1, 2, 3]
- c) [1, 2, 1, 2]
- d) Raise an Error

Question 16: Which is the output of the following program?

```

1 def my_function(my_data):
2     rs = 0
3     for i in my_data:
4         rs = rs + i
5     return rs
6
7 my_list = [1, 2, 3]
8 print(my_function(my_data = my_list))

```

- a) 5
- b) 6
- c) 8
- d) 9

Question 17: Which is the output of the following program?

```

1 def my_function(my_data):
2     result = []
3     for element in data:
4         if element not in result:
5             result.append(element)
6     return result
7
8 data = [ {'id': 'M12'}, 10, 20, 30]
9 print(my_function(data))

```

- a) [{‘id’: ‘M12’}, 10, 20, 30]
- b) [‘id’, ‘M12’, 10, 20, 30]
- c) [10, 20, 30]
- d) Raise an Error

Question 18: Which is the output of the following program?

```

1 def my_function(data, max, min):
2     result = []
3     for i in data:
4         if i < min:
5             result.append(min)
6         elif i > max:
7             result.append(max)
8         else:
9             result.append(i)
10    return result
11
12 my_list = [10, 2, 5, 0, 1]
13 max = 2
14 min = 1
15 print(my_function(max = max, min = min, data = my_list))

```

- a) [10, 2, 5, 1, 1]
- b) [0, 2, 2, 0, 0]
- c) [2, 2, 2, 1, 1]
- d) Raise an Error

Question 19: Which is the output of the following program?

```

1 def my_function(x, y):
2     x.extend(y)
3     return x
4
5 list_num1 = [1, 2]
6 list_num2 = [3, 4]
7 list_num3 = [0, 0]
8
9 my_function(list_num1, my_function(list_num2, list_num3))

```

- a) [1, 2, 3, 4, 0, 0]
- b) [1, 2, [3, 4, 0, 0]]
- c) [[1, 2, 3, 4, 0, 0]]
- d) Raise an Error

Question 20: Which is the output of the following program?

```

1 def my_function(n):
2     x = n[0]
3     for i in n:
4         if i < x:
5             x = i
6     return x
7
8 my_list = [1, 2, 3, -1]
9 my_function(my_list)

```

- a) None
- b) Raise an Error
- c) -1
- d) 3

Question 21: Which is the output of the following program?

```

1 def my_function(n):
2     x = n[0]
3     for i in n:
4         if i > x:
5             x = i
6     return x
7
8 my_list = [1, 9, 9, 0]
9 my_function(my_list)

```

- a) None
- b) Raise an Error
- c) 0
- d) 9

Question 22: Which is the output of the following program?

```

1 def my_function(integers, number = 1):
2     return any([True if i == number else False for i in integers])
3
4 my_list = [1, 2, 3, 4]
5 my_function(my_list, 2)

```

- a) 1
- b) 4
- c) True
- d) False

Question 23: Which is the output of the following program?

```

1 def my_function(list_nums = [0, 1, 2]):
2     var = 0
3     for i in list_nums:
4         var += i
5     return var/len(list_nums)
6
7 my_function()

```

- a) 1.0
- b) 2.0
- c) Raise an Error
- d) A and C

Question 24: Which is the output of the following program?

```

1 def my_function(signal1, signal2):
2     var = False
3     for s1 in signal1:
4         for s2 in signal2:
5             if s1 == s2:
6                 var = True
7             return var
8 print(my_function([1, 1, 1], [2, 2, 2]))

```

- a) False
- b) True
- c) Raise an Error
- d) None

Question 25: Which is the output of the following program?

```

1 def my_function(signal1, signal2):
2     var = False
3     for s1 in signal1:
4         for s2 in signal2:
5             if s1 == s2:
6                 var = True
7             return var
8 print(my_function([1, 2, 3], [2, 2]))

```

- a) False
- b) True
- c) Raise an Error
- d) None

Question 26: Which is the output of the following program?

```

1 def my_function(data):
2     var = []
3     for i in data:
4         if i%3 == 0:
5             var.append(i)
6     return var
7
8 print(my_function([1, 2, 3, 5, 6]))

```

- a) [3, 6]
- b) [1, 2, 3, 5, 6]
- c) a and d
- d) [5, 1]

Question 27: Which is the output of the following program?

```

1 def my_function(x):
2     for i in range(x):
3         for j in range(x):
4             if i == j:
5                 print("1 ", end=" ")
6             else:
7                 print("0 ", end=" ")
8             print()

```

```

9
10 my_function(2)

```

- a)
1 0
- b)
0 1
- c)
0 1
- d)
1 0
- c) [1, 0, 0, 1]
- d) None

Question 28: Which is the output of the following program?

```

1 def my_function(y):
2     var = 1
3     while(y > 1):
4         var = var * y
5         y = y - 1
6     return var
7
8 print(my_function(4))

```

- a) 0
- b) 20
- c) 24
- d) Raise an Error

Question 29: Which is the output of the following program?

```

1 def my_function(signal):
2     var = True
3     while var:
4         var = False
5         for i in range(len(signal) - 1):
6             if signal[i] > signal[i + 1]:
7                 signal[i], signal[i + 1] = signal[i + 1], signal[i]
8             var = True
9
10 my_signal = [1, 2, 3, 0]
11 my_function(my_signal)
12 print(my_signal)

```

- a) [0, 1, 2, 3]
- b) [1, 1, 1, 1]
- c) [3, 2, 1, 0]
- d) Raise an Error

Question 30: Which is the output of the following program?

```

1 def sum_function(num1, num2):
2     """
3     Sum of two numbers
4     input: num1, num2
5     output: num1+num2
6     """
7     return num1 + num1
8
9 print(sum_function.__doc__)

```

- a)
 Sum of two numbers
 input: num1, num2
 output: num1+num2
 b) 10
 c) Raise an Error
 d) 100!

Question 31: Which is the output of the following program?

```

1 def my_function(x):
2     res = ""
3     for i in x:
4         res = i + res
5     return res
6
7 x = 'apricot'
8 print(my_function(x))

```

- a) apricot
 b) tocirpa
 c) Raise a Error
 d) None

Question 32: Which is the output of the following program?

```

1 def function_helper(x):
2     if x > 0:
3         return 'T'
4     else:
5         return 'N'
6
7 def my_function(data):
8     res = [function_helper(x) for x in data]
9     return res
10
11 data = [2, 3, 5, -1]
12 print(my_function(data))

```

- a) ['N', 'T', 'T', 'N']
 b) ['T', 'N', 'T', 'N']
 c) ['T', 'T', 'T', 'N']
 d) Raise an Error

Question 33: Which is the output of the following program?

```

1 def function_helper(x, data):
2     for i in data:
3         if x == i:
4             return 0
5
6     return 1
7
8 def my_function(data):
9     res = []
10    for i in data:
11        if function_helper(i, res):
12            res.append(i)

```

```
13
14     return res
15
16 lst = [9, 9, 8, 1, 1]
17 print(my_function(lst))
```

- a) [9, 8, 1]
- b) [1, 1, 1]
- c) [9, 9, 8, 1, 1]
- d) Raise an Error

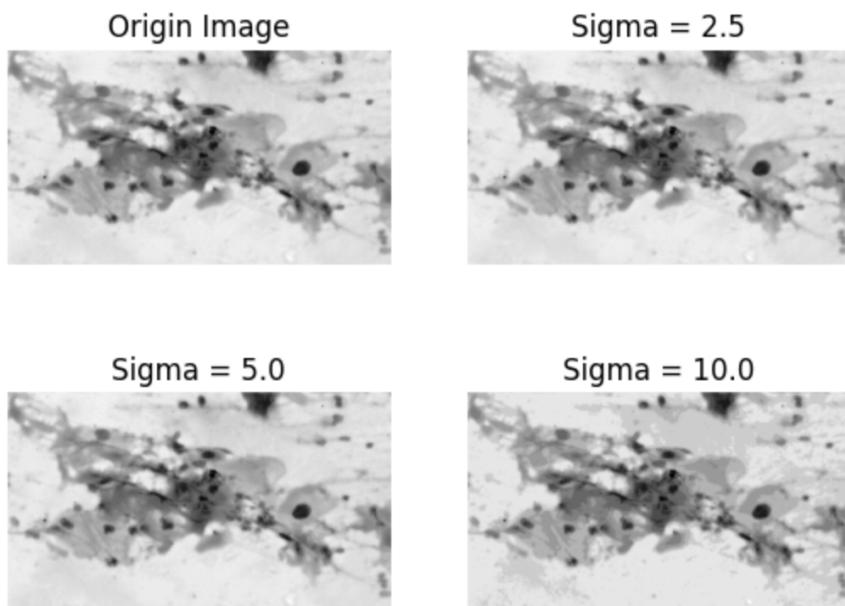
Basic Python - Work with Image Data

Hoàng-Nguyễn Vũ

1. Mô tả:

- **Bộ lọc Gauss (Gaussian Blur)** là một kỹ thuật xử lý ảnh phổ biến được sử dụng để làm mịn ảnh và loại bỏ nhiễu. Nó hoạt động bằng cách áp dụng một ma trận Gauss (Gaussian kernel) lên từng pixel của ảnh.
- **Hàm Gauss** là một hàm phân phối xác suất có dạng hình chuông. Giá trị của hàm Gauss tại mỗi pixel được xác định bởi độ lệch chuẩn (sigma) của hàm. Sigma càng lớn, mức độ làm mịn càng cao. Công thức tổng quát của Gaussian Filter như sau:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \cdot \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (1)$$



- OpenCV cung cấp hai hàm để thực hiện bộ lọc Gauss:

- **cv2.GaussianBlur:** Hàm này cung cấp một cách đơn giản để áp dụng bộ lọc Gauss. Ví dụ:

```

1 # Load image
2 img = cv2.imread('image.png')
3
4 # Apply Gaussian Filter with kernel matrix 5x5, sigma = 1
5 img_filtered = cv2.GaussianBlur(img, (5, 5), 1)

```

- **cv2.filter2D:** Hàm này cho phép áp dụng một ma trận tùy ý lên ảnh, bao gồm cả ma trận Gauss. Ví dụ:

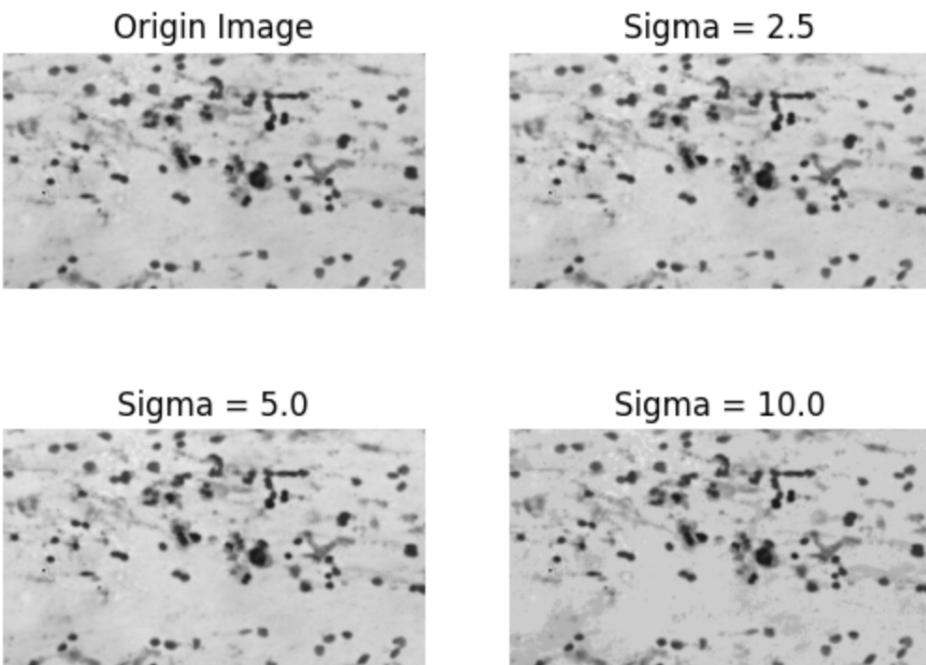
```

1 def gaussian_kernel(size, sigma):
2     if size % 2 == 0:
3         size = size + 1
4
5     max_point = size // 2 # both directions (x,y) maximum
6     cell start point
7     min_point = -max_point # both directions (x,y) minimum
8     cell start point
9
10    K = np.zeros((size, size)) # kernel matrix
11    for x in range(min_point, max_point + 1):
12        for y in range(min_point, max_point + 1):
13            value = #Gaussian Filter Forumla Here#
14            K[x - min_point, y - min_point] = value
15
16    return K
17
18 kernel = gaussian_kernel(5, 1.4)
19 img = cv2.imread("image_path", 0)
20 img_gaussian = cv2.filter2D(img, -1, kernel)

```

2. Bài tập: Dữ liệu ảnh Y Khoa - Nguồn: Kaggle

- Hãy đọc và hiển thị ảnh có tên 2.jpg trong tập dữ liệu trên, áp dụng kỹ thuật Gaussian Filter trong thư viện OpenCV theo 2 cách cài đặt với các sigma khác nhau như [2.5, 5.0, 10.0] Kết quả:



Basic Python - Work with SQLite Database

Hoàng-Nguyễn Vũ



1. Mô tả: Làm quen với Cơ sở dữ liệu SQLite

- **SQLite** là một hệ quản trị cơ sở dữ liệu hay còn gọi là hệ thống cơ sở dữ liệu quan hệ nhỏ gọn, khác với các hệ quản trị khác như MySQL, SQL Server, Oracle, PostgreSQL... SQLite là một thư viện phần mềm mà triển khai một SQL Database Engine truyền thống, không cần mô hình client-server nên rất nhỏ gọn. SQLite được sử dụng vào rất nhiều chương trình từ desktop đến mobile hay là website..
- Ngoài những lý do trên thì không thể không kể đến những ưu điểm khi sử dụng SQLite, sau đây là phần ưu điểm của SQLite. **Ưu điểm của SQLite:**
 - **Nhẹ và dễ sử dụng:** SQLite không yêu cầu cài đặt máy chủ riêng biệt và có thể được nhúng trực tiếp vào ứng dụng Python của bạn.
 - **Hiệu suất cao:** SQLite cung cấp hiệu suất truy vấn nhanh và có thể xử lý lượng dữ liệu lớn.
 - **Đa nền tảng:** SQLite có sẵn trên hầu hết các nền tảng, bao gồm Windows, MacOS, Linux và Android.
 - **Miễn phí và mã nguồn mở:** SQLite là phần mềm miễn phí và mã nguồn mở, có nghĩa là bạn có thể sử dụng và sửa đổi nó cho mục đích của riêng bạn.

2. Làm quen với các câu lệnh SQL trong SQLite Database:

- Ví dụ chúng ta cần quản lý thông tin khách hàng gồm các trường dữ liệu cơ bản như sau:

Bảng 1: Bảng quản lý thông tin của khách hàng

CUSTOMER		
Email	Name	Phone

Phân tích sơ lược, chúng ta dễ thấy mỗi khách hàng chỉ có duy nhất 1 email, nên email sẽ đóng vai trò là khóa chính, nhằm dễ xác định thông tin của một khách hàng trong hệ thống. Để quản lý các thông tin trên và và truy vấn chúng ta sẽ có những thao tác chính như sau: Tạo bảng (**CREATE TABLE**), lấy dữ liệu từ bảng (**SELECT**), thêm mới dữ liệu (**INSERT**), cập nhật dữ liệu (**UPDATE**), xóa dữ liệu (**DELETE**).

- + **Tạo bảng mới (CREATE TABLE):** Để tạo bảng trên ta sẽ sử dụng như sau:

```

1 -- Tạo Bảng CUSTOMERS gồm 3 cột email, name, phone với email là
2   khóa chính
3 CREATE TABLE customers (
4   email TEXT PRIMARY KEY,
5   name TEXT NOT NULL,
6   phone TEXT
7 );

```

- + **Lấy dữ liệu (SELECT):** Chúng ta có 2 cách lấy dữ liệu từ bảng trong cơ sở dữ liệu bao gồm: lấy hết dữ liệu của toàn bộ cột trong bảng và lấy dữ liệu từ các cột được chỉ định. Cách truy vấn (Query) có cấu trúc tổng quát như sau:

```

1 SELECT column1, column2, ...
2 FROM table_name;

```

A. Lấy toàn bộ dữ liệu của tất cả cột trong bảng:

```

1 -- Lấy toàn bộ dữ liệu của tất cả cột trong bảng CUSTOMERS
2 SELECT *
3 FROM CUSTOMERS;

```

B. Lấy dữ liệu từ các cột được chỉ định trong bảng:

```

1 -- Lấy dữ liệu từ các cột được chỉ định trong bảng CUSTOMERS
2 SELECT NAME, PHONE
3 FROM CUSTOMERS;

```

C. Lấy dữ liệu theo điều kiện trong bảng (Filter):

```

1 -- Lấy dữ liệu theo điều kiện email = giá trị truyền vào từ bảng
2   CUSTOMERS
3 SELECT NAME, PHONE
4 FROM CUSTOMERS
5 WHERE 1 = 1
6 AND EMAIL = 'aivietnam@aivietnam.edu.vn';

```

- + **Thêm dữ liệu (INSERT):** Để thêm dữ liệu vào bảng chúng ta có thể thực thi với câu lệnh có cấu trúc tổng quát như sau:

```

1 INSERT INTO table_name (column1, column2, column3, ...)
2 VALUES (value1, value2, value3, ...);

```

Ví dụ:

```

1 -- Thêm mới 1 dòng trong bảng CUSTOMERS
2 INSERT INTO CUSTOMERS(EMAIL, NAME, PHONE)
3 VALUES('nguyen@nguyen.com', 'Nguyen', '123456789');
4
5 -- Thêm mới nhiều dòng trong bảng CUSTOMERS
6 INSERT INTO CUSTOMERS(EMAIL, NAME, PHONE)
7 VALUES
8     ('nguyen@aivietnam.edu.vn', 'Nguyen', '123456789'),
9     ('admin@aivietnam.edu.vn', 'Vinh', '1122334455');

```

- + **Cập nhật dữ liệu (UPDATE):** Để cập nhật dữ liệu vào bảng chúng ta có thể thực thi với câu lệnh có cấu trúc tổng quát như sau:

```

1 UPDATE table_name
2 SET column1 = value1, column2 = value2, ...
3 WHERE condition;

```

Ví dụ: Cập nhật tên của email là nguyen@aivietnam.edu.vn thành Hoang Nguyen.

```

1 -- Cập nhật tên của email là nguyen@aivietnam.edu.vn thành Hoang
2   Nguyen
3 UPDATE CUSTOMERS
4 SET NAME = 'Hoang Nguyen'
5 WHERE 1 = 1
6 AND EMAIL = 'nguyen@aivietnam.edu.vn';

```

- + **Xóa dữ liệu (DELETE):** Để xóa dữ liệu khỏi bảng chúng ta có thể thực thi với câu lệnh có cấu trúc tổng quát như sau:

```

1 DELETE FROM table_name
2 WHERE condition;

```

Ví dụ: Xóa email nguyen@aivietnam.edu.vn khỏi bảng CUSTOMERS

```

1 -- Xóa email: nguyen@aivietnam.edu.vn khỏi bảng CUSTOMERS
2 DELETE FROM CUSTOMERS
3 WHERE 1 = 1
4 AND EMAIL = 'nguyen@aivietnam.edu.vn';

```

3. Sử dụng SQLite trong Python:

- **Tạo kết nối đến CSDL:** Để sử dụng SQLite và tương tác với cơ sở dữ liệu SQLite trong python, chúng ta cần tạo kết nối đến CSDL bằng câu lệnh như sau:

```

1 import sqlite3
2 # Tạo kết nối tới CSDL có tên là database.sqlite
3 # Nếu database.sqlite chưa tồn tại trong hệ thống thì nó sẽ
4 # tự tạo mới
5 connection = sqlite3.connect('database.sqlite')
6 cursor = connection.cursor()

```

→ Sau khi tạo kết nối đến cơ sở dữ liệu xong, chúng ta có Object có tên *cursor* nhằm giúp chúng ta thực thi và tương tác các câu lệnh truy vấn (SELECT, INSERT, UPDATE, DELETE,...) đến cơ sở dữ liệu.

- **Tạo bảng:** Để tạo bảng mới xong cơ sở dữ liệu chúng ta sẽ thực hiện như sau:

```

1 # Tạo bảng CUSTOMERS:
2 cursor.execute("""
3 CREATE TABLE CUSTOMERS (
4     EMAIL TEXT PRIMARY KEY,
5     NAME TEXT NOT NULL,
6     PHONE TEXT NOT NULL
7 );
8 """)
```

- **Thêm dữ liệu vào bảng (INSERT):** Sau khi thực hiện tạo bảng ở bước trên, lúc này bảng CUSTOMERS của chúng ta hoàn toàn không có dữ liệu, nên chúng ta sẽ thêm dữ liệu vào bảng bằng cách thực thi câu SQL như sau:

```

1 # Insert data mới
2 cursor.execute("""
3 INSERT INTO CUSTOMERS(EMAIL, NAME, PHONE)
4 VALUES
5     ('nguyen@aivietnam.edu.vn', 'Nguyen', '123456789'),
6     ('admin@aivietnam.edu.vn', 'Vinh', '1122334455');
7 """
8
9 connection.commit()
```

→ **Mở rộng:** Khi thực hiện thao tác INSERT/UPDATE/DELETE, khi thực hiện cursor.execute() trên cơ sở dữ liệu SQLite, những thay đổi này chỉ được lưu trữ tạm thời trong bộ nhớ. Nên chúng ta cần phải thực hiện commit để lưu những thay đổi này vào CSDL, đảm bảo chúng được lưu trữ vĩnh viễn.

- **Lấy dữ liệu (SELECT):** Sau khi thực hiện bước tạo dữ liệu ở trên, chúng ta sẽ kiểm tra dữ liệu vừa thêm bằng cách thực thi câu SQL như sau:

```

1 import pandas as pd
2 # Lấy tất cả data từ bảng CUSTOMER
3 data=pd.read_sql_query("SELECT * FROM CUSTOMERS", connection)
4 print(data)
```

→ Kết quả:

```
[5]: import pandas as pd
```

```
[7]: data=pd.read_sql_query("SELECT * FROM CUSTOMERS", connection)
print(data)
```

	EMAIL	NAME	PHONE
0	nguyen@aivietnam.edu.vn	Nguyen	123456789
1	admin@aivietnam.edu.vn	Vinh	1122334455

Hình 1: Kết quả sau khi thực hiện SELECT

- **Cập nhật dữ liệu (UPDATE):** Để thực hiện cập nhật dữ liệu trong CSDL, chúng ta sẽ thực thi câu SQL như sau:

Cập nhật tên của email : nguyen@aivietnam.edu.vn sang giá trị mới và kiểm tra kết quả sau khi cập nhật.

```

1 # Update name của email: nguyen@aivietnam.edu.vn
2 cursor.execute("""
3 UPDATE CUSTOMERS
4 SET NAME = 'Hoang Nguyen'
5 WHERE 1 = 1
6 AND EMAIL = 'nguyen@aivietnam.edu.vn';
7 """
8
9 connection.commit()
10
11 data=pd.read_sql_query("SELECT * FROM CUSTOMERS", connection)
12 print(data)

```

→ Kết quả:

	EMAIL	NAME	PHONE
0	nguyen@aivietnam.edu.vn	Hoang Nguyen	123456789
1	admin@aivietnam.edu.vn	Vinh	1122334455

Hình 2: Kết quả sau khi thực hiện UPDATE

- **Xóa dữ liệu (DELETE):** Để thực hiện xóa dữ liệu trong CSDL, chúng ta sẽ thực thi câu SQL như sau:

Xóa email : nguyen@aivietnam.edu.vn khỏi bảng CUSTOMERS và kiểm tra kết quả sau khi cập nhật.

```

1 # Update name của email: nguyen@aivietnam.edu.vn
2 cursor.execute("""
3 DELETE FROM CUSTOMERS
4 WHERE 1 = 1
5 AND EMAIL = 'nguyen@aivietnam.edu.vn';
6 """
7
8 connection.commit()
9
10 data=pd.read_sql_query("SELECT * FROM CUSTOMERS", connection)
11 print(data)

```

→ Kết quả:

	EMAIL	NAME	PHONE
0	admin@aivietnam.edu.vn	Vinh	1122334455

Hình 3: Kết quả sau khi thực hiện DELETE

4. Bài tập:

- Hãy tạo mới bảng có tên *PRODUCT* có các cột như sau:

Bảng 2: Bảng quản lý sản phẩm

PRODUCT		
Cột	Kiểu dữ liệu	Chú thích
ID	INTEGER	Khóa chính
NAME	TEXT	Not null
PRICE	INTEGER	Not null

Câu 1: Hãy thêm mới các dòng có giá trị như sau và kiểm tra kết quả:

PRODUCT		
ID	NAME	PRICE
1	iPhone 15	18000000
2	Galaxy Z-Fold 5	30000000

→ Kết quả:

	ID	NAME	PRICE
0	1	iPhone 15	18000000
1	2	Galaxy Z-Fold 5	30000000

Câu 2: Hãy cập nhật giá mới cho Galaxy Z-Fold 5 thành 50.000.000 và kiểm tra kết quả:

→ Kết quả:

	ID	NAME	PRICE
0	1	iPhone 15	18000000
1	2	Galaxy Z-Fold 5	50000000

Câu 3: Hãy xóa iPhone 15 ra khỏi CSDL và kiểm tra kết quả:

→ Kết quả:

	ID	NAME	PRICE
0	2	Galaxy Z-Fold 5	50000000

Basic Python - Work with SQLite Database

Hoàng-Nguyễn Vũ



1. Mô tả: Thống kê cơ bản trong SQLite

- **SQL** cung cấp nhiều hàm để thực hiện các phép tính thống kê cơ bản trên dữ liệu. Các hàm này được sử dụng để tính toán các giá trị như tổng, trung bình, giá trị tối đa, giá trị tối thiểu, v.v. Dưới đây là một số hàm thống kê cơ bản trong SQL:

- **SUM()**: Tính tổng của các giá trị trong một cột.
- **AVG()**: Tính trung bình của các giá trị trong một cột.
- **MIN()**: Tìm giá trị nhỏ nhất trong một cột.
- **MAX()**: Tìm giá trị lớn nhất trong một cột.
- **COUNT()**: Đếm số lượng các giá trị trong một cột.

Ví dụ: Giả sử bạn có một bảng *Product* với các cột *name*, *brand* và *price*. Bạn muốn biết:

- Tổng doanh thu của tất cả các sản phẩm.
- Doanh thu trung bình của các sản phẩm.
- Giá sản phẩm cao nhất.
- Số lượng các sản phẩm khác nhau.

Câu lệnh SQL cho các yêu cầu trên sẽ như sau:

```
1 # Query để lấy tổng giá bán toàn bộ sản phẩm trong bảng Product
2 query = """
3 SELECT SUM(price) AS total_revenue
4 FROM PRODUCT;
5 """
6
7 data_sum = pd.read_sql_query(query, connection)
8 print(data_sum)
9
```

```

10 # Query để lấy thông tin sản phẩm có giá bán cao nhất
11 query = """
12 SELECT NAME, MAX(price) AS PRICE
13 FROM PRODUCT;
14 """
15 data_max = pd.read_sql_query(query, connection)
16 print(data_max)

```

→ Kết quả:

```

data=pd.read_sql_query("SELECT * FROM PRODUCT", connection)
print('*** All data ***')
print(data)

```

```

*** All data ***
   ID      NAME BRAND    PRICE
0  1    iPhone 15  Apple 18000000
1  2  Galaxy Z-Fold 5 Samsung 30000000
2  3        Find X  Oppo 20000000
3  4    iPhone 14  Apple 16000000
4  5  Galaxy Z-Flip Samsung 17000000
5  6 iPhone 15 Pro Max  Apple 48000000

```

```

query = """
SELECT SUM(price) AS total_revenue
FROM PRODUCT;
"""

data_sum = pd.read_sql_query(query, connection)
print(data_sum)

total_revenue
0      149000000

```

```

query = """
SELECT NAME, MAX(price) AS PRICE
FROM PRODUCT;
"""

data_max = pd.read_sql_query(query, connection)
print(data_max)

```

```

          NAME      PRICE
0  iPhone 15 Pro Max  48000000

```

- **Hàm thống kê với GROUP BY:**

GROUP BY là một mệnh đề trong SQL được sử dụng để nhóm các hàng dựa trên các giá trị chung trong một hoặc nhiều cột và thực hiện các phép tính tổng hợp trên các nhóm đó.

Hàm thống kê được sử dụng để tính toán các giá trị như tổng, trung bình, giá trị tối đa, giá trị tối thiểu, v.v. trên các nhóm dữ liệu.

Ví dụ: Giả sử bạn có một bảng *Product* với các cột *name*, *brand* và *price*. Bạn muốn biết:

- Doanh thu tổng cho mỗi hãng (BRAND) của sản phẩm.

- Giá bán thấp nhất của mỗi danh mục.

Cách thực thi câu lệnh SQL cho 2 yêu cầu trên sẽ như sau:

```

1 # Query để lấy tổng giá bán toàn bộ sản phẩm theo hãng trong bảng
   Product
2 query = """
3 SELECT BRAND, SUM(price) AS total_revenue
4 FROM PRODUCT
5 GROUP BY BRAND;
6 """
7 data_sum_by_brand = pd.read_sql_query(query, connection)
8 print(data_sum_by_brand)
9
10 # Query để lấy thông tin sản phẩm có giá bán thấp nhất
11 query = """
12 SELECT NAME, BRAND, MIN(price) AS PRICE
13 FROM PRODUCT
14 GROUP BY BRAND;
15 """
16 data_min_by_brand = pd.read_sql_query(query, connection)
17 print(data_min_by_brand)

```

→ Kết quả:

```

query = """
SELECT BRAND, SUM(price) AS total_revenue
FROM PRODUCT
GROUP BY BRAND;
"""

data_sum_by_brand = pd.read_sql_query(query, connection)
print(data_sum_by_brand)

```

	BRAND	total_revenue
0	Apple	82000000
1	Oppo	20000000
2	Samsung	47000000

```

query = """
SELECT NAME, BRAND, MIN(price) AS PRICE
FROM PRODUCT
GROUP BY BRAND;
"""

data_min_by_brand = pd.read_sql_query(query, connection)
print(data_min_by_brand)

```

	NAME	BRAND	PRICE
0	iPhone 14	Apple	16000000
1	Find X	Oppo	20000000
2	Galaxy Z-Flip	Samsung	17000000

2. Bài tập:

- Hãy tạo mới bảng có tên *STOCK* có các cột như sau:

Bảng 1: Bảng quản lý cổ phiếu

STOCK		
Cột	Kiểu dữ liệu	Chú thích
ID	INTEGER	Khóa chính
NAME	TEXT	Not null
BUY	INTEGER	Not null
INVESTOR	TEXT	Not null

Hãy thêm mới các dòng có giá trị như sau:

STOCK			
ID	NAME	BUY	INVESTOR
1	ACB	29.45	Nguyen
2	VIC	44.55	Nguyen
3	GMD	74.30	Nguyen
4	ACB	28.45	Vinh
5	VIC	40.55	Vinh
6	GMD	60.30	Vinh

Câu 1: Hãy viết lệnh SQL để truy vấn và in ra kết quả thống kê tổng giá bán (BUY) của bảng STOCK:

Kết quả: Tổng giá bán = 277.69

Câu 2: Hãy viết lệnh SQL để thống kê mã cổ phiếu có giá mua (BUY) lớn nhất theo nhà đầu tư (Investor):

→ **Kết quả:**

	INVESTOR	NAME	MAX_PRICE
0	Nguyen	GMD	74.39
1	Vinh	GMD	60.30

XỬ LÝ TỆP PDF ĐƠN GIẢN VỚI PYPDF

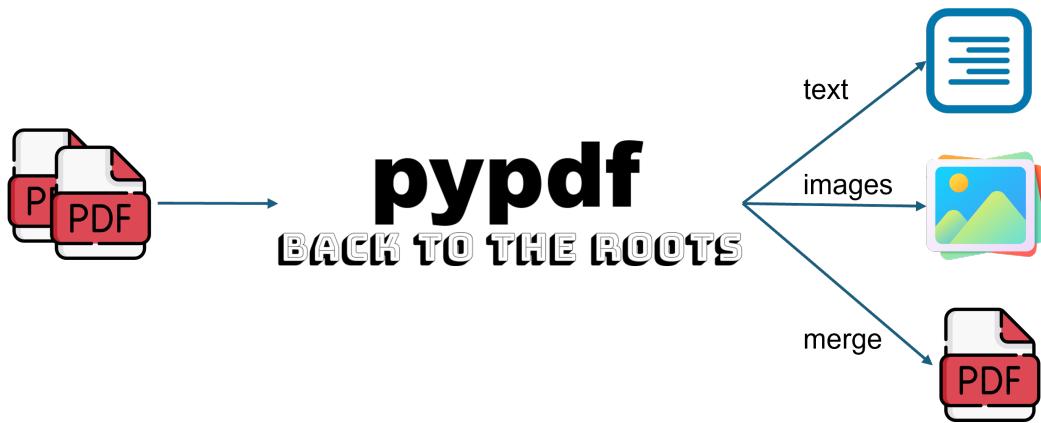
Dinh-Tiem Nguyen và Quang-Vinh Dinh

1 Mở đầu

Làm sao để trích xuất nội dung văn bản và hình ảnh trong file pdf? Làm thế nào để ghép nhiều file pdf thành một file duy nhất? Trong bài viết này, chúng ta sẽ trả lời những câu hỏi đó bằng cách sử dụng thư viện pypdf-một thư viện xử lý file pdf hiệu quả.

Yêu cầu:

- Máy tính đã cài đặt Python ≥ 3.7
- Biết lập trình Python cơ bản.



2 Hướng dẫn cài đặt và sử dụng pypdf

Để cài đặt pypdf ta sử dụng lệnh sau:

```
1 pip install pypdf
```

2.1 Trích xuất văn bản từ pdf

Pypdf cho phép ta trích xuất nội dung văn bản từ một hoặc nhiều trang trong tập tin PDF. Điều này rất hữu ích khi ta cần trích xuất thông tin hoặc dữ liệu văn bản từ tài liệu PDF để sử dụng trong các ứng dụng khác.

Ví dụ chương trình dưới đây, chúng ta khai báo sử dụng thư viện pypdf và sử dụng PdfReader để đọc file `yolov9.pdf` sau đó lưu kết quả vào biến reader. Tiếp theo ta sử dụng reader.pages, phương thức này trả về một danh sách các trang trong tập tin PDF đã được đọc bằng PdfReader. Mỗi phần tử trong danh sách này đại diện cho một trang trong tập tin PDF và có thể được truy cập bằng cách sử dụng chỉ mục hoặc vòng lặp. Ví dụ, để truy cập trang thứ hai trong tập tin PDF, ta có thể sử dụng `reader.pages[1]`, vì chỉ mục trong Python bắt đầu từ 0. Điều này sẽ trả về một đối tượng đại diện cho trang thứ hai trong tài liệu PDF. Để truy cập vào nội dung của một trang cụ thể, ta có thể sử dụng phương thức `extract_text()`, như trong ví dụ.

```

1 from pypdf import PdfReader
2
3 # Đọc file PDF
4 reader = PdfReader("yolov9.pdf")
5
6 # Lấy số trang
7 num_pages = len(reader.pages)
8
9 # Lấy nội dung trang đầu tiên
10 page_1 = reader.pages[0]
11 page_1_txt = page_1.extract_text()
12
13 # Lấy nội dung toàn bộ các trang
14 pages_txt = ""
15 for i in range(num_pages):
16     page = reader.pages[i]
17     pages_txt += page.extract_text() + "\n"

```

Abstract

Today's deep learning methods focus on how to design the most appropriate objective functions so that the prediction results of the model can be closest to the ground truth. Meanwhile, an appropriate architecture that can facilitate acquisition of enough information for prediction has to be designed. Existing methods ignore a fact that when input data undergoes layer-by-layer feature extraction and spatial transformation, large amount of information will be lost. This paper will delve into the important issues of data loss when data is transmitted through deep networks, namely information bottleneck and reversible functions. We proposed the concept of programmable gradient information (PGI) to cope with the various changes required by deep networks to achieve multiple objectives. PGI can provide complete input information for the target task to calculate objective function, so that reliable gradient information can be obtained to update network weights. In addition, a new lightweight network architecture - Generalized Efficient Layer Aggregation Network (GELAN), based on gradient path planning is designed. GELAN's architecture confirms that PGI has gained superior results on lightweight models. We verified the proposed GELAN and PGI on MS COCO dataset based object detection. The results show that GELAN only uses conventional convolution operators to achieve better parameter utilization than the state-of-the-art methods developed based on depth-wise convolution. PGI can be used for variety of models from lightweight to large. It can be used to obtain complete information, so that train-from-scratch models can achieve better results than state-of-the-art models pre-trained using large datasets, the comparison results are shown in Figure 1. The source codes are at: <https://github.com/WongKinYiu/yolov9>.


 PYPDF →
Abstract

Today's deep learning methods focus on how to design the most appropriate objective functions so that the prediction results of the model can be closest to the ground truth. Meanwhile, an appropriate architecture that can facilitate acquisition of enough information for prediction has to be designed. Existing methods ignore a fact that when input data undergoes layer-by-layer feature extraction and spatial transformation, large amount of information will be lost. This paper will delve into the important issues of data loss when data is transmitted through deep networks, namely information bottleneck and reversible functions. We proposed the concept of programmable gradient information (PGI) to cope with the various changes required by deep networks to achieve multiple objectives. PGI can provide complete input information for the target task to calculate objective function, so that reliable gradient information can be obtained to update network weights. In addition, a new lightweight network architecture - Generalized Efficient Layer Aggregation Network (GELAN), based on gradient path planning is designed. GELAN's architecture confirms that PGI has gained superior results on lightweight models. We verified the proposed GELAN and PGI on MS COCO dataset based object detection. The results show that GELAN only uses conventional convolution operators to achieve better parameter utilization than the state-of-the-art methods developed based on depth-wise convolution. PGI can be used for variety of models from lightweight to large. It can be used to obtain complete information, so that train-from-scratch models can achieve better results than state-of-the-art models pre-trained using large datasets, the comparison results are shown in Figure 1. The source codes are at: <https://github.com/WongKinYiu/yolov9> .

Hình 1: Trích xuất văn bản từ file PDF

2.2 Trích xuất hình ảnh từ pdf

Trích xuất hình ảnh từ các tập tin PDF là quá trình lấy các hình ảnh từ các trang trong tài liệu PDF và lưu chúng thành các tập tin hình ảnh độc lập, chẳng hạn như JPEG hoặc PNG. Việc này thường được thực hiện để xử lý và sử dụng hình ảnh từ tài liệu PDF trong các ứng dụng khác nhau, như xây dựng bộ sưu tập hình ảnh, phân tích hình ảnh...

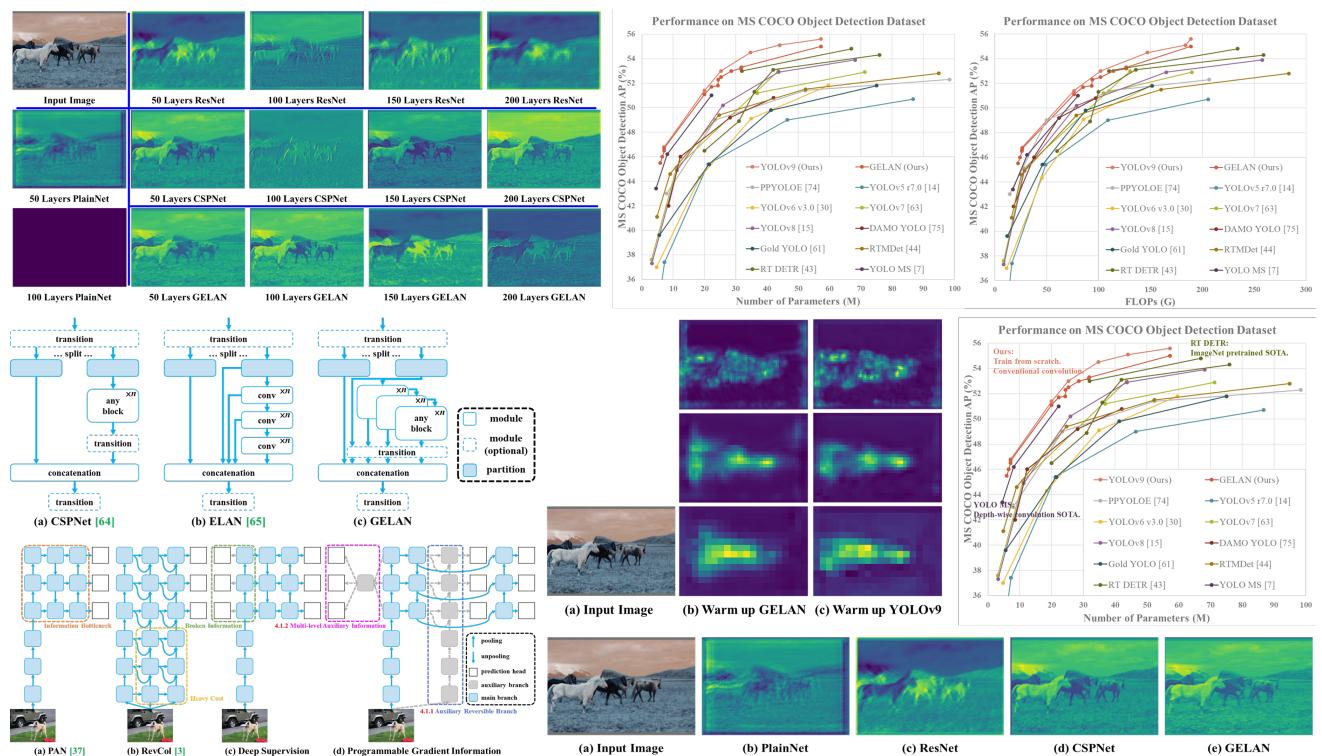
Trong pypdf cung cấp phương thức `page.images` để hỗ trợ trích xuất các hình ảnh trong tập tin pdf. Ta cùng xem ví dụ sau:

```

1 # extract images
2 from pypdf import PdfReader
3
4 reader = PdfReader("yolov9.pdf")
5 count = 0
6 for page in reader.pages:
7     for image_file_object in page.images:
8         with open(str(count) + image_file_object.name, "wb") as fp:
9             fp.write(image_file_object.data)
10            count += 1

```

Trong ví dụ trên, chúng ta đọc và sử dụng vòng lặp để duyệt qua từng trang trong file pdf, với mỗi trang chúng ta sẽ trích xuất các hình ảnh thông qua phương thức `page.images`, phương thức này sẽ trả về danh sách các hình ảnh mà nó trích xuất được từ 1 trang, sau đó chúng ta lưu lại file hình ảnh với tên hình ảnh là số thứ tự + tên hình ảnh trong file pdf ta lấy được qua phương thức `image_file_object.name`



Hình 2: Trích xuất các hình ảnh từ file PDF

2.3 Nối các file pdf

Merge PDF cho phép bạn tổ chức các tài liệu PDF riêng lẻ thành một tài liệu lớn hơn, giúp dễ dàng quản lý và tìm kiếm thông tin. Trong pypdf cung cấp tính năng PdfWriter để thực hiện điều này.

```

1 # Merge PDFs
2 from pypdf import PdfWriter
3
4 merger = PdfWriter()
5 for pdf in ["yolov6.pdf", "yolov7.pdf", "yolov9.pdf"]:
6     merger.append(pdf)
7
8 merger.write("merged-yolov-679.pdf")
9 merger.close()

```

Trong ví dụ trên, ta thực hiện nối ba bài báo yolov6, yolov7, yolov9 lại với nhau, đầu tiên chúng ta sẽ import PdfWriter từ thư viện pypdf, tiếp theo chúng ta tạo đối tượng merger từ PdfWriter(), đối tượng này sẽ được sử dụng để merge các tập tin PDF. Tiếp theo chúng ta duyệt qua từng file pdf và thêm chúng vào đối tượng merger qua phương thức append. Cuối cùng ta ghi dữ liệu từ đối tượng merger vào tệp tin mới có tên là merged-yolov-679.pdf và phương thức close() được gọi để đóng tập tin PDF đã được merge.

2.4 Nén file pdf

Chức năng nén file trong thư viện pypdf cho phép bạn giảm kích thước của các tập tin PDF bằng cách nén lại các luồng nội dung của mỗi trang. Việc này không chỉ giúp tiết kiệm không gian lưu trữ mà còn làm tăng tốc độ tải xuống và chia sẻ tập tin PDF.

```

1 from pypdf import PdfWriter
2
3 writer = PdfWriter(clone_from="yolov9.pdf")
4
5 for page in writer.pages:
6     page.compress_content_streams(level=8) # This is CPU intensive!
7
8 with open("out.pdf", "wb") as f:
9     writer.write(f)

```

Trong ví dụ trên, ta tạo một bản sao của tập tin PDF "yolov9.pdf", nén lại nội dung của mỗi trang trong tập tin PDF này với mức độ nén là 8, và sau đó lưu lại thành một tập tin PDF mới có tên là "out.pdf". Đầu tiên ta tạo một đối tượng PdfWriter mới gọi là writer với số clone_from chỉ định tên tập tin PDF mà chúng ta muốn tạo bản sao. Tiếp theo chúng ta lặp qua từng trang của pdf, sau đó sử dụng page.compress_content_streams(level=8) để nén, trong đó level là mức nén có giá trị từ 1 đến 9. Cuối cùng chúng ta lưu lại file có tên out.pdf.

Chúng ta có thể kiểm tra dung lượng file qua đoạn mã dưới đây:

```

1 # get size of file pdf
2 import os
3
4 file_size = os.path.getsize("yolov9.pdf") / (1024 * 1024)
5 print(f"yolo9.pdf size:{file_size}")
6
7 file_out = os.path.getsize("out.pdf") / (1024 * 1024)
8 print(f"out.pdf size:{file_out}")

```

===== Output =====
yolo9.pdf size:4.738467216491699
out.pdf size:4.718204498291016
=====

Có thể thấy dung lượng file out.pdf được nén từ file yolov9.pdf có dung lượng thấp hơn.

3 Kết Luận

Trong bài viết này, chúng ta đã tìm hiểu cách sử dụng thư viện pypdf trong Python để thực hiện hai tác vụ quan trọng là trích xuất văn bản và hình ảnh từ các tập tin PDF, cũng như thực hiện các tính năng nén, merge (gộp) các tập tin PDF lại với nhau. Qua bài viết, hy vọng có thể giúp mọi người biết thêm cách sử dụng công cụ hữu ích pypdf để xử lý dữ liệu pdf.

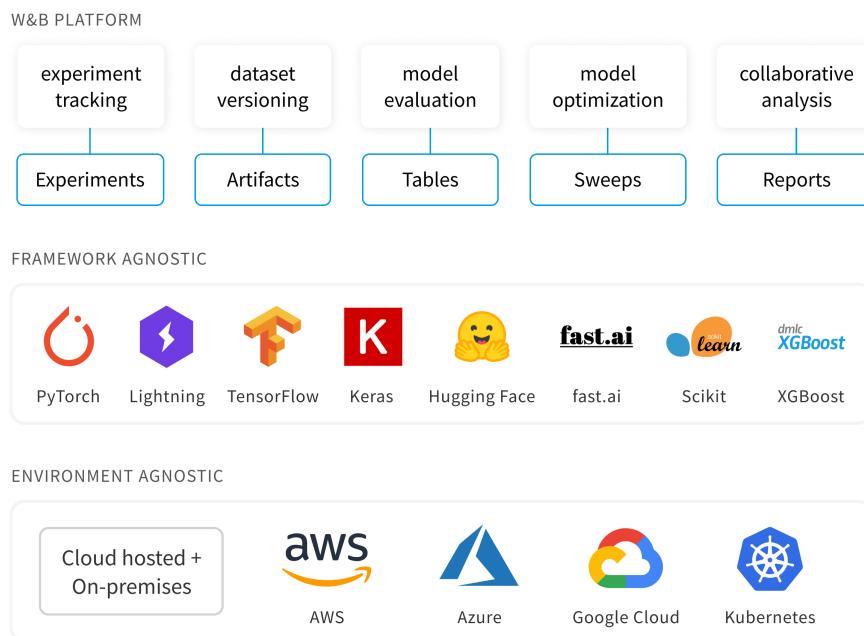
Basic Python - Getting Started with WandB

Hoàng-Nguyễn Vũ và Quang-Vinh Dinh

Weights & Biases

1. Mô tả:

- WandB hay còn gọi là **Weights and Biases** là một công cụ giúp các Data Scientist theo dõi mô hình, dữ liệu, thông tin hệ thống chỉ với vài dòng code. Weights & Bias hỗ trợ miễn phí cho các cá nhân hoặc tổ chức nghiên cứu. Công cụ này hỗ trợ rất nhiều nền tảng mà không cần chuyển đổi sang công cụ khác như TensorFlow, Keras, PyTorch, Sklearn, FastAI, ...
- Các thông tin cần quản lý và theo dõi sẽ được chuyển tới một giao diện (UI) của Weights & Biases. Giao diện này sẽ hiển thị và phân tích thông tin dễ dàng, việc so sánh các mô hình cũng như các tham số của nhiều thử nghiệm cũng diễn ra một cách nhanh chóng và trực tiếp. Một tiện ích nữa là bạn có thể chia sẻ các thông tin này cho team member của bạn. Dưới đây là một số công cụ trong WandB hỗ trợ bao gồm:



Hình 1: Tổng quan các công cụ của WandB.

- **Experiments:** Theo dõi các thử nghiệm trên các biểu đồ như evaluate function, loss function; thông tin về hệ thống như dung lượng bộ nhớ, thông tin xử lý GPU, ...
- **Artifacts:** Phiên bản dữ liệu, các phiên bản mô hình
- **Table:** Hiển thị các thông tin về các thử nghiệm và giá trị của các tham biến.
- **Sweeps:** Tối ưu hóa các tham số
- **Report:** Lưu trữ và chia sẻ các số liệu để có thể tái tạo mô hình

2. Cách triển khai WandB:

- **Tạo tài khoản:** Để bắt đầu, chúng ta cần tạo một tài khoản trên trang của Weights & Biases [tại đây](#). Sau khi đăng ký thành công, tại màn hình chính của trang web, chúng ta sẽ được cấp một API key để đăng nhập vào và sử dụng thư viện WandB.



Quickstart: Tracking your first run in Weights & Biases

Weights & Biases' tools make it easy for you to quickly track experiments, visualize results, spot regressions, and more. Simply put, Weights & Biases enables you to build better models faster and easily share findings with colleagues.

Visualize your model training with [Python / Pytorch](#) or [Open in Colab](#)

1. Set up the wandb library

Install the CLI and Python library for interacting with the Weights and Biases API.

```
pip install wandb
```

Next, log in and paste your API key when prompted.

```
wandb login
```



Your API key for logging in to the wandb library.

d0c

sd123...

Hình 2: Giao diện WandB sau khi đăng ký thành công.

• Cài đặt và đăng nhập vào thư viện WandB:

- Để cài đặt thư viện WandB, chúng ta sẽ thực thi dòng lệnh sau và đăng nhập dựa trên API Key được cấp ở bước đăng ký ở trên.

```
1 # Cài đặt thư viện WandB
2 !pip install wandb
3
4 # Đăng nhập với API Key
5 !wandb login
```

```
▶ 1 !wandb login
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit: .....  


```

Hình 3: Đăng nhập vào WandB với API Key.

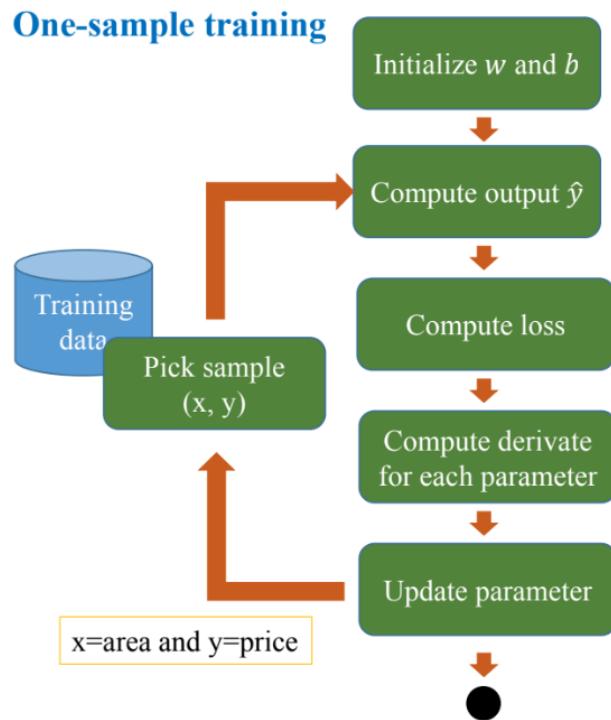
- **Làm quen với WandB:**

- Giả sử, chúng ta có tập dữ liệu giá nhà theo diện tích như bảng dưới, và chúng ta cần xây dựng mô hình hồi quy tuyến tính (Linear Regression) cơ bản để dự đoán giá nhà theo diện tích:

Bảng 1: Bảng dữ liệu giá nhà theo diện tích

Diện tích (m^2)	Giá
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

- Để xây dựng mô hình linear regression với tập dữ liệu trên, chúng ta sẽ xây dựng dựa theo lưu đồ dưới đây:



Hình 4: Lưu đồ cài đặt mô hình Linear Regression

Chúng ta sẽ cài đặt và theo dõi quá trình training của mô hình với WandB như sau:

A. Import thư viện và khởi tạo dataset:

```

1 import pandas as pd
2 import wandb
3
4 areas = [6.7, 4.6, 3.5, 5.5]
5 prices = [9.1, 5.9, 4.6, 6.7]
6
7 dataset = pd.DataFrame({
8     'areas': areas,
9     'prices': prices
10 })

```

B. Cài đặt mô hình Linear Regression kèm theo dõi quá trình huấn luyện với WandB:

- + Cài đặt mô hình:

```

1 # forward
2 def predict(x, w, b):
3     return x*w + b
4
5 # compute gradient
6 def gradient(y_hat, y, x):
7     dw = 2*x*(y_hat-y)
8     db = 2*(y_hat-y)
9
10    return (dw, db)
11
12 # update weights
13 def update_weight(w, b, lr, dw, db):
14     w_new = w - lr*dw
15     b_new = b - lr*db
16
17    return (w_new, b_new)

```

- + Huấn luyện mô hình và theo dõi quá trình với WandB:

```

1 # init weights
2 b = 0.04
3 w = -0.34
4 lr = 0.01
5 epochs = 10
6
7 # init project wandb
8 wandb.init(
9     # Set the project where this run will be logged
10     project="demo-linear-regression",
11     config={
12         "learning_rate": lr,
13         "epochs": epochs,
14     },
15 )

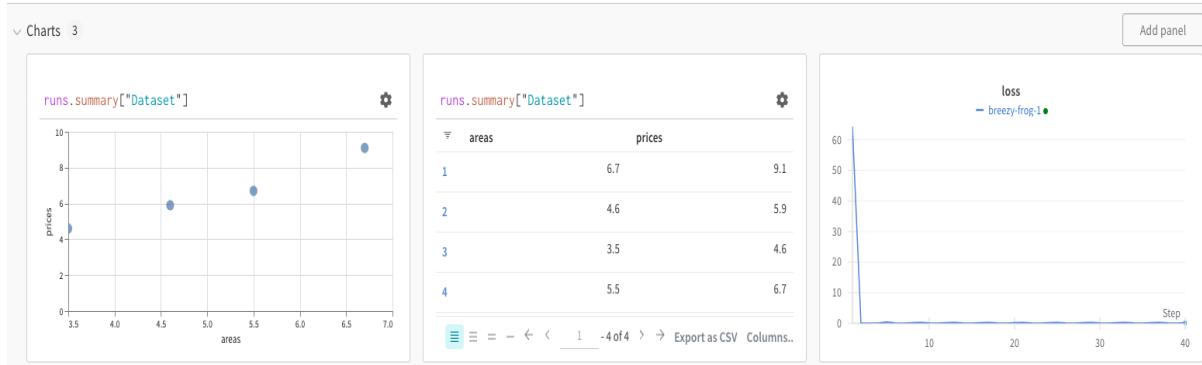
```

```

16
17 wandb.run.log({"Dataset" : wandb.Table(dataframe=dataset)})
18
19 X_train = dataset['areas']
20 Y_train = dataset['prices']
21
22 N = len(X_train)
23 # parameter
24 losses = [] # for debug
25
26 for epoch in range(epochs):
27     # for an epoch
28     for i in range(N):
29         # get a sample
30         x = X_train[i]
31         y = Y_train[i]
32
33         # predict y_hat
34         y_hat = predict(x, w, b)
35
36         # compute loss
37         loss = (y_hat-y)*(y_hat-y) / 2.0
38
39         # tracking loss with WandB
40         wandb.log({"loss": loss})
41
42         # compute gradient
43         (dw, db) = gradient(y_hat, y, x)
44
45         # update weights
46         (w, b) = update_weight(w, b, lr, dw, db)
47
48 # Mark a run as finished, and finish uploading all data.
49 wandb.finish()

```

- Trong quá trình training mô hình, chúng ta có thể theo dõi trực tiếp tại trang web của WandB, với kết quả training khi thực thi dòng lệnh trên như sau:

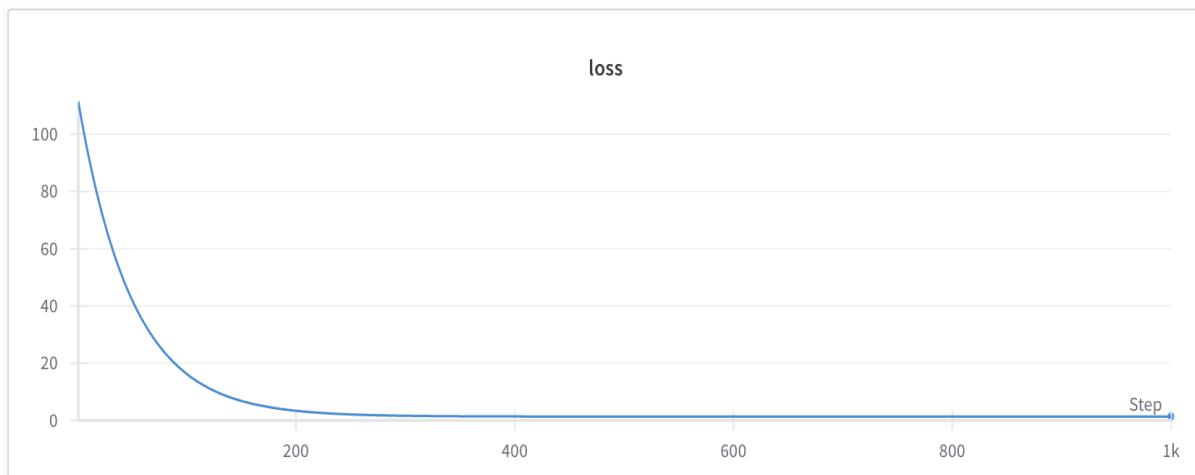


Hình 5: Kết quả theo dõi quá trình training mô hình Linear Regression

→ **Nhận xét:** Qua biểu đồ hàm loss được theo dõi ở trên, giúp chúng ta dễ dàng theo dõi quá trình training của mô hình.

3. **Bài tập:** Hãy đọc dữ liệu tại file sau: [advertising.csv](#). Xây dựng mô hình Linear Regression với 1000 epochs, khởi tạo các giá trị trọng số $w_1 = w_2 = w_3 = 0$, $b = 1$ và thực hiện tracking quá trình training với WandB cho tập data trên.

→ **Kết quả:**



Tables 1

	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	12
4	151.5	41.3	58.5	16.5
5	180.8	10.8	58.4	17.9
6	8.7	48.9	75	7.2
7	57.5	32.8	23.5	11.8

Hình 6: Kết quả theo dõi quá trình training mô hình Linear Regression

- Hết -

NÂNG CAO HIỆU QUẢ VIẾT CODE VỚI TYPE HINTS VÀ MYPY

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1 Mở đầu

Làm sao để viết code Python dễ đọc, dễ hiểu và dễ bảo trì hơn? Làm thế nào để có thể kiểm tra tính đúng đắn của code trước khi thực thi chúng? Và liệu có cách nào để phát hiện và sửa lỗi kiểu dữ liệu một cách dễ dàng và nhanh chóng không? Trong bài viết này sẽ hướng dẫn sử dụng Type Hints để xác định và gợi ý kiểu dữ liệu của các biến, giá trị trả về của các hàm, phương thức... Đồng thời kết hợp với Mypy-một công cụ kiểm tra kiểu dữ liệu static mạnh mẽ cho các chương trình Python, giúp nâng cao hiệu quả viết code trong quá trình phát triển dự án.

Yêu cầu:

- Máy tính đã cài đặt python ≥ 3.10 , Vscode.
- Đã biết lập trình Python cơ bản.



2 Hướng dẫn sử dụng Type Hints và Mypy

2.1 Hướng dẫn sử dụng Type Hints

Type Hints là một tính năng đã được tích hợp sẵn trong Python, cho phép ta khi viết code có thể khai báo kiểu dữ liệu của các biến, tham số và giá trị trả về của hàm. Và điều thú vị là khi thực thi chương trình Python trình thông dịch sẽ bỏ qua các gợi ý kiểu dữ liệu trong code. Chúng ta sẽ hiểu rõ hơn thông qua các ví dụ.

2.1.1 Type hints cho biến

Chúng ta sẽ bắt đầu bằng ví dụ tạo biến theo cách thông thường:

```
1 # Cách không dùng type hints
2 name = "AI VIETNAM"
3 year = 2024
4
5 print(f"Welcome to {name} {year}!")
```

```
===== Output =====
Welcome to AI VIETNAM 2024!
=====
```

Để gợi ý về kiểu dữ liệu của một biến, ta có thể sử dụng Type Hints như sau:

```

1 name: str = "AI VIETNAM"
2 year: int = 2024
3
4 print(f"Welcome to {name} {year}!")
5 print(__annotations__)

```

```

=====
Output =====
Welcome to AI VIETNAM 2024!
{'name': <class 'str'>, 'year': <class 'int'>}
=====
```

Trong ví dụ trên chúng ta khai báo kiểu dữ liệu cho name là string, year với kiểu int tuy nhiên khi thực thi chương trình thì đều cho kết quả giống nhau vì Python sẽ bỏ qua phần gợi ý kiểu dữ liệu. Khác biệt duy nhất là khi sử dụng type hints chương trình sẽ tạo ra một thuộc tính đặc biệt `__annotations__` để chứa thông tin về kiểu dữ liệu của các biến, điều này sẽ giúp IDE(Vscode) đưa ra những gợi ý cú pháp chính xác. Ngoài ra thuộc tính này sẽ là thông tin quan trọng để các công cụ hỗ trợ như Mypy có thể kiểm tra kiểu dữ liệu của chương trình.

Khi sử dụng type hints, có nhiều trường hợp gợi ý kiểu cho biến là kiểu list, tuple, boolean, hoặc biến đó có thể có nhiều kiểu dữ liệu khác nhau.

```

1 # Biến với Type Hint là list chứa các số nguyên
2 numbers: list[int]
3
4 # Biến với Type Hint là tuple chứa một chuỗi và một số nguyên
5 person: tuple[str, int]
6
7 # Biến với Type Hint là int hoặc float
8 value: int|float
9
10 # Biến với Type Hint là list chứa chuỗi và tuple chứa số nguyên
11 data: list[str]|tuple[int, int]
12
13 # Biến với Type Hint là Tuple chứa một list các số nguyên và một dict
14 info: tuple[list[int], dict]
15
16 # Biến với Type hint là boolean
17 is_active: bool

```

Trong ví dụ trên, chúng ta đã sử dụng Type Hints để chỉ định kiểu dữ liệu của các biến numbers, person, value, data, và info. Khi chúng ta tạo type hints cho biến mà không gán giá trị như vậy, thì các biến này vẫn chưa thực sự được tạo ra. Mà chỉ có `__annotations__` được tạo ra chứa thông tin gợi ý kiểu dữ liệu cho từng biến. Chúng ta có thể kiểm tra điều này bằng cách in biến đó ra để xem thông tin.

```

1 print(__annotations__)
2 print(numbers, person, value, data, info)

```

```

=====
Output =====
{'numbers': list[int], 'person': tuple[str, int], 'value': int | float, 'data': list[
    str] | tuple[int, int], 'info': tuple[list[int], dict], 'is_active': <class 'bool
    '>}
Traceback (most recent call last):
  File "create_type_hints.py", line 20, in <module>
    print(numbers, person, value, data, info, is_active)
NameError: name 'numbers' is not defined
=====
```

2.1.2 Type hints cho hàm

Để gợi ý về kiểu dữ liệu của các tham số và giá trị trả về của một hàm, ta có thể sử dụng Type Hints như ví dụ phía dưới. Trong hàm add, chúng ta gợi ý rằng x và y là các số nguyên (int) và hàm trả về là một số nguyên (int). Nhưng nếu ta cố tình sử dụng kiểu dữ liệu float thì chương trình vẫn hoạt động bình thường, không một cảnh báo lỗi nào xuất hiện.

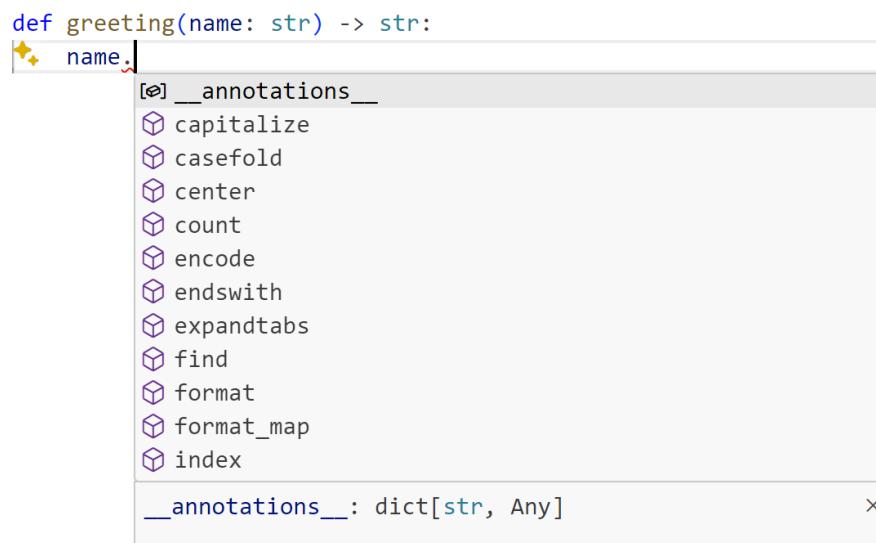
```

1 def add(x: int, y: int) -> int:
2     return x + y
3
4 if __name__ == "__main__":
5     print(add(x= 1.5, y=2))
6     print(add.__annotations__)

```

===== Output =====
3.5
{'x': <class 'int'>, 'y': <class 'int'>,
'return': <class 'int'>}
=====

Ưu điểm của việc sử dụng type hints là giúp code của chúng ta dễ đọc hơn, ngoài ra thì giúp công cụ IDE code gợi ý cú pháp hiệu quả hơn. Thông thường, khi chúng ta tạo hàm, nếu các tham số trong hàm không được gán giá trị mặc định thì IDE không biết kiểu dữ liệu của tham số đó là gì, nên không đưa ra gợi ý code cho chúng ta được. Nhưng khi sử dụng type hints, điều này lại được khắc phục.



Hình 1: Gợi ý code trong Vscode xuất hiện khi sử dụng type hints

2.1.3 Type hints cho class

Để sử dụng type hints cho class, đối với thuộc tính chúng ta tạo type hints cho chúng như cách làm với biến, đối với các phương thức thì chúng ta tạo type hints như cách chúng ta làm với hàm.

```

1 class Person:
2     name: str
3     age: int
4
5     def __init__(self, name: str, age: int) -> None:
6         self.name = name
7         self.age = age
8
9     def greet(self) -> str:
10        return f"Xin chào, Tôi là {self.name} năm nay tôi {self.age} tuổi."
11
12 if __name__ == "__main__":
13     person_1 = Person("Tom", 25)
14     print(person_1.greet())

```

===== Output =====
Tôi là Tom năm nay tôi 25 tuổi.
=====

Trong lớp Person trên, chúng ta gọi ý rằng thuộc tính name là một chuỗi (str), thuộc tính age là một số nguyên (int) và phương thức greet trả về một chuỗi (str).

Lưu ý: Việc tạo type hints trong code chương trình là không bắt buộc, không có chúng thì chương trình vẫn chạy bình thường. Vậy thì khi nào nên và không nên sử dụng type hints? Thì sẽ tùy vào từng dự án, nhóm bạn làm việc có muốn sử dụng type hints không. Nhìn chung, type hints có nhiều ưu điểm giúp code chương trình rõ ràng và dễ phát hiện và sửa lỗi bảo trì hơn so với nhược điểm như phải dành thêm nhiều thời gian hơn để viết mã.

2.2 Kết hợp Type Hints với Mypy

Type hints giúp code của chúng ta rõ ràng, dễ đọc và dễ sửa lỗi hơn, tuy nhiên việc phát hiện lỗi chúng ta vẫn thực hiện thủ công, người lập trình phải tự đọc hiểu và cố gắng kiểm soát các kiểu dữ liệu để đảm bảo tính logic nhưng điều này có thể gây mất nhiều thời gian mà không hiệu quả.

Công cụ mypy ra đời để khắc phục điều này, nó là một công cụ kiểm tra kiểu dữ liệu static Python, cho phép chúng ta kiểm tra kiểu dữ liệu trong chương trình mà không cần phải thực thi chúng. Nó được phát triển bởi Jukka Lehtosalo và được công bố dưới dạng mã nguồn mở.

Để sử dụng mypy, trước tiên chúng ta cần cài đặt nó thông qua câu lệnh sau:

```
1 pip install mypy
```

Sau khi cài đặt thành công, chúng ta có thể sử dụng mypy để kiểm tra kiểu dữ liệu trong chương trình của chúng ta bằng cách chạy lệnh:

```
1 mypy my_script.py
```

Trong đó my_script.py là file chứa code chương trình và các type hints. Ta cần lưu ý là mypy chỉ kiểm tra được cho tệp có đuôi .py, đối với tệp notebook.ipynb chúng ta cần sử dụng công cụ khác như nbqa. Trong phạm vi bài viết này, chúng ta chỉ kiểm tra kiểu đối với file .py.

Chúng ta cùng quay trở lại ví dụ tính tổng hai số, ta tạo type hints cho các tham số là kiểu int, nhưng trong ví dụ, chúng ta truyền giá trị cho hai tham số là kiểu float và int thì chương trình vẫn chạy bình thường. Dưới đây, chúng ta sẽ kiểm tra kiểu bằng mypy.

```
1 def add(x: int, y: int) -> int:
2     return x + y
3
4 if __name__ == "__main__":
5     print(add(x= 1, y=2))
6     print(add(x= 1.5, y=2))
7     print(add.__annotations__)
```

```
=====
mypy add.py
add.py:6: error: Argument "x" to "add" has incompatible type "float"; expected "int"
[arg-type]
Found 1 error in 1 file (checked 1 source file)
=====
```

Kết quả mypy trả về cho thấy trong file code của chúng ta add.py xuất hiện lỗi kiểu ở dòng 6 tức là dòng print(add(x= 1.5, y=2)), lỗi này do x=1.5 có kiểu là float không khớp với kiểu dữ liệu dự kiến từ type hints là int. Sau khi xác định lỗi, chúng ta dễ dàng sửa lỗi và chương trình sẽ trở lên đúng đắn hơn. Ta thấy mypy thật tiện lợi phải không?

3 Bài tập

Viết một chương trình Python để kiểm tra xem một số có phải là số nguyên tố hay không. Số nguyên tố là số nguyên dương lớn hơn 1 và chỉ có hai ước số dương là 1 và chính nó.

Yêu cầu:

- Viết một hàm có tên là `is_prime` nhận một số nguyên dương `n` và trả về một giá trị boolean. Nếu `n` là số nguyên tố, hàm sẽ trả về `True`, ngược lại trả về `False`.
- Sử dụng Type Hints để gợi ý về kiểu dữ liệu của tham số và giá trị trả về của hàm.
- Sử dụng `mypy` để kiểm tra kiểu dữ liệu trong chương trình

Numpy Array và Pytorch/Tensorflow Tensor

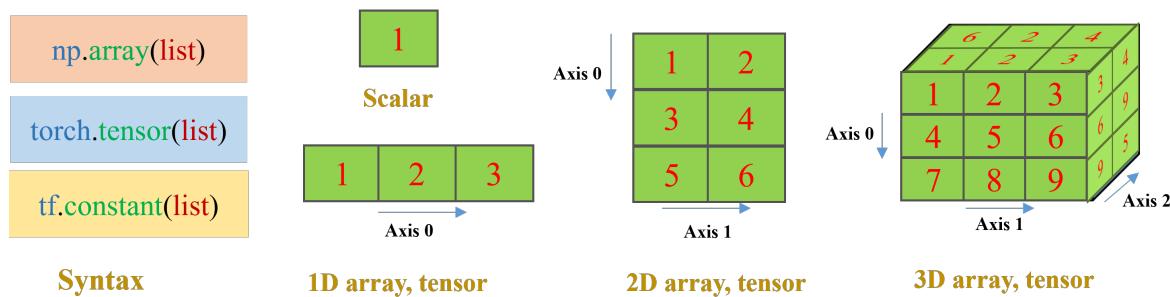
Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Array trong Numpy và Tensor trong các thư viện Pytorch, Tensorflow là những thành phần nền tảng cốt lõi cho việc tính toán của các thư viện này. Trong đó:

- **Array** là một cấu trúc dữ liệu đa chiều mạnh mẽ, giúp lưu trữ và thực hiện các phép toán toán học trên dữ liệu số. Array có thể là mảng 0 chiều (scalar), mảng một chiều (vector), mảng hai chiều (ma trận), hoặc nhiều chiều hơn.
- **Tensor trong Pytorch và Tensorflow** là một cấu trúc dữ liệu nhiều chiều, tương tự như array trong Numpy, nhưng được thiết kế để tương thích với học sâu và tính toán song song, đặc biệt là trên các thiết bị như GPU và TPU. Trong Pytorch và Tensorflow, tensors là đơn vị cơ bản để lưu trữ và xử lý dữ liệu.

Có nhiều cách để tạo tensor (hay array trong Numpy), ta có thể sử dụng cú pháp sau đây để khởi tạo chúng từ list Python.



Hình 1: Minh họa và cú pháp tạo Array và Tensor.

Khi làm việc với cấu trúc dữ liệu tensor, ta có thể kiểm tra thuộc tính của chúng thông qua một số phương thức sau:

- **shape** cho biết kích thước của mảng hoặc tensor, tức là số phần tử trong mỗi chiều của chúng.
- **dtype** cho biết kiểu dữ liệu của các phần tử trong mảng hoặc tensor.
- **type** cho biết kiểu của đối tượng mảng hoặc tensor.
- **device** chỉ có ở trong Pytorch và Tensorflow, và nó cho biết nơi lưu trữ tensor, có thể là CPU hoặc GPU.

2. Bài tập

Câu 1: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor từ danh sách 1 chiều?

Câu 2: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor từ danh sách 2 chiều. Sau đó thực hiện kiểm tra thuộc tính shape, dtype, type, device từ các array, tensor vừa tạo ?

3. Đáp án

Có nhiều cách khác nhau để ta tạo array hay tensor. Cách đầu tiên, ta tạo chúng từ list-kiểu dữ liệu quen thuộc trong Python.

```

1 import numpy as np
2 import torch
3 import tensorflow as tf
4
5 # Tạo một danh sách 1 chiều
6 list_1D = [1, 2, 3, 4, 5, 6, 7, 8, 9]
7
8 # Tạo một mảng 1 chiều từ danh sách
9 arr_1D = np.array(list_1D)
10 print("Mảng 1 chiều NumPy:\n",
11      arr_1D, type(arr_1D))
12
13 # Tạo một tensor PyTorch từ danh sách
14 tensor_1D_pt = torch.tensor(list_1D)
15 print("Tensor PyTorch 1 chiều:\n",
16      tensor_1D_pt)
17
18 # Tạo một tensor TensorFlow từ danh sách
19 tensor_1D_tf = tf.convert_to_tensor(
20             list_1D)
21 print("Tensor TensorFlow 1 chiều:\n",
22      tensor_1D_tf)

```

```

=====
Output =====
Mảng 1 chiều NumPy:
[1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>

Tensor PyTorch 1 chiều:
tensor([1, 2, 3, 4, 5, 6, 7, 8, 9])

Tensor TensorFlow 1 chiều:
tf.Tensor([1 2 3 4 5 6 7 8 9],
shape=(9,), dtype=int32)

=====
```

Ví dụ trên thực hiện tạo array, tensor từ một list Python. Để tạo array chúng ta dùng cú pháp `np.array(list_1D)`, với pytorch thì ta dùng `torch.tensor(list_1D)` và với Tensorflow ta dùng `tf.convert_to_tensor(list_1D)` hoặc `tf.constant(list_1D)`.

Đối với việc tạo array, tensor từ list 2D, chúng ta cũng thực hiện tương tự cách trên, Ví dụ Numpy:

```

1 # NumPy code
2 import numpy as np
3 # Tao một danh sách 2 chiều
4 list_2D = [[1, 2, 3],
5             [4, 5, 6],
6             [7, 8, 9]]
7 # Tạo một mảng 2 chiều
8 arr_2D = np.array(list_2D)
9 # Kiểm tra hình dạng, kiểu dữ liệu và loại
10 print("Hình dạng của mảng: ",
11       arr_2D.shape)
12 print("Kiểu dữ liệu của mảng: ",
13       arr_2D.dtype)
14 print("Loại của mảng: ",
15       type(arr_2D))
16 print("Mảng:\n", arr_2D)

```

```

=====
Output =====
Hình dạng của mảng: (3, 3)
Kiểu dữ liệu của mảng: int32
Loại của mảng: <class 'numpy.ndarray'>
Mảng:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

=====
```

Trong đoạn mã Numpy trên, chúng ta bắt đầu bằng việc tạo ra một danh sách 2D (`list_2D`) đại diện cho một ma trận có 3 hàng và 3 cột. Sau đó, chúng ta sử dụng Numpy để chuyển danh sách này thành một mảng 2 chiều (`arr_2D`). Dòng mã tiếp theo sử dụng các hàm tích hợp trong Numpy để in ra hình dạng (`shape`), kiểu dữ liệu (`dtype`), và kiểu của mảng (`type`). Cuối cùng, chúng ta in ra nội dung của mảng để kiểm tra kết quả

Ví dụ Pytorch:

```

1 # PyTorch code
2 import torch
3
4 # Tạo một danh sách 2 chiều
5 list_2D = [[1, 2, 3],
6             [4, 5, 6],
7             [7, 8, 9]]
8 # Tạo một tensor 2 chiều
9 tensor_2D_pt = torch.tensor(list_2D)
10
11 # Kiểm tra hình dạng, kiểu dữ liệu, loại,
12 # thiết bị lưu trữ của tensor
12 print("Hình dạng của tensor: ",
13       tensor_2D_pt.shape)
14 print("Kiểu dữ liệu của tensor: ",
15       tensor_2D_pt.dtype)
16 print("Loại của tensor: ",
17       type(tensor_2D_pt))
18 print("Thiết bị lưu trữ của tensor: ",
19       tensor_2D_pt.device)
20 print("Tensor:\n", tensor_2D_pt)

```

```

=====
Output
=====
Hình dạng của tensor: torch.Size([3, 3])
Kiểu dữ liệu của tensor: torch.int64
Loại của tensor: <class 'torch.Tensor'>
Thiết bị lưu trữ của tensor: cpu
Tensor:
tensor([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
=====
```

Trong đoạn mã Pytorch trên, chúng ta bắt đầu bằng cách tạo một danh sách 2D (**list_2D**) đại diện cho ma trận 3x3. Sau đó, chúng ta sử dụng Pytorch để chuyển đổi danh sách này thành tensor 2 chiều (**tensor_2D_pt**).

Dòng mã tiếp theo sử dụng các thuộc tính tích hợp trong Pytorch để in ra hình dạng (**shape**), kiểu dữ liệu (**dtype**), kiểu của tensor (**type**), và thiết bị lưu trữ của tensor (**device**). Cuối cùng, chúng ta in ra nội dung của tensor để kiểm tra kết quả.

Ví dụ Tensorflow:

```

1 import tensorflow as tf
2
3 # Tạo một danh sách 2 chiều
4 list_2D = [[1, 2, 3],
5             [4, 5, 6],
6             [7, 8, 9]]
7 # Tạo một tensor 2 chiều từ danh sách
8 tensor_2D_tf = tf.convert_to_tensor(
9                         list_2D)
10
11 # Kiểm tra hình dạng, kiểu dữ liệu, loại,
12 # và thiết bị mà tensor được lưu trữ
13 print("Hình dạng của tensor: ",
14       tensor_2D_tf.shape)
15 print("Kiểu dữ liệu của tensor: ",
16       tensor_2D_tf.dtype)
17 print("Loại của tensor: ",
18       type(tensor_2D_tf))
19 print("Thiết bị mà tensor được lưu trữ: ",
20       tensor_2D_tf.device)
21 print(tensor_2D_tf)

```

```

=====
Output
=====
Hình dạng của tensor: (3, 3)
Kiểu dữ liệu của tensor: <dtype: 'int32'>
Loại của tensor: <class 'tensorflow.python.framework.ops.EagerTensor'>
Thiết bị mà tensor được lưu trữ: /job:localhost/replica:0/task:0/device:CPU:0
tf.Tensor(
[[1 2 3]
 [4 5 6]
 [7 8 9]], shape=(3, 3), dtype=int32)
=====
```

Tương tự như Pytorch, trong đoạn mã Tensorflow trên, chúng ta bắt đầu bằng cách tạo một danh sách

2D (**list _ 2D**) đại diện cho ma trận 3x3. Sau đó, chúng ta sử dụng Tensorflow để chuyển đổi danh sách này thành một tensor 2 chiều (**tensor _ 2D _ tf**) bằng hàm **tf.convert _ to _ tensor()**.

Dòng mã tiếp theo sử dụng các thuộc tính tích hợp trong Tensorflow để in ra hình dạng (**shape**), kiểu dữ liệu (**dtype**), kiểu của tensor (**type**), và thiết bị lưu trữ của tensor (**device**). Cuối cùng, chúng ta in ra nội dung của tensor để kiểm tra kết quả.

Các Hàm Khởi Tạo Numpy Array và Pytorch/Tensorflow Tensor - Phần 1

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Khi lập trình với các thư viện Numpy, Pytorch, Tensorflow có một số cách để nhanh chóng tạo ra các array với giá trị, kích thước khác nhau. Trong bài tập này, chúng ta sẽ tìm hiểu các sử dụng 3 hàm **zeros**, **ones**, **full**.

a) **zeros**

zeros là hàm thực hiện chức năng tạo array, tensor toàn giá trị 0 với đầu vào là kích thước và kiểu dữ liệu ta muốn. Cả 3 thư viện đều sử dụng hàm trên, với cú pháp tương tự nhau.

np.zeros(shape)	zeros() function									
torch.zeros(shape)	<table border="1"> <tr> <td>0</td><td>1</td><td>2</td></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> </table>	0	1	2	0	0	0	1	0	0
0	1	2								
0	0	0								
1	0	0								
tf.zeros(shape)										

Hình 1: Minh họa và cú pháp sử dụng hàm zeros.

Nhìn chung, hàm **zeros** chủ yếu dùng để khởi tạo mảng hoặc tensor với các giá trị 0, thường được dùng trong quá trình xây dựng các mô hình máy học và thực hiện các phép toán số học.

b) **ones**

Tương tự với **zeros**, hàm **ones** tạo mảng chứa toàn số 1 với đầu vào là kích thước do người dùng chỉ định.

Syntax

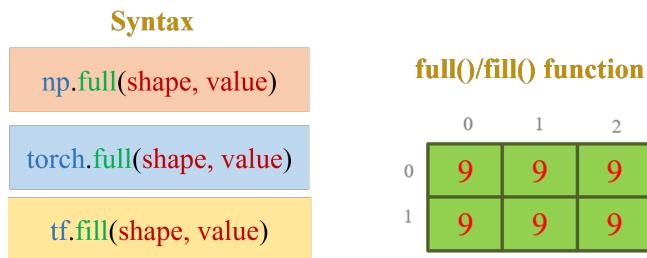
np.ones(shape)	ones() function									
torch.ones(shape)	<table border="1"> <tr> <td>0</td><td>1</td><td>2</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> </table>	0	1	2	0	1	1	1	1	1
0	1	2								
0	1	1								
1	1	1								
tf.ones(shape)										

Hình 2: Minh họa và cú pháp sử dụng hàm ones.

c) **full, fill**

Hàm **full** giúp chúng ta tạo array, tensor(Pytorch) với phần tử là giá trị chỉ định, đầu vào của hàm **full** gồm kích thước array và giá trị hằng số muốn tạo. Khác với hai thư viện trên, Tensorflow sử dụng hàm **fill**.

Hàm **full** với Numpy, Pytorch hay **fill** với Tensorflow giúp tạo array hoặc tensor với kích thước và giá trị được chỉ định, có ích khi ta cần khởi tạo các cấu trúc dữ liệu với giá trị đồng nhất.



Hình 3: Minh họa và cú pháp sử dụng hàm full/fill.

2. Bài tập

Câu 1: Hãy viết chương trình sử dụng hàm zeros tạo Numpy array, Tensorflow tensor, Pytorch tensor chỉ chứa giá trị là số 0 với kích thước (3, 4)?

Câu 2: Hãy viết chương trình sử dụng hàm ones tạo Numpy array, Tensorflow tensor, Pytorch tensor chỉ chứa giá trị là số 1 với kích thước (3, 4)?

Câu 3: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor chỉ chứa giá trị là số 5 với kích thước (3, 4)?

3. Đáp án

Câu 1: zeros

Chương trình sau tạo array với kích thước (3, 4), đối với thư viện Numpy sử dụng cú pháp **np.zeros**, bên trong hàm này chúng ta truyền vào kích thước array chúng ta mong muốn là (3, 4).

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo array toàn số 0
5 arr_zeros = np.zeros((3, 4))
6 print(arr_zeros)

```

```

=====
Output =====
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
=====
```

Tương tự như thư viện Numpy, Pytorch sử dụng cú pháp **torch.zeros**

```

1 #Pytorch code
2 import torch
3
4 # Tạo tensor toàn số 0
5 tensor_zeros = torch.zeros((3, 4))
6 print(tensor_zeros)

```

```

=====
Output =====
tensor([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
=====
```

Tương tự, Tensorflow sử dụng cú pháp **tf.zeros**

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor toàn số 0
5 tensor_zeros = tf.zeros((3, 4))
6 print(tensor_zeros)

```

```

=====
Output =====
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]], shape=(3, 4), dtype=
float32
=====
```

Câu 2: ones

Chương trình sau tạo array với kích thước (3, 4), đối với thư viện Numpy sử dụng cú pháp **np.ones**

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo array toàn số 1
5 arr_ones = np.ones((3, 4))
6 print(arr_ones)

```

```

=====
Output =====
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
=====
```

Với Pytorch và Tensorflow cú pháp thực hiện tương tự với cú pháp torch.ones, tf.ones

```

1 #PyTorch code
2 import torch
3
4 # Tạo tensor toàn số 1
5 tensor_ones = torch.ones((3, 4))
6 print(tensor_ones)

```

```

=====
Output =====
tensor([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
=====
```

```

1 #TensorFlow code
2 import tensorflow as tf
3
4
5 # Tạo tensor toàn số 1
6 tensor_ones = tf.ones((3, 4))
7 print(tensor_ones)

```

```

=====
Output =====
tf.Tensor(
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]], shape=(3, 4), dtype=
float32)
=====
```

Câu 3: full, fill

Chương trình sau tạo array với kích thước (3, 4), giá trị hằng số là 5. Đối với thư viện Numpy sử dụng cú pháp *np.full*

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo array toàn số 5
5 arr_full = np.full((3, 4), 5)
6 print(arr_full)

```

```

=====
Output =====
[[5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]]
=====
```

Tương tự như thư viện Numpy, Pytorch sử dụng cú pháp *torch.full*

```

1 #Pytorch code
2 import torch
3
4 # Tạo tensor toàn số 5
5 tensor_full = torch.full((3, 4), 5)
6 print(tensor_full)

```

```

=====
Output =====
tensor([[5, 5, 5, 5],
       [5, 5, 5, 5],
       [5, 5, 5, 5]])
=====
```

Khác với hai thư viện trên, Tensorflow sử dụng hàm **fill** để tạo một tensor với giá trị được chỉ định.

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor toàn số 5
5 tensor_full = tf.fill((3, 4), 5)
6 print(tensor_full)

```

```

=====
Output =====
tf.Tensor(
[[5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]], shape=(3, 4), dtype=int32)
=====
```

Các Hàm Khởi Tạo Numpy Array và Pytorch/Tensorflow Tensor - Phần 2

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Trong bài tập này, chúng ta tiếp tục tìm hiểu các hàm có chức năng tạo array, tensor đặc biệt. Chúng ta sẽ tìm hiểu và sử dụng các hàm **arange**, **range**, **eye**, và **random**.

a) **arange**, **range**

Trong Numpy, Pytorch, hàm **arange** được sử dụng để tạo một mảng chứa dãy số có giá trị tăng dần với khoảng cách cố định. Trong Tensorflow chúng ta sẽ sử dụng hàm **range**. Cú pháp như sau:

Syntax	arange() / range() function
<code>np.arange(start=0, stop, step=1)</code>	
<code>torch.arange(start=0, stop, step=1)</code>	$\text{arr1} = \begin{array}{ c c c c c } \hline 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 2 & & \\ \hline \end{array}$
<code>tf.range(start=0, stop, step=1)</code>	$\text{arr2} = \begin{array}{ c c c } \hline 0 & 2 & 4 \\ \hline \end{array}$

Hình 1: Minh họa và cú pháp sử dụng hàm arange, range.

Hàm **arange** (hoặc **range** trong Tensorflow) rất hữu ích khi ta muốn tạo ra một dãy số tăng dần với các giá trị cách đều nhau.

b) **eye**

Hàm **eye** được sử dụng để tạo ma trận đơn vị, tức là một ma trận vuông có tất cả các phần tử trên đường chéo chính có giá trị 1, các phần tử còn lại là 0. Cú pháp sử dụng như sau:

Syntax	eye() function
<code>np.eye(N, M)</code>	
<code>torch.eye(N, M)</code>	
<code>tf.eye(N, M)</code>	$\begin{array}{ c c c } \hline 0 & 1 & 2 \\ \hline 1 & 1 & 0 \\ \hline 2 & 0 & 1 \\ \hline \end{array}$ <p>N: number of rows M: number of columns defaults to N</p>

Hình 2: Minh họa và cú pháp sử dụng hàm eye.

Hàm **eye** giúp tạo array hoặc tensor đơn vị, đặc biệt hữu ích trong nhiều ứng dụng toán học và xử lý dữ liệu.

c) **random**

Trong thư viện Numpy, Pytorch cung cấp hàm **rand** để tạo mảng với các giá trị ngẫu nhiên trong khoảng [0.0, 1.0] với kích cỡ (shape) được chỉ định. Ngoài ra có thể sử dụng hàm **randint** để tạo mảng với các số nguyên ngẫu nhiên. Đối với thư viện Tensorflow chúng ta sử dụng hàm **random.uniform**.

Syntax	rand() function	Syntax	randint() function
np.random.rand(shape)	0 1 2 0 0.574 0.682 0.704 2 0.806 0.844 0.799	np.random.randint(low, hight, shape) torch.randint(low, hight, shape) tf.random.uniform(shape, minval, maxval, dtype)	0 1 2 0 6 3 9 2 0 1 -5
torch.rand(shape)			
tf.random.uniform(shape, minval, maxval, dtype)			

Hình 3: Minh họa và cú pháp sử dụng hàm random.

Các hàm này giúp việc tạo dữ liệu ngẫu nhiên trở nên dễ dàng trong quá trình phát triển mô hình Machine Learning, Deep Learning.

2. Bài tập

Câu 1: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor trong khoảng [0, 10) với step=1?

Câu 2: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor là ma trận đơn vị với kích thước (3, 3).

Câu 3: Hãy viết chương trình tạo Numpy array, Tensorflow tensor, Pytorch tensor ngẫu nhiên trong khoảng giá trị [0, 1] với kích thước (3, 4). Tiếp theo hãy tạo array, tensor với các giá trị là số nguyên trong khoảng [-10, 10]. Lưu ý trong bài tập này, các bạn sẽ sử dụng seed = 2024 để đảm bảo kết quả giống đáp án, cách sử dụng như sau:

```

1 # Numpy code
2 import numpy as np
3 np.random.seed(2024)
4
5 # Pytorch code
6 import torch
7 torch.manual_seed(2024)
8
9 # Tensorflow code
10 import tensorflow as tf
11 tf.random.set_seed(2024)

```

3. Đáp án

Câu 1: *arange, range*

Trong numpy chúng ta sử dụng hàm **np.arange**

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo array trong khoảng [0, 10),
5 # bước nhảy 1
6 arr_arange = np.arange(10)
7 print(arr_arange)

```

```

=====
Output =====
[0 1 2 3 4 5 6 7 8 9]
=====
```

Trong PyTorch, để tạo tensor chứa dãy số, ta sử dụng hàm **torch.arange**:

```
1 #Pytorch code
2 import torch
3
4 # Tạo tensor trong khoảng [0, 10],
5 # bước nhảy 1
6 tensor_arange = torch.arange(10)
7 print(tensor_arange)
```

```
===== Output =====
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
=====
```

Trong Tensorflow, ta có thể sử dụng hàm **tf.range** để thực hiện chức năng tương tự:

```
1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor trong khoảng [0, 10],
5 # bước nhảy 1
6 tensor_arange = tf.range(10)
7 print(tensor_arange)
```

```
===== Output =====
tf.Tensor([0 1 2 3 4 5 6 7 8 9], shape=(10,), dtype=int32)
=====
```

Câu 2: *eye*

Trong numpy chúng ta sử dụng hàm `np.eye` để tạo ma trận đơn vị, trong đó giá trị chúng ta truyền vào là số lượng hàng của ma trận đầu ra, số lượng cột mặc định bằng số lượng hàng, nếu muốn số lượng cột khác bạn cần truyền vào hàm `eye` số lượng cột chỉ định.

```
1 # Numpy code
2 import numpy as np
3
4 # Tạo array với đường chéo chính là 1,
5 # còn lại là 0
6 arr_eye = np.eye(3)
7 print(arr_eye)
```

```
===== Output =====
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
=====
```

Trong Pytorch, để tạo tensor đơn vị, ta sử dụng hàm **torch.eye**:

```
1 #Pytorch code
2 import torch
3
4 # Tạo tensor với đường chéo chính là 1,
5 # còn lại là 0
6 tensor_eye = torch.eye(3)
7 print(tensor_eye)
```

```
===== Output =====
tensor([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])
=====
```

Trong Tensorflow, hàm **eye** cũng được sử dụng để tạo constant tensor đơn vị:

```
1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor với đường chéo chính là 1,
5 # còn lại là 0
6 tensor_eye = tf.eye(3)
7 print(tensor_eye)
```

```
===== Output =====
tf.Tensor(
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]], shape=(3, 3), dtype=float32)
=====
```

Câu 3: *random*

Trong Numpy, chúng ta sẽ sử dụng hàm **random.rand** để tạo mảng với các giá trị ngẫu nhiên trong khoảng giá trị mặc định là [0, 1). Để tạo mảng với giá trị ngẫu nhiên là số nguyên chúng ta dùng **random.randint**, khi sử dụng hàm này ta cần truyền vào khoảng giá trị lấy ngẫu nhiên và kích thước đầu ra mong muốn.

```

1 # Numpy code
2 import numpy as np
3
4 # Đặt seed để đảm bảo kết quả ngẫu nhiên
5 # có thể tái tạo được
6 np.random.seed(2024)
7
8 # Tạo một mảng với các giá trị ngẫu nhiên
9 # trong khoảng [0, 1) với kích thước (3,
10 # 4)
11 arr_rand = np.random.rand(3, 4)
12 print("Mảng ngẫu nhiên trong khoảng
13     [0, 1):\n", arr_rand)
14
15 # Tạo một mảng với các giá trị ngẫu nhiên
16 # trong khoảng [-10, 10)
17 arr_randint = np.random.randint(-10, 10,
18                                 size=(3, 4))
19 print("Mảng ngẫu nhiên trong khoảng
20     [-10, 10):\n", arr_randint)

```

===== Output =====

```

Mảng ngẫu nhiên trong khoảng [0, 1):
[[0.58801452 0.69910875 0.18815196
  0.04380856]
 [0.20501895 0.10606287 0.72724014
  0.67940052]
 [0.4738457 0.44829582 0.01910695
  0.75259834]]

Mảng ngẫu nhiên trong khoảng [-10, 10):
[[ 5  1 -3  8]
 [-1 -4  0 -9]
 [-5  9 -6 -2]]

```

=====

Trong Pytorch, các hàm tương tự cũng có sẵn thông qua module torch. **torch.rand** tạo tensor với các giá trị ngẫu nhiên từ phân phối đều trong khoảng [0.0, 1.0], **torch.randint** dùng để tạo tensor với các số nguyên ngẫu nhiên trong một khoảng cụ thể.

```

1 # Pytorch code
2 import torch
3
4 # Thiết lập seed để đảm bảo kết quả ngẫu
5 # nhiên có thể tái tạo được
6 torch.manual_seed(2024)
7
8 # Tạo tensor với các giá trị ngẫu nhiên
9 # trong khoảng [0, 1) với kích thước (3,
10 # 4)
11 tensor_rand = torch.rand((3, 4))
12 print("Tensor ngẫu nhiên trong khoảng
13     [0, 1):\n", tensor_rand)
14
15 # Tạo tensor với các giá trị ngẫu nhiên
16 # trong khoảng [-10, 10)
17 tensor_randint = torch.randint(-10, 10,
18                                 size=(3, 4))
19 print("Tensor ngẫu nhiên trong khoảng
20     [-10, 10):\n", tensor_randint)

```

===== Output =====

```

Tensor ngẫu nhiên trong khoảng [0, 1):
tensor([[0.5317, 0.8313, 0.9718, 0.1193],
        [0.1669, 0.3495, 0.2150, 0.6201],
        [0.4849, 0.7492, 0.1521, 0.5625]])

Tensor ngẫu nhiên trong khoảng [-10, 10):
tensor([[ 1,  8,  0, -2],
        [-9, -10, -9,  0],
        [-3, -7, -4,  8]])

```

=====

Trong Tensorflow, ta cũng có các hàm tương tự là **tf.random.uniform** để tạo tensor với các giá trị ngẫu nhiên trong khoảng [0.0, 1.0], và sử dụng **tf.random.uniform** với **dtype = tf.dtypes.int32** để tạo tensor với các số nguyên ngẫu nhiên trong một khoảng cụ thể.

```
1 #TensorFlow code
2 import tensorflow as tf
3
4
5 # Thiết lập seed để đảm bảo kết quả ngẫu
6 # nhiên có thể tái tạo được
7 tf.random.set_seed(2024)
8
9 # Tạo tensor với các giá trị ngẫu nhiên
10 # trong khoảng [0, 1)
11 tensor_rand = tf.random.uniform((3, 4))
12 print("Tensor ngẫu nhiên trong khoảng
13     [0, 1):\n", tensor_rand)
14
15 # Tạo tensor với các giá trị ngẫu nhiên
16 # trong khoảng [-10, 10)
17 tensor_randint = tf.random.uniform((3, 4),
18                                   minval=-10, maxval=10,
19                                   dtype=tf.dtypes.int32)
20 print("Tensor ngẫu nhiên trong khoảng
21     [-10, 10):\n", tensor_randint)
```

```
=====
Tensor ngẫu nhiên trong khoảng [0, 1):
tf.Tensor(
[[0.90034294 0.19453335 0.36069036
 0.66361904]
[0.76605344 0.2159369  0.6261736
 0.07380784]
[0.22062695 0.934368   0.93327904
 0.69267046]], shape=(3, 4), dtype=float32)

Tensor ngẫu nhiên trong khoảng [-10, 10):
tf.Tensor(
[[-3 -7  9  3]
 [ 2 -5 -3 -5]
 [ 4 -3 -3  5]], shape=(3, 4), dtype=int32)
=====
```

Numpy, Pytorch và Tensorflow

Hàm Stack và Concatenate

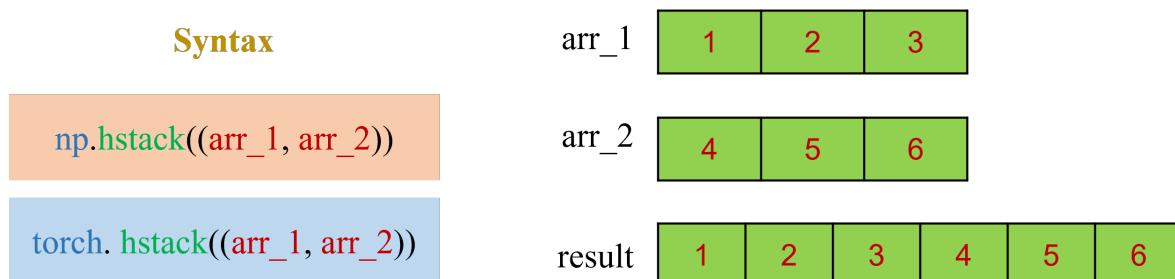
Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Trong các thư viện Numpy, Pytorch, Tensorflow các hàm **hstack**, **vstack**, và **concatenate** được sử dụng để nối các array hoặc các tensor lại với nhau theo các hướng khác nhau. Trong bài tập này, chúng ta sẽ tìm hiểu cách sử dụng ba hàm trên.

a) **hstack**

Hàm **hstack** trong NumPy, PyTorch được sử dụng để nối các array hoặc tensor theo chiều ngang, tức là nối chúng thành một cấu trúc dữ liệu lớn hơn theo chiều thứ hai. Quá trình này giúp kết hợp dữ liệu từ các nguồn khác nhau hoặc mở rộng kích thước của cấu trúc dữ liệu hiện có.

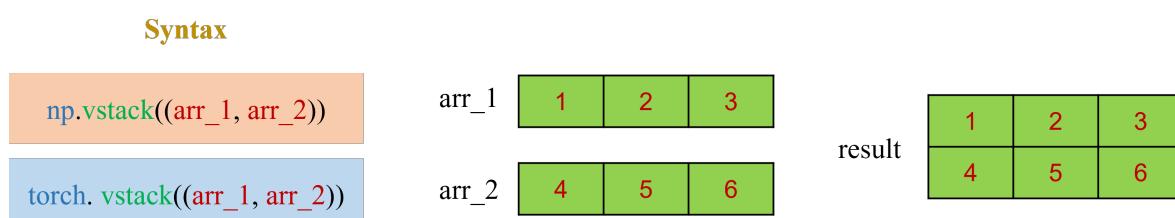


Hình 1: Minh họa và cú pháp sử dụng hstack

Lưu ý khi sử dụng **hstack** là các array hoặc tensor cần phải có cùng độ dài trong chiều dọc (cùng số hàng). Nếu các array/tensor không có cùng độ dài trong chiều dọc sẽ gây ra lỗi khi thực thi chương trình.

b) **vstack**

Hàm **vstack** trong NumPy, PyTorch được sử dụng để nối các array hoặc tensor theo chiều dọc, tức là nối chúng thành một cấu trúc dữ liệu lớn hơn theo chiều thứ nhất. Quá trình này giúp kết hợp dữ liệu từ các nguồn khác nhau hoặc mở rộng kích thước của cấu trúc dữ liệu hiện có.

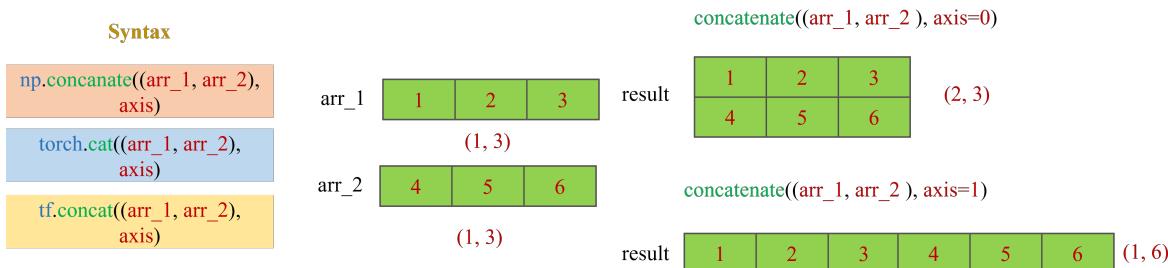


Hình 2: Minh họa và cú pháp sử dụng vstack

Lưu ý khi sử dụng **vstack** là các array hoặc tensor cần phải có cùng số cột. Nếu các array/tensor không có cùng số cột sẽ gây ra lỗi khi thực thi chương trình.

c) **concatenate**

Trong Tensorflow không có hàm **hstack** và **vstack**, tuy nhiên chúng ta có thể sử dụng hàm **concatenate** để thay thế. Khác với hai hàm trên, hàm **concatenate** được sử dụng để nối các array hoặc tensor theo một chiều cụ thể.



Hình 3: Minh họa và cú pháp sử dụng concatenate

Khi sử dụng concatenate các bạn cần lưu ý, đối với việc nối hai array, tensor 1D theo chiều 1 sẽ gấp lõi, lí do là vì các array, tensor 1D chỉ có 1 chiều là chiều 0. Chính vì vậy mà ta cần chuyển đổi chúng sang dạng 2D trước khi nối chúng lại theo chiều 1. Một lựa chọn dễ dàng hơn là sử dụng vstack.

2. Bài tập

Câu 1: Hãy viết chương trình tạo Numpy 2 array, Pytorch tensor với từ list 1 có giá trị [1, 2, 3], list 2 có giá trị [4, 5, 6]. Sau đó sử dụng hàm **hstack** để nối theo trực ngang và **vstack** để nối theo trực dọc?

Câu 2: Hãy viết chương trình tạo Numpy 2 array, Pytorch tensor, Tensorflow tensor từ list 2D là [[1 , 2 , 3] , [4 , 5 , 6]] và [[7 , 8 , 9] , [10 , 11 , 12]]. Sau đó sử dụng hàm **concatenate** để nối theo trực ngang và vstack để nối theo trực dọc?

3. Đáp án

Câu 1: **hstack**

Chương trình sử dụng hstack được sử dụng để nối các array theo chiều ngang trong Numpy:

```

1 #Numpy code
2 import numpy as np
3
4 # Tạo hai mảng
5 arr_1 = np.array([1, 2, 3])
6 arr_2 = np.array([4, 5, 6])
7
8 # Nối hai mảng theo trực ngang (axis=0)
9 arr_3 = np.hstack((arr_1, arr_2))
10
11 # In kết quả
12 print("Mảng 1:\n", arr_1)
13 print("Mảng 2:\n", arr_2)
14 print("Mảng sau khi nối theo trực ngang:\n",
      " ", arr_3)

```

```

=====
Output =====
[1 2 3 4 5 6]
=====
```

Trong Pytorch sử dụng tương tự như Numpy với cú pháp **torch.hstack**.

```

1 #Pytorch code
2 import torch
3
4 # Tạo hai tensor
5 tensor_1 = torch.tensor([1, 2, 3])
6 tensor_2 = torch.tensor([4, 5, 6])
7
8 # Nối hai tensor theo trục ngang
9 tensor_3 = torch.hstack((tensor_1,
   tensor_2))
10 print("Tensor 1:\n", tensor_1)
11 print("Tensor 2:\n", tensor_2)
12 print("Tensor sau khi nối theo trục ngang
   :\n", tensor_3)

```

```

=====
Output =====
Tensor 1:
 tensor([1, 2, 3])

Tensor 2:
 tensor([4, 5, 6])

Tensor sau khi nối theo trục ngang:
 tensor([1, 2, 3, 4, 5, 6])

=====

```

Câu 1: *vstack*

Hàm **vstack** được sử dụng để nối (hoặc "stack") các array hoặc tensor theo chiều dọc (theo trục hàng). Trong Numpy, **vstack** được sử dụng để nối các array theo chiều dọc. Chương trình:

```

1 # Numpy code
2 import numpy as np
3 # Tạo hai array
4 arr_1 = np.array([1, 2, 3])
5 arr_2 = np.array([4, 5, 6])
6 # Nối hai array theo chiều dọc
7 arr_3 = np.vstack((arr_1, arr_2))
8 # In kết quả
9 print("Array 1:\n", arr_1)
10 print("Array 2:\n", arr_2)
11 print("Array sau khi nối theo chiều dọc:\n",
   arr_3)

```

```

=====
Output =====
Array 1:
 [1 2 3]

Array 2:
 [4 5 6]

Array sau khi nối theo chiều dọc:
 [[1 2 3]
 [4 5 6]]

=====

```

```

1 #Pytorch code
2 import torch
3 # Tạo hai tensor
4 tensor_1 = torch.tensor([1, 2, 3])
5 tensor_2 = torch.tensor([4, 5, 6])
6 # Nối hai tensor theo chiều dọc bằng cách
   sử dụng torch.vstack
7 tensor_3 = torch.vstack((tensor_1,
   tensor_2))
8 # In kết quả
9 print("Tensor 1:\n", tensor_1)
10 print("Tensor 2:\n", tensor_2)
11 print("Tensor sau khi nối theo chiều dọc:\n",
   tensor_3)

```

```

=====
Output =====
Tensor 1:
 tensor([1, 2, 3])

Tensor 2:
 tensor([4, 5, 6])

Tensor sau khi nối theo chiều dọc:
 tensor([[1, 2, 3],
 [4, 5, 6]])

=====

```

Câu 2: Concatenate

Trong Numpy, **concatenate** được sử dụng để nối các array theo chiều ngang và dọc. Chương trình:

```

1 # Numpy code
2 import numpy as np
3 # Tạo hai array 2D
4 arr_1 = np.array([[1, 2, 3],
5                 [4, 5, 6]])
6 arr_2 = np.array([[7, 8, 9],
7                 [10, 11, 12]])
8 # Nối hai array theo chiều dọc axis = 0
9 arr_3 = np.concatenate((arr_1, arr_2),
10                      axis=0)
11 # Nối hai array theo chiều ngang axis = 1
12 arr_4 = np.concatenate((arr_1, arr_2),
13                      axis=1)
14 # In kết quả
15 print("Array 1:\n", arr_1)
16 print("Array 2:\n", arr_2)
17 print("Nối theo chiều dọc (axis=0):\n",
18       arr_3)
19 print("Nối theo chiều ngang (axis=1):\n",
20       arr_4)

```

```

=====
Output =====
Array 1:
[[1 2 3]
 [4 5 6]]

Array 2:
[[ 7   8   9]
 [10 11 12]]

Nối theo chiều dọc (axis=0):
[[ 1   2   3]
 [ 4   5   6]
 [ 7   8   9]
 [10 11 12]]

Nối theo chiều ngang (axis=1):
[[ 1   2   3   7   8   9]
 [ 4   5   6   10 11 12]]
=====
```

Trong chương trình trên ta sử dụng **np.concatenate** để nối hai array theo chiều dọc (axis=0) và chiều ngang (axis=1). Trong đó **arr_1** và **arr_2** là hai array 2D được tạo bằng NumPy. Tiếp theo, **arr_3** được tạo bằng cách nối **arr_1** và **arr_2** theo chiều dọc (axis=0), nghĩa là nối theo hàng. Cuối cùng **arr_4** được tạo bằng cách nối **arr_1** và **arr_2** theo chiều ngang (axis=1), nghĩa là nối theo cột.

Trong Pytorch, để thực hiện nối tensor theo một chiều cụ thể, ta sử dụng **torch.cat** với tham số **dim**, trong đó **dim = 1** sẽ nối theo chiều ngang, **dim = 0** sẽ nối theo chiều dọc:

```

1 #Pytorch code
2 import torch
3 # Tạo hai tensor 2D
4 tensor_1 = torch.tensor([[1, 2, 3],
5                         [4, 5, 6]])
6 tensor_2 = torch.tensor([[7, 8, 9],
7                         [10, 11, 12]])
8 # Nối hai tensor theo chiều dọc axis = 0
9 tensor_3 = torch.cat((tensor_1, tensor_2),
10                      dim=0)
11 # Nối hai tensor theo chiều ngang axis = 1
12 tensor_4 = torch.cat((tensor_1, tensor_2),
13                      dim=1)
14 # In kết quả
15 print("Tensor 1:\n", tensor_1)
16 print("Tensor 2:\n", tensor_2)
17 print("Nối theo chiều dọc (axis=0):\n",
18       tensor_3)
19 print("Nối theo chiều ngang (axis=1):\n",
20       tensor_4)

```

```

=====
Output =====
Tensor 1:
tensor([[1, 2, 3],
       [4, 5, 6]])

Tensor 2:
tensor([[ 7,  8,  9],
       [10, 11, 12]])

Nối theo chiều dọc (axis=0):
tensor([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])

Nối theo chiều ngang (axis=1):
tensor([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
=====
```

Trong Tensorflow, **concatenate** tương đương với **tf.concat** với tham số **axis** để chỉ định kết nối theo chiều nào:

```
1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo hai tensor 2D
5 tensor_1 = tf.constant([[1, 2, 3],
6                         [4, 5, 6]])
7 tensor_2 = tf.constant([[7, 8, 9],
8                         [10, 11, 12]])
9
10 # Nối hai tensor theo chiều dọc axis = 0
11 tensor_3 = tf.concat((tensor_1, tensor_2),
12                      axis=0)
13
14 # Nối hai tensor theo chiều ngang axis = 1
15 tensor_4 = tf.concat((tensor_1, tensor_2),
16                      axis=1)
17
18 # In kết quả
19 print("Tensor 1:\n", tensor_1)
20 print("Tensor 2:\n", tensor_2)
21 print("Nối theo chiều dọc (axis=0):\n",
22       tensor_3)
23 print("Nối theo chiều ngang (axis=1):\n",
24       tensor_4)
```

```
===== Output =====
Tensor 1:
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)

Tensor 2:
tf.Tensor(
[[ 7   8   9]
 [10 11 12]], shape=(2, 3), dtype=int32)

Nối theo chiều dọc (axis=0):
tf.Tensor(
[[ 1   2   3]
 [ 4   5   6]
 [ 7   8   9]
 [10 11 12]], shape=(4, 3), dtype=int32)

Nối theo chiều ngang (axis=1):
tf.Tensor(
[[ 1   2   3   7   8   9]
 [ 4   5   6   10 11 12]], shape=(2, 6),
dtype=int32)
=====
```

Hàm **concatenate** là một công cụ linh hoạt, cho phép ta nối các array hoặc tensor theo bất kỳ trục nào mà ta mong muốn.

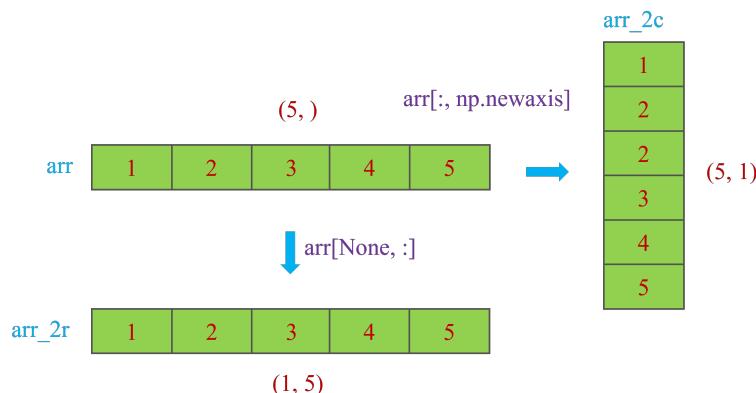
Các Hàm Quan Trọng Trong Numpy, Pytorch, Tensorflow - Phần 2

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

a) Thêm chiều dữ liệu

Khi thực hiện tính toán với array, tensor, chúng ta sẽ thường xuyên phải điều chỉnh số chiều của chúng. Để thêm một chiều mới, chúng ta sử dụng `newaxis` hoặc `expand_dims`.



Hình 1: Minh họa cách thêm chiều dữ liệu

Ví dụ sau đây thực hiện thêm một chiều mới vào array sử dụng **newaxis**:

```
1 #Numpy code
2 import numpy as np
3 # Tạo một array 1D
4 arr_1 = np.array([1, 2, 3])
5 # 1D -> 2D
6 arr_2 = arr_1[np.newaxis, :]
7 # 2D -> 5D
8 arr_3 = arr_2[np.newaxis, :, np.newaxis,
                  :, np.newaxis]
9 print("array 1: ", arr_1, arr_1.shape)
10 print("array 2: ", arr_2, arr_2.shape)
11 print("array 3:\n ", arr_3, arr_3.shape)
```

```
===== Output =====
array 1: [1 2 3] (3,)

array 2: [[1 2 3]] (1, 3)

array 3:
[[[[[1]
[2]
[3]]]]] (1, 1, 1, 3, 1)

=====
```

Trong ví dụ trên, ta tạo một array 1D có giá trị [1, 2, 3]. Tiếp theo ta sử dụng `np.newaxis` để thêm một chiều mới ở vị trí 0, chuyển từ array 1D thành array 2D. Cuối cùng ta dùng `np.newaxis` để thêm chiều mới, chuyển từ array 2D thành array 5D. Các chiều mới được thêm vào ở vị trí 0, 2, và 4.

Ở đây, dấu ":" Thể hiện chiều của array cũ, dấu hai chấm đầu tiên là chiều thứ nhất, dấu hai chấm thứ hai là chiều thứ 2 của array cũ. Ta có thể đặt dấu hai chấm này ở các vị trí khác nhau, tùy thuộc vào mục đích tạo array mới. `"np.newaxis"` là chiều mới ta muốn tạo, có thể đặt ở các vị trí khác nhau, ta cũng có thể thay thế nó bằng `"None"` hoặc `"..."`.

Ngoài ra ta cũng có thể sử dụng cách thứ hai, **np.expand_dims** để thêm chiều mới cho array.

```

1 # Numpy code
2 import numpy as np
3 # Tạo một array 1D
4 arr_1 = np.array([1, 2, 3])
5 # Thêm chiều mới, chuyển từ 1D -> 2D
6 arr_2 = np.expand_dims(arr_1, axis=0)
7 # Thêm nhiều chiều mới, chuyển từ 2D -> 5D
8 arr_3 = np.expand_dims(arr_2,
9                         axis=(0, 2, 4))
10 # In kết quả
11 print("array 1:", arr_1, arr_1.shape)
12 print("array 2:", arr_2, arr_2.shape)
13 print("array 3:\n", arr_3, arr_3.shape)

```

```

=====
Output =====
array 1: [1 2 3] (3,)

array 2: [[1 2 3]] (1, 3)

array 3:
[[[[[1]
   [2]
   [3]]]]] (1, 1, 1, 3, 1)

=====
```

Trong Pytorch, Tensorflow, cách thêm chiều cũng thực hiện tương tự. Dưới đây là ví dụ Pytorch:

```

1 #Pytorch code
2 import torch
3 # Tạo một tensor 1D
4 tensor_1 = torch.tensor([1, 2, 3])
5 # Thêm chiều mới, chuyển từ 1D -> 2D
6 tensor_2 = tensor_1[None, :]
7 # Thêm nhiều chiều mới, chuyển từ 2D -> 5D
8 tensor_3 = tensor_2[None, :, None, :, None
9                      ]
9 # In kết quả
10 print("tensor 1:", tensor_1, tensor_1.
11       shape)
11 print("tensor 2:", tensor_2, tensor_2.
12       shape)
12 print("tensor 3:\n", tensor_3, tensor_3.
13       shape)

```

```

=====
tensor 1: tensor([1, 2, 3])
torch.Size([3])

tensor 2: tensor([[1, 2, 3]])
torch.Size([1, 3])

tensor 3:
tensor([[[[1],
   [2],
   [3]]]]) torch.Size([1, 1, 1, 3, 1])

=====
```

Ví dụ Tensorflow:

```

1 #TensorFlow code
2 import tensorflow as tf
3 # Tạo một tensor 1D
4 tensor_1 = tf.constant([1, 2, 3])
5 # Thêm chiều mới, chuyển từ 1D -> 2D
6 tensor_2 = tf.expand_dims(tensor_1,
7                           axis=0)
8 # Thêm nhiều chiều mới, chuyển từ 2D -> 5D
9 tensor_3 = tensor_2[None, :, None, :, None
10                      ]
10 # In kết quả
11 print("tensor 1:", tensor_1, tensor_1.
12       shape)
12 print("tensor 2:", tensor_2, tensor_2.
13       shape)
13 print("tensor 3:\n", tensor_3, tensor_3.
14       shape)

```

```

=====
Output =====
tensor 1: tf.Tensor([1 2 3], shape=(3,), dtype=int32) (3,)

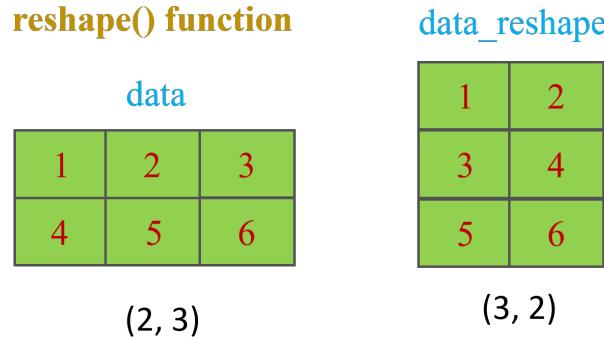
tensor 2: tf.Tensor([[1 2 3]], shape=(1, 3), dtype=int32) (1, 3)

tensor 3:
tf.Tensor(
[[[[[1]
   [2]
   [3]]]]], shape=(1, 1, 1, 3, 1),
dtype=int32) (1, 1, 1, 3, 1)

=====
```

b) *reshape*

reshape là một hàm được sử dụng để thay đổi hình dạng của một array hoặc tensor mà không làm thay đổi dữ liệu bên trong nó. Hình dạng mới được chỉ định thông qua tham số của hàm **reshape**.



Hình 2: Minh họa cách sử dụng hàm reshape

Cú pháp sử dụng reshape:

```

1 # Numpy
2 np.reshape(input, newshape)
3
4 # Pytorch
5 torch.reshape(input, newshape)
6
7 # Tensorflow
8 torch.reshape(input, newshape)

```

Trong đó:

- input: là array, tensor đầu vào.
- newshape là hình dạng mới ta muốn chuyển đổi.

Trong ví dụ sau, ta sẽ sử dụng **reshape** để giảm chiều array, tensor:

```

1 # Numpy code
2 import numpy as np
3 # Tạo một array 2D
4 arr_2D = np.array([[1, 2, 3], [4, 5, 6]])
5 # Chuyển từ 2D -> 1D
6 arr_1D = np.reshape(arr_2D,
7                     newshape=(6, ))
8 print("array 2D:\n", arr_2D, arr_2D.shape)
9 print("array 1D:\n", arr_1D, arr_1D.shape)

```

```

=====
Output =====
array 2D:
 [[1 2 3]
 [4 5 6]] (2, 3)

array 1D:
 [1 2 3 4 5 6] (6,)

=====
```

Trong ví dụ trên, ta tạo một array 2D có 2 hàng và 3 cột. Sử dụng **np.reshape** để thay đổi hình dạng của array từ 2D thành 1D với hình dạng mới là (6,), nghĩa là một array có 6 phần tử. Ở đây $6 = 2 \times 3$, bằng với số phần tử của cũ, ta cần lưu ý khi thay đổi shape thì cần đảm bảo số lượng phần tử không được thay đổi khi **reshape**. Với Tensorflow, Pytorch cũng thực hiện tương tự.

2. Bài tập

Câu 1: Viết chương trình tạo một Numpy array, Pytorch tensor, Tensorflow tensor từ list 1D có giá trị [2, 3, 5, 7]. Sau đó thực hiện thêm một chiều mới ở vị trí 0 để tạo array, tensor 2D. Sau đó tiếp tục thêm các chiều mới ở vị trí 1, 3 và 4 để tạo array, tensor 5D.

Câu 2: Hãy viết chương trình để tạo một numpy array, một PyTorch tensor và một TensorFlow tensor từ một list 3D có giá trị như sau: [[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]], [[13, 14, 15], [16, 17, 18]]].

Sau đó, sử dụng hàm reshape để thay đổi hình dạng của array và tensor 3D vừa tạo có shape (3, 2, 3) thành array và tensor 2D có shape (3, 6).

3. Đáp án

Đáp án sẽ được gửi vào buổi tối trên nhóm.

Các Hàm Quan Trọng Trong Numpy, Pytorch, Tensorflow - Phần 3

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Hàm **clip** được sử dụng để giới hạn giá trị của một array hoặc tensor để nằm trong một phạm vi được chỉ định. Các thư viện như NumPy, PyTorch và TensorFlow đều cung cấp hàm clip với cú pháp tương tự nhau.

Trong NumPy, hàm clip được sử dụng để giới giá trị của một mảng trong một khoảng được chỉ định. Cú pháp sử dụng:

```
1 np.clip(a, a_min, a_max)
```

Trong đó:

- a: Mảng đầu vào.
- a_min: Giá trị nhỏ nhất mà các phần tử của a được lấy.
- a_max: Giá trị lớn nhất mà các phần tử của a được lấy.

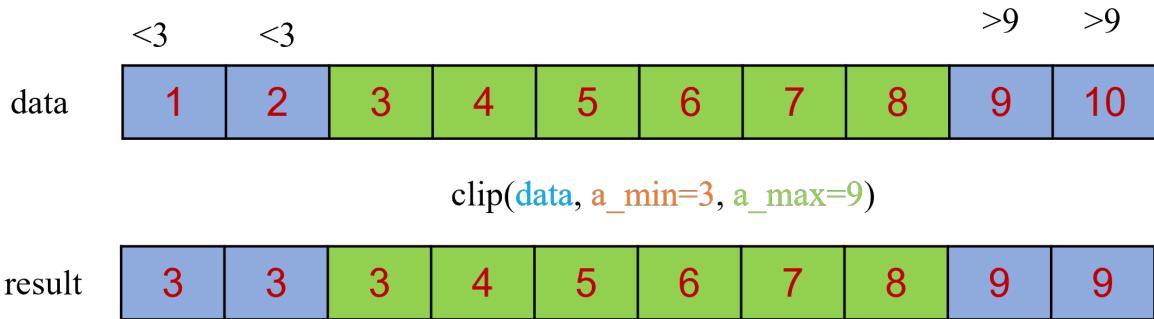
Ví dụ:

```
1 # Numpy code
2 import numpy as np
3
4 # Thiết lập seed để đảm bảo kết quả ngẫu
    nhiên có thể tái tạo được
5 np.random.seed(2024)
6
7 # Tạo một array với các giá trị ngẫu nhiên
    trong khoảng [-10, 10) với kích thước
    (3, 3)
8 arr_rand = np.random.randint(-10, 10,
9                                (3, 3))
10 print("Mảng ngẫu nhiên trong khoảng
11      [-10, 10):\n", arr_rand)
12
13 # Sử dụng hàm clip để cắt các giá trị nằm
    ngoài khoảng [3, 8]
14 arr_clip = np.clip(arr_rand, a_min=3,
15                     a_max=8)
16 print("Mảng sau khi được cắt trong khoảng
17      [3, 8]:\n", arr_clip)
```

```
=====
Output =====
Mảng ngẫu nhiên trong khoảng [-10, 10):
[[ -2 -10 -10]
 [ -6 -1 -9]
 [ -7 0 -8]]

Mảng sau khi được cắt trong khoảng [3, 8]:
[[3 3 3]
 [3 3 3]
 [3 3 3]]
```

Trong ví dụ trên, ta thiết lập seed giúp đảm bảo rằng kết quả ngẫu nhiên có thể được tái tạo. Tiếp theo, sử dụng `np.random.randint` để tạo một array với các giá trị ngẫu nhiên trong khoảng [-10, 10) với kích thước (3, 3). Sau đó `np.clip` được sử dụng để cắt các giá trị trong array sao cho chúng không vượt qua khoảng [3, 8]. Tức là giá trị nào nhỏ hơn 3 thì sẽ gán bằng 3, giá trị nào lớn hơn 8 sẽ gán bằng 8, các giá trị trong khoảng [3, 8] thì giữ nguyên.



Hình 1: Minh họa cách sử dụng hàm clip

Trong Pytorch, ta có thể sử dụng **clamp** hoặc **clip** đều được, vì trong Pytorch **clip** là tên bí danh hoặc có thể hiểu là tên viết tắt của **clamp**. Cú pháp:

```
1 # Pytorch
2 torch.clamp(input, min, max)
3 torch.clip(input, min, max)
```

Dưới đây là ví dụ:

```
1 #Pytorch code
2 import torch
3
4 # Thiết lập seed để đảm bảo kết quả ngẫu
# nhiên có thể tái tạo được
5 torch.manual_seed(2024)
6
7 # Tạo một tensor với các giá trị
8 # ngẫu nhiên trong khoảng [-10, 10] với
9 # kích thước (3, 4)
10 tensor_rand = torch.randint(-10, 10, size
    =(3, 4))
11 print("Tensor ngẫu nhiên trong khoảng
    [-10, 10]:\n", tensor_rand)
12
13 # Sử dụng hàm clamp để cắt các giá trị nằm
# ngoài khoảng [3, 8]
14 tensor_clip = torch.clamp(tensor_rand, min
    =3, max=8)
15 print("Tensor sau khi được cắt trong khoảng
    [3, 8]:\n", tensor_clip)
```

```
=====
Output =====
Tensor ngẫu nhiên trong khoảng [-10, 10]:
tensor([[ 2,   0, -6, -10],
        [-7, -10,   0,   1],
        [ 3,   9,   7, -6]])

Tensor sau khi được cắt trong khoảng [3,
8]:
tensor([[3, 3, 3, 3],
        [3, 3, 3, 3],
        [3, 8, 7, 3]])

=====
```

Trong Tensorflow, ta có thể sử dụng hàm **tf.clip_by_value** để giới hạn giá trị của tensor. Cú pháp:

```
1 #Tensorflow
2 tf.clip_by_value(t, clip_value_min, clip_value_max)
```

Dưới đây là ví dụ:

```

1 # Tensorflow code
2 import tensorflow as tf
3
4
5 # Thiết lập seed để đảm bảo kết quả ngẫu
6 # nhiên có thể tái tạo được
7 tf.random.set_seed(2024)
8
9 # Tạo một tensor với các giá trị
10 # ngẫu nhiên trong khoảng [-10, 10]
11 # với kích thước (3, 4)
12 tensor_random = tf.random.uniform((3, 4),
13                                   minval=-10, maxval=10, dtype=tf.int32)
14 print("Tensor ngẫu nhiên trong khoảng
15       [-10, 10]:\n", tensor_random)
16
17 # Sử dụng hàm clip_by_value để cắt các giá
18 # trị nằm ngoài khoảng [3, 8]
19 tensor_clip = tf.clip_by_value(
20     tensor_random, clip_value_min=3,
21     clip_value_max=8)
22
23 print("Tensor sau khi được cắt trong khoảng
24       [3, 8]:\n", tensor_clip)

```

```

=====
Output =====
Tensor ngẫu nhiên trong khoảng [-10, 10]:
tf.Tensor(
[[[-6 -2  0 -2]
 [ 4  4 -9  3]
 [ 7  1  2 -5]], shape=(3, 4),
dtype=int32)

Tensor sau khi được cắt trong khoảng
[3, 8]:
tf.Tensor(
[[3 3 3 3]
 [4 4 3 3]
 [7 3 3 3]], shape=(3, 4), dtype=int32)
=====
```

Hàm **clip** rất hữu ích khi ta muốn đảm bảo rằng các giá trị trong array hoặc tensor không vượt quá một khoảng cụ thể, giúp kiểm soát và chuẩn hóa dữ liệu, đặc biệt trong quá trình tiền xử lý dữ liệu cho mô hình máy học.

2. Bài tập

Câu 1: Viết chương trình tạo một Numpy array, Pytorch tensor, Tensorflow tensor với các giá trị số nguyên ngẫu nhiên trong khoảng [-5, 10] với kích thước (4, 3). Sử dụng hàm clip để cắt các giá trị nằm ngoài khoảng [-2, 0]? Lưu ý: Sử dụng seed=2024

```

1 # Numpy code
2 import numpy as np
3 np.random.seed(2024)
4
5 # Pytorch code
6 import torch
7 torch.manual_seed(2024)
8
9 # Tensorflow code
10 import tensorflow as tf
11 tf.random.set_seed(2024)

```

3. Đáp án

Đáp án sẽ được gửi vào buổi tối trên nhóm.

Indexing Trong Numpy, Pytorch và Tensorflow - Phần 1

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Slicing là một cách để trích xuất một phần của array, tensor dựa trên các chỉ số hoặc các điều kiện. Điều này cho phép ta lấy ra các phần tử theo các quy luật cụ thể, giúp thao tác dữ liệu dễ dàng hơn. Trong Numpy, cú pháp để thực hiện slicing là:

```
1 array[axis n row:column, axis n-1 row:column, ..., axis 0 row:column,
2      axis 1 row:column]
```

Trong đó: axis n row:column nghĩa là tại chiều thứ n, lấy phần tử từ hàng nào đến cột nào.

data	data[0, 1]	data[1:3]	data[0:1, 0]	data[:, :]
0 1 0 1 2 1 3 4 2 5 6	0 1 0 1 2 1 3 4 2 5 6	0 1 0 1 2 1 3 4 2 5 6	0 1 0 1 2 1 3 4 2 5 6	0 1 0 1 2 1 3 4 2 5 6
0 1 0 1 2 1 3 4 2 5 6	0 1 0 1 2 1 3 4 2 5 6	0 1 0 1 2 1 3 4 2 5 6	0 1 0 1 2 1 3 4 2 5 6	0 1 0 1 2 1 3 4 2 5 6

Hình 1: Minh họa cách sử dụng slicing

ví dụ:

```
1 # Import thư viện Numpy
2 import numpy as np
3 # Tạo mảng 2D
4 array_2d = np.array([[1, 2, 3], [4, 5, 6]])
5 # Slicing mảng tại row = 0, column = 1
6 # Giữ nguyên số chiều của mảng
7 sliced_array_1 = array_2d[0:1, 1:2]
8 print("sliced_array_1 (Giữ nguyên số chiều):\n")
9 print(sliced_array_1)
10 # Giảm số chiều của mảng
11 sliced_array_2 = array_2d[0, 1:2]
12 print("\nsliced_array_2 (Giảm số chiều):\n")
13 print(sliced_array_2)
14 # Giảm số chiều của mảng
15 sliced_array_3 = array_2d[0, 1]
16 print("\nsliced_array_3 (Giảm số chiều):\n")
17 print(sliced_array_3)
18 # Lấy toàn bộ mảng
19 sliced_array_4 = array_2d[:, :]
20 print("\nsliced_array_4 (Giữ nguyên số chiều):\n")
21 print(sliced_array_4)
```

```
=====
Output =====
sliced_array_1 (Giữ nguyên số chiều):
[[2]]

sliced_array_2 (Giảm số chiều):
[2]

sliced_array_3 (Giảm số chiều):
2

sliced_array_4 (Giữ nguyên số chiều):
[[1 2 3]
 [4 5 6]]
```

Trong ví dụ trên là các cách ta thực hiện slicing array, ở đây có hiện tượng giảm số chiều của mảng xuất hiện khi ta trích xuất một phần nhỏ của mảng thông qua slicing và chỉ giữ lại một số chiều của nó.

Ở trường hợp đầu tiên `sliced_array_1 = array_2d[0:1, 1:2]`, ta đang trích xuất một phần của mảng bằng cách chỉ định slicing cho cả hai chiều (hàng và cột). Kết quả là một mảng 2D với số chiều không thay đổi.

Trường hợp thứ hai `sliced_array_2 = array_2d[0, 1:2]`, ta đang trích xuất một hàng (0) và một phần của cột (từ 1 đến 2). Kết quả là một mảng 1D, và số chiều của mảng đã giảm xuống.

Tương tự như `sliced_array_2`, trường hợp `sliced_array_3 = array_2d[0, 1]` ta chỉ đang trích xuất một phần nhỏ của mảng, nên kết quả là một mảng 0D (véc-tơ). Trong Numpy, mảng 0D còn được gọi là scalar.

Trong trường hợp cuối, ta lấy toàn bộ mảng bằng cách không chỉ định slicing cho bất kỳ chiều nào cả. Kết quả là một mảng 2D và số chiều không thay đổi.

Quy tắc thực hiện slicing trong Numpy cũng áp dụng cho tensor trong Pytorch và Tensorflow. Ví dụ trong Pytorch:

```

1 # Pytorch code
2 import torch
3
4
5 # Tạo tensor 2D
6 tensor_2d = torch.tensor([[1, 2, 3], [4,
5, 6]])
7
8 # Slicing tensor tại vị trí row= 0,
9 # column = 1
10 # Giữ nguyên số chiều của tensor
11 sliced_tensor_1 = tensor_2d[0:1, 1:2]
12 print("sliced_tensor_1 (Giữ nguyên số
chiều):")
13 print(sliced_tensor_1)
14
15 # Giảm số chiều của tensor
16 sliced_tensor_2 = tensor_2d[0, 1:2]
17 print("\nsliced_tensor_2 (Giảm số chiều):"
)
18 print(sliced_tensor_2)
19
20 # Giảm số chiều của tensor
21 sliced_tensor_3 = tensor_2d[0, 1]
22 print("\nsliced_tensor_3 (Giảm số chiều):"
)
23 print(sliced_tensor_3)
24
25 # Lấy toàn bộ tensor
26 sliced_tensor_4 = tensor_2d[:, :]
27 print("\nsliced_tensor_4 (Giữ nguyên số
chiều):")
28 print(sliced_tensor_4)
29

```

```

=====
Output =====
sliced_tensor_1 (Giữ nguyên số chiều):
tensor([[2]])

sliced_tensor_2 (Giảm số chiều):
tensor([2])

sliced_tensor_3 (Giảm số chiều):
tensor(2)

sliced_tensor_4 (Giữ nguyên số chiều):
tensor([[1, 2, 3],
       [4, 5, 6]])


=====
```

Ví dụ trong Tensorflow:

```

1 # Tensorflow code
2 import tensorflow as tf
3 # Tạo tensor 2D
4 tensor_2d = tf.constant([[1, 2, 3], [4, 5,
   6]])
5 # Slicing tensor tại vị trí row=0, column
   =1
6 # Giữ nguyên số chiều của tensor
7 sliced_tensor_1 = tensor_2d[0:1, 1:2]
8 print("sliced_tensor_1 (Giữ nguyên số
   chiều):")
9 print(sliced_tensor_1)
10 # Giảm số chiều của tensor
11 sliced_tensor_2 = tensor_2d[0, 1:2]
12 print("\nsliced_tensor_2 (Giảm số chiều):"
   )
13 print(sliced_tensor_2)
14 # Giảm số chiều của tensor
15 sliced_tensor_3 = tensor_2d[0, 1]
16 print("\nsliced_tensor_3 (Giảm số chiều):"
   )
17 print(sliced_tensor_3)
18 # Lấy toàn bộ tensor
19 sliced_tensor_4 = tensor_2d[:, :]
20 print("\nsliced_tensor_4 (Giữ nguyên số
   chiều):")
21 print(sliced_tensor_4)
22
===== Output =====
sliced_tensor_1 (Giữ nguyên số chiều):
tf.Tensor([[[2]]], shape=(1, 1),
dtype=int32)

sliced_tensor_2 (Giảm số chiều):
tf.Tensor([2], shape=(1,), dtype=int32)

sliced_tensor_3 (Giảm số chiều):
tf.Tensor(2, shape=(), dtype=int32)

sliced_tensor_4 (Giữ nguyên số chiều):
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
=====
```

Tùy thuộc vào mục đích mà ta có thể lựa chọn cách slice khác nhau.

2. Bài tập

Hãy viết chương trình để tạo một numpy array, PyTorch tensor và TensorFlow tensor từ một list 3D có giá trị như sau: `[[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]], [[13, 14, 15], [16, 17, 18]]]`. Sau đó sử dụng slicing thực hiện các yêu cầu sau:

- Sử dụng slicing để lấy một phần của array, tensor tại vị trí index (0, 1) của mảng 2D trong mảng 3D và giữ nguyên số chiều
- Sử dụng slicing để lấy một phần của array, tensor tại vị trí index (0, 1) của mảng 2D trong mảng 3D, nhưng giảm đi một chiều.

3. Đáp án

Đáp án sẽ được gửi vào buổi tối trên nhóm.

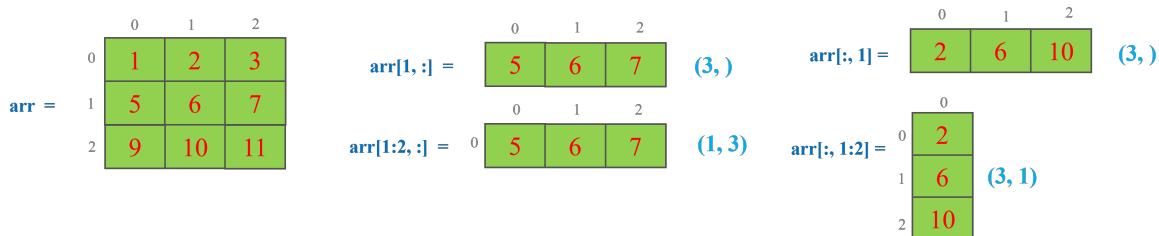
Indexing Trong Numpy, Pytorch và Tensorflow - Phần 2

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

a) *Truy xuất row, column*

Để truy cập một cột(column) hoặc một hàng(row) từ một array, tensor trong Numpy, Pytorch và Tensorflow có thể được thực hiện bằng cách sử dụng indexing.



Hình 1: Minh họa cách truy xuất row, column

Dưới đây là ví dụ và giải thích cách thực hiện:

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo mảng 2D
5 arr_2d = np.array([[1, 2, 3],
6                 [4, 5, 6],
7                 [7, 8, 9]])
8
9 # Hai cách để lấy hàng index=1
10 # Cách 1: giảm số chiều của mảng
11 row_1_arr_1 = arr_2d[1, :]
12 print("row_1_arr_1 (Giảm số chiều):",
      row_1_arr_1)
13
14 # Cách 2: giữ nguyên số chiều của mảng
15 row_1_arr_2 = arr_2d[1:2, :]
16 print("row_1_arr_2 (Giữ nguyên số chiều):",
      , row_1_arr_2)
17
18 # Hai cách để lấy cột index=2
19 # Cách 1: giảm số chiều của mảng
20 column_2_arr_1 = arr_2d[:, 2]
21 print("\ncolumn_2_arr_1 (Giảm số chiều):\n",
      , column_2_arr_1)
22 # Cách 2: giữ nguyên số chiều của mảng
23 column_2_arr_2 = arr_2d[:, 2:3]
24 print("\ncolumn_2_arr_2\n",
      (Giữ nguyên số chiều):", column_2_arr_2)
25
===== Output =====
row_1_arr_1 (Giảm số chiều): [4 5 6]
row_1_arr_2 (Giữ nguyên số chiều): [[4 5
6]]
column_2_arr_1 (Giảm số chiều):
[3 6 9]
column_2_arr_2 (Giữ nguyên số chiều):
[[3]
[6]
[9]]
=====

```

Ví dụ thư viện Pytorch, thực hiện tương tự như Numpy:

```

1 # Pytorch code
2 import torch
3 # Tạo tensor 2D
4 tensor_2d = torch.tensor([[1, 2, 3], [4,
5, 6], [7, 8, 9]])
5 # Hai cách để lấy hàng index=1
6 # Cách 1: giảm số chiều của tensor
7 row_1_tensor_1 = tensor_2d[1, :]
8 print("row_1_tensor_1 (Giảm số chiều):",
row_1_tensor_1)
9 # Cách 2: giữ nguyên số chiều của tensor
10 row_1_tensor_2 = tensor_2d[1:2, :]
11 print("row_1_tensor_2 (Giữ nguyên số chiều):",
row_1_tensor_2)
12 # Hai cách để lấy cột index=2
13 # Cách 1: giảm số chiều của tensor
14 column_2_tensor_1 = tensor_2d[:, 2]
15 print("\ncolumn_2_tensor_1 (Giảm số chiều):
:\n", column_2_tensor_1)
16 # Cách 2: giữ nguyên số chiều của tensor
17 column_2_tensor_2 = tensor_2d[:, 2:3]
18 print("\ncolumn_2_tensor_2 (Giữ nguyên số
chiều):\n", column_2_tensor_2)

```

```

=====
Output =====
row_1_tensor_1 (Giảm số chiều):
tensor([4, 5, 6])

row_1_tensor_2 (Giữ nguyên số chiều):
tensor([[4, 5, 6]])

column_2_tensor_1 (Giảm số chiều):
tensor([3, 6, 9])

column_2_tensor_2 (Giữ nguyên số chiều):
tensor([[3],
       [6],
       [9]])

=====
```

```

1 # Tensorflow code
2 import tensorflow as tf
3 # Tạo tensor 2D
4 tensor_2d = tf.constant([[1, 2, 3], [4, 5,
6], [7, 8, 9]])
5 # Hai cách để lấy hàng index=1
6 # Cách 1: giảm số chiều của tensor
7 row_1_tensor_1 = tensor_2d[1, :]
8 print("row_1_tensor_1 (Giảm số chiều):",
row_1_tensor_1)
9 # Cách 2: giữ nguyên số chiều của tensor
10 row_1_tensor_2 = tensor_2d[1:2, :]
11 print("row_1_tensor_2 (Giữ nguyên số chiều):
:\n", row_1_tensor_2)
12 # Hai cách để lấy cột index=2
13 # Cách 1: giảm số chiều của tensor
14 column_2_tensor_1 = tensor_2d[:, 2]
15 print("\ncolumn_2_tensor_1 (Giảm số chiều):
:\n", column_2_tensor_1)
16 # Cách 2: giữ nguyên số chiều của tensor
17 column_2_tensor_2 = tensor_2d[:, 2:3]
18 print("\ncolumn_2_tensor_2 (Giữ nguyên số
chiều):\n", column_2_tensor_2)

```

```

=====
Output =====
row_1_tensor_1 (Giảm số chiều):
tf.Tensor([4 5 6], shape=(3,), dtype=
int32)

row_1_tensor_2 (Giữ nguyên số chiều):
tf.Tensor([[4 5 6]], shape=(1, 3), dtype=
int32)

column_2_tensor_1 (Giảm số chiều):
tf.Tensor([3 6 9], shape=(3,), dtype=
int32)

column_2_tensor_2 (Giữ nguyên số chiều):
tf.Tensor(
[[3]
 [6]
 [9]], shape=(3, 1), dtype=int32)

=====
```

Trong các ví dụ trên, chúng ta có một mảng, tensor hai chiều và chúng ta thực hiện một số thao tác để lấy dữ liệu từ hàng và cột cụ thể của mảng:

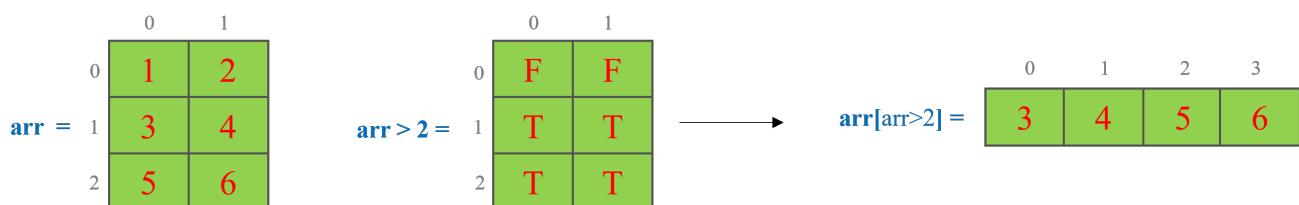
- Lấy hàng index=1: Chúng ta sử dụng hai cách khác nhau để lấy hàng thứ hai ('[4, 5, 6]'). Cách đầu tiên giảm số chiều của mảng, tensor, trong khi cách thứ hai giữ nguyên số chiều.

- Lấy cột index=2: Chúng ta sử dụng hai cách khác nhau để lấy cột thứ ba của mảng, tensor ('[3, 6, 9]' hoặc '[3], [6], [9]]'). Cách đầu tiên giảm số chiều, trong khi cách thứ hai giữ nguyên số chiều bằng cách chỉ rõ là lấy từ vị trí bắt đầu và kết thúc của cột muốn lấy.

Tóm lại trong cả ba thư viện, ta có thể sử dụng phép indexing với dấu hai chấm ("::") để chọn tất cả các phần tử trong một chiều của mảng hoặc tensor. Để truy cập một cột, ta giữ nguyên chiều hàng và chỉ định chỉ số của cột. Để truy cập một hàng, ta giữ nguyên chiều cột và chỉ định chỉ số của hàng.

b) Boolean indices

Boolean indices (hoặc boolean indexing) là một cách để lấy hoặc lọc các phần tử từ một mảng dựa trên một mảng boolean có cùng hình dạng. Mảng boolean này được gọi là mask, và nó được sử dụng để chọn các phần tử tương ứng từ mảng gốc.



Hình 2: Minh họa cách sử dụng boolean indices

```

1 # Import thư viện Numpy
2 import numpy as np
3
4 # Tạo mảng 2D
5 array_2d = np.array([[1, 2, 3], [4, 5,
6   6], [7, 8, 9]])
7
8 # Lấy các phần tử có giá trị lớn hơn 5
9 boolean_mask = array_2d > 5
10 filtered_result = array_2d[boolean_mask]
11
12 # In kết quả
13 print("Mảng 2D ban đầu:\n", array_2d)
14 print("\n Boolean mask:\n", boolean_mask)
15 print("\nCác phần tử lớn hơn 5:\n",
16   filtered_result)

```

```

=====
Output =====
Mảng 2D ban đầu:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Boolean mask:
[[False False False]
 [False False True]
 [ True  True  True]]

Các phần tử lớn hơn 5:
[6 7 8 9]
=====
```

Trong ví dụ trên, **boolean mask** được tạo ra bằng cách kiểm tra xem mỗi phần tử của arr có lớn hơn 5 hay không. Kết quả là một mảng **boolean mask** với các giá trị True hoặc False. Sau đó, **boolean mask** được sử dụng để lọc các phần tử tương ứng từ arr, chỉ giữ lại những phần tử có giá trị True.

Tương tự, cả Pytorch và Tensorflow cũng hỗ trợ boolean indexing. Quy trình là giống nhau như trong Numpy Ví dụ Pytorch:

```

1 # Pytorch code
2 import torch
3
4 # Tạo tensor 2D
5 tensor_2d = torch.tensor([[1, 2, 3],
6                           [4, 5, 6],
7                           [7, 8, 9]])
8
9 # Lấy các phần tử có giá trị lớn hơn 5
10 boolean_mask = tensor_2d > 5
11 filtered_result = tensor_2d[boolean_mask]
12
13 # In kết quả
14 print("Tensor 2D ban đầu:\n", tensor_2d)
15 print("\n Boolean mask:\n", boolean_mask)
16 print("\nCác phần tử lớn hơn 5:\n",
      filtered_result)

```

```

=====
Output =====
Tensor 2D ban đầu:
tensor([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])

Boolean mask:
tensor([[False, False, False],
        [False, False, True],
        [True, True, True]])

Các phần tử lớn hơn 5:
tensor([6, 7, 8, 9])

=====
```

Ví dụ Tensorflow:

```

1 # Tensorflow code
2 import tensorflow as tf
3
4
5 # Tạo tensor 2D
6 tensor_2d = tf.constant([[1, 2, 3],
7                           [4, 5, 6],
8                           [7, 8, 9]])
9
10 # Lấy các phần tử có giá trị lớn hơn 5
11 boolean_mask = tensor_2d > 5
12 filtered_result = tf.boolean_mask(
    tensor_2d, boolean_mask)
13
14 # In kết quả
15 print("Tensor 2D ban đầu:\n", tensor_2d)
16 print("\n Boolean mask:\n", boolean_mask)
17 print("\nCác phần tử lớn hơn 5:\n",
      filtered_result)

```

```

=====
Output =====
Tensor 2D ban đầu:
tf.Tensor(
[[1 2 3]
 [4 5 6]
 [7 8 9]], shape=(3, 3), dtype=int32)

Boolean mask:
tf.Tensor(
[[False False False]
 [False False True]
 [True True True]], shape=(3, 3), dtype=bool)

Các phần tử lớn hơn 5:
tf.Tensor([6 7 8 9], shape=(4,), dtype=int32)

=====
```

2. Bài tập

Câu 1: Cho một list 2D như sau:

```

1 [[1, 0, 1],
2  [0, 1, 0],
3  [1, 0, 1]]

```

Hãy tạo array, tensor(Pytorch, Tensorflow) từ list 2D này sau đó thực hiện các yêu cầu sau:

- Lấy hàng thứ 2(index=1) theo 2 cách giữ nguyên hoặc giảm số chiều
- Lấy cột thứ nhất(index=0) theo 2 cách giữ nguyên hoặc giảm số chiều

Câu 2: Viết chương trình tạo một Numpy array, Pytorch tensor, Tensorflow tensor với các giá trị số nguyên ngẫu nhiên trong khoảng [-10, 10) với kích thước (3, 3). Sau đó sử dụng Boolean indices để lọc các phần tử lớn hơn 0. Lưu ý: sử dụng seed=2024

3. Đáp án

Đáp án sẽ được gửi vào buổi tối trên nhóm.

Các Phép Tính Numpy, Pytorch và Tensorflow - Phần 1

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

Trong phần này, chúng ta sẽ tìm hiểu về các phép tính cơ bản nhưng thật sự quan trọng khi làm việc với array, tensor. Chúng ta sẽ tạm thời quên đi vòng for và sẽ thấy được sức mạnh tính toán của những thư viện này thật sự kinh ngạc.

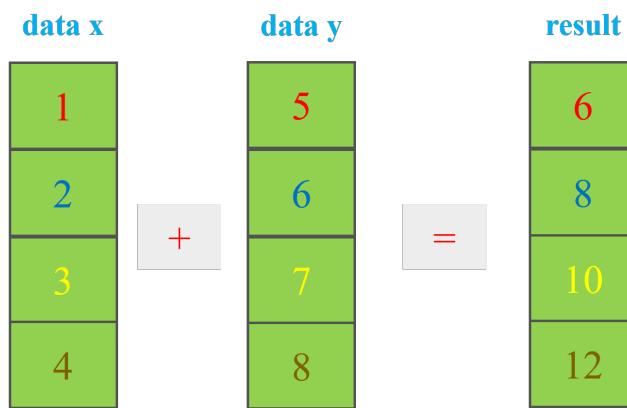
a) *Addition*

Chúng ta đã quá quen với các phép tính cộng, trừ, nhân, chia vì chúng ta sử dụng nó hằng ngày. Chúng ta biết về cách tính tuần tự, và gặp khó khăn khi đối mặt với số lượng tính toán lớn. Thư viện Numpy, Pytorch, Tensorflow được xây dựng để giải quyết vấn đề đó, chúng hỗ trợ việc tính toán song song, tức là thực hiện nhiều phép tính đồng thời.

Trước khi bắt đầu với các ví dụ thì ta hãy xem lại định nghĩa về phép cộng: Cho hai số a và b , phép cộng được biểu diễn bằng ký hiệu '+' và kết quả của phép cộng là tổng của a và b , thường được ký hiệu là $a + b$.

Đối với array và tensor thì phép cộng giữa chúng được thực hiện theo cùng một nguyên tắc cơ bản, là phép cộng element-wise (theo từng phần tử). Phép cộng này được định nghĩa như sau: Cho hai array (hoặc tensor) A và B cùng kích thước, phép cộng giữa chúng ($A + B$) tạo ra một array mới C , với mỗi phần tử C_i được tính bằng cách cộng phần tử A_i với phần tử B_i .

$$C_i = A_i + B_i$$



Hình 1: Minh họa hàm addition

Trong các ví dụ dưới đây, ta thực hiện phép cộng các array, tensor bằng cách sử dụng toán tử "+" hoặc hàm add().

Ví dụ với Numpy, phép cộng giữa hai array có cùng kích thước là việc thực hiện phép cộng giữa các phần tử tương ứng của chúng.

```

1 # Numpy code
2 import numpy as np
3 # Tạo hai array 1D
4 arr_1 = np.array([1, 2, 3, 4])
5 arr_2 = np.array([5, 6, 7, 8])
6 # Thực hiện phép cộng hai array
7 # Cách 1: Sử dụng toán tử '+'
8 arr_add_1 = arr_1 + arr_2
9 # Cách 2: Sử dụng hàm np.add()
10 arr_add_2 = np.add(arr_1, arr_2)
11 # In ra màn hình
12 print(f"arr_1 = \n{arr_1}")
13 print(f"arr_2 = \n{arr_2}")
14 print("arr_1 + arr_2")
15 print(f"arr_add_1 = \n{arr_add_1}")
16 print(f"arr_add_2 = \n{arr_add_2}")

```

```

===== Output =====
arr_1 =
[1 2 3 4]

arr_2 =
[5 6 7 8]

arr_1 + arr_2

arr_add_1 =
[ 6  8 10 12]

arr_add_2 =
[ 6  8 10 12]

=====

```

Kết quả thực hiện phép cộng sẽ là một array mới là [6, 8, 10, 12] với mỗi phần tử là tổng của các phần tử tương ứng trong arr1 và arr2.

Tương tự, chúng ta có thể sử dụng toán tử "+" và hàm add() trong thư viện Pytorch và Tensorflow:

```

1 #Pytorch code
2 import torch
3 # Tạo hai tensor 1D
4 tensor_1 = torch.tensor([1, 2, 3, 4])
5 tensor_2 = torch.tensor([5, 6, 7, 8])
6 # Thực hiện phép cộng hai tensor
7 # Cách 1: Sử dụng toán tử '+'
8 tensor_add_1 = tensor_1 + tensor_2
9 # Cách 2: Sử dụng hàm torch.add()
10 tensor_add_2 = torch.add(tensor_1,
                           tensor_2)
11 # In ra màn hình
12 print(f"tensor_1 = \n{tensor_1}")
13 print(f"tensor_2 = \n{tensor_2}")
14 print("tensor_1 + tensor_2")
15 print(f"tensor_add_1 = \n{tensor_add_1}")
16 print(f"tensor_add_2 = \n{tensor_add_2}")

```

```

===== Output =====
tensor_1 =
tensor([1, 2, 3, 4])

tensor_2 =
tensor([5, 6, 7, 8])

tensor_1 + tensor_2

tensor_add_1 =
tensor([ 6,  8, 10, 12])

tensor_add_2 =
tensor([ 6,  8, 10, 12])

=====

```

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo hai tensor 1D
5 tensor_1 = tf.constant([1, 2, 3, 4])
6 tensor_2 = tf.constant([5, 6, 7, 8])
7
8 # Thực hiện phép cộng hai tensor
9 # Cách 1: Sử dụng toán tử '+'
10 tensor_add_1 = tensor_1 + tensor_2
11 # Cách 2: Sử dụng hàm tf.add()
12 tensor_add_2 = tf.add(tensor_1, tensor_2)
13
14 print(f"tensor_1 = \n{tensor_1}")
15 print(f"tensor_2 = \n{tensor_2}")
16 print("tensor_1 + tensor_2")
17 print(f"tensor_add_1 = \n{tensor_add_1}")
18 print(f"tensor_add_2 = \n{tensor_add_2}")

```

```

===== Output =====
tensor_1 =
[1 2 3 4]

tensor_2 =
[5 6 7 8]

tensor_1 + tensor_2

tensor_add_1 =
[ 6  8 10 12]

tensor_add_2 =
[ 6  8 10 12]

=====

```

Từ các ví dụ trên, ta có thể thấy phép cộng array, tensor có tính chất sau: Phép cộng array, tensor là một phép toán element-wise, tức là mỗi phần tử trong kết quả được tính toán dựa trên các phần tử tương ứng của các array, tensor gốc. Điều này giúp thực hiện các phép toán một cách hiệu quả trên toàn bộ array, tensor, làm giảm sự cần thiết phải sử dụng vòng lặp.

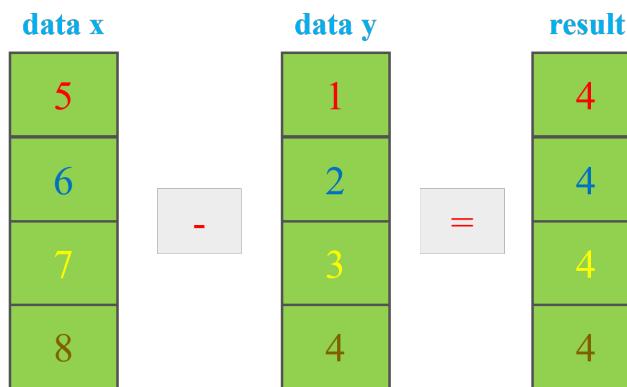
b) *Subtraction*

Trong toán học, phép trừ hai số được định nghĩa như sau: Giả sử có hai số a và b , thì $a - b$ là kết quả của phép trừ. Kết quả này cho biết khoảng cách giữa a và b trên trục số.

Đối với phép trừ hai array hoặc tensor cùng kích thước A và B , phép trừ giữa chúng ($A - B$) tạo ra một array mới C , với mỗi phần tử C_i được tính bằng cách trừ phần tử B_i khỏi phần tử A_i .

$$C_i = A_i - B_i$$

Tương tự như phép cộng array, tensor, phép trừ cũng là một phép toán element-wise, tức là mỗi phần tử của array kết quả được tính bằng cách trừ phần tử tương ứng của array thứ nhất cho phần tử tương ứng của array thứ hai.



Hình 2: Minh họa hàm subtraction

Ví dụ phép trừ hai array trong Numpy, chúng ta có thể sử dụng toán tử `"-"`hoặc hàm `np.subtract()`

```

1 # Numpy code
2 import numpy as np
3 # Tạo hai array 1D
4 arr_1 = np.array([1, 2, 3, 4])
5 arr_2 = np.array([5, 6, 7, 8])
6 print("arr_1:", arr_1)
7 print("arr_2:", arr_2)
8 # Hiệu hai array
9 # Cách 1: Sử dụng toán tử '-'
10 array_sub_1 = arr_1 - arr_2
11 # Cách 2: Sử dụng hàm np.subtract()
12 array_sub_2 = np.subtract(arr_1, arr_2)
13 # In ra màn hình
14 print("Cách 1\n", array_sub_1)
15 print("Cách 2\n", array_sub_2)

```

```

=====
Output =====
arr_1: [1 2 3 4]
arr_2: [5 6 7 8]
Cách 1
[-4 -4 -4 -4]
Cách 2
[-4 -4 -4 -4]
=====
```

Đối với Pytorch, ta dùng `torch.sub()` hoặc `torch.subtract()`.

```

1 # Pytorch code
2 import torch
3 # Tạo hai tensor 1D
4 tensor_1 = torch.tensor([1, 2, 3, 4])
5 tensor_2 = torch.tensor([5, 6, 7, 8])
6 # In ra giá trị của hai tensor
7 print("tensor 1:", tensor_1)
8 print("tensor 2:", tensor_2)
9 # Hiệu hai tensor
10 # Cách 1: Sử dụng toán tử '-'
11 tensor_sub_1 = tensor_1 - tensor_2
12 # Cách 2: Sử dụng hàm torch.sub()
13 tensor_sub_2 = torch.sub(tensor_1,
    tensor_2)
14 # In ra màn hình
15 print("Cách 1\n", tensor_sub_1)
16 print("Cách 2\n", tensor_sub_2)

```

```

=====
Output =====
tensor 1: tensor([1, 2, 3, 4])
tensor 2: tensor([5, 6, 7, 8])
Cách 1
tensor([-4, -4, -4, -4])
Cách 2
tensor([-4, -4, -4, -4])
=====
```

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo hai tensor 1D
5 tensor_1 = tf.constant([1, 2, 3, 4])
6 tensor_2 = tf.constant([5, 6, 7, 8])
7
8 # In ra giá trị của hai tensor
9 print("tensor 1:", tensor_1)
10 print("tensor 2:", tensor_2)
11
12 # Hiệu hai tensor
13 # Cách 1: Sử dụng toán tử '-'
14 tensor_sub_1 = tensor_1 - tensor_2
15
16 # Cách 2: Sử dụng hàm tf.subtract()
17 tensor_sub_2 = tf.subtract(tensor_1,
    tensor_2)
18
19 # In ra màn hình
20 print("Cách 1\n", tensor_sub_1)
21 print("Cách 2\n", tensor_sub_2)

```

```

=====
Output =====
tensor 1: tf.Tensor([1 2 3 4], shape=(4,), dtype=int32)
tensor 2: tf.Tensor([5 6 7 8], shape=(4,), dtype=int32)
Cách 1
tf.Tensor([-4 -4 -4 -4], shape=(4,), dtype=int32)
Cách 2
tf.Tensor([-4 -4 -4 -4], shape=(4,), dtype=int32)
=====
```

2. Bài tập

Cho hai list 2D như sau:

```

1 lst_1 = [[1, -2, 1],
2     [-3, 1, 0],
3     [-2, 5, 1]]
4
5 lst_2 = [[1, 3, 5],
6     [2, 4, 6],
7     [3, 5, 7]]

```

Hãy tạo 2 array, tensor(Pytorch, Tensorflow) từ list 2D này sau đó thực hiện các phép tính addition và subtraction giữa hai array, tensor vừa tạo.

3. Đáp án

Đáp án sẽ được cập nhật vào buổi tối trong nhóm!

Các Phép Tính Numpy, Pytorch và Tensorflow - Phần 2

Dinh-Tiem Nguyen và Quang-Vinh Dinh

1. Mô tả

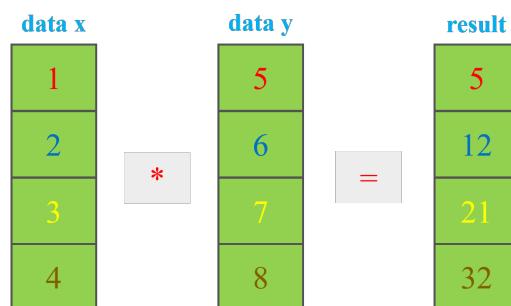
Trong phần này, chúng ta sẽ tiếp tục tìm hiểu về các phép tính cơ bản nhưng thắt sự quan trọng khi làm việc với array, tensor.

a) Multiplication

Phép nhân **Multiplication** array, tensor được thực hiện theo nguyên tắc element-wise: Cho hai array (hoặc tensor) cùng kích thước A và B , phép nhân giữa chúng ($A \cdot B$) tạo ra một array (hoặc tensor) mới C , với mỗi phần tử C_{ij} được tính bằng cách nhân phần tử A_{ij} với phần tử B_{ij} .

$$C_{ij} = A_{ij} \cdot B_{ij}$$

Để thực hiện multiplication, ta sử dụng toán tử "*" hoặc hàm **multiply**.



Hình 1: Minh họa phép nhân multiplication

Ví dụ với thư viện Numpy:

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo hai array 1D
5 arr_1 = np.array([1, 2, 3, 4])
6 arr_2 = np.array([5, 6, 7, 8])
7 # In ra giá trị của hai array
8 print("arr_1:\n", arr_1)
9 print("arr_2:\n", arr_2)
10 # Tích hai array
11 # Cách 1: Sử dụng toán tử '*'
12 result_mul_1 = arr_1 * arr_2
13 # Cách 2: Sử dụng hàm np.multiply()
14 result_mul_2 = np.multiply(arr_1, arr_2)
15 # In ra màn hình
16 print("Cách 1:\n", result_mul_1)
17 print("Cách 2:\n", result_mul_2)

```

```

=====
Output =====
arr_1:
[1 2 3 4]

arr_2:
[5 6 7 8]

Cách 1:
[ 5 12 21 32]

Cách 2:
[ 5 12 21 32]
=====
```

Ví dụ với thư viện Pytorch

```

1 # Pytorch code
2 import torch
3
4 # Tạo hai tensor 1D
5 tensor_1 = torch.tensor([1, 2, 3, 4])
6 tensor_2 = torch.tensor([5, 6, 7, 8])
7
8 # In ra giá trị của hai tensor
9 print("tensor 1:", tensor_1)
10 print("tensor 2:", tensor_2)
11
12 # Tích hai tensor
13 # Cách 1: Sử dụng toán tử '*'
14 result_mul_1 = tensor_1 * tensor_2
15
16 # Cách 2: Sử dụng hàm torch.multiply()
17 result_mul_2 = torch.multiply(tensor_1,
18                               tensor_2)
19
20 # In ra màn hình
21 print("Cách 1\n", result_mul_1)
22 print("Cách 2\n", result_mul_2)

```

===== Output =====
tensor 1: tensor([1, 2, 3, 4])

tensor 2: tensor([5, 6, 7, 8])

Cách 1

tensor([5, 12, 21, 32])

Cách 2

tensor([5, 12, 21, 32])

Ví dụ với thư viện Tensorflow

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo hai tensor 1D
5 tensor_1 = tf.constant([1, 2, 3, 4])
6 tensor_2 = tf.constant([5, 6, 7, 8])
7
8 # In ra giá trị của hai tensor
9 print("tensor 1:\n", tensor_1)
10 print("tensor 2:\n", tensor_2)
11
12 # Tích hai tensor
13 # Cách 1: Sử dụng toán tử '*'
14 result_mul_1 = tensor_1 * tensor_2
15
16 # Cách 2: Sử dụng hàm tf.multiply()
17 result_mul_2 = tf.multiply(tensor_1,
18                           tensor_2)
19
20 # In ra màn hình
21 print("Cách 1:\n", result_mul_1)
22 print("Cách 2:\n", result_mul_2)

```

===== Output =====
tensor 1:
tf.Tensor([1 2 3 4], shape=(4,), dtype=int32)

tensor 2:

tf.Tensor([5 6 7 8], shape=(4,), dtype=int32)

Cách 1:

tf.Tensor([5 12 21 32], shape=(4,), dtype=int32)

Cách 2:

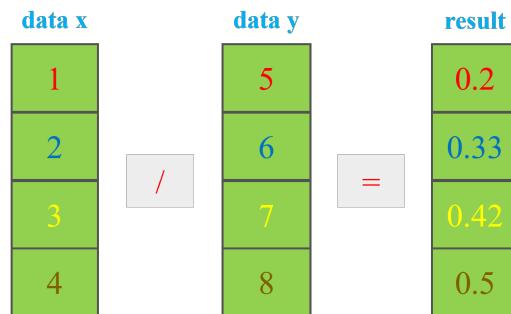
tf.Tensor([5 12 21 32], shape=(4,), dtype=int32)

b) *Division*

Phép chia (division) array và tensor được thực hiện theo nguyên tắc element-wise. Cho hai array (hoặc tensor) cùng kích thước A và B , phép chia giữa chúng ($A \div B$) tạo ra một array (hoặc tensor) mới C , với mỗi phần tử C_{ij} được tính bằng cách chia phần tử A_{ij} cho phần tử B_{ij} .

$$C_{ij} = \frac{A_{ij}}{B_{ij}}$$

Để thực hiện phép chia array, tensor, ta có thể sử dụng toán tử "/" hoặc hàm divide().



Hình 2: Minh họa phép chia Division

Ví dụ với thư viện Numpy:

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo hai array 1D
5 arr_1 = np.array([1, 2, 3, 4])
6 arr_2 = np.array([5, 6, 7, 8])
7 # In ra giá trị của hai array
8 print("arr_1:\n", arr_1)
9 print("arr_2:\n", arr_2)
10 # Chia hai array
11 # Cách 1: Sử dụng toán tử '/'
12 result_div_1 = arr_1 / arr_2
13 # Cách 2: Sử dụng hàm np.divide()
14 result_div_2 = np.divide(arr_1, arr_2)
15 # In ra màn hình
16 print("Cách 1:\n", result_div_1)
17 print("Cách 2:\n", result_div_2)

```

```

=====
Output =====
arr_1:
[1 2 3 4]

arr_2:
[5 6 7 8]

Cách 1:
[0.2  0.33333333 0.42857143 0.5]

Cách 2:
[0.2  0.33333333 0.42857143 0.5]

=====
```

Ví dụ với thư viện Pytorch:

```

1 # Pytorch code
2 import torch
3
4 # Tạo hai tensor 1D
5 tensor_1 = torch.tensor([1, 2, 3, 4])
6 tensor_2 = torch.tensor([5, 6, 7, 8])
7 # In ra giá trị của hai tensor
8 print("tensor 1:\n", tensor_1)
9 print("tensor 2:\n", tensor_2)
10 # Chia hai tensor
11 # Cách 1: Sử dụng toán tử '/'
12 result_div_1 = tensor_1 / tensor_2
13 # Cách 2: Sử dụng hàm torch.div()
14 result_div_2 = torch.divide(tensor_1,
                           tensor_2)
15 # In ra màn hình
16 print("Cách 1:\n", result_div_1)
17 print("Cách 2:\n", result_div_2)

```

```

=====
Output =====
tensor 1:
tensor([1, 2, 3, 4])

tensor 2:
tensor([5, 6, 7, 8])

Cách 1:
tensor([0.2000, 0.3333, 0.4286, 0.5000])

Cách 2:
tensor([0.2000, 0.3333, 0.4286, 0.5000])

=====
```

Ví dụ với thư viện Tensorflow:

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo hai tensor 1D
5 tensor_1 = tf.constant([1, 2, 3, 4])
6 tensor_2 = tf.constant([5, 6, 7, 8])
7 # In ra giá trị của hai tensor
8 print("tensor 1:\n", tensor_1)
9 print("tensor 2:\n", tensor_2)
10 # Chia hai tensor
11 # Cách 1: Sử dụng toán tử '/'
12 result_div_1 = tensor_1 / tensor_2
13 # Cách 2: Sử dụng hàm tf.divide()
14 result_div_2 = tf.divide(tensor_1,
15                         tensor_2)
16 # In ra màn hình
17 print("Cách 1:\n", result_div_1)
18 print("Cách 2:\n", result_div_2)

```

```

=====
tensor 1:
tf.Tensor([1 2 3 4], shape=(4,), dtype=int32)

tensor 2:
tf.Tensor([5 6 7 8], shape=(4,), dtype=int32)

Cách 1:
tf.Tensor([0.2 0.33333333 0.42857143
          0.5],
          shape=(4,), dtype=float64)

Cách 2:
tf.Tensor([0.2 0.33333333 0.42857143
          0.5],
          shape=(4,), dtype=float64)
=====
```

2. Bài tập

Cho hai list 2D như sau:

```

1 lst_1 = [[1, -2, 1],
2             [-3, 1, 4],
3             [-2, 5, 1]]
4
5 lst_2 = [[1, 3, 5],
6             [2, 4, 6],
7             [3, 5, 7]]

```

Hãy tạo 2 array, tensor(Pytorch, Tensorflow) từ list 2D này sau đó thực hiện các phép tính multiplication và division giữa hai array, tensor vừa tạo.

3. Đáp án

Các Phép Tính Numpy, Pytorch và Tensorflow - Phần 3

Dinh-Tiem Nguyen và Quang-Vinh Dinh

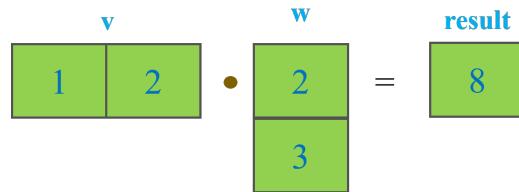
1. Mô tả

a) Inner product

Phép tích tích vô hướng (inner product), còn được biết đến với tên gọi khác là dot product. Phép tích vô hướng giữa hai vector có thể được tính để đo lường sự tương quan hoặc hướng của chúng.

Cho hai vectơ $\mathbf{a} = [a_1, a_2, \dots, a_n]$ và $\mathbf{b} = [b_1, b_2, \dots, b_n]$ trong không gian Euclidean \mathbb{R}^n , phép tích vô hướng của chúng được xác định bởi công thức:

$$\mathbf{a} \cdot \mathbf{b} = a_1b_1 + a_2b_2 + \dots + a_nb_n$$



Hình 1: Minh họa inner

Trong thư viện Numpy, hàm `np.dot()` được sử dụng để thực hiện phép nhân ma trận, và nó cũng hỗ trợ tích vô hướng cho các vectơ(array 1D). Cú pháp:

```
1 np.dot(a, b)
```

Trong đó a, b là các array để thực hiện phép nhân. Các chiều của a và b phải thích hợp để thực hiện phép nhân (ví dụ, số cột của a phải bằng số hàng của b).

```
1 # Numpy code
2 import numpy as np
3
4 # Tạo hai array 1D
5 arr_1 = np.array([1, 2])
6 arr_2 = np.array([2, 3])
7 # In ra giá trị của hai array
8 print("arr_1:\n", arr_1)
9 print("arr_2:\n", arr_2)
10 # Sử dụng hàm np.dot()
11 result_dot = np.dot(arr_1, arr_2)
12 # In ra màn hình
13 print("Kết quả tích vô hướng arr_1, arr_2:\n", result_dot)
```

```
===== Output =====
arr_1:
[1 2]

arr_2:
[2 3]

Kết quả tích vô hướng arr_1, arr_2:
8

=====
```

Trong Pytorch, hàm `torch.dot()` được sử dụng để thực hiện tích vô hướng (dot product) giữa hai vectơ. Tuy nhiên khác với Numpy có thể tính tích của các array nhiều chiều thì hàm này chỉ cho phép tính tích vô hướng của hai vector 1D. Cú pháp:

```
1 torch.dot(tensor1, tensor2)
```

Trong đó, tensor1, tensor2: Hai tensor có cùng kích thước để thực hiện tích vô hướng. Kết quả trả về sẽ là một số vô hướng.

```
1 # Pytorch code
2 import torch
3
4 # Tạo hai tensor 1D
5 tensor_1 = torch.tensor([1, 2])
6 tensor_2 = torch.tensor([2, 3])
7 # In ra giá trị của hai tensor
8 print("tensor 1:\n", tensor_1)
9 print("tensor 2:\n", tensor_2)
10 # Sử dụng hàm torch.dot()
11 result_dot = torch.dot(tensor_1, tensor_2)
12 # In ra màn hình
13 print("Kết quả tích vô hướng tensor_1,\n    tensor_2:\n", result_dot)
```

```
===== Output =====
arr_1:
[1 2]

arr_2:
[2 3]

Kết quả tích vô hướng arr_1, arr_2:
8

=====
```

Trong Tensorflow, hàm **tf.tensordot()** được sử dụng để thực hiện phép nhân tensor với nhau theo các chiều chỉ định. Nó có thể được sử dụng để thực hiện nhiều loại phép nhân tensor, bao gồm cả phép nhân ma trận và tích vô hướng. Cú pháp:

```
1 tf.tensordot(a, b, axes)
```

Trong đó, các tham số được định nghĩa như sau:

- a, b: Hai tensor để thực hiện phép nhân.
- axes: Một tuple hoặc số nguyên chỉ định các chiều của a và b được sử dụng trong phép nhân. Nếu là một số nguyên, nó xác định số chiều được sử dụng trong cả a và b.

```
1 # TensorFlow code
2 import tensorflow as tf
3
4 # Tạo hai tensor 1D
5 tensor_1 = tf.constant([1, 2])
6 tensor_2 = tf.constant([2, 3])
7
8 # In ra giá trị của hai tensor
9 print("tensor 1:\n", tensor_1)
10 print("tensor 2:\n", tensor_2)
11
12 # Tích vô hướng hai tensor
13 result_dot = tf.tensordot(tensor_1,
    tensor_2, axes=1)
14
15 # In ra màn hình
16 print("Kết quả tích vô hướng tensor_1,\n    tensor_2:\n", result_dot)
```

```
===== Output =====
arr_1:
[1 2]

arr_2:
[2 3]

Kết quả tích vô hướng arr_1, arr_2:
8

=====
```

Tóm lại thì khi thực hiện tích vô hướng hai vector thì chúng ta dùng dot product. Với thư viện Numpy và Tensorflow thì chúng ta có thể áp dụng dot product với array, tensor nhiều chiều phức tạp hơn.

b) *Matrix-matrix multiplication*

Trong toán học, phép nhân ma trận (matrix multiplication) thường được biểu diễn bằng dấu chấm (\cdot) hoặc dấu nhân (\times). Các ma trận A và B có thể được nhân với nhau để tạo ra một ma trận kết quả C , với điều kiện số cột của ma trận A bằng số hàng của ma trận B .

Giả sử A là một ma trận có kích thước $m \times n$ (m hàng, n cột), và B là một ma trận có kích thước $n \times p$ (n hàng, p cột), thì ma trận kết quả C sẽ có kích thước $m \times p$. Quy tắc này còn được biết đến như là quy tắc hàng-cột, vì mỗi phần tử của ma trận kết quả C được tính bằng cách lấy tổng của tích từng phần tử của hàng tương ứng của ma trận A và cột tương ứng của ma trận B .

Phép nhân ma trận $C = A \times B$ có thể được mô tả như sau:

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}$$

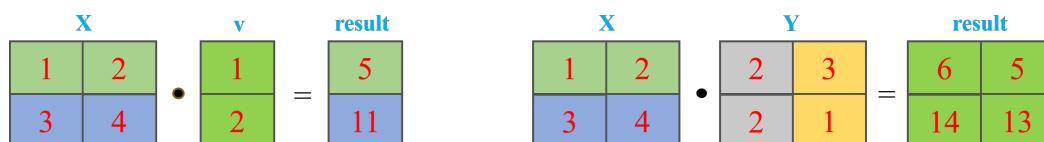
Trong đó:

- C_{ij} là phần tử ở hàng thứ i và cột thứ j của ma trận C .
- A_{ik} là phần tử ở hàng thứ i và cột thứ k của ma trận A .
- B_{kj} là phần tử ở hàng thứ k và cột thứ j của ma trận B .
- Phép toán $\sum_{k=1}^n$ biểu thị việc tính tổng qua tất cả các giá trị của k từ 1 đến n .

Ví dụ:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Trong trường hợp này, $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$, $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$, và tương tự cho các phần tử còn lại của ma trận C .



Hình 2: Minh họa phép nhân ma trận

Trong thư viện Numpy, hàm **np.matmul()** được sử dụng để thực hiện phép nhân ma trận. Hàm này cung cấp một cách linh hoạt để thực hiện nhân ma trận giữa các mảng (hoặc tensor) Numpy. Sử dụng cú pháp:

```
1 np.matmul(a, b, out=None)
```

Trong đó các tham số được định nghĩa như sau:

- a, b: Các ma trận (hoặc mảng) để thực hiện phép nhân. Số cột của ma trận a phải bằng số hàng của ma trận b.
- out: Mảng kết quả. Nếu được chỉ định, kết quả sẽ được lưu trữ vào mảng này.

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo 2 array 2D
5 arr_1 = np.array([[1, 2], [3, 4]])
6 arr_2 = np.array([[5, 6], [7, 8]])
7
8 # In ra giá trị của hai array
9 print("arr_1:\n", arr_1)
10 print("arr_2:\n", arr_2)
11
12 # Sử dụng hàm np.matmul()
13 result_matmul = np.matmul(arr_1, arr_2)
14
15 # In ra màn hình
16 print("Kết quả tích arr_1, arr_2:\n",
      result_matmul)

```

```

=====
Output =====
arr_1:
 [[1 2]
 [3 4]]

arr_2:
 [[5 6]
 [7 8]]

Kết quả tích arr_1, arr_2:
 [[19 22]
 [43 50]]

=====

```

Ví dụ PytorchPytorch:

```

1 # Pytorch code
2 import torch
3
4 # Tạo hai tensor 2D
5 tensor_1 = torch.tensor([[1, 2], [3, 4]])
6 tensor_2 = torch.tensor([[5, 6], [7, 8]])
7
8 # In ra giá trị của hai tensor
9 print("tensor 1:\n", tensor_1)
10 print("tensor 2:\n", tensor_2)
11
12 # Sử dụng hàm torch.matmul()
13 result_matmul = torch.matmul(tensor_1,
     tensor_2)
14
15 # In ra màn hình
16 print("Kết quả tích tensor_1, tensor_2:\n",
      result_matmul)

```

```

=====
Output =====
tensor 1:
 tensor([[1, 2],
 [3, 4]])

tensor 2:
 tensor([[5, 6],
 [7, 8]])

Kết quả tích tensor_1, tensor_2:
 tensor([[19, 22],
 [43, 50]])

=====

```

Trong Pytorch, hàm **torch.matmul()** được sử dụng để thực hiện phép nhân ma trận (matrix multiplication) giữa hai tensor. Hàm này hỗ trợ cả tính tích vô hướng cho vectơ và cũng có thể được sử dụng cho các trường hợp phức tạp hơn. Cú pháp sử dụng là:

```
1 torch.matmul(input, other, out=None)
```

Hàm này sẽ trả về kết quả của phép nhân ma trận, trong đó các tham số được định nghĩa như sau:

- input: Tensor đầu tiên để thực hiện phép nhân ma trận.
- other: Tensor thứ hai để thực hiện phép nhân ma trận.
- out: Tensor để lưu trữ kết quả. Nếu không được chỉ định, một tensor mới sẽ được tạo.

Hàm **tf.matmul()** trong Tensorflow được sử dụng để thực hiện phép nhân ma trận giữa hai tensor. Nó hỗ trợ cả việc thực hiện phép nhân ma trận và tích vô hướng cho các tensor có số chiều khác nhau. Cú pháp:

```
1 tf.matmul(a, b, transpose_a=False, transpose_b=False)
```

Hàm này trả về kết quả là phép nhân ma trận, trong đó các tham số được định nghĩa như sau:

- a, b: Hai tensor để thực hiện phép nhân ma trận.
- transpose_a, transpose_b: Xác định xem có nên chuyển vị các tensor a và b trước khi thực hiện phép nhân hay không.

Ví dụ:

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo hai tensor 2D
5 tensor_1 = tf.constant([[1, 2], [3, 4]])
6 tensor_2 = tf.constant([[5, 6], [7, 8]])
7
8 # In ra giá trị của hai tensor
9 print("tensor 1:\n", tensor_1)
10 print("tensor 2:\n", tensor_2)
11
12 # Tích hai tensor
13 result_matmul = tf.matmul(tensor_1,
14                           tensor_2)
15
16 # In ra màn hình
17 print("Kết quả tích tensor_1, tensor_2:\n",
18       , result_matmul)

===== Output =====
tensor 1:
tf.Tensor(
[[1 2]
 [3 4]], shape=(2, 2), dtype=int32)

tensor 2:
tf.Tensor(
[[5 6]
 [7 8]], shape=(2, 2), dtype=int32)

Kết quả tích tensor_1, tensor_2:
tf.Tensor(
[[19 22]
 [43 50]], shape=(2, 2), dtype=int32)
=====
```

Mặc dù hàm matmul khá giống với dot product ở phần trước, nhưng hai hàm này là khác nhau nên ta cần chú ý khi sử dụng chúng.

2. Bài tập

Câu 1: Cho hai vector $a = [1, 4, 7]$, $b = [9, 2, 3]$ Tính tích vô hướng của hai vector này bằng ba thư viện Numpy, Pytorch và Tensorflow

Câu 2: Viết chương trình tạo hai Numpy array, Pytorch tensor, Tensorflow tensor với các giá trị số ngẫu nhiên trong khoảng [-10, 10] với kích thước (3, 3). Hãy tính matrix multiplication hai ma trận này bằng ba thư viện Numpy, Pytorch và Tensorflow. Lưu ý: sử dụng seed=2024

3. Đáp án

Các Phép Tính Numpy, Pytorch và Tensorflow Transpose và Summation

Dinh-Tiem Nguyen và Quang-Vinh Dinh

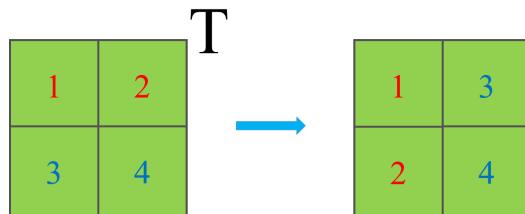
1. Mô tả

a) Chuyển vị - Transpose

Chuyển vị (Transpose) là một phép toán quan trọng trong đại số tuyến tính, nó thực hiện thay đổi vị trí của các hàng thành cột và ngược lại trong một ma trận. Kí hiệu của phép toán chuyển vị thường được biểu diễn bằng kí hiệu T hoặc \top . Nếu A là một ma trận, thì chuyển vị của A được kí hiệu là A^T hoặc A^\top .

Công Thức Chuyển Vị: Nếu A là một ma trận với các phần tử a_{ij} , thì chuyển vị của A , ký hiệu là A^T hay A^\top , có kích thước là số cột của A trở thành số hàng của A và ngược lại. Cụ thể, nếu A có kích thước $m \times n$, thì A^T có kích thước $n \times m$.

$$(A^T)_{ij} = A_{ji}$$



Hình 1: Minh họa transpose

Ví dụ:

Giả sử có ma trận A như sau:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Chuyển vị của A , ký hiệu là A^T , sẽ là:

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Một số tính chất của phép chuyển vị:

- Chuyển vị của một ma trận chuyển vị lại sẽ cho ra ma trận ban đầu: $(A^T)^T = A$.
- Chuyển vị của tổng hai ma trận bằng tổng chuyển vị của từng ma trận: $(A + B)^T = A^T + B^T$.
- Chuyển vị của tích hai ma trận bằng tích đảo ngược vị trí của chuyển vị từng ma trận: $(AB)^T = B^T A^T$.

Ví dụ:

```

1 # Numpy code
2 import numpy as np
3
4 #Tạo array 2d
5 arr_1 = np.array([[1, 2], [3, 4]])
6 # Chuyển vị array
7 # Cách 1: Sử dụng hàm np.transpose()
8 arr_transposed_1 = np.transpose(arr_1)
9 # Cách 2: Sử dụng toán tử T
10 arr_transposed_2 = arr_1.T
11 # In ra màn hình
12 print("array 1:\n", arr_1)
13 print("array 1 sau khi chuyển vị, cách 1:\n", arr_transposed_1)
14 print("array 1 sau khi chuyển vị, cách 2:\n", arr_transposed_2)

```

```

=====
Output =====
array 1:
[[1 2]
 [3 4]]

array 1 sau khi chuyển vị, cách 1:
[[1 3]
 [2 4]]

array 1 sau khi chuyển vị, cách 2:
[[1 3]
 [2 4]]

=====
```

Trong Numpy, chuyển vị của một mảng (array) có thể được thực hiện bằng cách sử dụng hàm `np.transpose()` hoặc toán tử chuyển vị `.T`. Chuyển vị thay đổi vị trí của các hàng thành cột và ngược lại trong array.

Trong Pytorch, chuyển vị của tensor có thể được thực hiện bằng cách sử dụng toán tử chuyển vị `.T`, hàm `torch.t()` hoặc `torch.transpose()`. Chuyển vị thay đổi vị trí của các hàng thành cột và ngược lại trong tensor. Cú pháp:

```

1 #cách 1:
2 torch.t(input)
3
4 #Cách 2
5 torch.transpose(input, dim0, dim1)

```

Kết quả trả về là một tensor mới là kết quả của phép chuyển vị. Trong đó:

- input: Tensor cần được chuyển vị.
- dim0, dim1: Các chiều được chọn để thực hiện chuyển vị.

Ví dụ:

```

1
2 #Pytorch code
3 import torch
4
5 #Tạo tensor 2d
6 tensor_1 = torch.tensor([[1, 2], [3, 4]])
7 # Chuyển vị tensor
8 # Cách 1: Sử dụng hàm torch.t()
9 tensor_transposed_1 = torch.t(tensor_1)
10 # Cách 2: Sử dụng hàm torch.transpose()
11 tensor_transposed_2 = torch.transpose(
12     tensor_1, 0, 1)
13 # In ra màn hình
14 print("tensor 1:\n", tensor_1)
15 print("tensor 1 sau khi chuyển vị, cách
1: \n", tensor_transposed_1)
16 print("tensor 1 sau khi chuyển vị, cách
2: \n", tensor_transposed_2)

```

```

=====
Output =====
tensor 1:
tensor([[1, 2],
       [3, 4]])

tensor 1 sau khi chuyển vị, cách 1:
tensor([[1, 3],
       [2, 4]])

tensor 1 sau khi chuyển vị, cách 2:
tensor([[1, 3],
       [2, 4]])

=====
```

Trong Tensorflow, không có toán tử chuyển vị **T**, mà chỉ dùng hàm **tf.transpose()**. Cách sử dụng cũng tương tự như Numpy và Pytorch:

Ví dụ:

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor 2d
5 tensor_1 = tf.constant([[1, 2], [3, 4]])
6 # Chuyển vị tensor
7 tensor_transposed = tf.transpose(tensor_1)
8
9 # In ra màn hình
10 print("tensor 1:\n", tensor_1)
11 print("tensor 1 sau khi chuyển vị:\n",
      tensor_transposed)

```

```

=====
Output =====
tensor 1:
tf.Tensor(
[[1 2]
 [3 4]], shape=(2, 2), dtype=int32)

tensor 1 sau khi chuyển vị:
tf.Tensor(
[[1 3]
 [2 4]], shape=(2, 2), dtype=int32)
=====
```

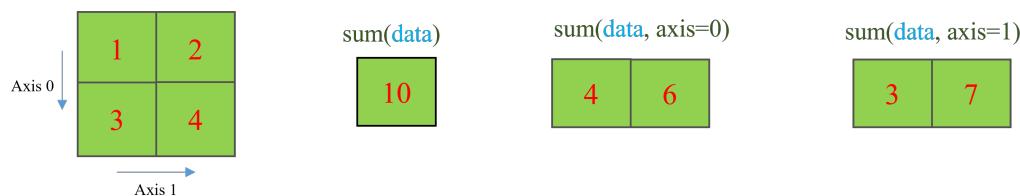
b) *Summation*

Trong thư viện Numpy, hàm **np.sum()** được sử dụng để tính tổng của các phần tử trong mảng (array). Hàm này có thể được áp dụng trên các mảng 1D, 2D hoặc có số chiều cao hơn. Cú pháp:

```
1 np.sum(a, axis=None, dtype=None, keepdims=False, initial=0, where=True)
```

Trong đó:

- a: Mảng đầu vào.
- axis: (Tùy chọn) Chiều hoặc các chiều trên đó tổng sẽ được thực hiện. Mặc định là None, tức là tổng của tất cả các phần tử trong mảng.
- dtype: (Tùy chọn) Kiểu dữ liệu của kết quả.
- keepdims: (Tùy chọn) Nếu là True, giữ chiều của mảng kết quả (nếu có) giống với chiều của mảng đầu vào.
- initial: (Tùy chọn) Giá trị khởi tạo cho tổng.
- where: (Tùy chọn) Một mảng Boolean chỉ định vị trí các phần tử được sử dụng trong phép toán.



Hình 2: Minh họa summation

Trong ví dụ sau, **np.sum()** được sử dụng để tính tổng của mảng array. Tham số sử dụng ở đây là mặc định khi tính tổng các phần tử trong toàn bộ array, sử dụng tham số axis để tính tổng theo cột hoặc hàng. Kết quả được in ra màn hình bao gồm tổng của tất cả các phần tử, tổng theo cột, và tổng theo hàng của mảng.

```

1 # Numpy code
2 import numpy as np
3
4 # Tạo mảng 2D
5 arr = np.array([[1, 2, 3],
6                 [4, 5, 6]])
7
8 # Tính tổng của tất cả các phần tử trong mảng
9 total_sum = np.sum(arr)
10
11 # Tính tổng theo cột (theo chiều dọc)
12 column_sum = np.sum(arr, axis=0)
13
14 # Tính tổng theo hàng (theo chiều ngang)
15 row_sum = np.sum(arr, axis=1)
16
17 # In ra màn hình
18 print("Mảng:\n", arr)
19 print("Tổng của tất cả các phần tử trong mảng:\n", total_sum)
20 print("Tổng theo cột:\n", column_sum)
21 print("Tổng theo hàng:\n", row_sum)

```

```

===== Output =====
Mảng:
[[1 2 3]
 [4 5 6]]

Tổng của tất cả các phần tử trong mảng:
21

Tổng theo cột:
[5 7 9]

Tổng theo hàng:
[ 6 15]

=====

```

Trong Pytorch, chúng ta tính sum tương tự như trong Numpy, ví dụ:

```

1 #Pytorch code
2 import torch
3
4 # Tạo tensor 2d
5 tensor_1 = torch.tensor([[1, 2], [3, 4]])
6 # Tính tổng tensor
7 tensor_sum = torch.sum(tensor_1)
8 # Tính tổng tensor theo cột axis = 0
9 tensor_sum_axis_0 = torch.sum(tensor_1,
                               axis=0)
10 # Tính tổng tensor theo hàng axis = 1
11 tensor_sum_axis_1 = torch.sum(tensor_1,
                               axis=1)
12
13 # In ra màn hình
14 print("tensor:\n", tensor_1)
15 print("Tổng các phần tử trong tensor:\n",
      tensor_sum)
16 print("Tổng theo cột:\n",
      tensor_sum_axis_0)
17 print("Tổng theo hàng:\n",
      tensor_sum_axis_1)

```

```

===== Output =====
tensor:
tensor([[1, 2],
       [3, 4]])

Tổng các phần tử trong tensor:
tensor(10)

Tổng theo cột:
tensor([4, 6])

Tổng theo hàng:
tensor([3, 7])

=====

```

Ví dụ tính sum trong Tensorflow:

```

1 # Tensorflow code
2 import tensorflow as tf
3
4 # Tạo tensor 2d
5 tensor_1 = tf.constant([[1, 2], [3, 4]])
6 # Tính tổng tensor
7 tensor_sum = tf.reduce_sum(tensor_1)
8 # Tính tổng tensor theo cột axis = 0
9 tensor_sum_axis_0 = tf.reduce_sum(tensor_1
10 , axis=0)
11 # Tính tổng tensor theo hàng axis = 1
12 tensor_sum_axis_1 = tf.reduce_sum(tensor_1
13 , axis=1)
14
15 # In ra màn hình
16 print("tensor:\n", tensor_1)
17 print("Tổng các phần tử trong tensor:\n",
18      tensor_sum)
19 print("Tổng theo cột:\n",
20      tensor_sum_axis_0)
21 print("Tổng theo hàng:\n",
22      tensor_sum_axis_1)

```

===== Output =====

tensor:
tf.Tensor(
[[1 2]
[3 4]], shape=(2, 2), dtype=int32)

Tổng các phần tử trong tensor:
tf.Tensor(10, shape=(), dtype=int32)

Tổng theo cột:
tf.Tensor([4 6], shape=(2,), dtype=int32)

Tổng theo hàng:
tf.Tensor([3 7], shape=(2,), dtype=int32)

=====

2. Bài tập

Câu 1: Viết chương trình tạo hai Numpy array, Pytorch tensor, Tensorflow tensor với các giá trị số nguyên ngẫu nhiên trong khoảng [-10, 10) với kích thước (3, 4). Sau đó chuyển vị array, tensor thứ 2 và thực hiện phép nhân matrix multiplication. Lưu ý: sử dụng seed=2024

Câu 2: Viết chương trình tạo một Numpy array, Pytorch tensor, Tensorflow tensor với các giá trị số nguyên ngẫu nhiên trong khoảng [-10, 10) với kích thước (3, 3). Sau đó hãy tính tổng của toàn bộ tensor, array, tiếp theo tính tổng theo chiều dọc, chiều ngang . Lưu ý: sử dụng seed=2024

3. Đáp án

Đáp án sẽ được gửi cho các bạn vào khoảng 8h tối trên group Code.