

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

LÊ THANH PHONG

PYTHON
VÀ ỨNG DỤNG
TRONG KỸ THUẬT

Bản quyền thuộc tác giả

MỤC LỤC

PHẦN 1: HƯỚNG DẪN LẬP TRÌNH BẰNG PYTHON.

1. Giới thiệu Python.....	1
2. Cài đặt.....	1
2.1. Cài đặt Python	1
2.2. Cài đặt VSCode	2
3. Sử dụng trên online.....	4
3.1. Sử dụng trực tiếp từ địa chỉ https://jupyter.org/	4
3.2. Sử dụng trực tiếp từ địa chỉ https://colab.research.google.com/	7
4. Soạn thảo văn bản trên Markdown	8
5. Các ký tự đặc biệt trong Python	9
6. Một số hàm cơ bản có sẵn trên Python gốc	10
6.1. Hàm print.....	10
6.2. Hàm round.....	11
6.3. Hàm abs.....	11
6.4. Hàm type	11
6.5. Hàm range	12
6.5.1.Hàm range tạo một tập hợp số có cấu trúc.	12
6.5.2.Hàm range kết hợp với vòng lặp for và hàm append trong gói numpy dùng để tạo mảng	13
7. Cách sử dụng hàm trong các gói cài đặt bổ sung	14
8. Hàm do người dùng tự định nghĩa, tạo ra	15
8.1. Tạo hàm bằng từ khóa lambda	15
8.2. Tạo hàm bằng từ khóa def.....	16
8.3. Vị trí tạo hàm và nơi lưu trữ của hàm	17
9. Các toán tử trong Python	18
9.1. Toán tử số học	18
9.2. Toán tử gán	18
9.3. Toán tử quan hệ	19
9.4. Toán tử logic.....	19
9.5. Toán tử tìm kiếm, xác thực.....	19
10. Các mệnh đề, vòng lặp điều khiển trong Python	20
10.1. Điều kiện if.....	20

10.1.1. Điều kiện if một nhánh rẽ	20
10.1.2. Điều kiện if hai nhánh rẽ.....	20
10.1.3. Điều kiện if nhiều hơn hai nhánh rẽ.....	21
10.1.4. Điều kiện if nhiều nhánh rẽ lồng nhau.....	21
10.2. Vòng lặp for	21
10.3. Vòng lặp while	22
11. Hướng dẫn sử dụng gói numy cơ bản.....	22
11.1. Mảng trong numpy	22
11.1.1. Tạo mảng một chiều (1D).....	22
11.1.2. Tạo mảng hai chiều (2D)	24
11.1.3. Tạo mảng 3-D	25
11.2. Các phép toán trên mảng.....	26
11.2.1. Phép toán cơ bản từng cặp phần tử tương ứng giữa hai mảng	26
11.2.2. Phép toán nhân ma trận, véc tơ.....	27
11.3. Chuyển đổi chiều và dạng của mảng.....	28
11.4. Chuyển trí (chuyển vị) của mảng	29
11.5. Hàm tạo mảng đặc biệt.....	29
11.5.1. Hàm arange	29
11.5.2. Hàm linspace	30
11.5.3. Hàm zeros	31
11.5.4. Hàm ones	31
11.5.5. Hàm eye	32
11.5.6. Hàm tri	32
11.5.7. Hàm tril	32
11.5.8. Hàm triu	33
11.6. Tạo mảng bằng cách ghép hai mảng đã có.....	33
11.7. Mở rộng mảng, thêm phần tử vào mảng đã có.....	34
11.8. Cách tham chiếu đến phần tử của mảng.....	35
11.9. Đại số tuyến tính (Linear algebra) trong numpy	38
11.9.1. Tìm giá trị và vector riêng	39
11.9.2. Tính chuẩn của một matrix hoặc vector	39
11.9.3. Tính định thức của matrix.....	40
11.9.4. Tìm nghịch đảo của matrix	40
11.9.5. Giải hệ phương trình đại số tuyến tính	40
12. Hướng dẫn sử dụng gói matplotlib cơ bản	41

12.1. Hàm plot.....	43
12.2. Hàm figure.....	45
12.3. Hàm subplot	46
13. Hướng dẫn sử dụng gói sympy cơ bản	47
13.1. Hàm tạo biến trong sympy	47
13.2. Hàm subs	47
13.3. Hàm evalf	47
13.4. Hàm diff	48
13.5. Hàm integrate	48
13.6. Hàm plot.....	49
13.7. Hàm solve.....	49
13.8. Hàm linsolve	50
13.9. Hàm nonlinsolve	50
13.10. Gói sympy.physics.continuum_mechanics.beam tính dầm	51
14. Hướng dẫn sử dụng gói tabulate cơ bản	52

PHẦN 2: ÚNG DỤNG TRONG KỸ THUẬT.

1. Đạo hàm của hàm số, dãy số54
 - 1.1. Đạo hàm của hàm số54
 - 1.2. Đạo hàm của dãy số
2. Tích phân của hàm số, dãy số.
 - 2.1. Đạo hàm của hàm số
 - 2.2. Đạo hàm của dãy số
3. Nội suy và xấp xỉ.
4. Giải phương trình phi tuyến một biến.
5. Giải hệ phương trình tuyến tính.
6. Giải hệ phương trình phi tuyến.
7. Giải phương trình và hệ phương trình vi phân bài toán trị đầu.
8. Giải phương trình và hệ phương trình vi phân bài toán trị biên.

PHẦN 1: HƯỚNG DẪN LẬP TRÌNH BẰNG PYTHON.

1. Giới thiệu Python.

Python là một ngôn ngữ lập trình thông dịch (interpreted), hướng đối tượng (object-oriented), và là một ngôn ngữ bậc cao (high-level) ngữ nghĩa động (dynamic semantics).

Python hỗ trợ các module và gói (packages), có thể tạo chương trình module hóa và tái sử dụng mã.

Trình thông dịch Python và thư viện chuẩn mở rộng có sẵn dưới dạng mã nguồn mở hoặc dạng nhị phân miễn phí cho tất cả các nền tảng chính và có thể được phân phối tự do.

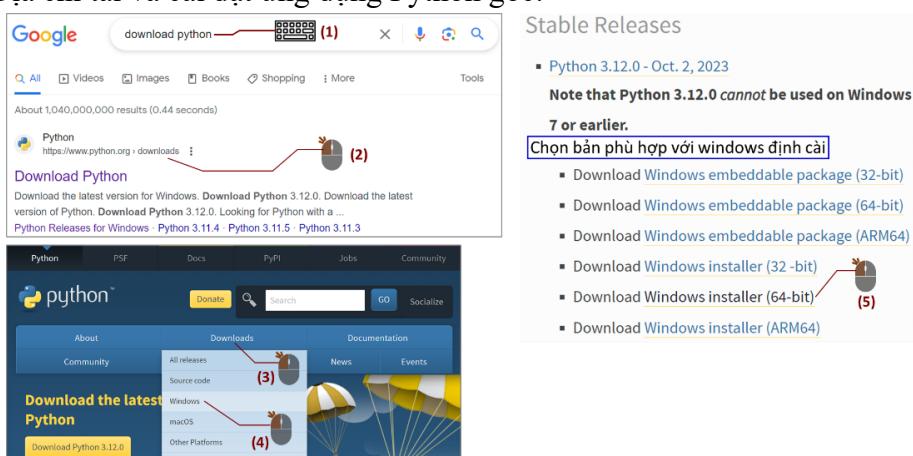
Các điểm nổi bật:

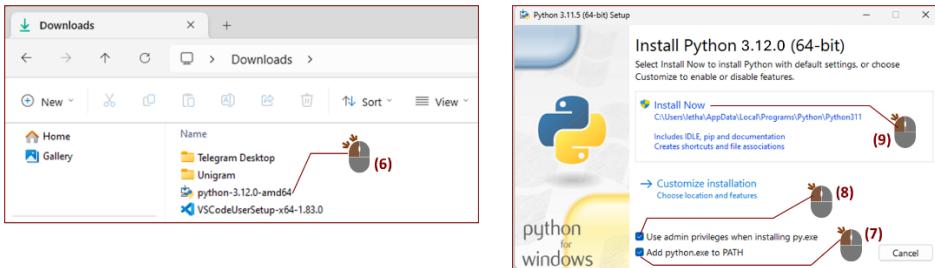
- Đơn giản và dễ học: Lập trình bằng Python có hình thức sáng sủa, cấu trúc rõ ràng, cú pháp ngắn gọn, có cộng đồng lập trình rất lớn, hệ thống thư viện chuẩn được chia sẻ trên mạng.
- Là ngôn ngữ mã nguồn mở: Cài đặt Python dùng giấy phép nguồn mở nên được sử dụng và phân phối tự do, ngay cả trong việc thương mại.
- Là ngôn ngữ có khả năng chạy trên nhiều nền tảng: Python có cho mọi hệ điều hành như Windows, Linux/Unix, Mac, Android...
- Với cùng một mã nguồn (Code) sẽ chạy giống nhau trên mọi nền tảng. Dễ dàng kết nối với các ngôn ngữ khác: Python có thể viết, tạo các thư viện để nhúng, liên kết với C, C++, Excel, Autocad... và ngược lại.

2. Cài đặt.

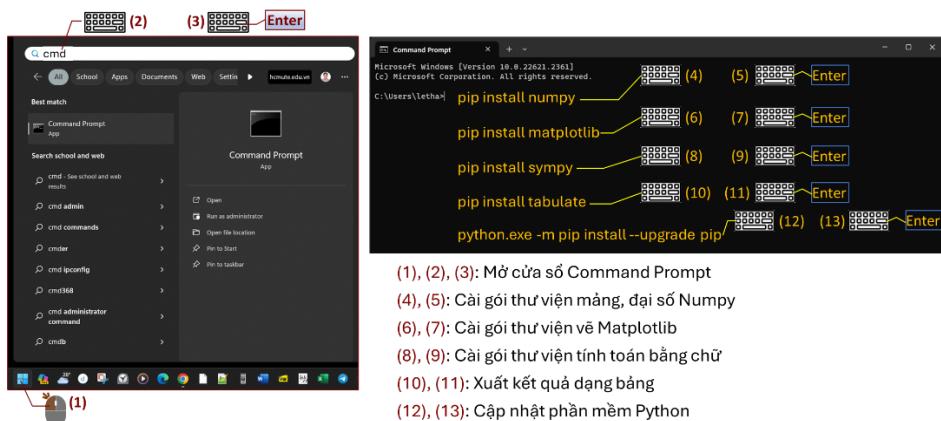
2.1. Cài đặt Python.

Địa chỉ tải và cài đặt ứng dụng Python gốc:





Cài đặt các gói bổ sung cho Python:



Các gói cài đặt bổ sung (thư viện hàm ứng dụng) rất đa dạng, cho rất nhiều lĩnh vực khoa học khác nhau. Những gói phổ biến trong kỹ thuật bao gồm:

numpy: (Địa chỉ: <https://numpy.org/>) Là gói cung cấp thư viện với đầy đủ hàm dùng để xử lý các phép toán trên mảng một chiều – véc tơ (vector), mảng hai chiều – ma trận (matrix), và mảng ba chiều. Các phép toán này thường dùng trong đại số tuyến tính và được ứng dụng trong các tính toán kỹ thuật.

matplotlib: (Địa chỉ: <https://matplotlib.org/>) Là một trong những gói mở rộng của Python phổ biến nhất được sử dụng để trực quan hóa dữ liệu. Thư viện bao gồm các hàm có thể sử dụng trên đa nền tảng để tạo ra nhiều loại biểu đồ, đồ thị từ dữ liệu trong các mảng.

sympy: (Địa chỉ: <https://www.sympy.org/>) Là gói mở rộng của Python, cung cấp thư viện bao gồm các hàm dùng để tính toán về số học, đại số, giải tích mà lời giải cho kết quả bằng chữ (symbol).

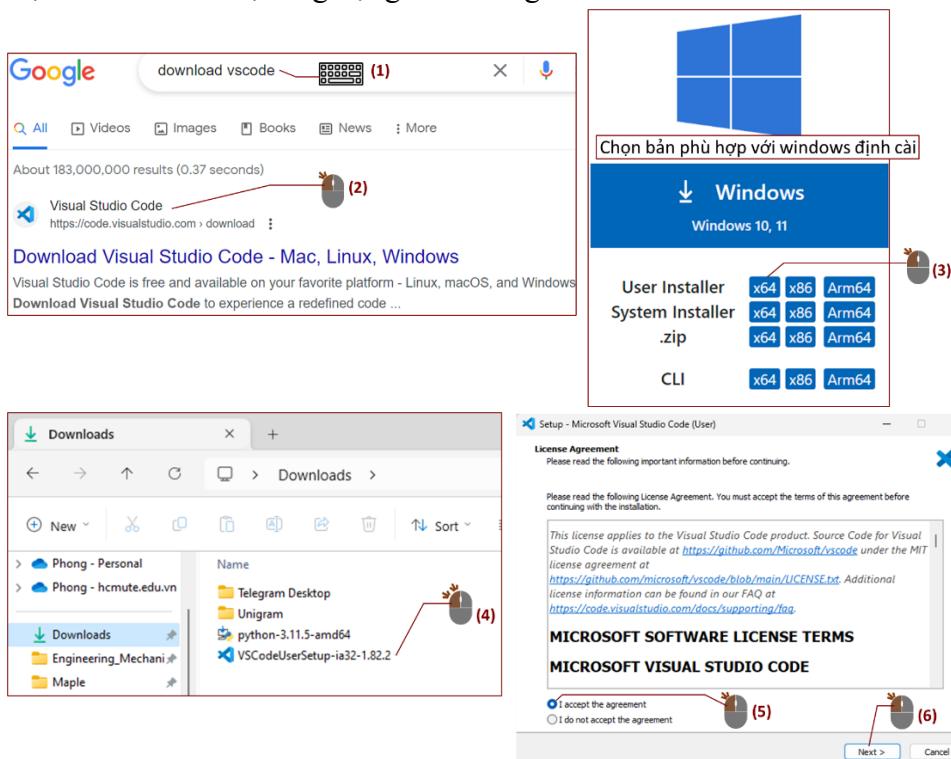
2.2. Cài đặt VSCode.

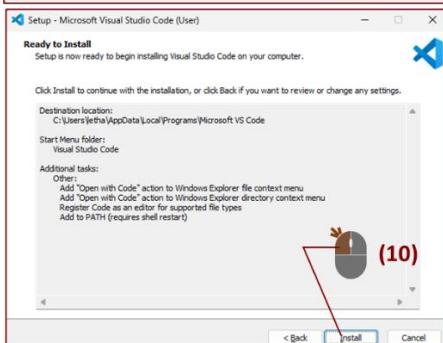
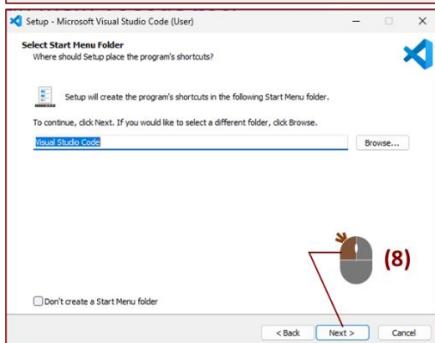
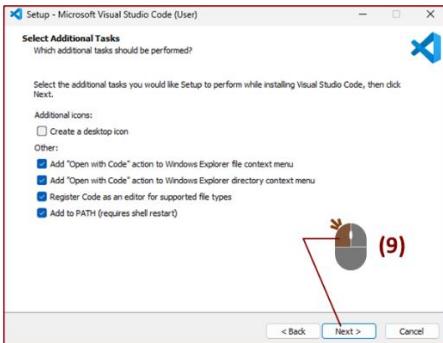
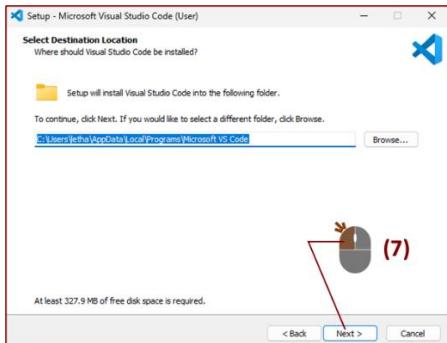
Jupyter là một thuật ngữ được ghép từ ba ngôn ngữ lập trình Julia, Python và R. Trước đây Jupyter Notebook có tên là IPython Notebook, đến năm 2014 tách ra khỏi IPython và đổi tên thành Jupyter Notebook. Trong ứng dụng Jupyter có thể làm được:

- Tạo tài liệu dạng văn bản, chèn bảng biểu, công thức toán học, hình ảnh, video trong môi trường Markdown.
- Viết chương trình (Code) bằng ngôn ngữ Python hay nhiều ngôn ngữ khác.
- Chạy chương trình và xuất kết quả dạng chữ, số, đồ thị trực quan.
- Toàn bộ văn bản, code, kết quả, hình ảnh, đồ thị... đều nằm trong một tài liệu, có thể xuất ra dạng .html hay .pdf dùng để xem hoặc in ấn.

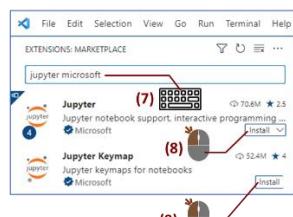
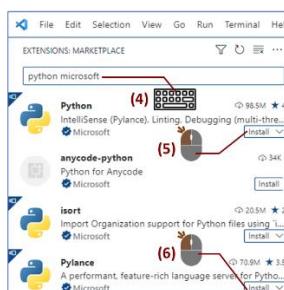
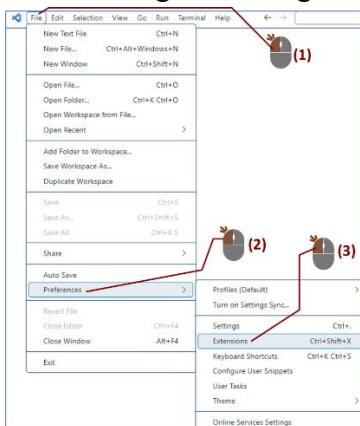
Ứng dụng VSCode có thể chạy đơn lẻ Python hay Jupyter Notebook nên rất thuận tiện cho việc lập trình.

Địa chỉ tải và cài đặt ứng dụng VSCode gốc:





Cài đặt các gói mở rộng cho VSCode:

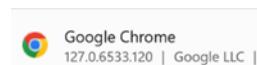


(1), (2), (3), phím tắt **[Ctrl]+[Shift]+[X]**: Vào phần cài đặt mở rộng cho VSCode
(4), (5), (6): Tim và cài đặt phần mở rộng Python trên VSCode
(7), (8), (9): Tim và cài đặt phần mở rộng Jupyter trên VSCode

3. Sử dụng trên online.

3.1. Sử dụng trực tiếp từ địa chỉ <https://jupyter.org/>

Mở một trình duyệt web:



Truy cập vào địa chỉ <https://jupyter.org/>

The screenshot shows the official Jupyter website at https://jupyter.org/. The navigation bar includes links for 'Try', 'Install', 'Get Involved', and 'Documentation'. The 'Try' button is highlighted with a red border.

Click chọn Try

The screenshot shows the 'Try' page of the Jupyter website. It features three cards: 'JupyterLab' (selected), 'Jupyter Notebook', and 'JupyterLite'. Each card has a small icon and a brief description.

Click chọn JupyterLab hoặc Jupyter Notebook

Detailed description of the cards:

- JupyterLab:** The latest web-based interactive development environment. It features the Jupyter logo icon.
- Jupyter Notebook:** The original web application for creating and sharing computational documents. It features the Python logo icon.
- JupyterLite:** JupyterLite (Wasm powered Jupyter) deployed as static GitHub Pages. It features a lightbulb icon with the letter 'jl' inside.

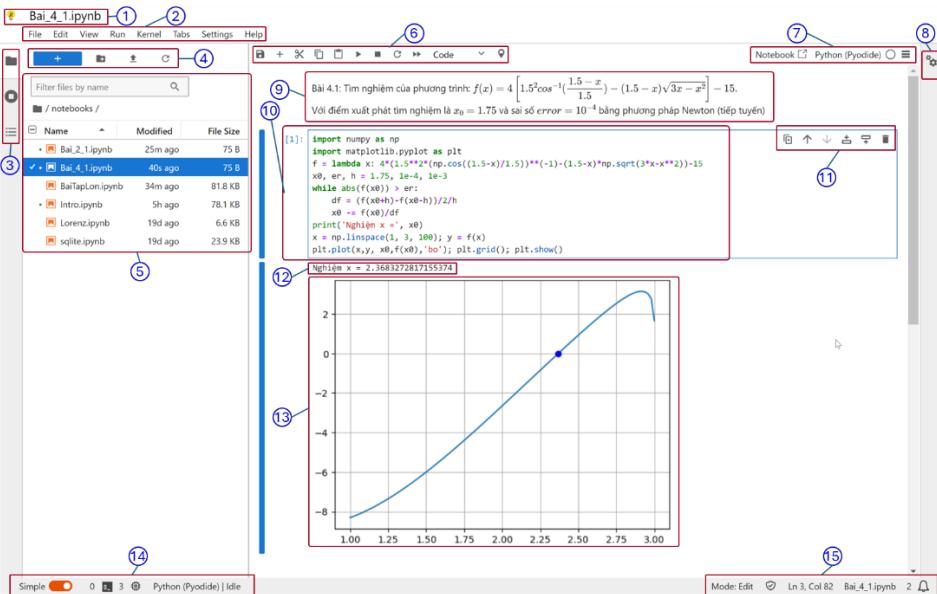
JupyterLab: Ưu điểm là có nhiều cửa sổ dạng thẻ dùng để quản lý thư mục, tập tin, soạn thảo chương trình, upload file từ máy tính, download file về máy tính, tùy chỉnh giao diện, trạng thái... Dẫn đến nhược điểm là giao diện phức tạp, không tập trung vào vùng lập trình soạn thảo.

Jupyter Notebook: Ưu điểm là có giao diện lập trình soạn thảo đơn giản, nhẹ nhàng giúp tập trung vào việc lập trình nhưng không cung cấp các tùy chọn nâng cao khác.

Tùy theo mục đích công việc để chọn JupyterLab hay Jupyter Notebook. Việc chuyển đổi qua lại giữa hai môi trường cũng đơn giản:

The screenshot shows two side-by-side Jupyter interface windows. The top window is 'BàiTapLon.ipynb' in 'Notebook' mode, with a red box around the 'Notebook' tab. The bottom window is also 'BàiTapLon.ipynb' but in 'JupyterLab' mode, with a red box around the 'JupyterLab' tab. Both windows show the same content: 'Từ JupyterLab muốn chuyển qua Jupyter Notebook' and 'Từ Jupyter Notebook muốn chuyển qua JupyterLab'.

Giao diện và các vùng làm việc trên JupyterLab:



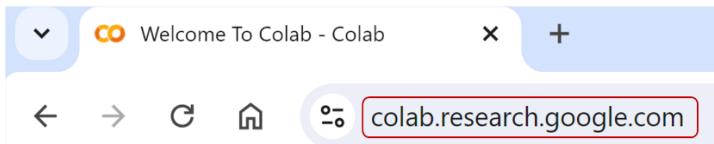
- (1). Tiêu đề (Tên file)
- (2). Menu sổ xuống (Có đa số các cài đặt)
- (3). Ân, hiện: Thư mục, file; đầu cuối, nhân; Nội dung bảng biểu.
- (4). Tạo file mới; Tạo thư mục; Upload file từ máy tính; Làm tươi.
- (5). Hiển thị thư mục, file đã tạo.
- (6). Lưu file; Chèn, cắt, chép, dán cell (ô); Chạy cell hiện hành; Dừng chạy; Khởi động lại nhân; Khởi động lại nhân và chạy tất cả các cell; Thay đổi môi trường soạn thảo Markdown hoặc Code; Xem hướng dẫn.
- (7). Chuyển đổi môi trường giữa JupyterLab và Jupyter Notebook; Thay đổi nhân; Hiển thị toàn vùng soạn thảo.
- (8). Cài đặt giao diện nâng cao.
- (9). Vùng soạn thảo văn bản bằng Markdown.
- (10). Vùng soạn thảo chương trình (code).
- (11). Quản lý cell: Sao chép, di chuyển cell, chèn cell mới, xóa cell.
- (12). Xuất kết quả dạng văn bản.
- (13). Xuất kết quả dạng đồ thị.
- (14). Trạng thái hiển thị thẻ, nhân, chuyển đổi nhân.
- (15). Hiển thị trạng thái của file, vị trí dòng cột con trỏ, tên file.

3.2. Sử dụng trực tiếp từ địa chỉ <https://colab.research.google.com/>

Mở một trình duyệt web:



Truy cập vào địa chỉ <https://colab.research.google.com/>



Click chọn: + New notebook (Hoặc Welcome To Colab)

A screenshot of the "Open notebook" dialog box. It lists recent notebooks under "Recent": "Welcome To Colab" (selected), "ViDu_3_10.ipynb", and "ViDu1.ipynb". A blue button labeled "+ New notebook" is highlighted with a red border at the bottom left. A "Cancel" button is at the bottom right.

Bắt đầu sử dụng:

A screenshot of a Colab notebook. The sidebar on the left has icons for file operations. The main area shows a code cell with the following Python code:

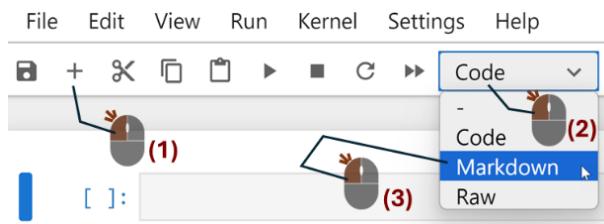
```
1 import numpy as np
2 f = lambda x: (3*x**2-np.pi)*np.sin(x)
3 x = np.array([1, 2, 3])
4 fx = f(x)
5 print('f(x) =', fx)
```

The output cell shows the result of running the code:

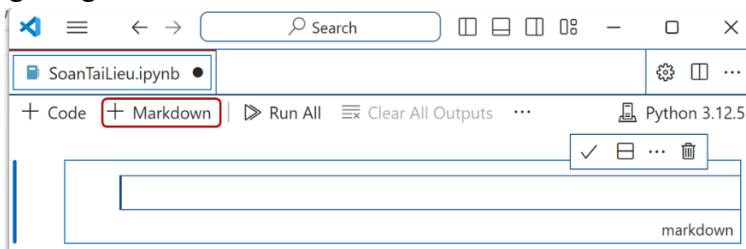
```
f(x) = [-0.11914611  8.05492701  3.36689864]
```

4. Soạn thảo văn bản trên Markdown.

Trên trang <http://jupyter.org/>



Trên ứng dụng VSCode:



Soạn thảo	Hiển thị
<p>Không tiêu đề</p> <p># Tiêu đề cấp 1</p> <p>## Tiêu đề cấp 2</p> <p>### Tiêu đề cấp 3</p> <p>#### Tiêu đề cấp 4</p> <p>##### Tiêu đề cấp 5</p> <p>###### Tiêu đề cấp 6</p>	<p>Không tiêu đề</p> <p>Tiêu đề cấp 1</p> <p>Tiêu đề cấp 2</p> <p>Tiêu đề cấp 3</p> <p>Tiêu đề cấp 4</p> <p>Tiêu đề cấp 5</p> <p>Tiêu đề cấp 6</p>
<p>Chữ thường</p> <p>*Chữ đậm* (Hoặc thay ** bằng __)</p> <p>*<i>Chữ nghiêng</i>* (Hoặc thay * bằng __)</p> <p>***Chữ đậm + nghiêng*** (Hoặc thay *** bằng __)</p>	<p>Chữ thường</p> <p>Chữ đậm</p> <p><i>Chữ nghiêng</i></p> <p>Chữ đậm + nghiêng</p>
<p>$\alpha, \beta, \gamma, \Delta, \delta, \varepsilon, \mu, \pi, \rho, \Sigma, \sigma, \tau, \phi, \varphi, \Omega, \omega$</p>	<p>$\alpha, \beta, \gamma, \Delta, \delta, \varepsilon, \lambda, \mu, \pi, \rho, \Sigma, \sigma, \tau, \phi, \varphi, \Omega, \omega$</p>

$\$ \times, \#, \leq, \geq, \approx, \neq, \pm$ $\$ \Rightarrow, \rightarrow, \Leftrightarrow$													
$\$x^2, y_1, z_3^4, \sqrt{6x+y}, \sqrt[5]{xyz}$ $\$ \int (3x+5)dx, \int (x^2-x)dx$ $\$ \int_{-5}^3 6x^4 dx, \int_1^5 x^4 dx$ $\$ \sum_{i=1}^n (3i+5)$													
$\$ \frac{x^5-4x^3+2}{2x-3}, \frac{y^6+3y-2}{4x+3}$													
$\$ A=\begin{matrix}1&2\\3&4\end{matrix};$ $B=\begin{pmatrix}5&6\\7&8\end{pmatrix};$ $C=\begin{vmatrix}9&0\\1&2\end{vmatrix};$ $D=\begin{bmatrix}3&4\\5&6\end{bmatrix};$ $E=\begin{Bmatrix}7&8\\9&0\end{Bmatrix};$ $\$ f=\left\{\begin{array}{l} -5x_1+4x_2=4 \\ 2x_1+7x_2=-1 \end{array}\right.$	$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}; B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}; C = \begin{vmatrix} 9 & 0 \\ 1 & 2 \end{vmatrix};$ $D = \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}; E = \begin{Bmatrix} 7 & 8 \\ 9 & 0 \end{Bmatrix}$ $f = \begin{cases} -5x_1 + 4x_2 = 4 \\ 2x_1 + 7x_2 = -1 \end{cases}$												
$ i 0 1 2 $ $:--: :--: :--: :--: $ $ \$x_i\$ 4 11 18 $ $ \$y_i\$ 26.5 279.8 1189.1 $	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>i</th><th>0</th><th>1</th><th>2</th></tr> <tr> <td>x_i</td><td>4</td><td>11</td><td>18</td></tr> <tr> <td>y_i</td><td>26.5</td><td>279.8</td><td>1189.1</td></tr> </table>	i	0	1	2	x_i	4	11	18	y_i	26.5	279.8	1189.1
i	0	1	2										
x_i	4	11	18										
y_i	26.5	279.8	1189.1										

5. Các ký tự đặc biệt trong Python.

Chữ HOA khác với chữ thường, do đó cần lưu ý khi đặt tên biến.

A = 5

a = 4

Chú thích trong một khối được mở và đóng ba nháy đôi hoặc đơn.

....

Tiêu đề, mô tả...

Trong môi trường Python
xem trong KHỐI này là chú
thích, không biên dịch

....

....

Tiêu đề, mô tả...

Trong môi trường Python
xem trong KHỐI này là chú
thích, không biên dịch

....

Chú thích sau dòng lệnh được sử dụng dấu tăng.

x = (3 + 9)/4 # Chú thích cho dòng lệnh

Viết nhiều dòng lệnh trên một dòng soạn thảo thì ngăn cách các câu lệnh bằng dấu chấm phẩy.

```
x = 4; y = 7; z = 5
```

Ngắt dòng lệnh (quá dài) thành nhiều dòng soạn thảo bằng ký tự \

```
b = 5; c = 8; h = 12  
Iy = (2 * b * h ^ 3 + (2 * b + 3 * c) ^ 2 \\  
     * b * h + 3 * c * h ^ 3) / 18
```

Gán một lần cho nhiều biến ngăn cách bằng dấu phẩy.

```
x, y, z = 3, 8, 6
```

6. Một số hàm cơ bản có sẵn trên Python gốc.

6.1. Hàm print.

Sử dụng rất nhiều dùng để xuất, trả kết quả, chuỗi, dòng ký tự...

```
x = (5 + 9)/4  
y = 2 + 7  
print('In chuỗi ký tự. x =', x, '; y =', y)
```

[1] ✓ 0.0s
... In chuỗi ký tự. x = 3.5; y = 9

Giữ chỗ kết quả xen lẩn trong chuỗi (dòng text)

```
name = 'Họ Và Tên'  
age = 18  
weight = 65.2  
print('Tên: {}, Tuổi: {}, Nặng: {}kg'.format(name, age, weight))
```

[1] ✓ 0.0s
... Tên: Họ Và Tên, Tuổi: 18, Nặng: 65.2kg

Giữ chỗ kết quả xen lẩn trong chuỗi (dòng text), kết quả xuống dòng.

```
x = 15/7  
y = 4*x**3 - 5*x  
print(f' x = {x} \n y = {y} ')
```

[1] ✓ 0.0s
... x = 2.142857142857143
y = 28.644314868804663

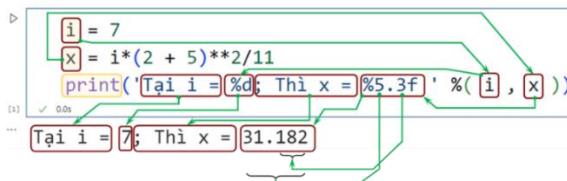
Định dạng và giữ chỗ của kết quả xen lẫn trong chuỗi (dòng text)

%d : Xác kết quả theo kiểu số nguyên.

%T.Sf : Xác kết quả theo kiểu số thập phân.

T là Tổng của số chữ số muốn xuất.

S là tổng số chữ số sau dấu phẩy muốn giữ lại.



6.2. Hàm round.

Dùng để làm tròn số. Tham số thứ nhất là số muốn làm tròn, tham số thứ hai là số chữ số sau dấu phẩy muốn giữ lại.

```
1 a = 1.2345; print('In giá trị: a =', a)
2 b = round(a, 1); print('In kết quả làm tròn: b =', b)
[1] ✓ 0.0s
...
In giá trị: a = 1.2345
In kết quả làm tròn: b = 1.2
```

6.3. Hàm abs.

Dùng để lấy giá trị tuyệt đối của một số.

```
1 c = -2.435; print('In giá trị: c =', c)
2 d = abs(c); print('In kết quả lấy trị tuyệt đối: d =', d)
[2] ✓ 0.0s
...
In giá trị: c = -2.435
In kết quả lấy trị tuyệt đối: d = 2.435
```

6.4. Hàm type.

Xem định dạng kiểu của biến, môi trường...

```
a = 5 ;           print('type(a):', type(a))
b = 5. ;          print('type(b):', type(b))
c = 2+3j ;        print('type(c):', type(c))
d = 'Dòng tex'; print('type(d):', type(d))
e = [1, 2, 3];   print('type(e):', type(e))
f = (1, 2, 3);   print('type(f):', type(f))
g = 3>5;         print('type(g):', type(g))
[1] ✓ 0.0s
...
type(a): <class 'int'>
type(b): <class 'float'>
type(c): <class 'complex'>
type(d): <class 'str'>
type(e): <class 'list'>
type(f): <class 'tuple'>
type(g): <class 'bool'>
```

6.5. Hàm range.

6.5.1. Hàm range tạo một tập hợp số có cấu trúc.

`range(stop)` tạo mảng có *stop* phần tử:

- Phần tử thứ '0' (đầu tiên) bằng 0 (mặc định).
- Bước bằng 1 (mặc định).
- Phần tử thứ '*stop - 1*' (cuối cùng) bằng *stop - 1*.

```
> v
    stop = 4
    x = range(stop)
    print('x =', x)
    print('x =', x[0], x[1], x[2], x[3])
[1] ✓ 0.0s
...
... x = range(0, 4)
x = 0 1 2 3
```

`range(start, stop)` tạo mảng có *stop - start* phần tử:

- Phần tử thứ '0' (đầu tiên) bằng *start*.
- Bước bằng 1 (mặc định).
- Phần tử thứ '*stop - start - 1*' (cuối cùng) bằng *stop - 1*.

```
> v
    start, stop = 5, 9
    x = range(start, stop)
    print('x =', x)
    print('x =', x[0], x[1], x[2], x[3])
[1] ✓ 0.0s
...
... x = range(5, 9)
x = 5 6 7 8
```

`range(start, stop, step)` tạo mảng có *n* phần tử:

- Phần tử thứ '0' (đầu tiên) bằng *start*.
- Bước bằng *step*.
- Phần tử thứ '*n - 1*' (cuối cùng) bằng *start + (n - 1) × step ≤ stop - 1*

```
> v
    start, stop, step = 2, 14, 3
    x = range(start, stop, step)
    print('x =', x)
    print('x =', x[0], x[1], x[2], x[3])
[1] ✓ 0.0s
...
... x = range(2, 14, 3)
x = 2 5 8 11
```

6.5.2. Hàm range kết hợp với vòng lặp for và hàm append trong gói numpy dùng để tạo mảng.

for i in range(stop):

```
import numpy as np
stop = 4; In = []; x = []
for i in range(stop):
    In = np.append(In, i)
    Tinh = i * 4
    x = np.append(x, Tinh)
print('Các giá trị của i =', In)
print('Các giá trị của mảng x =', x)
```

[1] ✓ 0.1s
... Các giá trị của i = [0. 1. 2. 3.]
Các giá trị của mảng x = [0. 4. 8. 12.]

- i lần lượt nhận các giá trị:

0 1 2 ... stop - 1

for i in range(start, stop):

```
import numpy as np
start, stop = 4, 9; In = []; x = []
for i in range(start, stop):
    In = np.append(In, i)
    Tinh = i * 2.7
    x = np.append(x, Tinh)
print('Các giá trị của i =', In)
print('Các giá trị của mảng x =', x)
```

[1] ✓ 0.0s
... Các giá trị của i = [4. 5. 6. 7. 8.]
Các giá trị của mảng x = [10.8 13.5 16.2 18.9 21.6]

- i lần lượt nhận các giá trị:

start start + 1 start + 2 ... stop - 1

for i in range(start, stop, step):

```
import numpy as np
start, stop, step = 2, 14, 3; In = []; x = []
for i in range(start, stop, step):
    In = np.append(In, i)
    Tinh = i * 2.7
    x = np.append(x, Tinh)
print('Các giá trị của i =', In)
print('Các giá trị của mảng x =', x)
```

[1] ✓ 0.0s
... Các giá trị của i = [2. 5. 8. 11.]
Các giá trị của mảng x = [5.4 13.5 21.6 29.7]

- i lần lượt nhận các giá trị:

start + 0 * step start + 1 * step start + 2 * step ... start + n * step $\leq (stop - 1)$

7. Cách sử dụng hàm trong các gói cài đặt bổ sung.

Để sử dụng được thư viện hàm có trong các gói (package), module đã cài đặt thông qua lệnh pip thì phải nạp (import) package, module đó trước dòng lệnh sử dụng hàm.

Cách nạp thư viện vào	Chú thích
<pre>▷ > import numpy x = numpy.sin(numpy.pi/2) print('x =', x) [1] ✓ 0.0s ... x = 1.0</pre>	Nạp gói numpy và sử dụng hai hàm sin, pi có trong gói này. Cách này khi sử dụng dài, phức tạp
<pre>▷ > import numpy as np y = np.sqrt(5**2 + np.pi) print('y =', y) [1] ✓ 0.0s ... y = 5.3048649986205865</pre>	Nạp gói numpy và rút gọn tên gói thành tên định danh np . Khi dùng hai hàm căn và pi thông qua tên rút gọn này (Dùng phổ biến)
<pre>▷ > from numpy import sqrt, exp x = exp(-sqrt(3)) print('x =', x) [1] ✓ 0.0s ... x = 0.17692120631776423</pre>	Từ gói numpy, chỉ nạp vào hai hàm căn bậc hai và e mũ. Cách này tuy gọn nhưng khó kiểm soát trong một dự án lớn.
<pre>▷ > from numpy import * x = array([sin(3*pi/4), pi, cos(3*pi/4)]) print('x =', x) [1] ✓ 0.1s ... x = [0.70710678 3.14159265 -0.70710678]</pre>	Từ gói numpy, nạp vào tất cả các hàm. Không khuyến khích vì có nhiều hàm tồn tại trên các gói khác nhau, không kiểm soát.

Để in danh sách tất cả các hàm trong gói sympy, sử dụng hàm dir có trong Python.

```
▷ > import sympy as sp
      print('Danh sách hàm trong gói sympy: \n', dir(sp))
[1]   ✓ 0.4s
...
Danh sách hàm trong gói sympy:
['Abs', 'AccumBounds', 'Add', 'Adjoint', 'AlgebraicField',
```

Để in mô tả công dụng, hướng dẫn cách sử dụng, ví dụ... của hàm thì sử dụng đuôi `__doc__` vào sau tên hàm.

```

> v
    import matplotlib.pyplot as plt
    print('Xem hướng dẫn sử dụng hàm plot: \n', plt.plot.__doc__)
[1]  ✓ 0.4s
...
Xem hướng dẫn sử dụng hàm plot:
Plot y versus x as lines and/or markers.

```

Call signatures::

```

plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

```

The coordinates of the points or line nodes are given by *x*, *y*.

The optional parameter *fmt* is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the *Notes* section below.

```

>>> plot(x, y)          # plot x and y using default line style and color
>>> plot(x, y, 'bo')   # plot x and y using blue circle markers
>>> plot(y)            # plot y using x as index array 0..N-1
>>> plot(y, 'r+')      # ditto, but with red plusses

```

8. Hàm do người dùng tự định nghĩa, tạo ra.

Hàm là một đoạn code (khối lệnh) được tạo ra theo cú pháp được qui định cụ thể. Mục đích tạo hàm là để tái sử dụng nhiều lần trong chương trình. Thông thường, sử dụng hàm là truyền vào các tham biến và trả về các giá trị.

8.1. Tạo hàm bằng từ khóa lambda.

Hàm tạo ra phải nằm trên một dòng và hàm chỉ trả về một giá trị.

```

import numpy as np
Wr = lambda d: pi*d**3/16; pi = np.pi
Wx = lambda b, h: b*h**2/6
d, Mz = 6, 9
ta = Mz / Wr(d); print('tau =', ta, 'kN/cm^2')
b, h, Mx = 5, 9, 7
si = Mx / Wx(b,h); print('sigma =', si, 'kN/cm^2')

```

```

tau = 0.2122065907891938 kN/cm^2
sigma = 0.1037037037037037 kN/cm^2

```

8.2. Tạo hàm bằng từ khóa def.

```

def BacHai( a , b=1 , c=2 ) :
    """
    Hàm giải: a*x^2 + b*x + c = 0
    Nhập: a - Bắt buộc
           b - Tùy chọn. (Không nhập thì b=1)
           c - Tùy chọn. (Không nhập thì c=0)
    Xuất: x1, x2 ''
    """
    Delta = b**2 - 4*a*c
    if Delta < 0:
        x = 'Phương trình vô nghiệm'
    elif Delta == 0:
        x = round(-b/2/a, 2)
    else:
        x1 = round((-b + Delta**(1/2))/2/a, 2)
        x2 = round((-b - Delta**(1/2))/2/a, 2)
        x = [x1, x2]
    return x

print(BacHai.__doc__)
x = BacHai(5); print('5x^2+x+2=0 -> x =', x)
print('-3x^2+2x+5=0 -> x =', BacHai(-3,2,5))

```

[1] ✓ 0.0s

Hàm giải: a*x^2 + b*x + c = 0
 Nhập: a - Bắt buộc
 b - Tùy chọn. (Không nhập thì b=1)
 c - Tùy chọn. (Không nhập thì c=0)
 Xuất: x1, x2
 $5x^2+x+2=0 \rightarrow x =$ Phương trình vô nghiệm
 $-3x^2+2x+5=0 \rightarrow x = [-1.0, 1.67]$

- (1). Từ khóa tạo hàm. (Bắt buộc)
- (2). Tên hàm. (Không có khoảng trắng và các ký tự đặc biệt)
- (3). Tham biến bắt buộc.
- (4). Tham biến tùy chọn (đứng sau tham biến bắt buộc).
- (5). Kết thúc tiêu đề tạo hàm.
- (6). Thụt vào một Tab. (Bắt buộc)
- (7). Lệnh in (xuất) ra docstring: Mô tả, hướng dẫn sử dụng hàm.
- (8). Lời gọi sử dụng hàm, kết quả được gán cho biến x.
- (9). Tạo docstring: Mô tả, hướng dẫn sử dụng hàm.
- (10). Phần thân của hàm.
- (11). Từ khóa trả giá trị kết quả của hàm. (Bắt buộc)
- (12). In ra kết quả khi gọi hàm.
- (13). Xuất docstring.
- (14). Xuất kết quả khi gọi hàm.
- (15). Xuất kết quả khi gọi hàm tiếp theo.

8.3. Vị trí tạo hàm và nơi lưu trữ của hàm.

Hàm được tạo trước lời gọi sử dụng hàm trong cùng nơi của chương trình chính (trên một file hay một cell)

```
MatCatChuNhat.ipynb X
+ Code + Markdown | Run All | Clear All Outputs | Outline ...
Wx = lambda b, h: b*h**3/12 ①
def Sigma(Mx, Wx):
    Si = Mx/Wx
    return Si ②
M, b, h = 2e3, 5, 10
W = Wx(b, h)
S = Sigma(M, W)
print('Üng suất trên tiết diện sigma =', S) ③
[1] ...
Üng suất trên tiết diện sigma = 4.8
```

- (1). Tạo hàm trước khi sử dụng.
- (2). Tạo hàm trước khi sử dụng.
- (3). Chương trình chính, sử dụng hàm.

Hàm được tạo trên một file khác với định dạng `.py` và lưu vào cùng thư mục với file của chương trình chính.

- Đối với file dùng để tạo hàm thì phải lưu dưới định dạng `.py`
- Trên một file, thường được gọi là module có thể tạo ra nhiều hàm khác nhau.
- Với file dùng để viết chương trình chính thì có thể lưu định dạng `.py` hoặc `.ipynb` đều được.
- Ví dụ:
 - (1). Tạo folder có tên **My_Python** trong ô D: D:\My_Python
 - (2). Tạo file (module) có tên **stress.py** trong nó tạo các hàm: **doc**, **xoan**, **uon**, **PhucTap** và lưu vào folder **My_Python**
 - (3). Tạo file (Chương trình chính) có tên **SucBenVatLieu.ipynb** có chứa dòng code: **import stress** và các nội dung khác rồi lưu vào folder **My_Python**
- (4). Chạy chương trình chính → sinh ra:

Folder con: **__pycache__** ở trong folder **My_Python**
File: **stress.cpython-311.pyc** ở trong folder **__pycache__**
- (5). Lúc này, ta có thể sử dụng các hàm tồn tại trong module **stress**: **stress.doc**, **stress.xoan**, **stress.uon**, **stress.PhucTap**

```

1 import numpy as np
2 def doc(Nz, F, scp):
3     s = Nz/F; dkb = 100*s/scp
4     return dkb
5 def xoan(Mz, d, tcp):
6     t = Mz*16*np.pi/d**3
7     dkb = 100*t/tcp
8     return dkb
9 def uon(Mx, Jx, ymax, scp):
10    s = Mx*ymax/Jx
11    dkb = 100*s/scp
12    return dkb
13 def PhucTap(Mx, My, Mz, Wu, scp):
14     s4 = np.sqrt(Mx**2 + My**2 + 0.75*Mz**2)/Wu
15     dkb = 100*s4/scp
16     return dkb

```

```

import stress
Nz, F, scp = 9, 0.4, 15
dkb = stress.doc(Nz, F, scp)
print('sigma = ', dkb, '% của sigma cho phép')

```

```

sigma = 150.0 % của sigma cho phép

```

9. Các toán tử trong Python.

9.1. Toán tử số học.

Toán tử	Mô tả	Ví dụ	Kết quả
+	Phép cộng hai số	$2 + 3$	5
-	Phép trừ hai số	$4 - 7$	-3
*	Phép nhân hai số	$6 * 5$	30
/	Phép chia hai số	$15 / 7$	2.142857142857143
//	Phép chia hai số lấy phần nguyên	$15 // 7$	2
%	Phép chia hai số lấy phần dư	$15 \% 7$	1
**	Phép lũy thừa của một số	$2 ** 3$	8
**(1/2)	Phép lấy căn bậc 2 của một số	$3 ** (1/2)$	1.7320508075688772

Thứ tự các phép toán được thực hiện:

Trong ngoặc → Lũy thừa → Nhân và chia → Cộng và trừ

9.2. Toán tử gán.

Dùng để gán giá trị của một đối tượng cho một đối tượng khác.

Toán tử	Mô tả	Ví dụ	Kết quả
=	Lấy giá trị của biến bên phải gán cho biến bên trái.	$a = 4$ $b = a$	$a = 4$ $b = 4$
+=	Lấy giá trị của biến bên trái (đã có) cộng với biến bên phải, kết quả gán cho biến bên trái (cập nhật)	$a = 4$ $a += 1$ $\sim (a = a + 1)$	$a = 4$ $a = 5$
-=	Lấy giá trị của biến bên trái (đã có) trừ cho biến bên phải, kết quả gán cho biến bên trái (cập nhật)	$a = 4$ $a -= 5$ $\sim (a = a - 5)$	$a = 4$ $a = -1$
*=	Lấy giá trị của biến bên trái (đã có) nhân cho biến bên phải, kết quả gán cho biến bên trái (cập nhật)	$a = 4$ $a *= 2$ $\sim (a = a * 2)$	$a = 4$ $a = 8$

$/=$	Lấy giá trị của biến bên trái (đã có) chia cho biến bên phải, kết quả gán cho biến bên trái (cập nhật)	$a = 4$ $a /= 2$ $\sim (a = a/2)$	$a = 4$ $a = 2$
------	--	---	--------------------

9.3. Toán tử quan hệ.

Dùng để so sánh các giá trị với nhau, kết quả trả về là **True** nếu đúng và **False** nếu sai. Thường được dùng trong các câu lệnh điều kiện.

Toán tử	Mô tả	Ví dụ	Kết quả
$=$	So sánh bằng nhau giữa hai đối số	$2 == 3$	False
$!=$ hoặc $<>$	So sánh khác nhau giữa hai đối số	$2 != 3$	True
$<$	So sánh liệu số bên trái có nhỏ hơn số bên phải không	$2 < 3$	True
$>$	So sánh liệu số bên trái có lớn hơn số bên phải không	$2 > 3$	False
\leq	So sánh liệu số bên trái có nhỏ hơn hoặc bằng số bên phải không	$4 \leq 4$	True
\geq	So sánh liệu số bên trái có lớn hơn hoặc bằng số bên phải không	$3 \geq 2$	True

9.4. Toán tử logic.

Dùng để kết hợp hoặc loại trừ của các kết quả của **True**, **False**. Kết quả trả về là **True** hoặc **False**. Thường được dùng trong các câu lệnh điều kiện.

Toán tử	Mô tả	Ví dụ	Kết quả
and	True and True = True	$a = 4$	$a = 4$
	True and False = False	$a \geq 3 \text{ and } a \leq 5 \sim 3 \leq a \leq 5$	True
	False and True = False	$a > 4 \text{ and } a < 5 \sim 4 < a < 5$	True
	False and False = False	$a > 5 \text{ and } a < 3 \sim 5 < a < 3$	False
or	True or True = True	$a = 4$	$a = 4$
	True or False = True	$a \geq 3 \text{ or } a \leq 5$	True
	False or True = True	$a > 4 \text{ or } a < 5$	True
	False or False = False	$a > 5 \text{ or } a < 3$	False
not	Phủ định của kết quả	$a = 4$ not $a > 4$	$a = 4$ True

9.5. Toán tử tìm kiếm, xác thực.

Dùng biến vé trái tìm trong một tập hợp. Nếu tìm có thì trả kết quả **True** và ngược lại thì **False**.

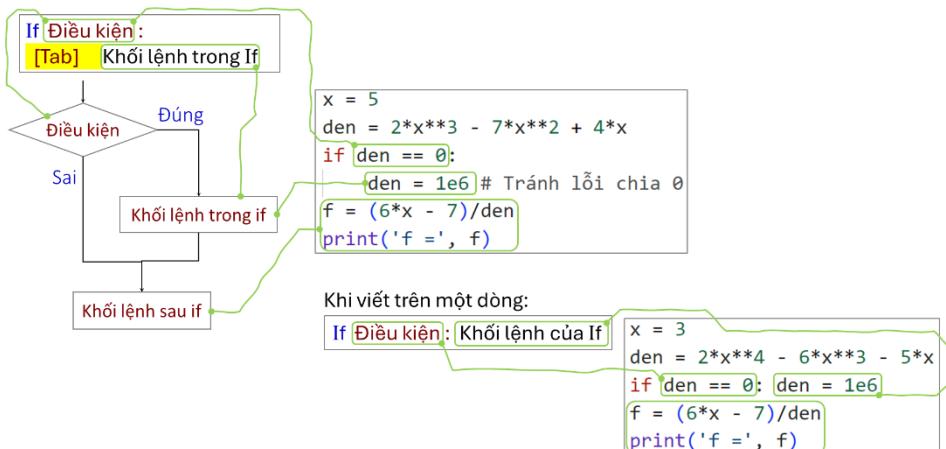
Toán tử	Mô tả	Ví dụ	Kết quả
in	Tìm giá trị vé trái xem có trong tập hợp bên vé phải không	$a = 4$ $b = [1, 3, 5]$ in b	$a = 4$ $b = [1, 3, 5]$ False
not in	Tìm giá trị vé trái xem liệu không có trong tập hợp bên vé phải không	$a = 4$ $b = [1, 3, 5]$	$a = 4$ $b = [1, 3, 5]$

		a not in b	True
is	Tìm giá trị về trái xem liệu có đúng bằng giá trị bên vệ phải không	a = 4; b = 5 a is b	a = 4; b = 5 False
not is	Tìm giá trị về trái xem có phải không bằng giá trị vệ phải không	a = 4; b = 5 a not is b	a = 4; b = 5 True

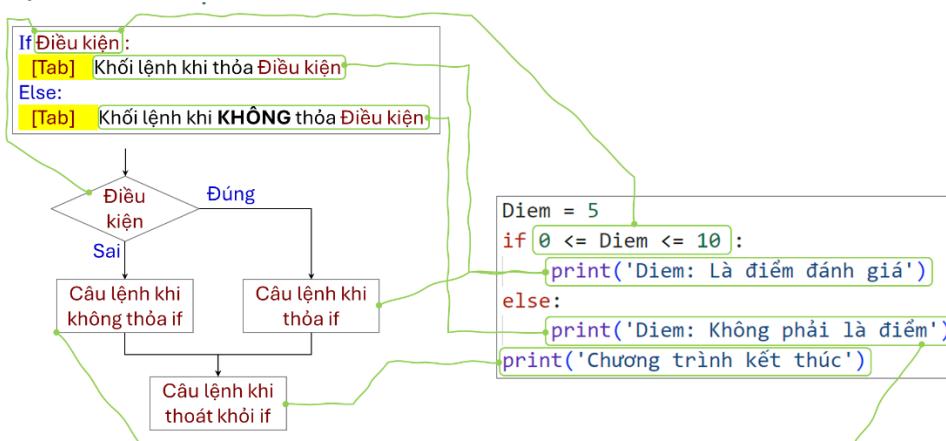
10. Các mệnh đề, vòng lặp điều khiển trong Python.

10.1. Điều kiện if.

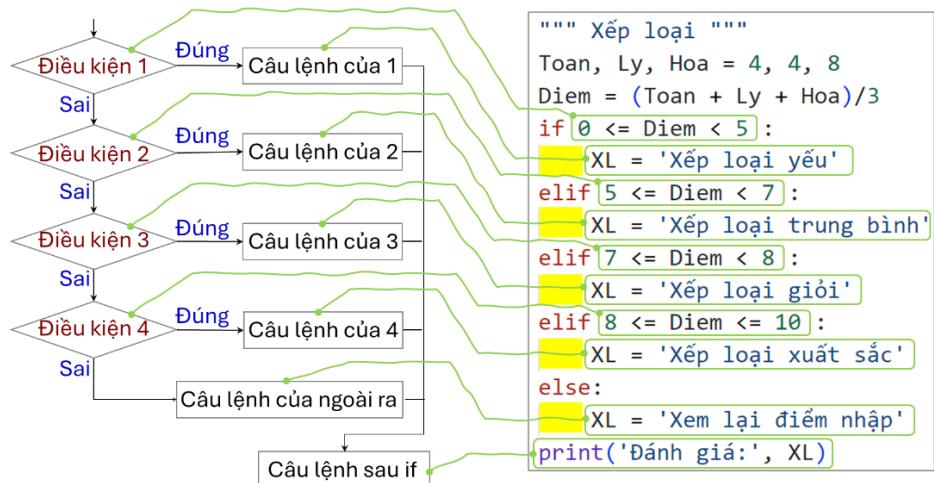
10.1.1. Điều kiện if một nhánh rẽ.



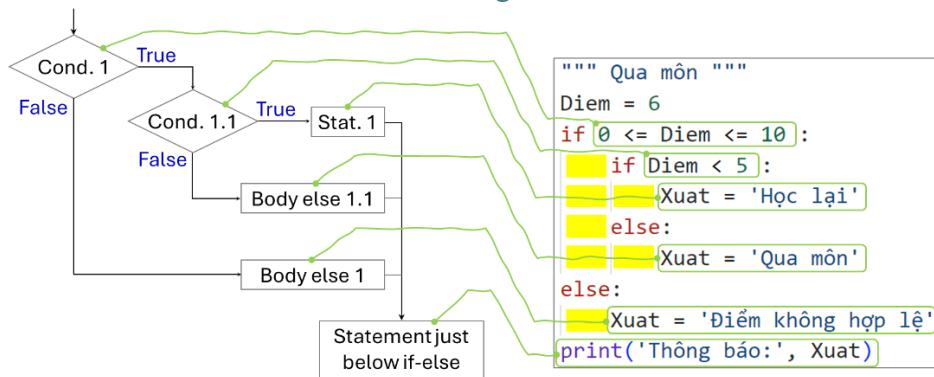
10.1.2. Điều kiện if hai nhánh rẽ.



10.1.3. Điều kiện if nhiều hơn hai nhánh rẽ.

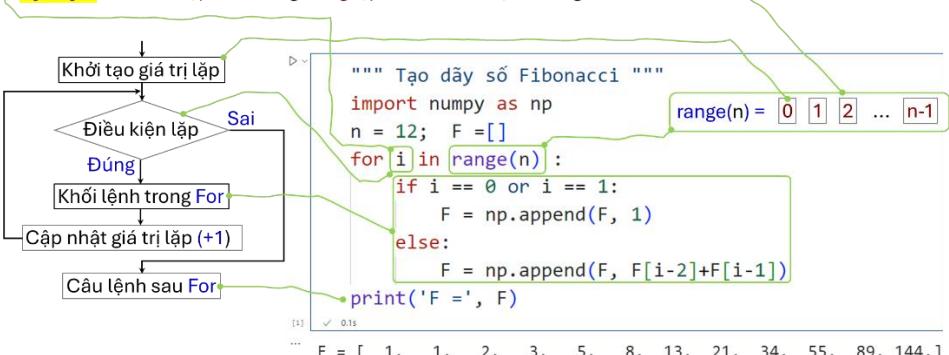


10.1.4. Điều kiện if nhiều nhánh rẽ lồng nhau.



10.2. Vòng lặp for.

`For Biến lặp in Vùng lặp:`
 [Tab] Nếu Biến lặp còn trong Vùng lặp thì làm khối lệnh trong For



10.3. Vòng lặp while.

Khởi tạo giá trị lặp

While Điều kiện lặp :

[Tab] Khối lệnh trong While

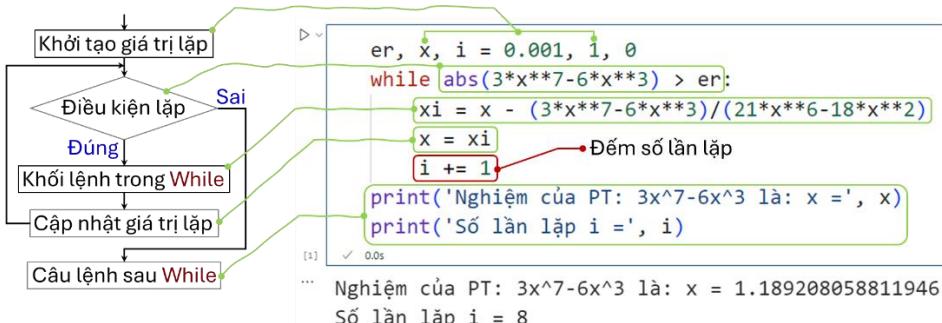
[Tab] Cập nhật giá trị lặp

Chưa biết số lần lặp, kiểm tra điều kiện trước khi làm.

Trong khi điều kiện còn đúng thì cứ làm.

- Nếu “Điều kiện còn đúng” → Thị làm.

- Nếu “Điều kiện không còn đúng” → Thị dừng.



11. Hướng dẫn sử dụng gói numpy cơ bản.

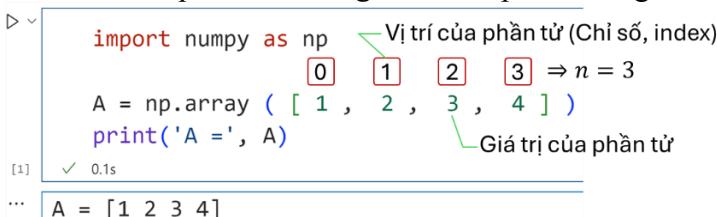
`import numpy as np`

11.1. Mảng trong numpy.

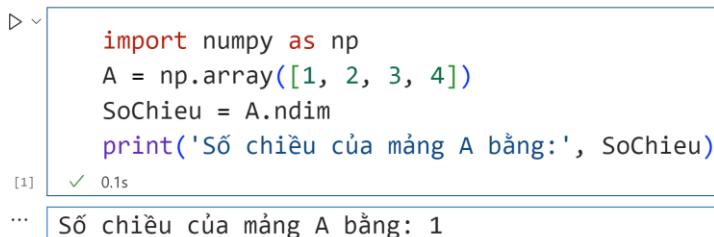
Hàm `array` dùng để tạo mảng trong numpy mà matrix hay vector thuộc tập con của nó (**mảng hai chiều**). Có ba loại mảng một chiều, hai chiều và ba chiều, các phép toán trên mảng muốn thực hiện được thì điều đầu tiên là phải cùng số chiều.

11.1.1. Tạo mảng một chiều (1D).

`A = np.array([a0, a1, a2, ..., an])` → Tạo mảng một chiều với a_i ($0 \leq i \leq n$) là các phần tử được gán trực tiếp cho mảng.



A. `ndim = 1` → Số chiều của mảng (Sử dụng một cặp ngoặc vuông []).



A. `size` = $n + 1 \rightarrow$ Số phần tử của mảng.

```
▷ 
    import numpy as np
    A = np.array([1, 2, 3, 4])
    SoPhanTu = A.size
    print('Số phần tử của mảng A bằng:', SoPhanTu)
[1] ✓ 0.1s
... Số phần tử của mảng A bằng: 4
```

A. `shape` = $(n + 1,) \rightarrow$ Dạng của mảng. (Giá trị của chiều thứ nhất bằng số phần tử của mảng và bằng $n + 1$, chiều thứ hai bỏ trống)

```
▷ 
    import numpy as np
    A = np.array([1, 2, 3, 4])
    Dang = A.shape
    print('Dạng của mảng A là:', Dang)
[1] ✓ 0.1s
... Dạng của mảng A là: (4,)
```

A. `dtype` → Kiểu dữ liệu của mảng. (Phụ thuộc dữ liệu khi tạo:
int32, int64 → kiểu số nguyên, float64 → kiểu số thực)

- Khai báo tất cả các phần tử của mảng đều là số nguyên.

```
▷ 
    import numpy as np
    A = np.array([1, 2, 3, 4])
    KieuDuLieu = A.dtype
    print('Kiểu dữ liệu của mảng A là:', KieuDuLieu)
[1] ✓ 0.1s
... Kiểu dữ liệu của mảng A là: int64
```

- Khi khai báo chỉ cần một phần tử không phải số nguyên.

```
▷ 
    import numpy as np
    A = np.array([1., 2, 3, 4]) Số thực
    KieuDuLieu = A.dtype
    print('Kiểu dữ liệu của mảng A là:', KieuDuLieu)
[1] ✓ 0.1s
... Kiểu dữ liệu của mảng A là: float64
```

- Khai báo kiểu dữ liệu trong khi tạo mảng.

```
▷ 
    import numpy as np
    A = np.array([1, 2, 3, 4], dtype='float64') Khai báo kiểu dữ liệu khi tạo mảng
    KieuDuLieu = A.dtype
    print('Kiểu dữ liệu của mảng A là:', KieuDuLieu)
[1] ✓ 0.1s
... Kiểu dữ liệu của mảng A là: float64
```

- Thay đổi kiểu dữ liệu của mảng đã có.

```

> 
import numpy as np
A = np.array([1, 2, 3, 4], dtype='float64')
B = A.astype('int32')Thay đổi kiểu dữ liệu
print('Kiểu dữ liệu của mảng A:', A.dtype)
print('Kiểu dữ liệu của mảng B:', B.dtype)
[1] ✓ 0.1s
...
Kiểu dữ liệu của mảng A: float64
Kiểu dữ liệu của mảng B: int32

```

11.1.2. Tạo mảng hai chiều (2D).

$A = \text{np.array}([[a_{00}, a_{01}, \dots, a_{0n}], [a_{10}, a_{11}, \dots, a_{1n}], [\dots, \dots, \dots, \dots], [a_{m0}, a_{m1}, \dots, a_{mn}]])$

→ Tạo mảng A hai chiều với a_{ij} ($0 \leq i \leq m; 0 \leq j \leq n$) là các phần tử được gán cho mảng.

```

> 
import numpy as npVị trí cột của mảng
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])Vị trí hàng của mảng
print('A =\n', A)
[1] ✓ 0.0s
...
A =
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

```

A. $\text{ndim} = 2 \rightarrow$ Số chiều của mảng (Sử dụng hai cặp ngoặc vuông [[]]).
A. $\text{size} = (m + 1) \times (n + 1) \rightarrow$ Số phần tử của mảng A .

A. $\text{shape} = (m + 1, n + 1) \rightarrow$ Dạng của mảng A (số phần tử trên chiều thứ nhất (hàng) $m+1$ và chiều thứ hai (cột) $n+1$).

A. $\text{dtype} \rightarrow$ Kiểu dữ liệu của mảng A (Phụ thuộc dữ liệu khi tạo: Số nguyên, số thực, số phức).

Các phép toán trên ma trận (matrix) và véc tơ (vector) chủ yếu được thực hiện trên mảng hai chiều.

```

> import numpy as np
A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
SoChieu = A.ndim;    print('Số chiều:', SoChieu)
SoPhanTu = A.size;   print('Số phần tử:', SoPhanTu)
Dang = A.shape;     print('Đạng:', Dang)
KieuDuLieu = A.dtype; print('Kiểu dữ liệu:', KieuDuLieu)
[1]: ✓ 0.1s
...
Số chiều: 2
Số phần tử: 12
Đạng: (3, 4)
Kiểu dữ liệu: int64

```

11.1.3. Tạo mảng ba chiều (3D).

Trong một số trường hợp đặc biệt thì phải cần đến mảng ba chiều.

$$A = \text{np.array}([[[a_{000}, a_{010}, \dots, a_{0n0}], \\ [a_{010}, a_{011}, \dots, a_{01n}], \\ [\dots, \dots, \dots, \dots], \\ [a_{0m0}, a_{0m1}, \dots, a_{0mn}]] \\ [[a_{100}, a_{101}, \dots, a_{10n}], \\ [a_{110}, a_{111}, \dots, a_{11n}], \\ [\dots, \dots, \dots, \dots], \\ [a_{1m0}, a_{1m1}, \dots, a_{1mn}]] \\ \dots, \dots, \dots, \dots \\ [[a_{k00}, a_{k01}, \dots, a_{k0n}], \\ [a_{k10}, a_{k11}, \dots, a_{k1n}], \\ [\dots, \dots, \dots, \dots], \\ [a_{km0}, a_{km1}, \dots, a_{kmn}]]])$$

→ Tạo mảng A hai chiều với a_{qij} ($0 \leq q \leq k; 0 \leq i \leq m; 0 \leq j \leq n$) là các phần tử được gán cho mảng.

```

> 
import numpy as np
A = np.array([[1, 2, 3, 4],  

             [5, 6, 7, 8],  

             [9, 10, 11, 12]],  

             [[13, 14, 15, 16],  

              [17, 18, 19, 20],  

              [21, 22, 23, 24]]])
print('Số chiều của A =', A.ndim)
print('Số phần tử của A =', A.size)
print('Dạng của mảng A:', A.shape)

```

[1] ✓ 0.1s

... Số chiều của A = 3
 Số phần tử của A = 24
 Dạng của mảng A: (2, 3, 4)

11.2. Các phép toán trên mảng.

11.2.1. Phép toán cơ bản từng cặp phần tử tương ứng giữa hai mảng.

Với các phép toán này thì dạng (shape) của hai mảng phải giống nhau.

- Phép toán trên mảng một chiều:

```

> 
import numpy as np
A = np.array([7, 5, 6])
B = np.array([1, 2, 3])
print('A+B=', A+B, '\nA-B=', A-B)
print('A*B=', A*B, '\nA/B=', A/B)

```

[1] ✓ 0.1s

... A+B= [8 7 9]
 A-B= [6 3 3]
 A*B= [7 10 18]
 A/B= [7. 2.5 2.]

- Phép toán trên mảng hai chiều:

```
▷ 
import numpy as np
C = np.array([[7, 5, 6], [5, 9, 8]])
D = np.array([[4, 2, 3], [4, 1, 2]])
print('C+D=\n', C+D, '\nC-D=\n', C-D)
print('C*D=\n', C*D, '\nC/D=\n', C/D)
[1] ✓ 0.1s
```

... C+D=

```
[[11  7  9]
 [ 9 10 10]]
```

C-D=

```
[[3  3  3]
 [1  8  6]]
```

C*D=

```
[[28 10 18]
 [20  9 16]]
```

C/D=

```
[[1.75 2.5 2. ]
 [1.25 9.   4. ]]
```

11.2.2. Phép toán nhân ma trận, véc tơ $A@B$ hoặc $np.dot(A,B)$.

Phép nhân này chỉ áp dụng trong mảng hai chiều. Với A nhân B là hai ma trận hoặc véc tơ thì phép nhân chỉ có thể thực hiện được khi:

- A và B phải cùng số chiều là 2 ($A.ndim = B.ndim = 2$).
- Chiều thứ hai (cột) của A phải bằng chiều thứ nhất (hàng) của B .

$A.ndim = (1, 3)$ và $B.ndim = (3, 1)$.

```
▷ 
import numpy as np
A = np.array([[4, 5, 6]])
B = np.array([[7], [8], [9]])
print('A =\n', A, '\nB =\n', B)
print('A@B=\n', A@B, '\nB@A=\n', B@A)
[1] ✓ 0.1s
```

... A =

```
[[4 5 6]] → [1, 3]
```

B =

```
[[7]
 [8]
 [9]] → [3, 1]
```

A@B=

```
[[122]] → [1, 1]
```

B@A=

```
[[28 35 42]
 [32 40 48]
 [36 45 54]] → [3, 3]
```

$C.ndim = (2, 3)$ và $D.ndim = (3, 2)$.

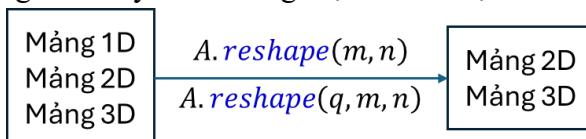
```
> v
    import numpy as np
    C = np.array([[4, 5, 6], [9, 7, 8]])
    D = np.array([[1, 2], [3, 4], [5, 6]])
    print('C =\n', C, '\nD =\n', D)
    print('C@D=\n', C@D)

[1]   ✓  0.1s
```

... C =
[[4 5 6] } → [2, 3]
[9 7 8]] }
D =
[[1 2] } → [3, 2]
[3 4]
[5 6]] }
C@D =
[[49 64] } → [2, 2]
[70 94]] }

11.3. Chuyển đổi chiều và dạng của mảng.

Có thể chuyển từ mảng có số chiều bất kỳ thành mảng hai chiều và mảng ba chiều, không thể chuyển về mảng một chiều được.



A.*reshape*(m, n): Số hàng m, số cột n, $m \times n =$ số phần tử của A.

A.*reshape*(q, m, n): Lớp, hàng, cột. $q \times m \times n =$ số phần tử của A.

```
> v
    import numpy as np
    A = np.array([1, 2, 3, 4, 5, 6, 7, 8]) #Tạo mảng một chiều
    B = A.reshape(1, 8) #Chuyển thành mảng hai chiều: 1 hàng, 8 cột
    C = A.reshape(2, 1, 4) #Chuyển thành mảng ba chiều: 2 lớp, 1 hàng, 4 cột
    D = C.reshape(2,4) #Chuyển thành mảng 2 chiều: 2 hàng, 4 cột
    print('A = ', A, '\nB = ', B, '\nC=\n', C, '\nD =\n', D)

[1]   ✓  0.1s
```

... A = [1 2 3 4 5 6 7 8]
B = [[1 2 3 4 5 6 7 8]]
C=
[[[1 2 3 4]]

[[5 6 7 8]]]
D =
[[[1 2 3 4]]

Nếu mảng A là 1D thì có thể sử dụng:

- $A[np.newaxis, :] \rightarrow$ Chuyển thành 2D có dạng $(n, 1)$
- $A[:, np.newaxis] \rightarrow$ Chuyển thành 2D có dạng $(1, n)$

11.4. Chuyển trí (chuyển vị) của mảng.

Chỉ áp dụng cho mảng hai chiều (2D). Chuyển trí của mảng A là phép biến đổi hàng thành cột và ngược lại. $B = A.T$

```
> <
    import numpy as np
    A = np.array([[1, 2, 3], [4, 5, 6]])
    B = A.T
    print('A =\n', A, '\nB =\n', B)
[1]   ✓ 0.1s
...
A =
[[1 2 3]
 [4 5 6]]
B =
[[1 4]
 [2 5]
 [3 6]]
```

11.5. Hàm tạo mảng đặc biệt.

11.5.1. Hàm arange.

Hàm `range` có trong python và hàm `arange` trong gói `numpy` có cách tạo giống nhau nhưng kết quả thì khác nhau:

Hàm `range` tạo ra một tập hợp số có cấu trúc có thể truy xuất đến phần tử bất kỳ nhưng không thể biểu diễn tường minh cả tập hợp và không có các phép toán cho tập hợp.

Hàm `arange` tạo ra mảng một chiều nên có thể truy xuất đến từng phần tử và thực hiện được các phép toán cho mảng một chiều.

$$A = np.arange(stop) \rightarrow [0, 1, 2, \dots, stop - 1]$$

```
> <
    import numpy as np
    A = np.arange(4)
    print('A =', A, '\nDim =', A.ndim, '\nShape =', A.shape)
[1]   ✓ 0.1s
...
A = [0 1 2 3]
Dim = 1
Shape = (4,)
```

$A = np.arange(start, stop) \rightarrow [start, start + 1, \dots, stop - 1]$

```
> ~
    import numpy as np
    A = np.arange(5, 11)
    print('A =', A, '\nDim =', A.ndim, '\nShape =', A.shape)
[1] ✓ 0.3s
...
A = [ 5  6  7  8  9 10]
Dim = 1
Shape = (6,)
```

$A = np.arange(start, stop, step)$

$\rightarrow [start, start + step, \dots, start + n \times step \leq stop - 1]$

```
> ~
    import numpy as np
    A = np.arange(4, 18, 3)
    print('A =', A, '\nDim =', A.ndim, '\nShape =', A.shape)
[1] ✓ 0.0s
...
A = [ 4  7 10 13 16]
Dim = 1
Shape = (5,)
```

11.5.2. Hàm linspace.

Hàm `linspace` cũng tạo ra mảng một chiều như hàm `arrange` tuy nhiên, điểm khác biệt là:

Hàm `arrange` có giá trị đầu tiên (`start`) và bước (`step`) cho trước, nhưng giá trị cuối cùng của mảng có nhiều khi nhỏ hơn (`stop - 1`)

Hàm `linspace` thì có điểm đầu tiên (`start`), điểm cuối cùng (`stop`) và số điểm cho trước nhưng bước (`step`) tự động tính.

Nhiều trường hợp trong kỹ thuật sử dụng hàm `linspace` thuận tiện hơn.

$x = np.linspace(start, stop, num)$

$$\rightarrow \left[start, start + \frac{stop - start}{num}, \dots, stop \right]$$

```
> ~
    import numpy as np
    x = np.linspace(5, 14, 5)
    print('x =', x, '; Size =', x.size, '; Shape =', x.shape)
[1] ✓ 0.1s
...
x = [ 5.      7.25  9.5   11.75 14. ] ; Size = 5 ; Shape = (5,)
```

$x = np.linspace(start, stop)$. Mặc định num = 50
 $\rightarrow \left[start, start + \frac{stop - start}{50}, \dots, stop \right]$

```
▷ 
    import numpy as np
    x = np.linspace(0, 5.6)
    print('Dim = ', x.ndim, '; Size = ', x.size, '; Shape = ', x.shape)
[1]    ✓ 0.0s
...
... Dim = 1 ; Size = 50 ; Shape = (50,)
```

11.5.3. Hàm zeros.

Hàm `zeros` dùng để tạo mảng có các phần tử bằng 0.

$x = np.zeros(num) \rightarrow$ Tạo mảng 1D có num phần tử.

```
▷ 
    import numpy as np
    x = np.zeros(4)
    print('x = ', x, '; Size = ', x.size, '; Shape = ', x.shape)
[1]    ✓ 0.0s
...
... x = [0. 0. 0. 0.] ; Size = 4 ; Shape = (4,)
```

$x = np.zeros((m, n)) \rightarrow$ Tạo mảng 2D có m hàng, n cột.

```
▷ 
    import numpy as np
    x = np.zeros((3, 4))
    print('x = ', x, )
    print('Dim = ', x.ndim, '; Size = ', x.size, '; Shape = ', x.shape)
[1]    ✓ 0.0s
...
... x = [[0. 0. 0. 0.]
        [0. 0. 0. 0.]
        [0. 0. 0. 0.]]
Dim = 2 ; Size = 12 ; Shape = (3, 4)
```

11.5.4. Hàm ones.

Hàm `ones` dùng tạo mảng có các phần tử bằng 1, cấu trúc như hàm `zeros`.

```
▷ 
    import numpy as np
    x = np.ones(4); print('x = ', x)
    y = np.ones((3, 5)); print('y =\n', y)
[1]    ✓ 0.0s
...
... x = [1. 1. 1. 1.]
y =
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

11.5.5. Hàm eye.

Hàm `eye` dùng để tạo mảng đơn vị, vuông trong 2D.

$I = np.eye(num) \rightarrow$ Tạo mảng 2D có $m = n = num$.

```
> ~
    import numpy as np
    I = np.eye(3); print('I =\n', I)
[1] ✓ 0.1s
...
I =
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

11.5.6. Hàm tri.

Hàm `tri` dùng để tạo mảng tam giác dưới, vuông trong 2D.

$I = np.tri(num) \rightarrow$ Tạo mảng 2D có $m = n = num$.

```
> ~
    import numpy as np
    Tri = np.tri(3); print('Tri =\n', Tri)
[1] ✓ 0.1s
...
Tri =
[[1. 0. 0.]
 [1. 1. 0.]
 [1. 1. 1.]]
```

11.5.7. Hàm tril.

Hàm `tril` dùng để chuyển mảng A hai chiều có sẵn thành mảng mới B với các phần tử phía trên đường chéo chính bằng 0.

$B = np.tril(A)$.

```
> ~
    import numpy as np
    A = np.arange(1, 10).reshape(3,3); print('A =\n', A)
    B = np.tril(A); print('B =\n', B)
[1] ✓ 0.0s
...
A =
[[1 2 3]
 [4 5 6]
 [7 8 9]]
B =
[[1 0 0]
 [4 5 0]
 [7 8 9]]
```

11.5.8. Hàm triu.

Hàm `triu` dùng để chuyển mảng A hai chiều có sẵn thành mảng mới B với các phần tử phía dưới đường chéo chính bằng 0.

$$B = np.triu(A).$$

```
▷ 
    import numpy as np
    A = np.arange(3, 12).reshape(3,3); print('A =\n', A)
    B = np.triu(A); print('B =\n', B)
[1] ✓ 0.0s
...
A =
[[ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
B =
[[ 3  4  5]
 [ 0  7  8]
 [ 0  0 11]]
```

11.6. Tạo mảng bằng cách ghép hai mảng đã có.

Hai mảng A, B cần ghép phải là mảng hai chiều (2D). Muốn ghép theo hàng thì số cột phải bằng nhau, ghép theo cột thì số hàng phải bằng nhau.

$$C = np.r_[A, B] \rightarrow \text{Ghép theo hàng, thì số cột phải bằng nhau.}$$

```
▷ 
    import numpy as np
    A = np.arange(1, 5).reshape(1, 4); print('A =\n', A)
    B = np.arange(5, 13).reshape(2, 4); print('B =\n', B)
    ArB = np.r_[A, B]; print('ArB =\n', ArB)
[1] ✓ 0.1s
...
A =
[[1 2 3 4]]
B =
[[ 5  6  7  8]
 [ 9 10 11 12]]
ArB =
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

$C = np.c_[A, B]$ \rightarrow Ghép theo cột, thì số hàng phải bằng nhau.

```
> import numpy as np
  A = np.arange(1, 5).reshape(2, 2); print('A =\n', A)
  B = np.arange(5, 15).reshape(2, 5); print('B =\n', B)
  AcB = np.c_[A, B]; print('AcB =\n', AcB)
[1] ✓ 0.1s
...
A =
[[1 2]
 [3 4]]
B =
[[ 5  6  7  8  9]
 [10 11 12 13 14]]
AcB =
[[ 1  2  5  6  7  8  9]
 [ 3  4 10 11 12 13 14]]
```

11.7. Mở rộng mảng, thêm phần tử vào mảng đã có.

Trong trường hợp chưa biết trước kích thước của mảng cần tính (1D), sử dụng hàm `append` để mở rộng và thêm một phần tử vừa tính được vào mảng đã có.

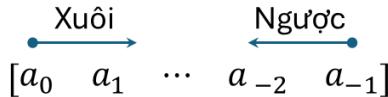
$A = np.append(A, x)$ \rightarrow thêm giá trị x vào mảng A đã có. $A = []$ là mảng rỗng được tạo.

```
> import numpy as np
  Mx = []
  for i in range(7):
    Tinh = 2*(0.25*i)**3
    Mx = np.append(Mx, Tinh)
    print(f'Mx({i}) = ', Mx)
[1] ✓ 0.1s
...
Mx(0) = [0.]
Mx(1) = [0.      0.03125]
Mx(2) = [0.      0.03125  0.25     ]
Mx(3) = [0.      0.03125  0.25     0.84375]
Mx(4) = [0.      0.03125  0.25     0.84375  2.      ]
Mx(5) = [0.      0.03125  0.25     0.84375  2.      3.90625]
Mx(6) = [0.      0.03125  0.25     0.84375  2.      3.90625  6.75     ]
```

11.8. Cách tham chiếu đến phần tử của mảng.

Để tham chiếu đến một phần tử hay một vài vùng của mảng thì phải sử dụng dấu ngoặc vuông [].

Đối với mảng 1D chỉ truyền vào một tham số $A[n]$, theo hai chiều khác nhau. Chiều xuôi bắt đầu từ 0, chiều ngược bắt đầu từ -1.



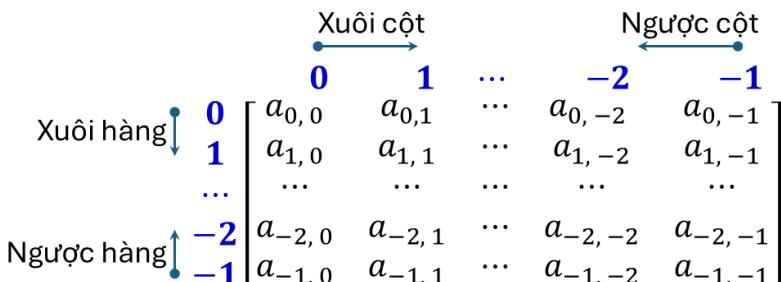
▷ v

```
import numpy as np
A = np.arange(1, 12); print('A =', A)
print('A[0] =', A[0], '; A[-3] =', A[-3])
[1] ✓ 0.1s
```

...

```
A = [ 1  2  3  4  5  6  7  8  9 10 11]
A[0] = 1 ; A[-3] = 9
```

Đối với mảng 2D thì có chiều xuôi và ngược của hàng và cột.



$A[m]$ hoặc $A[m, :]$ → Tham chiếu đến hàng m (xuôi, ngược) mọi cột.

▷ v

```
import numpy as np
A = np.arange(1, 21).reshape(4, 5)
print('A =\n', A, '\nA[-2] =', A[-2])
[1] ✓ 0.0s
```

...

```
A =
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
```

```
A[-2] = [11 12 13 14 15]
```

$A[:, n]$ → Tham chiếu đến cột n (xuôi, ngược) mọi hàng.

```
▷ 
import numpy as np
A = np.arange(1, 21).reshape(4, 5)
print('A =\n', A, '\nA[:,1] =', A[:,1])
[1] ✓ 0.1s
```

...

```
A =
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
A[:,1] = [ 2  7 12 17]
```

$A[m, n]$ → Tham chiếu đến một phần tử tại hàng m cột n (xuôi, ngược).

```
▷ 
import numpy as np
A = np.arange(1, 16).reshape(3, 5)
print('A =\n', A, '\nA[1,-2] =', A[1,-2])
[2] ✓ 0.0s
```

...

```
A =      -2 -1
0 [[ 1  2  3  4  5]
 1 [ 6  7  8  9 10]
 [11 12 13 14 15]]
A[1,-2] = 9
```

$A[np.s_[start :: step]]$ hoặc $A[np.s_[start :: step, :]]$ → Tham chiếu đến tất cả các hàng bắt đầu từ hàng $start$ cách nhau $step$ hàng.

```
▷ 
import numpy as np
A = np.arange(1, 25).reshape(4, 6)
B = A[np.s_[0::2]]
print('A =\n', A, '\nB =\n', B)
[1] ✓ 0.0s
```

...

```
A =
0 [[ 1  2  3  4  5  6]
 +2 { [ 7  8  9 10 11 12]
    [13 14 15 16 17 18]
    [19 20 21 22 23 24]]
B =
 [ [ 1  2  3  4  5  6]
   [13 14 15 16 17 18]]]
```

`A[np.s_[:, start :: step]]` → Tham chiếu đến tất cả các cột bắt đầu từ cột `start` cách nhau `step` cột.

```
▷ 
import numpy as np
A = np.arange(1, 22).reshape(3, 7)
B = A[np.s_[:,1::2]]
print('A =\n', A, '\nB =\n', B)
[1] ✓ 0.1s
```

... A = 1 +2 +2
[[1 2 3 4 5 6 7]
[8 9 10 11 12 13 14]
[15 16 17 18 19 20 21]]
B =
[[2 4 6]
[9 11 13]
[16 18 20]]

`np.delete(A, m, axis)` → Xóa hàng (`axis = 0`) thứ `m` (xuôi, ngược).

```
▷ 
import numpy as np
A = np.arange(1, 19).reshape(3, 6)
B = np.delete(A, 1, 0)
print('A =\n', A, '\nB =\n', B)
[1] ✓ 0.2s
```

... A =
0 [[1 2 3 4 5 6]
1 [7 8 9 10 11 12]
2 [13 14 15 16 17 18]]
B =
[[1 2 3 4 5 6]
[13 14 15 16 17 18]]

`np.delete(A, n, axis)` → Xóa cột (`axis = 1`) thứ `n` (xuôi, ngược).

```
▷ 
import numpy as np
A = np.arange(1, 19).reshape(3, 6)
B = np.delete(A, -2, 1)
print('A =\n', A, '\nB =\n', B)
[1] ✓ 0.1s
```

... A = -2 -1
[[1 2 3 4 5 6]
[7 8 9 10 11 12]
[13 14 15 16 17 18]]
B =
[[1 2 3 4 6]
[7 8 9 10 12]
[13 14 15 16 18]]

`np.setdiff1d(A, n)` → Xóa tất cả các phần tử có giá trị bằng n trong mảng A ; Chỉ giữ lại một phần tử giống nhau có trong A ; Sắp xếp lại mảng A theo thứ tự giá trị tăng dần.

▷ ▾

```
import numpy as np
A = np.array([1, 3, -5, 8, 8, -1, 4, -3, 2, -1, 8])
B = np.setdiff1d(A, -1)
print('A =', A, '\nB =', B)
```

[1] ✓ 0.1s

...

```
A = [ 1  3 -5  8  8 -1  4 -3  2 -1  8]
B = [-5 -3  1  2  3  4  8]
```

`np.ix_([...ri...], [...cj...])` → Lập chỉ mục giao giữa hàng r_i và cột c_j để tham chiếu đến mảng $A[np.ix_([...r_i...], [...c_j...])]$.

▷ ▾

```
import numpy as np
A = np.arange(15).reshape(3, 5); print('A =\n', A)
ix = np.ix_([0, 2], [1, 2, 4]); print('ix =\n', ix)
Aix = A[ix]; print('A[ix] =\n', Aix)
```

[1] ✓ 0.1s

...

```
A =
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
ix =
(array([[0],
       [2]]), array([[1, 2, 4]]))
A[ix] =
[[ 1  2  4]
 [11 12 14]]
```

Chỉ mục của ix:

$$\begin{bmatrix} (0,1) & (0,2) & (0,4) \\ (2,1) & (2,2) & (2,4) \end{bmatrix}$$

11.9. Đại số tuyến tính (Linear algebra) trong numpy.

Gói `linalg` nằm trong gói numpy do đó cách nạp gói sẽ là:

`import numpy.linalg as la`.

Gói `linalg` chuyên xử lý về matrix và vector, ứng dụng trong việc nghiên cứu hệ phương trình đại số tuyến tính.

Các phép toán sử dụng các hàm trong gói `linalg` chỉ thực hiện trên mảng hai chiều (2D).

11.9.1. Tìm giá trị riêng và vector riêng.

Hàm `eig` dùng để tính các giá trị riêng và vector riêng của một mảng vuông 2D (matrix).

```
> ~
    import numpy as np
    import numpy.linalg as la
    A = np.arange(9).reshape(3, 3); print('A =\n', A)
    Val, Vec = la.eig(A)
    print('Values =', Val, '\nVector =\n', Vec)
[1] ✓ 0.0s
...
A =
[[0 1 2]
 [3 4 5]
 [6 7 8]]
Values = [ 1.33484692e+01 -1.34846923e+00 -2.48477279e-16]
Vector =
[[ 0.16476382  0.79969966  0.40824829]
 [ 0.50577448  0.10420579 -0.81649658]
 [ 0.84678513 -0.59128809  0.40824829]]
```

11.9.2. Tính chuẩn của một matrix hoặc vector.

Hàm `norm` dùng để tính chuẩn của một matrix hoặc vector.

`la.norm(X, 1)` → Tính chuẩn cột.

`la.norm(X)` hoặc `la.norm(X, 2)` → Tính chuẩn Euclide.

`la.norm(X, 1)` → Tính chuẩn hàng.

```
> ~
    import numpy as np
    import numpy.linalg as la
    X = np.arange(9); print('X =', X)
    Xnorm1 = la.norm(X, 1); print('||X||_1 =', Xnorm1)
    Xnorm2 = la.norm(X); print('||X||_2 =', Xnorm2)
    Xnorm3 = la.norm(X, np.inf); print('||X||_3 =', Xnorm3)
[1] ✓ 0.1s
...
X = [0 1 2 3 4 5 6 7 8]
||X||_1 = 36.0
||X||_2 = 14.2828568570857
||X||_3 = 8.0
```

11.9.3. Tính định thức của matrix.

Hàm `det` dùng để tính định thức của một mảng vuông hai chiều (matrix).

```
▷ 
    import numpy as np
    import numpy.linalg as la
    A = np.array([[-5, 1, 4], [3, -2, 0], [6, 7, -8]])
    print('A =\n', A)
    det_A = la.det(A); print('det(A) =', det_A)
[1] ✓ 0.1s
...
A =
[[ -5  1  4]
 [ 3 -2  0]
 [ 6  7 -8]]
det(A) = 76.0
```

11.9.4. Tìm nghịch đảo của matrix.

Hàm `inv` dùng để tính nghịch đảo của một mảng vuông hai chiều.

```
▷ 
    import numpy as np
    import numpy.linalg as la
    A = np.array([[-5, 1, 4], [3, -2, 0], [6, 7, -8]])
    print('A =\n', A)
    inv_A = la.inv(A); print('inv(A) =\n', inv_A)
[1] ✓ 0.1s
...
A =
[[ -5  1  4]
 [ 3 -2  0]
 [ 6  7 -8]]
inv(A) =
[[ 0.21052632  0.47368421  0.10526316]
 [ 0.31578947  0.21052632  0.15789474]
 [ 0.43421053  0.53947368  0.09210526]]
```

11.9.5. Giải hệ phương trình đại số tuyến tính.

Hệ phương trình đại số tuyến tính:
$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

Viết dưới dạng matrix:
$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Hay: $Ax = b$, cần tìm nghiệm x của hệ. Nếu $\det(A) \neq 0$ thì hệ có nghiệm duy nhất.

$x = la.inv(A) @ b \rightarrow$ Tìm nghiệm thông qua hàm `inv` và `@`.

```
> import numpy as np
> import numpy.linalg as la
> A = np.array([[-5, 1, 4], [3, -2, 0], [6, 7, -8]])
> b = np.array([3, 2, -1])
> print('A =\n', A, '\nb =', b)
> x = la.inv(A) @ b; print('x =', x)
[1] ✓ 0.0s
```

```
... A =
[[ -5  1  4]
 [ 3 -2  0]
 [ 6  7 -8]]
b = [ 3  2 -1]
x = [1.47368421 1.21052632 2.28947368]
```

$x = la.solve(A, b) \rightarrow$ Tìm nghiệm thông qua hàm `solve`.

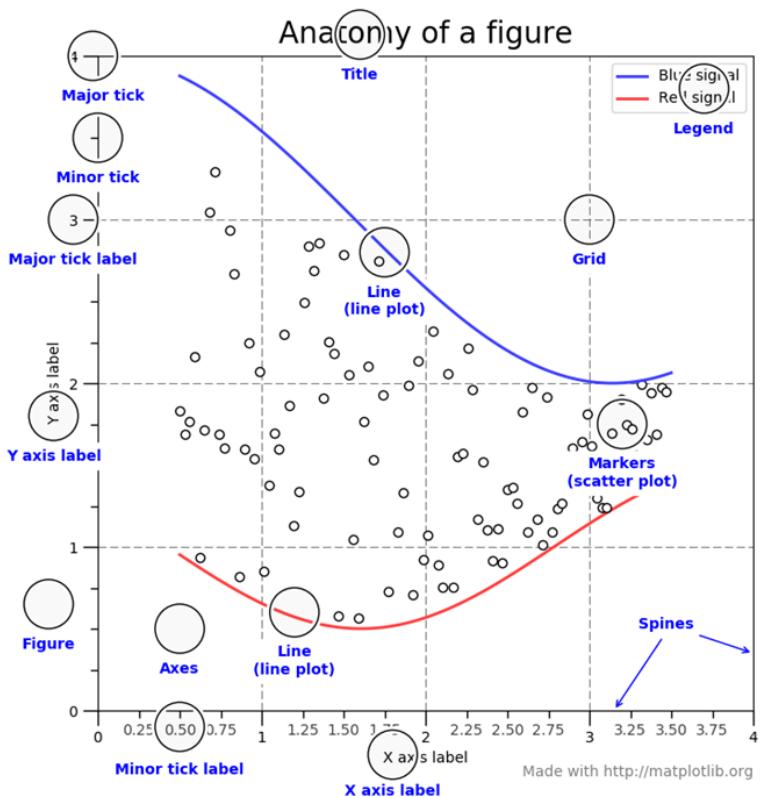
```
> import numpy as np
> import numpy.linalg as la
> A = np.array([[-5, 1, 4], [3, -2, 0], [6, 7, -8]])
> b = np.array([3, 2, -1])
> print('A =\n', A, '\nb =', b)
> x = la.solve(A, b); print('x =', x)
[1] ✓ 0.1s
```

```
... A =
[[ -5  1  4]
 [ 3 -2  0]
 [ 6  7 -8]]
b = [ 3  2 -1]
x = [1.47368421 1.21052632 2.28947368]
```

12. Hướng dẫn sử dụng gói matplotlib cơ bản.

Gói numpy thể hiện sức mạnh tính toán trên mảng, để trực quan số liệu kết quả thì phải cần đến gói `matplotlib.pyplot`. Nó có thể vẽ nhiều kiểu biểu đồ, điểm đường mặt 2D, 3D và có thể mô phỏng chuyển động... Trên một hình vẽ đồ họa (Figure) có rất nhiều thành phần khác nhau. Ở mức độ cơ bản, sử dụng các cài đặt mặc định cũng đủ đáp ứng tốt cho công việc.

`import matplotlib.pyplot as plt` → Nạp gói `pyplot` vào chương trình.



Dạng phân cấp quản lý của một vùng đồ họa (figure).



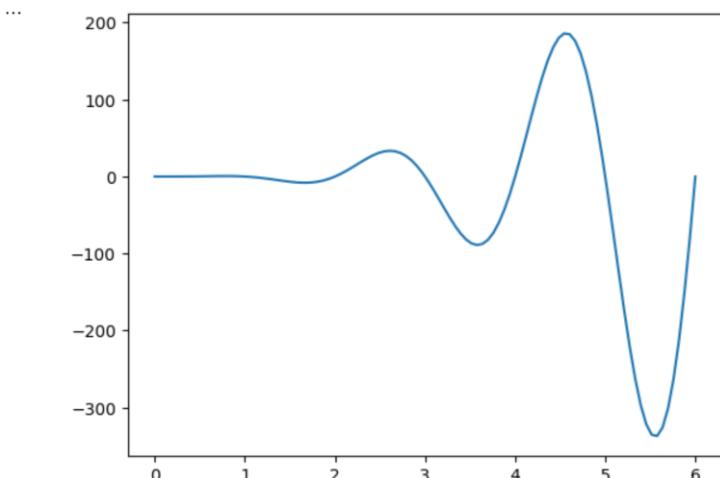
12.1. Hàm plot.

Hàm `plot` dùng để vẽ điểm, đường trong mặt phẳng 2D.

`plt.plot(x, y) → Vẽ một đường với hai mảng x, y có cùng dạng (shape).`

```
▷ import numpy as np
    import matplotlib.pyplot as plt
    x = np.linspace(0, 6, 100) # Tạo biến x: 0 <= x <= 6
    y = 2*x**3*np.sin(np.pi*x) # Hàm cần vẽ
    plt.plot(x, y); plt.show()
```

[1] ✓ 0.5s



Để thay đổi kiểu đường mặc định thì cần phải khai báo thêm các tham số cho hàm `plot`: dạng đường, màu, đánh dấu, bè rộng, nhẵn.

Khai báo dạng đường: `linestyle = '—'` hoặc `ls = '-.'`

Khai báo	Hiển thị
<code>ls = '—'</code>	
<code>ls = '-.'</code>	
<code>ls = '--'</code>	
<code>ls = ':'</code>	

Khai báo màu của đường: `color = 'red'` hoặc `c = 'b'`

Khai báo	Hiển thị
<code>c = 'k'</code>	
<code>c = 'b'</code>	
<code>c = 'g'</code>	
<code>c = 'y'</code>	
<code>c = 'r'</code>	
<code>c = 'm'</code>	

Khai báo đánh dấu các điểm: `marker = 's'`

Khai báo	Hiển thị
<code>marker = '+'</code>	
<code>marker = '.'</code>	
<code>marker = 'v'</code>	
<code>marker = '^'</code>	
<code>marker = 's'</code>	
<code>marker = 'o'</code>	
<code>marker = '1'</code>	

Để rút gọn việc khai báo ba tham số trên, không cần thêm tên của tham biến, khai báo cả ba tham biến cùng lúc thì cần phải để các giá trị khai báo trong cặp dấu nháy đơn '' không cần phải theo thứ tự: '`- k4`'

Khai báo	Hiển thị
<code>' - bs'</code>	
<code>' -.g'</code>	
<code>' : v'</code>	
<code>' k^'</code>	
<code>' - -'</code>	
<code>' r'</code>	
<code>' o'</code>	

Khai báo bì rộng nét vẽ: `linewidth = 2` hoặc `lw = 1.5`

Khai báo nhãn cho đường: `label = 'Mô tả đường'`

Nếu nhãn có chứa công thức, ký tự latin thì thêm `r` và cặp \$:
`label = r'$Ghi công thức theo kiểu markdown$'`

Một số hàm trong gói `pyplot` để hoàn thiện kết quả.

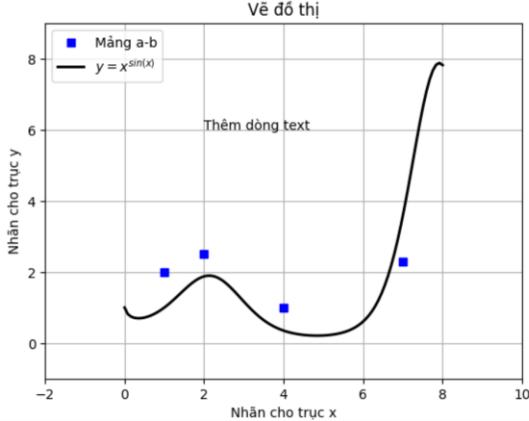
Khai báo	Mô tả
<code>plt.title('Tiêu đề')</code>	Tạo tiêu đề cho hình vẽ
<code>plt.xlabel('Nhãn cho trục x')</code>	Tạo nhãn cho trục x
<code>plt.ylabel('Nhãn cho trục y')</code>	Tạo nhãn cho trục y
<code>plt.text(x, y, 'Thêm dòng ký tự')</code>	Tạo chuỗi ký tự tại (x, y)
<code>plt.axis([x₁, x₂, y₁, y₂])</code>	Vùng hiển thị theo trục x và y
<code>plt.legend()</code>	Hiện nhãn của đồ thị, vị trí hiện
<code>plt.grid()</code>	Hiện lưới trên đồ thị
<code>plt.show()</code>	Hiện kết quả vẽ

```

import numpy as np
import matplotlib.pyplot as plt
a = np.array([1, 2, 4, 7]); b = np.array([2, 2.5, 1, 2.3])
x = np.linspace(0, 8, 100); y = x**np.sin(x)
plt.plot(a,b,'bs',label='Mảng a-b'); plt.plot(x,y,'k',lw=2,label=r'$y=x^{\sin(x)}$')
plt.title('Vẽ đồ thị');
plt.xlabel('Nhãn cho trục x')
plt.ylabel('Nhãn cho trục y')
plt.grid();
plt.legend();
plt.text(2, 6, 'Thêm dòng text')
plt.axis([-2, 10, -1, 9])
plt.show()

```

[1] ✓ 0.6s



12.2. Hàm figure.

Một figure tương ứng với một hình ảnh có thể xuất ra một file ảnh.

Mặc định khi chỉ vẽ trên một figure thì không cần sử dụng hàm figure.

Khi muốn thay đổi kích thước hay sử dụng nhiều hơn một figure thì phải dùng hàm figure.

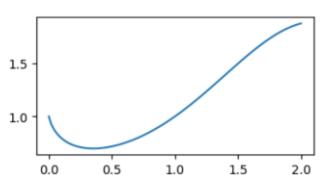
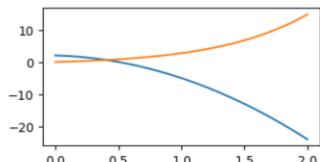
`plt.figure(n, figsize = (width, height))` → Tạo một figure thứ n (figure đầu tiên $n = 1$), width và height là chiều rộng và chiều cao của figure. Figure nào được gọi sau cùng thì nằm ở dưới cùng.

```

import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2, 100)
y1 = -6*x**2 - x + 2
y2 = x**np.sin(x)
y3 = x*np.exp(x)
plt.figure(1, figsize=(4,2)); plt.plot(x, y1)
plt.figure(2, figsize=(4,2)); plt.plot(x,y2)
plt.figure(1);
plt.figure(2);
plt.show()

```

[1] ✓ 0.5s



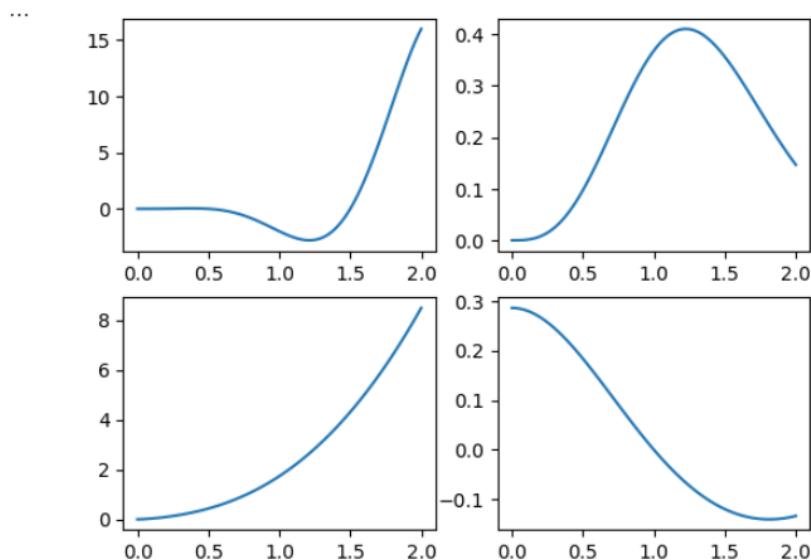
12.3. Hàm subplot.

Hàm subplot dùng để chia một figure thành nhiều vùng con để vẽ.

`plt.subplot(r, c, i)` hoặc `plt.subplot(r, c, i) →` Tạo ra $r \times c$ vùng con trên một figure, vùng con hiện hành là $i: 1 \leq i \leq r \times c$.

Với r (số hàng của subplot), c (số cột của subplot). r, c, i phải là số nguyên dương.

```
> 
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2, 100)
y1 = 2*x**3*np.cos(np.pi * x)
y2 = x**3*np.exp(-x**2)
y3 = x*np.sqrt(2*x**3+x)
y4 = (x**3-3*x**2+2)/(2*x**2+7)
plt.subplot(221); plt.plot(x, y1)
plt.subplot(222); plt.plot(x, y2)
plt.subplot(223); plt.plot(x, y3)
plt.subplot(224); plt.plot(x, y4); plt.show()
[1] ✓ 0.7s
```



13. Hướng dẫn sử dụng gói sympy cơ bản.

`import sympy as sp` → nạp gói sympy chứa thư viện hàm dùng để tính toán về số học, đại số, giải tích, vật lý, cơ học và nhiều ứng dụng khác.

13.1. Hàm tạo biến trong sympy.

Hàm `Symbol` dùng để tạo một biến có tên tự đặt.

Hàm `symbols` dùng để tạo một hoặc nhiều biến có tên tự đặt.

Gói `abc` chứa nhiều biến có sẵn.

Cú pháp	Giải thích
<code>import sympy as sp</code> <code>x = sp.Symbol('x')</code>	Tạo một biến <code>x</code> (Có thể đặt tên khác)
<code>import sympy as sp</code> <code>x, y = sp.Symbol('x, y')</code>	Lỗi. Hàm <code>Symbol</code> chỉ tạo được một biến
<code>import sympy as sp</code> <code>x = sp.symbols('x')</code>	Tạo một biến <code>x</code> (Có thể đặt tên khác)
<code>import sympy as sp</code> <code>x, y, z = sp.symbols('x y z')</code>	Tạo ba biến <code>x, y, z</code> (Có thể đặt tên khác)
<code>import sympy as sp</code> <code>x, y, z = sp.symbols('x, y, z')</code>	Tạo ba biến <code>x, y, z</code> (Có thể đặt tên khác)
<code>from sympy.abc import x, y, z</code>	Tạo ba biến <code>x, y, z</code>

13.2. Hàm subs.

Hàm subs dùng để thay thế các biến bằng giá trị trong biểu thức.

$f_{ThayThế}$

= `TenHam.subs([(biến1, thay1), (biến2, thay2), (biến3, thay3), ...])`

```
▷ ▾
    import sympy as sp
    from sympy.abc import x, y, z
    f = sp.sqrt(x**3*y**5*z**6) + sp.sin(x)**2*sp.cos(z)
    fs = f.subs([(x, 2), (y, 3), (z, 4)])
    print('f(x,y,z) =', f)
    print('f.subs(x=2,y=3,z=4) =', fs)

[1] ✓ 0.5s
...
    f(x,y,z) = sqrt(x**3*y**5*z**6) + sin(x)**2*cos(z)
    f.subs(x=2,y=3,z=4) = sin(2)**2*cos(4) + 1152*sqrt(6)
```

13.3. Hàm evalf.

Hàm evalf dùng để tính toán một biểu thức.

$f_{Tính} = TenHam.evalf()$

```

> import sympy as sp
  from sympy.abc import x, y, z
  f = sp.sqrt(x**3*y**5*z**6) + sp.sin(x)**2*sp.cos(z)
  fs = f.subs([(x, 2), (y, 3), (z, 4)])
  fe = fs.evalf()
  print('f(x,y,z) =', f)
  print('f.subs(x=2,y=3,z=4) =', fs)
  print('f.evalf() =', fe)
[1] ✓ 0.5s
...
f(x,y,z) = sqrt(x**3*y**5*z**6) + sin(x)**2*cos(z)
f.subs(x=2,y=3,z=4) = sin(2)**2*cos(4) + 1152*sqrt(6)
f.evalf() = 2821.27173688424

```

13.4. Hàm diff.

Hàm diff dùng để tính đạo hàm các cấp cho một hàm số.

*f*_ĐạoHàm

= *TenHam.diff((biến1, cấp_i), (biến2, cấp_j), (biến3, cấp_k), ...)*

```

> import sympy as sp
  from sympy.abc import x, y, z
  f = sp.sqrt(x)*sp.sin(y)**2*sp.cos(z)
  df_x2y3z4 = f.diff((x, 2), (y, 3), (z, 4))
  print('f =', f)
  print('df/(dx2 dy3 dz4) =', df_x2y3z4)
[1] ✓ 0.5s
...
f = sqrt(x)*sin(y)**2*cos(z)
df/(dx2 dy3 dz4) = 2*sin(y)*cos(y)*cos(z)/x**3/2

```

13.5. Hàm integrate.

Hàm integrate dùng để tính tích phân của hàm số.

Int = *TenHam.integrate((biến1, cd1, ctr1), (biến2, cd2, ctr2), ...)*

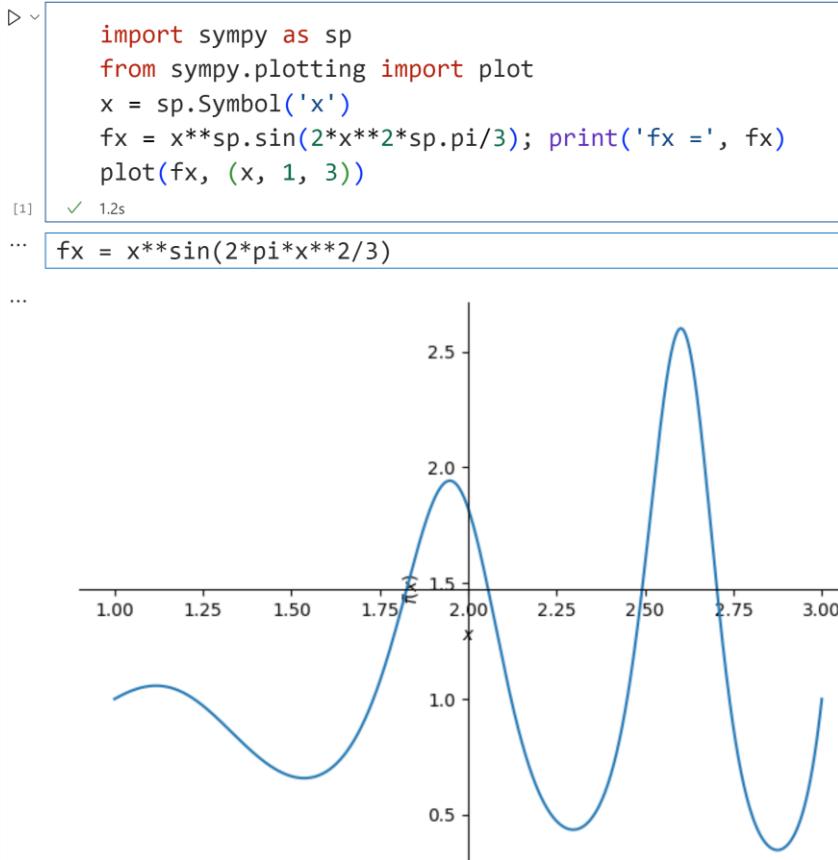
```

> import sympy as sp
  from sympy.abc import x, y
  f = sp.sqrt(x)*sp.sin(y); print('f =', f)
  Inte = f.integrate((x, 0, 2), (y, -1, 2)); print('Int =', Inte)
  In_evalf = Inte.evalf(); print('Int =', In_evalf)
[1] ✓ 0.5s
...
f = sqrt(x)*sin(y)
Int = -4*sqrt(2)*cos(2)/3 + 4*sqrt(2)*cos(1)/3
Int = 1.80349779856508

```

13.6. Hàm plot.

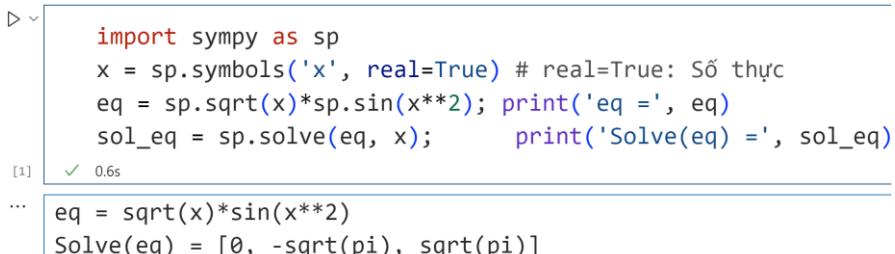
Gói con plotting của gói sympy có hàm plot dùng để vẽ đồ thị hàm số. Điểm khác biệt so với hàm plot trong gói matplotlib.pyplot là không cần phải tạo mảng cho biến.



13.7. Hàm solve.

Hàm `solve` dùng để giải phương trình hoặc hệ phương trình đại số tuyến tính (bằng chữ hoặc bằng số).

`sol_eq = sp.solve(TenHam, biến)` → Giải phương trình.



$sol_eqs = sp.solve([eq1, eq2, \dots, eqn], biến1, biến2, \dots, biếnn) \rightarrow$
Giải hệ phương trình.

```
> import sympy as sp
  x, y, z = sp.symbols('x y z', real=True)
  eq1 = 3*x + y - 5*z - 6
  eq2 = x - 4*y + 2*z + 4
  eq3 = -x - 8*y + 3*z - 7
  eqs = [eq1, eq2, eq3]
  sol_eqs = sp.solve(eqs, x, y, z)
  print('Nghiệm của hệ: ', sol_eqs)
  print('x =', sol_eqs[x].evalf())
[1] ✓ 0.5s
...
Nghiệm của hệ: {x: -250/67, y: -123/67, z: -255/67}
x = -3.73134328358209
```

13.8. Hàm linsolve.

Hàm **linsolve** dùng để giải hệ phương trình đại số tuyến tính (bằng chữ hoặc bằng số)

$sol_eqs = sp.linsolve([eq1, eq2, \dots, eqn], biến1, biến2, \dots, biếnn)$

```
> import sympy as sp
  x, y, z = sp.symbols('x y z', real=True)
  eq1 = 3*x + y - 5*z - 6
  eq2 = x - 4*y + 2*z + 4
  eq3 = -x - 8*y + 3*z - 7
  eqs = [eq1, eq2, eq3]
  sol_eqs = sp.linsolve(eqs, x, y, z)
  print('Nghiệm của hệ: (x, y, z) =', sol_eqs)
[1] ✓ 0.5s
...
Nghiệm của hệ: (x, y, z) = {(-250/67, -123/67, -255/67)}
```

13.9. Hàm nonlinsolve.

Hàm **nonlinsolve** dùng để giải hệ phương trình phi tuyến (bằng chữ hoặc bằng số)

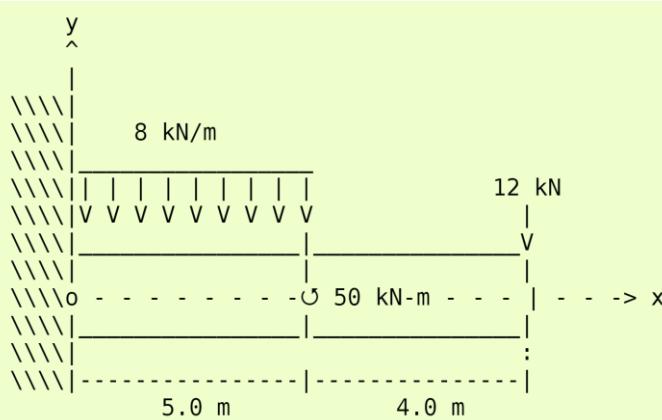
$sol_eqs = sp.nonlinsolve([eq1, eq2, \dots, eqn], biến1, biến2, \dots, biếnn)$

```

> import sympy as sp
x, y, z = sp.symbols('x y z', real=True)
eq1 = 3*x*y + 2*y - z - 1
eq2 = 2*x**2 - 2*y + 4*z + 2
eq3 = 2*x - x*y + 2*z
eqs =[eq1, eq2, eq3]
sol_eqs = sp.nonlinsolve(eqs, x, y, z)
print('Nghiệm của hệ: (x, y, z) =', sol_eqs)
[1] ✓ 0.7s
...
Nghiệm của hệ: (x, y, z) = {(-2/5, 7/5, 3/25), (1, 0, -1)}

```

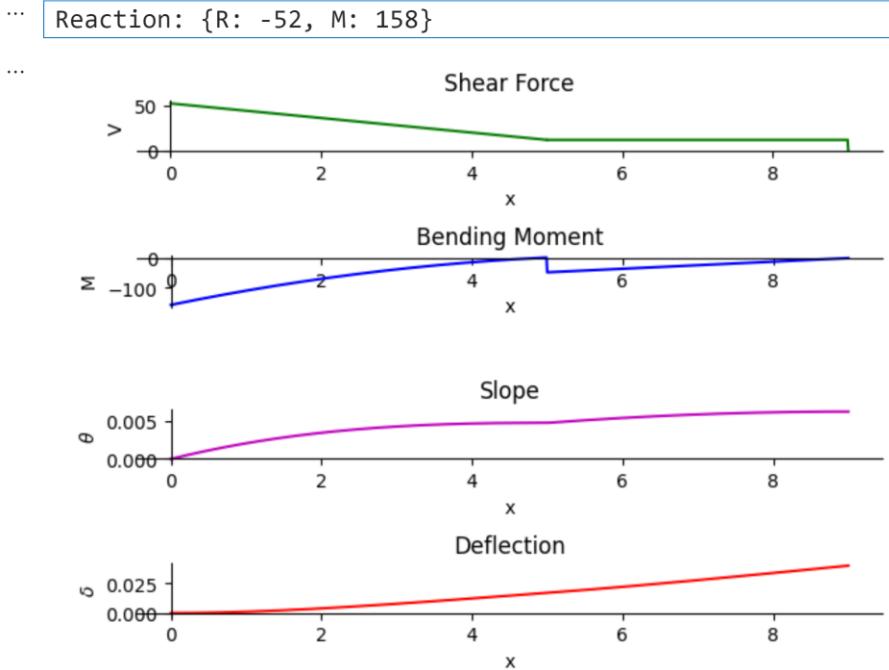
13.10. Gói sympy.physics.continuum_mechanics.beam tính đầm.



```

> import sympy as sp
from sympy.physics.continuum_mechanics.beam import Beam
E, I = sp.symbols('E, I')
b = Beam(9, E, I) # Khai báo đầm L = 9m
b.apply_load(12, 9, -1) # Lực tập trung (bậc -1)
b.apply_load(50, 5, -2) # Moment tập trung (bậc -2)
b.apply_load(8, 0, 0, end=5) # Lực phân bố (bậc 0)
b.bc_deflection.append((0, 0)) # Ràng buộc chuyển vị
b.bc_slope.append((0, 0)) # Ràng buộc góc xoay
R, M = sp.symbols('R, M')
b.apply_load(R, 0, -1) # Phản lực liên kết tại ngầm
b.apply_load(M, 0, -2) # Moment phản lực liên kết tại ngầm
b.solve_for_reaction_loads(R, M)
print('Reaction:', b.reaction_loads)
b.plot_loading_results(subs={E: 20E9, I: 3.25E-6})
[1] ✓ 2.9s

```



14. Hướng dẫn sử dụng gói tabulate cơ bản.

```
from tabulate import tabulate
```

Gói `tabulate` chỉ có khi cài đặt trên máy tính, khi sử dụng online sẽ không có gói này. Mục đích của hàm `tabulate` là để xuất dữ liệu dạng bảng có kẽ hàng, cột bằng ký tự.

Khi cần in ra kết quả tính toán là mảng 2D mô tả các đại lượng khác nhau giữa các cột.

Để rõ ràng hơn thì hàng trên cùng cần phải thêm tiêu đề (`headers`) và cột bên trái đánh số thứ tự của hàng.

Trên cơ sở bảng `KetQua` đã có.

- Tạo hai mảng 2D: `Index` và `TieuDe`
- Ghép cột: `Bang1 = np.c_[index, KetQua]`
- Ghép hàng: `Bang2 = np.r_[TieuDe, Bang1]`
- `print(tabulate(Bang2, headers = 'firstrow', tablefmt = 'Kiểu_Đường'))`
 - + `headers = 'firstrow'` → lấy hàng đầu tiên làm tiêu đề.

+ `tablefmt = 'Kiểu_Đường'`

Có thể chọn một trong số kiểu đường sau:

`tablefmt = 'rounded_outline'`

`tablefmt = 'mixed_grid'`

`tablefmt = 'double_grid'`

`tablefmt = 'fancy_grid'`

`tablefmt = 'simple_outline'`

`tablefmt = 'heavy_outline'`

`tablefmt = 'mixed_outline'`

`tablefmt = 'double_outline'`

`tablefmt = 'fancy_outline'`

`tablefmt = 'outline'`

▷ v

```
import numpy as np
from tabulate import tabulate
KetQua = np.arange(9, 25).reshape(4,4) # Kết quả tính toán
TieuDe = np.array([['Chỉ số', 'Cột 0', 'Cột 1', 'Cột 2', 'Cột 3']])
Index = np.arange(4).reshape(4,1)
Bang1 = np.c_[Index, KetQua]
Bang2 = np.r_[TieuDe, Bang1]
print(tabulate(Bang2, headers='firstrow', tablefmt='outline'))
```

[1] ✓ 0.1s

...

Chỉ số	Cột 0	Cột 1	Cột 2	Cột 3
0	9	10	11	12
1	13	14	15	16
2	17	18	19	20
3	21	22	23	24

PHẦN 2: ỨNG DỤNG TRONG KỸ THUẬT.

1. Đạo hàm của hàm số, dãy số.

1.1. Đạo hàm của một hàm số.

Xét hàm số $f(x)$, chọn trước bước sai phân, trong kỹ thuật thường chọn $h = 10^{-3}$. Công thức xấp xỉ đạo hàm với sai số cấp hai.

$f'(x_i) =$	$\frac{-3f(x_i) + 4f(x_{i+1}) - f(x_{i+2})}{2h} + \mathcal{O}(h^2)$	
	$\frac{f(x_{i-1}) - f(x_{i+1})}{2h} + \mathcal{O}(h^2)$	
	$\frac{f(x_{i-2}) - 4f(x_{i-1}) + 3f(x_i)}{2h} + \mathcal{O}(h^2)$	
$f''(x_i) =$	$\frac{2f(x_i) - 5f(x_{i+1}) + 4f(x_{i+2}) - f(x_{i+3})}{h^2} + \mathcal{O}(h^2)$	
	$\frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{h^2} + \mathcal{O}(h^2)$	
	$\frac{-f(x_{i-3}) + 4f(x_{i-2}) - 5f(x_{i-1}) + 2f(x_i)}{h^2} + \mathcal{O}(h^2)$	
$f'''(x_i) =$	$\frac{-5f(x_i) + 18f(x_{i+1}) - 24f(x_{i+2}) + 14f(x_{i+3}) - 3f(x_{i+4})}{2h^3} + \mathcal{O}(h^2)$	
	$\frac{-f(x_{i-2}) + 2f(x_{i-1}) - 2f(x_{i+1}) + f(x_{i+2})}{2h^3} + \mathcal{O}(h^2)$	
	$\frac{3f(x_{i-4}) - 14f(x_{i-3}) + 24f(x_{i-2}) - 18f(x_{i-1}) + 5f(x_i)}{2h^3} + \mathcal{O}(h^2)$	
$f^{IV}(x_i) =$	$\frac{3f(x_i) - 14f(x_{i+1}) + 26f(x_{i+2}) - 24f(x_{i+3}) + 11f(x_{i+4}) - 2f(x_{i+5})}{h^4} + \mathcal{O}(h^2)$	
	$\frac{f(x_{i-2}) - 4f(x_{i-1}) + 6f(x_i) - 4f(x_{i+1}) + f(x_{i+2})}{h^4} + \mathcal{O}(h^2)$	
	$\frac{-2f(x_{i-5}) + 11f(x_{i-4}) - 24f(x_{i-3}) + 26f(x_{i-2}) - 14f(x_{i-1}) + 3f(x_i)}{h^4} + \mathcal{O}(h^2)$	

Ví dụ 1.

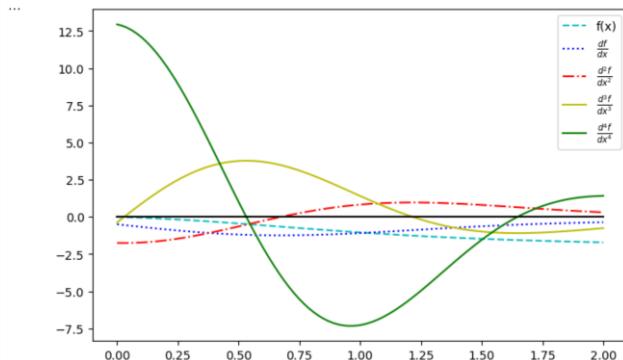
Cho hàm số: $f(x) = e^{-x^2} - \sqrt{x+1}$. Vẽ đồ thị hàm số và đạo hàm từ cấp một đến cấp 4 của $f(x)$, tính các giá trị tại $x = 1$

```
import numpy as np
import matplotlib.pyplot as plt
f = lambda x: np.exp(-x**2)-np.sqrt(x+1)
def diff(x):
    h=1e-3
    d1=(f(x+h)-f(x-h))/2/h
    d2=(f(x+h)-2*f(x)+f(x-h))/h**2
    d3=(f(x+2*h)-2*f(x+h)+2*f(x-h)-f(x-2*h))/2/h**3
    d4=(f(x+2*h)-4*f(x+h)+6*f(x)-4*f(x-h)+f(x-2*h))/h**4
    return [d1, d2, d3, d4]
x=np.linspace(0,2,100); out=diff(x); out1 = diff(1)
print('f(1) = ',f(1), '\n d1(1) = ',out1[0], '\n
      \nd2(1) = ',out1[1], '\n d3(1) = ',out1[2], '\n d4(1) = ',out1[3])
plt.figure(figsize=(8,5))
plt.plot(x,f(x), '--c', x,out[0], 'b',
          x,out[1], '-r', x,out[2], 'y',
          x,out[3], 'g', [0,2],[0,0], 'k')
plt.legend((('f(x)', r'$\frac{df}{dx}$'),
            ('d1', r'$\frac{d^2f}{dx^2}$'),
            ('d2', r'$\frac{d^3f}{dx^3}$'),
            ('d3', r'$\frac{d^4f}{dx^4}$')), loc=1); plt.show()
```

```

...
f(1) = -1.0463341212016528
d1(1) = -1.089312038731638
d2(1) = 0.8241466238345652
d3(1) = 1.4052270458364546
d4(1) = -7.274181257344025

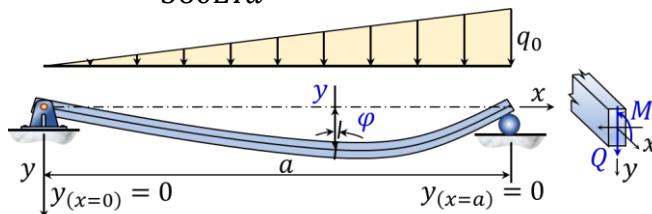
```



Ví dụ 2.

Độ vồng của đầm có phương trình:

$$y(x) = \frac{q_0}{360EIa} (-3x^5 + 10a^2x^3 - 7a^4x)$$



Liên hệ giữa các đại lượng trên đầm:

- Góc xoay của tiết diện: $\varphi = y^I$
- Moment uốn trên tiết diện: $M = -EI \times y^{II}$
- Lực cắt trên tiết diện: $Q = -EI \times y^{III}$

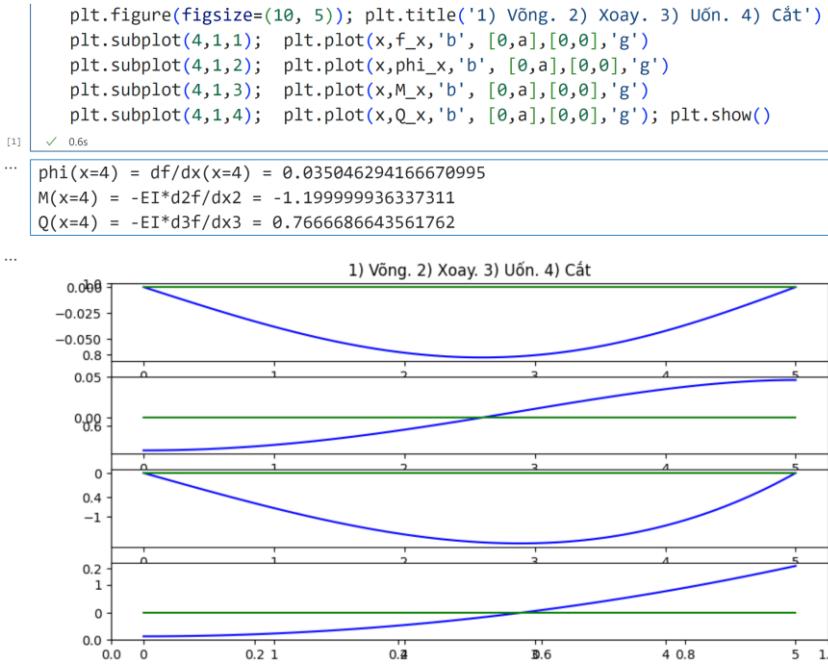
Biết: $a = 5m$; $E = 2.10^{11} N/m^2$; $I = 3 \times 10^{-7} m^4$; $q_0 = 1 kN/m$

Xác định góc xoay, moment uốn và lực cắt tại $x = 4m$.

```

> 
import numpy as np
import matplotlib.pyplot as plt
f = lambda x, a: -3*x**5+10*a**2*x**3-7*a**4*x
d1 = lambda x: (f(x+h,a)-f(x-h,a))/2/h; h=1e-3
d2 = lambda x: (f(x+h,a)-2*f(x,a)+f(x-h,a))/h**2; h=1e-3
d3 = lambda x: (f(x+2*h,a)-2*f(x+h,a)+2*f(x-h,a)-f(x-2*h,a))/2/h**3; h=1e-3
a=5; E=2e8; I=3e-7; q0=1; y0=q0/(360*E*I*a)
print('phi(x=4) = df/dx(x=4) =', y0*d1(4), \
      '\nM(x=4) = -EI*d2f/dx2 =', -E*I*y0*d2(4), \
      '\nQ(x=4) = -EI*d3f/dx3 =', -E*I*y0*d3(4))
x = np.linspace(0, a, 100)
f_x = y0*f(x, a); phi_x = y0*d1(x)
M_x = -E*I*y0*d2(x); Q_x = -E*I*y0*d3(x)

```

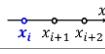
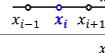
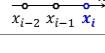
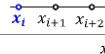
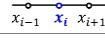
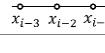
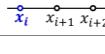
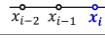
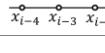
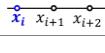


1.2. Đạo hàm của một dãy số.

Thường khi làm thực nghiệm sẽ có bảng số liệu dạng 2D như dưới.

<i>i</i>	0	1	...	i - 2	i - 1	<i>i</i>	i + 1	i + 2	...	n
<i>x_i</i>	<i>x₀</i>	<i>x₁</i>	...	<i>x_{i-2}</i>	<i>x_{i-1}</i>	<i>x_i</i>	<i>x_{i+1}</i>	<i>x_{i+2}</i>	...	<i>x_n</i>
<i>y_i</i>	<i>y₀</i>	<i>y₁</i>	...	<i>y_{i-2}</i>	<i>y_{i-1}</i>	<i>y_i</i>	<i>y_{i+1}</i>	<i>y_{i+2}</i>	...	<i>y_n</i>

Lúc này, các công thức xấp xỉ đạo hàm được viết lại.

$f'(x_i) =$	$\frac{-3y_i + 4y_{i+1} - y_{i+2} + \mathcal{O}(h^2)}{(x_{i+2} - x_i)}$	
	$\frac{y_{i-1} - y_{i+1}}{(x_{i+2} - x_i)} + \mathcal{O}(h^2)$	
	$\frac{y_{i-2} - 4y_{i-1} + 3y_i}{(x_{i+2} - x_i)} + \mathcal{O}(h^2)$	
$f''(x_i) =$	$\frac{2y_i - 5y_{i+1} + 4y_{i+2} - y_{i+3} + \mathcal{O}(h^2)}{(x_{i+1} - x_i)^2}$	
	$\frac{y_{i-1} - 2y_i + y_{i+1}}{(x_{i+1} - x_i)^2} + \mathcal{O}(h^2)$	
	$\frac{-y_{i-3} + 4y_{i-2} - 5y_{i-1} + 2y_i}{(x_i - x_{i-1})^2} + \mathcal{O}(h^2)$	
$f'''(x_i) =$	$\frac{-5y_i + 18y_{i+1} - 24y_{i+2} + 14y_{i+3} - 3y_{i+4}}{2(x_{i+1} - x_i)^3} + \mathcal{O}(h^2)$	
	$\frac{-y_{i-2} + 2y_{i-1} - 2y_{i+1} + y_{i+2}}{2(x_{i+1} - x_i)^3} + \mathcal{O}(h^2)$	
	$\frac{3y_{i-4} - 14y_{i-3} + 24y_{i-2} - 18y_{i-1} + 5y_i}{2(x_i - x_{i-1})^3} + \mathcal{O}(h^2)$	
$f^{IV}(x_i) =$	$\frac{3y_i - 14y_{i+1} + 26y_{i+2} - 24y_{i+3} + 11y_{i+4} - 2y_{i+5}}{(x_{i+1} - x_i)^4} + \mathcal{O}(h^2)$	

$\frac{y_{i-2} - 4y_{i-1} + 6y_i - 4y_{i+1} + y_{i+2}}{(x_{i+1} - x_i)^4} + \mathcal{O}(h^2)$	$\frac{-2y_{i-5} + 11y_{i-4} - 24y_{i-3} + 26y_{i-2} - 14y_{i-1} + 3y_i}{(x_i - x_{i-1})^4} + \mathcal{O}(h^2)$	
---	---	--

Lưu ý: để không mất điểm tính đạo hàm thì các điểm đầu dùng sai phân tiên, các điểm giữa dùng sai phân hướng tâm và các điểm cuối dùng sai phân lùi.

Ví dụ 3.

Bảng thực nghiệm đo được độ vồng theo phuong y dọc theo phuong x của đàm như bảng dưới. Biết: $E = 200GPa$; $I = 3 \times 10^{-10}m^4$



x[m]	0	0.5	1	1.5	2	2.5	3	3.5	4
y[cm]	0	-1.01	-1.89	-2.51	-2.78	-2.63	-2.07	-1.14	0

Dùng xấp xỉ của đạo hàm để tìm các giá trị moment uốn và lực cắt dọc theo đàm. Liên hệ: $M = -EI \times y''$; $Q = -EI \times y'''$

▷

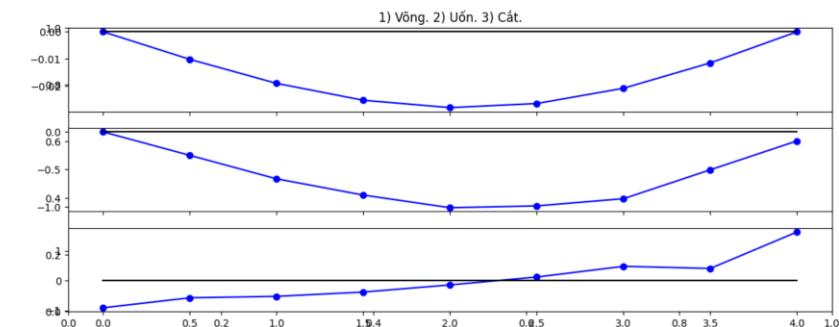
```

import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate
x=np.array([0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4])
y=np.array([0, -1.01, -1.89, -2.51, -2.78, -2.63, -2.07, -1.14, 0])
y=y*0.01; E=2e11; I=3e-10
M=[]; Q=[]
for i in range(len(x)):
    if i == 0:
        d2 = (2*y[i]-5*y[i+1]+4*y[i+2]-y[i+3])/(x[i+1]-x[i])**2
    elif 0 < i < len(x)-1:
        d2 = (y[i-1]-2*y[i]+y[i+1])/(x[i+1]-x[i])**2
    else:
        d2 = (-y[i-3]+4*y[i-2]-5*y[i-1]+2*y[i])/(x[i]-x[i-1])**2
    M = np.append(M, -E*I*d2)
    if i<=1:
        d3 = (-5*y[i]+18*y[i+1]-24*y[i+2]+14*y[i+3]-3*y[i+4])/2/(x[i+1]-x[i])**3
    elif 1 < i < len(x)-2:
        d3 = (-y[i-2]+2*y[i-1]-2*y[i+1]+y[i+2])/2/(x[i+1]-x[i])**3
    else:
        d3 = (3*y[i-4]-14*y[i-3]+24*y[i-2]-18*y[i-1]+5*y[i])/2/(x[i]-x[i-1])**3
    Q = np.append(Q, -E*I*d3)
title = np.array(['x', 'y', 'M', 'Q'])
table =np.c_[x, y, M, Q].T; table = np.c_[title, table]
print(tabulate(table, headers='firstrow', tablefmt='simple_outline'))
plt.figure(figsize=(8,5)); plt.title('1) Vồng. 2) Uốn. 3) Cắt.')
plt.subplot(311); plt.plot(x, y,'-bo', [0,x[-1]],[0,0],'k')
plt.subplot(312); plt.plot(x, M,'-bo', [0,x[-1]],[0,0],'k')
plt.subplot(313); plt.plot(x, Q,'-bo', [0,x[-1]],[0,0],'k'); plt.show()

```

[1] ✓ 0.6s

x	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
y	0	-0.0101	-0.0189	-0.0251	-0.0278	-0.0263	-0.0207	-0.0114	0
M	1.66533e-15	-0.312	-0.624	-0.84	-1.008	-0.984	-0.888	-0.504	-0.12
Q	-0.912	-0.576	-0.528	-0.384	-0.144	0.12	0.48	0.408	1.632



2. Tích phân xác định.
3. Nội suy và xấp xỉ.
4. Giải phương trình phi tuyến một biến.
5. Giải hệ phương trình tuyến tính.
6. Giải hệ phương trình phi tuyến.
7. Giải phương trình vi phân bài toán trị đầu.
8. Giải phương trình vi phân bài toán trị biên.