

BÀI 1. GIỚI THIỆU VỀ PYTHON VÀ CÁC THƯ VIỆN TÍNH TOÁN

Mục tiêu:

- *Nắm vững được Python để viết các đoạn lệnh.*
- *Nâng cao kỹ năng lập trình với phương pháp tư duy toán và vẻ đẹp của ngôn ngữ Python.*
- *Sử dụng được gói NumPy, SciPy để thực hiện các tác vụ đơn giản xử lý đại số.*

Nội dung chính:

1. Môi trường và một số lưu ý về lập trình Python với đại số tuyến tính

Với nền tảng kiến thức cơ bản về lập trình cơ bản Python, sinh viên cần thực hành thêm các lệnh trong ngôn ngữ Python dưới đây để có những kỹ năng xử lý cho các bài toán đại số:

1.1. Gói phần mềm Python tích hợp

Trong nội dung thực hành này, gói phần mềm Anaconda sẽ được sử dụng để minh họa về các thao tác tính toán về đại số tuyến tính với các câu lệnh Python. Anaconda là tập hợp các gói thư viện thực thi trên nền tảng Python.

Sinh viên có thể tự cài đặt gói Anaconda vào máy tính cá nhân để nghiên cứu, học tập và thực hành cũng như làm bài tập được giao từ trang web: <https://www.anaconda.com/distribution> phiên bản 3.x (1/2019 là 3.7).

Lưu ý 1: Khi cài đặt, nếu trong máy tính đã có các hệ thống sử dụng Python thì các biến môi trường không cần phải thay đổi để không ảnh hưởng đến các chương trình khác. Vị trí cài đặt có thể chọn là: C:\Anaconda hoặc D:\Anaconda để việc sử dụng dễ dàng.

Lưu ý 2: Hiện tại, ngoài Anaconda, nhiều gói phần mềm Python tương tự. Để cài đặt và sử dụng, yêu cầu cần đọc rõ các gói hỗ trợ, đặc biệt là các thư viện: numpy, scipy, networkx, sympy. Do đó, với hệ sinh thái phong phú của Python, sinh viên có thể sử dụng thư viện khác trong nghiên cứu. Tuy nhiên, trong lớp học, giảng viên chỉ giới thiệu các thư viện nền tảng và thống nhất sử dụng.

1.2. Môi trường thử nghiệm và lập trình

Mặc dù có nhiều môi trường phát triển phần mềm, trong khuôn khổ thực hành này, sinh viên được khuyến nghị sử dụng IDE có tên là IDLE nằm trong gói phần mềm Anaconda. Tập tin thực thi của IDLE là idle.exe có thể tìm thấy trong thư mục Anaconda sau khi cài đặt.

IDLE cung cấp một cửa sổ dòng lệnh đơn giản nhưng hiệu quả và dễ dàng thực thi câu lệnh.

Sau khi khởi động idle, ứng dụng sẽ có tên là **Python 3.x.y Shell** (ví dụ: 3.6.3) với dấu nhắc >>> được thể hiện trên màn hình để thực hiện các câu lệnh.

1.3. Các phép xử lý với danh sách

- Lưu ý về các phép toán xử lý trên list

Danh sách (**list**) là đối tượng thường được sử dụng trong Python. Như tên gọi, mục tiêu chính của cấu trúc dữ liệu danh sách chính là lưu trữ những đối tượng. Do đó, các phép tương tác trên cấu trúc dữ liệu **list** hướng đến xử lý về lưu trữ dữ liệu hơn là tính toán. Cụ thể là: phép toán + hai danh sách mang ý nghĩa ghép nối 2 danh sách. Ví dụ:

```
>>> danhsach1 = [1. , 3.]
>>> danhsach2 = [5. , 7.]
>>> danhsach = danhsach1 + danhsach2
..... ← Sinh viên điền kết quả.
>>> danhsach_gapdoi = 2 * danhsach
..... ← Sinh viên điền kết quả.
>>> danhsach * 2
..... ← Sinh viên điền kết quả.
>>> danhsach / 2
..... ← Sinh viên điền kết quả.
```

- Ghép các danh sách bằng lệnh zip:

Giả sử có 1 bạn học 4 môn với thứ tự các môn và điểm số như 3 danh sách bên dưới, chúng ta cần ghép cặp từng môn học

```
>>> mon_hoc = ["ToanCC", "DSTT", "ToanRR", "LaptrinhCB"]
>>> thu_tu = [2, 3, 4, 1]
>>> diem_so = [10, 9, 8, 7]
>>> anh_xa = zip(thu_tu, mon_hoc, diem_so)
>>> anh_xa
..... ← Sinh viên điền kết quả.
>>> tap_hop = set(anh_xa)          # ←chuyển thành dạng tập hợp
>>> tap_hop
..... ← Sinh viên điền kết quả.
```

Sau đó, từ dữ liệu được ghép cặp bằng zip, chúng ta có thể phân rã bằng phương pháp sau:

```
>>> lay_TT, lay_monhoc , lay_diem = zip(*anh_xa)
>>> lay_monhoc
..... ← Sinh viên điền kết quả.
```

- Xây dựng danh sách: Bên cạnh những cách khai báo danh sách thông thường như: khai báo sẵn hoặc khai báo list rỗng rồi bổ sung phần tử bằng lệnh append, cơ chế sinh tập hợp list trong Python còn có thể thực hiện bằng các cách sau:

```
>>> import itertools
>>> tap_sinh = list(itertools.chain(range(4), range(5,10), range(15,20) ))
>>> tap_sinh
```

..... ← Sinh viên điền kết quả.

Ngoài ra, với phép toán zip bên trên, chúng ta có thể tạo ra các tập là tập sinh từ 2 hoặc nhiều tập.

Ví dụ: Cần tạo 1 bộ 3 thành phần gồm: danh sách từ 0 đến 3; danh sách từ 7 đến 11 và ngược lại một danh sách từ 10. Lệnh Python như sau:

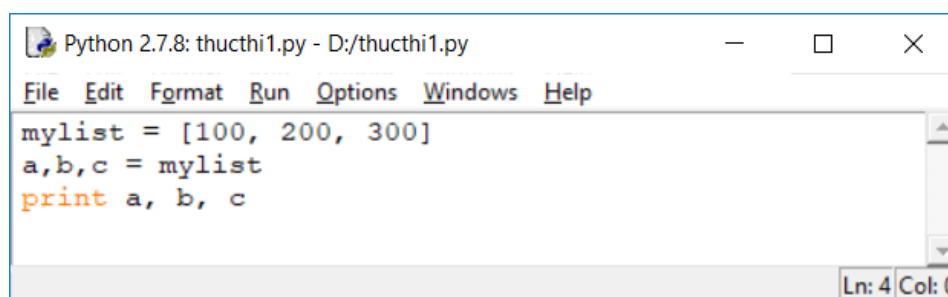
```
>>> list(zip(range(4), range(7, 12), reversed(range(11) ) ) )
```

..... ← Sinh viên điền kết quả.

1.4. Lệnh thực thi một tập tin python

Trong môi trường dòng lệnh Python, một tập tin Python (*.py) được thực thi thông qua hàm **execfile**. Sinh viên thực hiện minh họa dưới đây:

Bước 1: Tạo một tập tin thucthi1.py và đặt tại một thư mục (trong hình minh họa là đặt tại 'd:/')



Bước 2: Thực thi tập tin trên. Sinh viên thử nghiệm các lệnh sau và ghi kết quả đạt được:

```
>>> a = b = c = 0
```

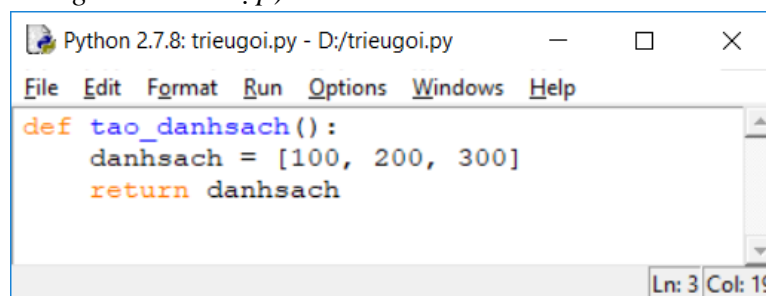
```
>>> mylist = []
```

```
>>> execfile('d:/thucthi1.py') # hoặc câu lệnh sau >>> execfile('d:\\thucthi1.py')
```

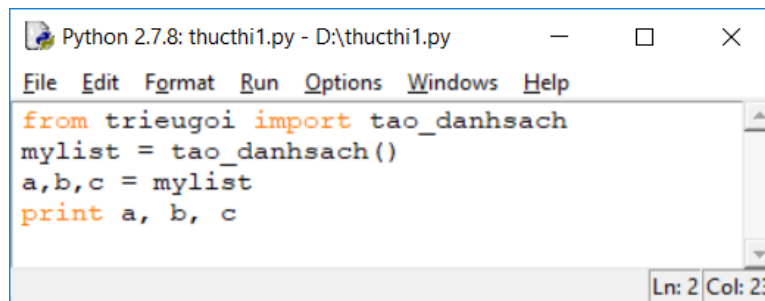
..... ← Sinh viên điền kết quả.

Ngoài ra, khi đã có một tập tin Python (.py) cùng thư mục, chúng ta có thể dễ dàng truy xuất đến hàm (def) của nó thông qua lệnh **from <tên tập tin> import <tên hàm xử lý>**. Ví dụ sau:

- Giả sử bổ sung thêm vào đường dẫn D:\ tập tin có tên là triegoi.py (*lưu ý: đường dẫn 'D:\' là đường dẫn minh họa, tùy thuộc vào máy tính sinh viên đang thực tập để chọn đường dẫn thích hợp*).



- Điều chỉnh tập tin thucthi1.py như sau:

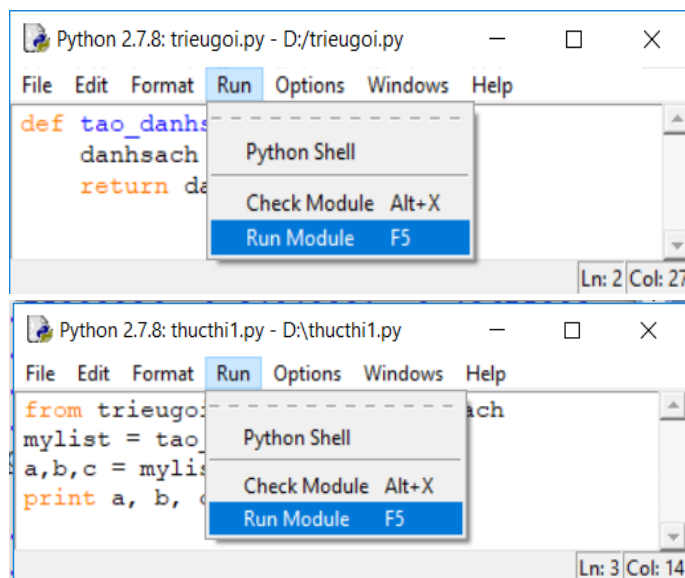


```
Python 2.7.8: thucthi1.py - D:\thucthi1.py
File Edit Format Run Options Windows Help
from trieugoi import tao_danhsach
mylist = tao_danhsach()
a,b,c = mylist
print a, b, c
Ln: 2 Col: 23
```

- Sau đó thử nghiệm là đoạn script trên:

```
>>> a = b = c = 0
>>> mylist = []
>>> execfile('d:/thucthi1.py')
```

Lưu ý: Trong một số IDE (như IDLE được tích hợp với gói Anaconda), để “biên dịch” vào môi trường chúng ta phải thực hiện việc Run Module để “tải” thư viện vào IDE (trước khi sử dụng) hoặc thực thi với IDE tập tin thucthi1.py:



1.5. Khởi lệnh bắt đầu với with

Từ khóa with được sử dụng để bắt đầu một khối đoạn lệnh với ý nghĩa là định nghĩa ngữ cảnh thực thi xác định.

Ví dụ: Để xây dựng một hàm đọc số dòng của một tập tin, chúng ta có thể viết như sau:

```
>>> def dem_dong(ten_taptin):
    with open(ten_taptin, 'r') as taptin:
        return len(taptin.readlines())
```

Về ý nghĩa, từ khóa with mở ra một khối lệnh xử lý với đối tượng được trả về sau một lệnh. Trong ví dụ trên, đó là đối tượng *taptin* dạng tập tin (file) được tạo thành từ lệnh **open** và xử lý trong khối lệnh đó. Với phong cách viết code như nêu một mệnh đề toán học và bên dưới là các triển khai cụ thể, cho thấy sự sáng sủa trong phong cách viết code của Python để người sử dụng dễ dàng định hướng khi xây dựng chương trình và trong kiểm lỗi.

Lưu ý: Từ phiên bản 2.6 trở đi, lệnh **with** được sử dụng trực tiếp. Trước đó, trong phiên bản 2.5, để sử dụng lệnh with, chúng ta phải khai báo thư viện: **from __future__ import with_statement**.

2. Các gói thư viện cơ bản tích hợp trong Python hỗ trợ toán học

Dưới đây là các gói thư viện cơ bản của Python sinh viên cần được trang bị để hỗ trợ tính toán đại số:

- **Gói math:** Python hỗ trợ tính toán các phép toán số học cơ bản với thư viện math. Để sử dụng thư viện, chúng ta đơn giản import vào như sau: **>>> import math** và sử dụng.

Bảng minh họa một số hàm, hằng và các xử lý toán học cơ bản trong thư viện **math** của python:

TT	Nhóm	Tên hàm	Ý nghĩa	Lệnh minh họa
1	Hàm lấy giá trị	floor	Giá trị nguyên lớn nhất không vượt số thực (số sàn)	>>> x = 5.4 >>> math.floor(x)
2		ceil	Giá trị nguyên nhỏ nhất lớn hơn số thực (số trần)	>>> x = 5.4 >>> math.ceil(x)
3		fabs	Lấy giá trị tuyệt đối của	>>> math.fabs(-x)
4	Mũ và lũy thừa	exp	Hàm lấy mũ e^x	>>> x = 1 >>> math.exp(x)
5		expm1	Hàm lấy mũ $e^x - 1$	>>> x = 1e-4 >>> math.expm1(x)
6		pow(x,y)	Tính giá trị e^x	>>> math.exp(2,3)
7		log	Hàm lấy logarith theo cơ số, mặc định là cơ số e.	>>> math.log(3) >>> math.log(8, 2)
8		log2	Lấy log cơ số 2 (từ phiên bản 3.5)	>>> math.log2(1024)
9		log10	Lấy log cơ số 10	>>> math.log10(1000)
10	Lượng giác	pi (viết thường)	Lấy giá trị số Pi.	>>> radius = 5 >>> S = math.pi *(radius ** 2)
11		radians(x)	Chuyển độ sang giá trị radian	>>> goc = 60 >>> radi = math.radians(goc)
12		degrees(x)	Chuyển giá trị radian sang độ	>>> math.degrees(1.0) >>> math.degrees(math.pi)

13		sin(rad)	Tính sin của góc radian	>>> math.sin(radi)
14		cos(rad)	Tính cos của góc radian	>>> math.cos(radi)
15		tan(rad)	Tính tan của góc radian	>>> math.tan(radi)
16		asin(x)	Lấy giá trị arcsin(x)	>>> math.degrees(math.asin(0.5))
17		acos(x)	Lấy giá trị arccos(x)	>>> math.degrees(math.acos(0.5))
18		atan(x)	Lấy giá trị arctan(x)	>>> math.atan(x)
19	Tính toán số học	sqrt(x)	Hàm lấy căn số thực	>>> math.sqrt(16) >>> math.sqrt(-9)
20		factorial(k)	Tính giai thừa của một số nguyên. Lỗi phát sinh khi nhập số thực	>>> math.factorial(5) # = 120
21		gcd(a, b)	[Phiên bản 3.5 trở lên] Ước số chung lớn nhất của hai số nguyên	>>> math.gcd(18, 27) # = 9
22		fsum(dãy)	Tính tổng của một list/...	>>> a = [1,2, 3, 4,5] >>> math.fsum(a)
23		hypot(x,y)	Tính $\sqrt{x^2 + y^2}$ với x,y là các số thực.	>>> math.hypot(3,4)
24	Hằng số	inf	Số vô cùng	>>> math.inf
25		nan	Không phải giá trị số. Để kiểm một biến có phải là số thực hay không	>>> math.nan

Sinh viên thực hành các lệnh:

```
>>> x = 1
```

```
>>> math.expm1(x) == math.exp(x) - 1
```

..... ← Sinh viên điền kết quả.

```
>>> x = 1e-5 # lưu ý: viết chữ e dính liền dấu – và số 1.
```

```
>>> math.expm1(x) == math.exp(x)- 1
```

..... ← Sinh viên điền kết quả (True hay False?).

```
>>> math.exp(x) – 1
```

..... ← Sinh viên điền kết quả.

```
>>> math.exp(1)
```

..... ← Sinh viên điền kết quả.

Như vậy, với các số có giá trị nhỏ, nếu sử dụng phương pháp tính $\exp(x)-1$ sẽ gây sai số!

Sinh viên có thể tham khảo và tiếp cận nhiều hàm khác tại:

<https://docs.python.org/3/library/math.html>.

Ngoài ra, một số gói thư viện sẽ được giới thiệu và thực hành kết hợp trong các bài tiếp theo như:

- **Gói random:** liên quan đến các vấn đề ngẫu nhiên.
- **Gói itertools:** liên quan đến các bài toán tập hợp như tổ hợp/chỉnh hợp.
- **Gói datetime:** liên quan đến xử lý thời gian, đặc biệt với dữ liệu có ngày giờ.
- **Gói numbers:** hỗ trợ xử lý các dạng số chung.
- **Gói cmath:** hỗ trợ tính toán số phức.
- **Gói decimal:** hỗ trợ tính toán và xử lý số thực.
- **Gói statistics:** hỗ trợ các công cụ tính các chỉ số thống kê cơ bản như: trung bình, mode, trung phương (median)... của một tập số.

3. Làm quen với thư viện NumPy

NumPy là một nền tảng tính toán của Python. Tài liệu của Numpy có thể tìm thấy dễ dàng trên mạng, như: <https://docs.scipy.org/doc/numpy/numpy-ref-1.16.1.pdf> (khoảng 1372 trang tính đến 01/2019). Chúng ta chỉ quan tâm đến một số lệnh của Numpy liên quan đến các bài toán đại số tuyến tính. Các lệnh còn lại sinh viên có thể chủ động tự tìm hiểu và nghiên cứu để sử dụng.

Để sử dụng numpy, trước tiên, chúng ta phải khai báo thư viện:

```
>>> import numpy as np
```

Giải pháp khai báo như thế sẽ được sử dụng trong các bài thực hành bên dưới. Khi đó, để sử dụng thư viện numpy, chúng ta phải sử dụng tiền tố **np**.

Lưu ý: Cách khác hơn (không giới thiệu trong bài thực hành này), chúng ta có thể khai báo như sau:

```
>>> from numpy import *
```

Với cách khai báo này, chúng ta có thể sử dụng trực tiếp các hàm của numpy mà không cần phải bổ sung tên thư viện numpy (**np**.) ở phía trước các câu lệnh. Tuy nhiên, nhược điểm của phương pháp là khó theo dõi các hàm thuộc numpy hoặc của thư viện khác với người mới/đang tiếp cận lập trình numpy.

3.1. Một số lệnh cơ bản numpy xử lý vector

Khác với kiểu danh sách ở Python chuẩn, numpy hỗ trợ việc định nghĩa một vector theo nghĩa đại số:

```
>>> vec1 = np.array([1., 3., 5.])
```

```
>>> vec1 * 2
```

```
..... ← Sinh viên điền kết quả.
>>> vec1 * vec1
..... ← Sinh viên điền kết quả và
cho biết đó phép nhân gì?
>>> vec1 / 2
..... ← Sinh viên điền kết quả.
>>> vec1 + vec1
..... ← Sinh viên điền kết quả.
>>> vec2 = array([2., 4.])
>>> vec1 + vec2
..... ← Sinh viên điền kết quả và
lí do.
>>> vec3 = array([2., 4., 6.])
>>> vec1 + vec3
..... ← Sinh viên điền kết quả.
>>> vec1 / vec3
..... ← Sinh viên điền kết quả.
>>> vec1 * vec3
..... ← Sinh viên điền kết quả.
>>> 2* vec1 + 5* vec3
..... ← Sinh viên điền kết quả.
```

Lưu ý:

Trong numpy, kiểu dữ liệu vector có tên là **array** là kiểu dữ liệu có kích thước không thay đổi và chỉ 1 kiểu dữ liệu cho toàn bộ vector. Tuy vậy, một số nét tương đồng giữa kiểu dữ liệu list trong python và vector (array) trong Python là:

- Truy xuất đến phần tử của vector:
>>> vec3[2]
..... ← Sinh viên điền kết quả.
- Tạo vector thông qua câu lệnh linspace(phần tử đầu, phần tử cuối, số lượng phần tử):
>>> vec4 = np.linspace(0, 20, 5)
>>> vec4
..... ← Sinh viên điền kết quả.

Ngoài ra, cách thức để tạo nhanh các vector như sau:

- Tạo các vector toàn 0:
>>> vec5 = np.zeros(5)
..... ← Sinh viên điền kết quả.
- Tạo các vector toàn 1:
>>> vec6 = np.ones(5)

- ← Sinh viên điền kết quả.
- Tạo vector ảo (giá trị rỗng):
`>>> vec7 = np.empty(5)`
..... ← Sinh viên điền kết quả.
- Tạo vector các giá trị ngẫu nhiên từ 0 đến 1:
`>>> np.rand(5)` # hoặc `np.random.random(5)` nếu 1 trong 2 lệnh bị lỗi
..... ← Sinh viên điền kết quả.

Các lệnh xử lý:

Các lệnh dưới đây sử dụng lệnh khai báo vector `v` như sau:

```
>>> v = np.array([1., 3., 5.])
```

Sinh viên thực tập các xử lý trên đối tượng vector của numpy:

- Lệnh lấy tổng các thành phần của vector:
`>>> np.sum(v)`
..... ← Sinh viên điền kết quả.
 - Lệnh xem số chiều của một vector:
`>>> v.shape`
..... ← Sinh viên điền kết quả.
 - Thử nghiệm “chuyển vị” vector:
`>>> v.transpose()`
..... ← Sinh viên điền kết quả.
 - Lệnh lấy một phần của vector:
`>>> v1 = v[:2]`
`>>> v1`
..... ← Sinh viên điền kết quả.
`>>> v[0] = 5`
`>>> v`
..... ← Sinh viên điền kết quả.
`>>> v1`
..... ← Sinh viên điền kết quả.
`>>> v1[:2] = [1., 2., 3.]`
..... ← Sinh viên điền kết quả.
.....
`>>> v1[:2] = [1., 2]`
`>>> v`
..... ← Sinh viên điền kết quả.
- Tuy nhiên, lưu ý với các phép gán có sử dụng các toán tử:
- ```
>>> v3 = 2 * v[:2] + v1 * 3
```
- ```
>>> v3
```
- ← Sinh viên điền kết quả.
- ```
>>> v1 = [4, 6]
```
- ```
>>> v3
```
- ← Sinh viên điền kết quả.

```
>>> v
```

```
..... ← Sinh viên điền kết quả.
```

- Các phép toán trên vector: trên vector, chúng ta có thể thực hiện phép toán như: lấy sin/cos:

Lưu ý rằng, các toán tử lấy căn bậc 2 (sqrt), sin, cos cũng phải là các toán tử của numpy

```
>>> v + 10.0
```

```
..... ← Sinh viên điền kết quả.
```

```
>>> np.sqrt(v)
```

```
..... ← Sinh viên điền kết quả.
```

```
>>> np.cos(v)
```

```
..... ← Sinh viên điền kết quả.
```

```
>>> np.sin(v)
```

```
..... ← Sinh viên điền kết quả.
```

- Tích vô hướng của hai vector (kết quả là một số):

$$\vec{v}_1 = (a, b), \vec{v}_2 = (c, d): \vec{v}_1 \vec{v}_2 = ac + bd$$

```
>>> np.dot(v1, v3)
```

```
..... ← Sinh viên điền kết quả.
```

```
>>> v1.dot(v3)
```

```
..... ← Sinh viên điền kết quả.
```

```
>>> v3.dot(v1)
```

```
..... ← Sinh viên điền kết quả.
```

- Độ dài (chuẩn 2) và chuẩn 1 của vector (độ dài theo Euclide):

$$\vec{v}_1 = (a, b): |\vec{v}_1| = \sqrt{a^2 + b^2}; \|\vec{v}_1\| = |a| + |b|$$

```
>>> np.linalg.norm(v)
```

```
..... ← Sinh viên điền kết quả.
```

*Lưu ý: gói thư viện **numpy.linalg** là gói thư viện chuyên xử lý các bài toán của đại số tuyến tính. Gói thư viện này chúng ta sẽ làm quen trong các chương tiếp theo.*

- Độ dài (kích thước) của vector:

```
>>> len(v)
```

```
..... ← Sinh viên điền kết quả.
```

- Kết nối các vector

*Lưu ý: Vector trong numpy không có khái niệm ghép nối (append) như list của python. Tuy nhiên, 2 lệnh thay thế là **hstack** và **vstack** sẽ kết nối vector theo chiều dọc và chiều ngang tương ứng. Sinh viên thực hành các lệnh dưới đây:*

```
>>> np.hstack([v, v3])
```

```
..... ← Sinh viên điền kết quả.
```

Bài tập thực hành:

Minh họa về xử lý vector như sau: Giả định có vector có chiều dài $2n$, n là số tự nhiên dương. Chúng ta muốn chuyển đoạn từ $n + 1$ đến $2n$ về đầu vector đồng thời chuyển đầu các giá trị đó, nghĩa là:

$$(x_0, x_1, \dots, x_{n-1}, x_n, x_{n+1}, \dots, x_{2n-1}) \rightarrow (-x_n, x_{n+1}, \dots, -x_{2n-1}, x_0, x_1, \dots, x_{n-1})$$

Đoạn mã xử lý:

```
>>> def symp(v):
    n = len(v) // 2
    return np.hstack([-v[-n:], v[:n]])
```

```
>>> v6 = np.array([1., 3., 5., 7., 9., 11.])
```

```
>>> v7 = symp(v6)
```

```
>>> v7
```

..... ← Sinh viên điền kết quả.

3.2. Thể hiện ma trận bằng numpy

Ma trận được thể hiện trong numpy là nhiều dãy [] trong kiểu khai báo **array**.

Ví dụ: ma trận $M_1 = \begin{pmatrix} 9 & 12 \\ 23 & 30 \end{pmatrix}$ được khai báo như sau:

```
>>> M1 = np.array([ [9, 12], [23, 30] ])
```

..... ← Sinh viên điền kết quả thể hiện trên màn hình

Khi đó, giả sử chúng ta có một vector $u = (2, 1)$ thì tích giữa ma trận và vector $M_1 u$ sẽ là:

```
>>> u = np.array([ 2, 1])
```

```
>>> tíchM1u = M1.dot(u)
```

```
>>> print(tíchM1u)
```

..... ← Sinh viên điền kết quả và giải thích.

Lưu ý: tích này sẽ khác kết quả với tích dưới đây:

Tích ma trận:

```
>>> tichuM1 = u.dot(M1)
```

```
>>> print(tichuM1)
```

..... ← Sinh viên điền kết quả và giải thích.

Sinh viên thực hành và hãy cho biết kết quả hai phép toán dưới đây:

```
>>> np.dot(M1, u)
```

..... ← Sinh viên điền kết quả và giải thích.

```
>>> np.dot(u, M1)
```

..... ← Sinh viên điền kết quả và giải thích.

Bài tập trên lớp: Sinh viên thực hành và cho biết kết quả các lệnh sau:

Lệnh: `>>> mat1 = np.zeros([5,5])`

Câu hỏi: Cho biết mat1?

Lệnh: `>>> mat2 = np.ones([5,5])`

Câu hỏi: Cho biết mat2?

Lệnh: `>>> mat3 = mat1 + 2* mat2`

Câu hỏi: Cho biết mat3?

Lệnh: `>>> mat4 = mat3`

Lệnh: `>>> mat3[3][2] = 10`

Câu hỏi: Cho biết mat3 và mat4? (mat3 thay đổi thì mat4 có thay đổi theo không?)

Lệnh: `>>> mat5 = np.copy(mat3)`

Lệnh: `>>> mat3[3][2] = 10`

Câu hỏi: Cho biết mat3, mat4 và mat5? (mat3 thay đổi thì mat4 và mat5 có thay đổi theo không?)

Lệnh: `>>> mat6 = np.empty([4, 5])`

Câu hỏi: hãy cho biết mat6?

Lệnh: `>>> mat7 = np.identity(4)`

Câu hỏi: hãy cho biết mat7?

Lệnh: `>>> mat8 = np.rand([4,5])` # hoặc lệnh: `mat8 = np.random.random([4,5])` nếu một trong hai lệnh báo lỗi!

Câu hỏi: hãy cho biết mat8?

4. Giới thiệu thư viện SciPy

Scipy là thư viện xử lý trên nền tảng của numpy. Scipy có nhiều gói xử lý khác nhau để phân tích dữ liệu, như: tích phân, tối ưu, nội suy, biến đổi Fourier, xử lý tín hiệu, xuất nhập tập tin, thống kê, xử lý ảnh nhiều chiều,... Trong đó, nhóm xử lý đại số (như gói **scipy.linalg** và gói **scipy.sparse**) hỗ trợ người sử dụng xử lý tính toán về đại số. Cụ thể:

- Gói **scipy.linalg**: xử lý các thuật toán đại số.
- Gói **scipy.sparse**: xử lý các bài toán về giá trị riêng thưa (sparse eigenvalue).

Để sử dụng SciPy, chúng ta phải import thư viện vào:

```
>>> import scipy as sp
```

Sau đó, những hàm thuộc thư viện sẽ được triệu gọi bằng cách: **sp**.

Trong các bài thực hành tiếp theo chúng ta sẽ làm quen nhiều lệnh hơn với thư viện scipy để xử lý về ma trận...

5. Tài nguyên Python trên mạng về tính toán đại số

Sympy là một trong những gói phần mềm toán học (hệ CAS – Computer Algebra System) miễn phí trên mạng. Sympy được tích hợp sẵn các gói thư viện thực thi như math, numpy,... để người sử dụng có thể tiếp cận. Địa chỉ của Sympy tại: www.sympy.org và địa chỉ sử dụng sympy trực tuyến là <https://live.sympy.org>.

Sympy hỗ trợ các thư viện và hàm để giải quyết các vấn đề về: tính toán tổ hợp, giải phương trình, giải tích, toán rời rạc, vẽ đồ thị, ma trận, hình học,... với nền tảng là gói mpmath. Cụ thể hơn, các tính năng của sympy sinh viên có thể tìm hiểu thêm tại: <https://www.sympy.org/en/features.html>.

Tuy nhiên, tính đến 1/2019, nhược điểm của sympy là hệ thống đang sử dụng là Python 2.7 (chưa phải là 3.x) và một số thư viện như scipy, networkx.

Trong các bài thực hành tiếp theo, một số vấn đề sẽ được thực hiện trên môi trường Sympy. Tuy vậy, đặc điểm lưu ý chính là Sympy có những đối tượng riêng của SymPy khi tương tác. Ví dụ sau có thể thực hiện trên môi trường IDE idle của Anaconda: Sử dụng Sympy để tính toán định thức ma trận.

Sinh viên thực hành:

```
>>> import sympy as sp
>>> M = sp.Matrix([[1, 3], [2, 4]])
>>> M
>>> M.det()
.....
>>> v1 = sp.Matrix([5,7])
.....
>>> M.dot(v1)
.....
```

Tài liệu tham khảo:

1. http://ctr.maths.lu.se/na/courses/NUMA01/course_media/material/unit06_u7JvFS1.pdf
2. <http://www-star.st-and.ac.uk/~pw31/CompAstro/IntroToPython.pdf>
3. http://codingthematrix.com/slides/The_Matrix.pdf
4. Sách Pro Python của Marty Alchin, NXB: Apress, 2010.
5. <https://meshlogic.github.io/posts/jupyter/linear-algebra/linear-algebra-numpy-2/>

BÀI TẬP CHƯƠNG 1

Bài 1. Điểm môn học của một nhóm SV được cho như bên dưới:

Sinh viên	BT cá nhân	Báo cáo	Giữa kỳ	Cuối kỳ
A	7	8	9	5
B	8	7	4	6
C	7	5	8	6
D	6	8	5	4
E	9	9	8	7
F	10	8	9	8

Tỷ lệ % điểm:

BT cá nhân	10%
Báo cáo	20%
Giữa kỳ	30%
Cuối kỳ	40%

Hãy nhập dữ liệu điểm vào ở dạng ma trận, tính ra điểm trung bình của từng HS và cho biết SV nào qua môn (điểm TB ≥ 5.0).

Bài 2.

Sinh ngẫu nhiên ra 2 ma trận kích thước 3×3 và có các phần tử nguyên từ 0 đến 10.

- Tính tổng phần tử của mỗi ma trận.
- So sánh norm của vector hàng 1 và vector hàng 3 của hai ma trận.
- Tìm ma trận C, D, E là tổng, hiệu, tích của hai ma trận trên.

Gợi ý: sử dụng lệnh `np.random.randint(?, size=(?, ?))`.

Bài 3.

Hãy dùng lệnh để sinh ra vector độ dài 21 với các phần tử từ a đến b nào đó. Chẳng hạn: $a = 0$, $b = 100$ hoặc $a = 10.5$, $b = 50$.

- Dùng lệnh `split`, chia nhỏ vector kia ra thành 3 vector con, đặt là u, v, w.
- Tính $u*v$ và $v.dot(w)$
- Tính shape của u trước và sau khi transpose, nhận xét.

Gợi ý: lệnh `np.linspace(a, b, c)` với a, b là các số thực và c là số nguyên dương cho biết số phần tử cách đều nhau trong $[a;b]$