LẬP TRÌNH PYTHON TỪ CƠ BẢN ĐẾN NÂNG CAO

Thực hiện: TS.Trần Bình Long

NỘI DUNG

1. Tổng quan kỹ thuật lập trình
2. Ma trận
3. Đồ thị
4. Giao diện người dùng

"Algorithm + Data structure = Program"
("Giải thuật + Cấu trúc dữ liệu = Chương trình")
Niklaus Wirth

Cách đặt tên

Tên định danh phải thể hiện được ý nghĩa:

- Biến nguyên i, j, k làm biến chạy trong vòng lặp;
- x, y dùng làm biến tọa độ, hoặc biến đại diện cho các số bất kỳ…
- Biến lưu trữ dữ liệu khác nên đặt gợi nhớ, tránh dài dòng: biến đếm "count, dem, so_luong…", biến trọng lượng “weight, trong_luong”, chiều cao “height” ; …
- Đặt quá ngắn như c cho biến đếm, hay w cho khối lượng, sẽ khó hiểu ý nghĩa. Đặt tên quá dài như "the_first_number, the_second_number,…" cho số bất kỳ, sẽ dư thừa, rườm rà.

Cách đặt tên

Tên phải xác định được kiểu dữ liệu lưu trữ,

Phong cách lập trình tốt là khi nhìn vào biến thì xác định ngay kiểu dữ liệu và tên đối tượng mà biến đó lưu trữ.

Tên biến thường là **danh từ** (tên đối tượng) kèm theo tiền tố mang ý nghĩa kiểu dữ liệu. Biến đếm số lần có thể đặt iNumber, i là kiểu số nguyên, strContent là kiểu chuỗi, CPoint là lớp Point…

Có nhiều quy ước đặt tên biến, có thể tham khảo một số quy ước:

Cách đặt tên

Cụ thể:

- Cú pháp Hungary: hình thức chung của cú pháp là thêm tiền tố chứa kiểu dữ liệu vào tên biến.

Tiền tố	Kiểu dữ liệu	Ví dụ minh họa
b	Bool	bool bEmpty, bChecked ;
c	Char	char cInChar, cOutChar ;
str/s	String	string strFirstName, strIn, strOut ;
i/n	Integer	int iCount, nNumElement ;
li	long integer	long liPerson, liStars ;

Cách đặt tên

f	Float	float fPercent ;
d	Double	double dMiles, dFraction ;
if	Input file stream	ifstream ifInFile ;
of	Output file stream	ofstream ofOutFile ;
S	Struct	struct sPoint{…} ;
C	Class	class CStudent,CPerson

Đối với hằng thì tất cả các ký tự đều viết HOA.

Ví dụ:

```
#define MAXSIZE 100
const float PI = 3.14 ;
```

Cách đặt tên cho hàm

Hàm bắt đầu với ký tự đầu tiên viết thường, các ký tự đầu từ phía sau viết hoa, hoặc các từ cách nhau bằng dấu _ (underscore) và không có tiền tố.

Điều này tùy theo ngôn ngữ lập trình.

Hàm có chức năng thực hiện một nhiệm vụ, nên tên là động từ hoặc cụm động từ, thường bắt đầu bằng các động từ: get, set, do, is, make…

Ví dụ:

string setName();

int countElement(); void importArr();

Phong cách viết mã nguồn

Canh lề chương trình:

Khi soạn thảo mã nguồn nên dùng tab hoặc 4 hay 8 khoảng cách để canh lề. Thói quen này giúp cho chương trình được rõ ràng, dễ đọc, dễ quản lý.

Phong cách viết mã nguồn

Không nên	Nên

void docFile (SV a[], int &n)
{
ifstream in;
char* filename="filein.txt"; in.open
(filename);
in>>n;
for(int i=0;i < n;i++)
{
in>>a[i].Masv; in>>a[i].hoten;
in>>a[i].diem;
}
}
	void docFile (SV a[], int &n)
	{
	ifstream in; char* filename
	= "filein.txt";
	in.open (filename); in >> n;
	for (int i=0; i < n; i++)
	{
	in >> a[i].Masv; in >>
	a[i].hoten; in >> a[i].diem;
	}
	}

Phong cách viết mã nguồn

Sử dụng khoảng trắng : chương trình sẽ dễ nhìn hơn

Không nên	Nên

int iCount =0 ; for(int
i=0;i<n;i++)
{
iCount++;
}
cout<<"Ket qua la:"<<iCount;
	int iCount = 0 ;
	for (int i = 0 ; i < n ; i++)
	{
	iCount ++;
	}
	cout << "Ket qua la:" <<
	iCount;

Phong cách viết mã nguồn

Tránh viết nhiều lệnh trên một dòng.

Không nên	Nên

if(a>5){b=a; a++}	if (a > 5)
	{
	b = a; a ++;
	}

Định nghĩa các hằng số.

Hằng số thường xuyên sử dụng nhưng không định nghĩa, dẫn đến xuất hiện những số khó hiểu trong chương trình, được gọi là “magic mumber”.

Không nên	Nên

...
for (int = 0; i < 100;i++)
a[i] = Rand (100);
...
k = InputNum();
int j = 0;
while (A[j] != k && j < 100)
j ++;
...
	#define MAX_LENGTH 100
	#define MAX_NUM 100
	...
	for (int = 0; i < MAX_LENGTH; i ++)
	a[i] = Rand (MAX_NUM);
	...
	k = InputNum ();
	int j = 0;
	while (a[j] != k && j < MAX_LENGTH)
	j ++;
	...

Viết chú thích cho chương trình

Trong khi lập trình cần ghi chú thích cho các đoạn mã trong chương trình.

Việc chú thích giúp hiểu rõ ràng và tương minh hơn, giúp dễ hiểu khi quay lại sửa hoặc cải tiến chương trình.

Giúp chia sẻ và phát triển chương trình theo nhóm làm việc.

Viết chú thích cho chương trình

Đối với mỗi hàm và đặc biệt là các hàm quan trọng, phức tạp, cần xác định và ghi chú thích về những vấn đề cơ bản sau :

+ Mục đích của hàm là gì ?
+ Biến đầu vào của hàm (tham số) là gì ?
+ Các điều kiện ràng buộc của các biến đầu vào (nếu có) ?
+ Kết quả trả về của hàm là gì ?
+ Các ràng buộc của kết quả trả về (nếu có).
+ Ý tưởng giải thuật các thao tác trong hàm.

Viết chú thích cho chương trình

Không phải lệnh nào cũng chú thích, việc chú thích tràn lan không có ý nghĩa, còn làm cho chương trình khó nhìn hơn.

- Không nên chú thích câu lệnh đơn giản này

```
//Nếu nhiệt độ vượt quá mức qui định thì phải cảnh báo
if (nhietDo > nhietDoCB)
    cout<<" Nhiet do vuot muc qui dinh" ;
//i là biến chạy trong vòng lặp for để xác định các chỉ số phần tử mảng a.
for (int i = 0 ; i<n ; i++) cout<< a[i] ;
```

Biểu thức điều kiện

– Nên viết biểu thức điều kiện mang tính tự nhiên : biểu thức nên viết dưới dạng khẳng định, việc viết biểu thức dạng phủ định sẽ làm khó hiểu.

– Viết các lệnh rõ ràng, tối ưu sự thực thi mã nguồn.

Không nên	Nên
if (!(i < a) || !(i >= b))	if ((i >= a) || (i < b))

Chia nhỏ chương trình

Nên sử dụng chiến lược " chia để trị", nghĩa là chương trình được chia nhỏ thành các chương trình con.

Việc chia nhỏ thành các chương trình con làm tăng tính modun của chương trình và người lập trình có thể tái sử dụng mã code.

Lưu ý: độ dài mỗi chương trình con không nên vượt quá một trang màn hình để lập trình viên có thể kiểm soát tốt hoạt động của chương trình con đó.

Biến toàn cục

Hạn chế dùng biến toàn cục.

Nên hạn chế sử dụng biến toàn cục.

Khi nhiều hàm cùng sử dụng một biến toàn cục, việc thay đổi giá trị biến toàn cục của một hàm dẫn đến những thay đổi không mong muốn ở các hàm khác.

Biến toàn cục sẽ làm cho các hàm trong chương trình không độc lập với nhau.

Phân tích thuật giải

Khi giải một bài toán, có nhiều cách giải (giải thuật) khác nhau, vì vậy cần đánh giá các giải thuật đó để chọn giải thuật tốt nhất.

Thông thường căn cứ vào các tiêu chuẩn sau:

- Giải thuật đúng đắn.
- Giải thuật đơn giản.
- Giải thuật thực hiện nhanh.

Phân tích thuật giải

Kiểm tra tính đúng đắn của giải thuật:

Cài đặt giải thuật và thực hiện với một số bộ dữ liệu mẫu, lấy kết quả so sánh với kết quả cần đạt được.

- Cách làm này không chắc chắn vì giải thuật có thể đúng với các bộ dữ liệu đã thử nhưng lại sai với bộ dữ liệu nào đó.
- Cách làm này giúp phát hiện giải thuật sai, chưa chứng minh được là đúng.

Tính đúng đắn của giải thuật cần được chứng minh bằng toán học. Điều không đơn giản với bài toán phức tạp.

Phân tích thuật giải

Chương trình sử dụng ít lần thì yêu cầu thứ 2 quan trọng nhất. Cần giải thuật để chương trình có kết quả nhanh chóng, thời gian thực hiện chương trình không được quan tâm vì chương trình chỉ sử dụng vài lần.

Chương trình sử dụng nhiều lần thì cần tiết kiệm thời gian thực hiện chương trình, đặc biệt với chương trình cần nhập dữ liệu lớn, khi đó yêu cầu thứ 3 sẽ được xem xét, còn gọi là hiệu quả thời gian thực hiện của giải thuật.

Thời gian thực hiện

Để xác định hiệu quả thời gian thực hiện giải thuật: lập trình và đo thời gian thực hiện trên máy tính với tập dữ liệu vào.

Thời gian thực hiện phụ thuộc:

- Giải thuật
- Tập lệnh của máy tính,
- Chất lượng của máy tính
- Kĩ xảo của người lập trình.

Sự phức tạp của thời gian thực hiện được chấp nhận như một sự đo lường sự thực thi của giải thuật.

Thời gian thực hiện

Thời gian thực hiện chương trình là hàm T của kích thước dữ liệu vào, ký hiệu T(n) với n là kích thước (độ lớn) của dữ liệu vào.

Ví dụ:

Chương trình tính tổng n có thời gian thực hiện là

T(n) = c.n với c là hằng số. Thời gian thực hiện là một hàm không âm, tức là T(n) ≥ 0 ∀ n ≥ 0.

T(n) là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thước n, tức là: T(n) là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có kích thước n.

Tỷ suất tăng

Hàm không âm T(n) có tỷ suất tăng (growth rate) *f*(n) nếu tồn tại hằng số C và N_0 sao cho T(n) ≤ C*f*(n) với mọi n ≥ N_0.

Có thể chứng minh rằng "*Cho một hàm không âm T(n) bất kỳ, ta luôn tìm được tỷ suất tăng f(n) của nó*".

Ví dụ:

$$T(n) = \begin{cases} 1 \ với \ n = 0 \\ 4 \ với \ n = 1 \\ (n+1)^2 \ với \ các \ n > 1 \end{cases}$$

Đặt N_0 = 1 và C = 4 thì với mọi n ≥1, dễ dàng chứng minh được rằng: T(n) = $(n+1)^2 \le 4n^2$ với mọi n ≥ 1, tức là tỷ suất tăng của T(n) là n^2 .

Độ phức tạp của thuật toán

Cho một hàm T(n), T(n) gọi là có độ phức tạp f(n) nếu tồn tại các hằng C, N_0 sao cho T(n) ≤ Cf(n) với mọi n ≥ N_0 (tức là T(n) có tỷ suất tăng là f(n)) và kí hiệu T(n) là O(f(n)) (đọc là "ô của f(n)").

Ví dụ:

T(n)= $3n^3 + 2n^2$ có tỷ suất tăng là n^3 nên T(n)= $3n^3 + 2n^2$ là O(n^3).

Lưu ý: O(C.f(n)) = O(f(n)) với C là hằng số. Đặc biệt O(C)=O(1).

Độ phức tạp của thuật toán

Độ phức tạp của giải thuật là một hàm chặn trên của hàm thời gian. Vì C là hằng số trong hàm chặn trên không có ý nghĩa nên có thể bỏ qua.

Hàm thể hiện độ phức tạp có các dạng thường gặp: $\log_2 n$, n, $n\log_2 n$, n^2, n^3, 2^n, n!, n^n. Ba hàm cuối gọi là dạng hàm mũ, các hàm khác gọi là hàm đa thức.

Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được tức là có thể cài đặt để thực hiện, còn các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật.

Độ phức tạp của thuật toán

Ký hiệu $\log_2 n$ thường có mặt trong độ phức tạp nên sẽ dùng **logn** thay thế **$\log_2 n$** cho gọn trong cách viết.

Khi nói đến độ phức tạp của giải thuật là nói đến hiệu quả thời gian thực hiện của chương trình nên, có thể xem việc xác định thời gian thực hiện của chương trình chính là xác định độ phức tạp của giải thuật.

Cách tính độ phức tạp

Qui tắc cộng

T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và T1(n)=O(f(n)), T2(n)=O(g(n)) thì thời gian thực hiện của đoạn hai chương trình đó nối tiếp nhau là:
T(n)=O(max(f(n),g(n)))

Ví dụ: Lệnh gán x:=15 tốn một hằng thời gian hay O(1), Lệnh đọc dữ liệu READ(x) tốn một hằng thời gian hay O(1).Vậy thời gian thực hiện cả hai lệnh trên nối tiếp nhau là O(max(1,1))=O(1).

Cách tính độ phức tạp

Qui tắc nhân

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1và P2 và T1(n) = O(f(n)), T2(n) = O(g(n)) thì thời gian thực hiện của đoạn hai đoạn chương trình đó lồng nhau là T(n) = O(f(n).g(n)).

Qui tắc tổng quát

- Thời gian thực hiện mỗi lệnh gán, nhập/xuất là O(1).
- Thời gian thực hiện một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng, là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.
- Thời gian thực hiện IF là thời gian kiểm tra điều kiện và thời gian lớn nhất thực hiện khối lệnh sau IF hoặc sau ELSE. Thường thời gian kiểm tra điều kiện là O(1).
-Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

Qui tắc tổng quát

Tính thời gian thực hiện của giải thuật bubbleSort (Nổi bọt) để sắp xếp mảng 1 chiều a tăng dần.

```
 void BubbleSort (int a[], int n)
{
    for(int i = 0; i < n-1;      i++)          (1)
        for(int j = n-1; j > i; j--)           (2)
            if (a[j-1] > a[j] )                (3)
            {
                int t = a[j-1];                (4)
                a[j-1] = a[j];                 (5)
                a[j] = t;                      (6)
             }
}
```

Qui tắc tổng quát

Toàn bộ chương trình gồm một lệnh lặp (1), lồng trong lệnh (1) là lệnh (2), lồng trong lệnh (2) là lệnh (3) và lồng trong lệnh (3) là 3 lệnh nối tiếp nhau: (4), (5), (6). Tiến hành tính độ phức tạp theo thứ tự từ trong ra.

- Cả ba lệnh gán (4), (5), (6) đều tốn O(1) thời gian, việc so sánh a[j-1] > a[j] cũng tốn O(1) thời gian, do đó lệnh (3) tốn O(1) thời gian.
- Vòng lặp (2) thực hiện (n-i) lần, mỗi lần O(1) do đó vòng lặp (2) tốn O((n-i).1) = O(n-i). Vòng lặp (1) có i chạy từ 1 đến n-1 nên thời gian thực hiện của vòng lặp (1) và cũng là độ phức tạp của giải thuật là $T(n) = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2)$

Qui tắc tổng quát

Chú ý: Trong trường hợp vòng lặp không xác định được số lần lặp thì phải lấy số lần lặp trong trường hợp xấu nhất.

Qui tắc tổng quát

Xét giải thuật tìm kiếm tuyến tính (Linear Search).

```
 bool LinearSearch (int a[], int n, int x)
{
    for (int i = 0; i < n; i++)          (1)
        if(a[i] == x)                    (2)
            return true;                 (3)
    return false;                        (4)
}
```

Qui tắc tổng quát

Các lệnh (1) và (4) nối tiếp nhau, do đó độ phức tạp của hàm chính là độ phức tạp lớn nhất trong 2 lệnh này. Lệnh (4) có độ phức tạp O(1) do đó độ phức tạp của hàm Linear Search chính là độ phức tạp của lệnh (1). Lồng trong lệnh (1) là lệnh (2), lệnh (2) có lệnh lồng là lệnh (3). Lệnh (2) có độ phức tạp là O(1). Trong trường hợp xấu nhất (tất cả các phần tử của mảng a đều khác x) thì vòng lặp (1) thực hiện n lần, vậy ta có T(n) = O(1.n) = O(n).

Ma trận

Tạo ma trận trong Python bằng cách sử dụng **nested list** (danh sách lồng nhau) và thư viện **NumPy**.
Ma trận có cấu trúc dữ liệu hai chiều, trong đó các số được sắp xếp thành các hàng và cột. Ví dụ:

4 Columns

$$\begin{bmatrix} 2 & -5 & -11 & 0 \\ -9 & 4 & 6 & 13 \\ 4 & 7 & 2 & -2 \end{bmatrix}$$

3 rows

Đây là ma trận 3 x 4

Ma trận

Nested list là dạng danh sách lồng nhau, nghĩa là một list là phần tử của một list khác. Ví dụ:
A = [1, 4, 5, [8, 9]]
Nếu in A[3] output là [8, 9].
Nested list thường được dùng để trình bày ma trận trong Python. Ví dụ:

```
A = [[1, 4, 5],
     [-5, 8, 9]]
```

Danh sách này là một ma trận gồm 2 hàng và 3 cột.

Nested list

Xuất phần tử từ ma trận, chọn hàng của ma trận theo cách thông thường hoặc dùng chỉ số kép, chỉ số thứ nhất chọn hàng, chỉ số thứ hai chọn cột. Ví dụ:

```
>>> A = [[ 1, 4,  5, 12],
         [-5, 8,  9,  0],
         [-6, 7, 11, 19]]
    print("A =", A)
    print("A[1] =", A[1]) # Hàng thứ 2 của ma trận
    print("A[1][2] =", A[1][2]) # Phần tử thứ 3 hàng thứ 2
    print("A[0][-1] =", A[0][-1]) # Phần tử cuối của hàng thứ 1
    column = [];
```

Nested list

```
>>> for row in A:
        column.append(row[2])
        print("Cột thứ 3 =", column)
```

Output được trả về là:

```
>>> A = [[1, 4, 5, 12], [-5, 8, 9, 0], [-6, 7, 11, 19]]
    A[1] = [-5, 8, 9, 0]
    A[1][2] = 9
    A[0][-1] = 12
   Cột thứ 3 = [5, 9, 11]
```

Dùng nested list để biểu diễn ma trận là cách thông dụng và thường dùng trong các phép tính đơn giản. Cách hay hơn là sử dụng thư viện NumPy.

Sử dụng NumPy cho ma trận

NumPy là thư viện Python phục vụ cho việc tính toán, hỗ trợ kiểu dữ liệu đa chiều giúp cho việc tính toán, lập trình, làm việc với các hệ cơ sở dữ liệu thuận tiện. Để tạo một ma trận sử dụng ndarray (viết tắt là array) của NumPy.

Array là đối tượng mảng đa chiều mọi phần tử đều cùng 1 kiểu. Ví dụ:

```
>>> import numpy as np
    a = np.array([1, 2, 3])
    print(a)
    # Output: [1, 2, 3]
    print(type(a))
    # Output: <class 'numpy.ndarray'>
```

Install các thư viện Python

Lưu ý: trong Windows, mở cửa sổ CMD (Command Prompt): lệnh kiểm tra version

>python –version # >pip -v

Lệnh cài thư viện: pip install, kiểm tra thư viện: pip list

>pip install numpy # thư viện toán học

>pip install matplotlib # thư viện vẽ đồ thị

>pip install scipy # thư viện toán đại số, xử lý ảnh . . .

>pip install opencv-python # thị giác máy tính

>pip install pandas # bổ sung cho numpy và scipy

>pip install scikit-learn # máy học

>pip install keras # deep learning/ tensorflow(438mb)

>pip instal tk # giao diện # pip3 install ipython

NumPy

Tạo (array) mảng số nguyên, số thực, số phức (integer, float, complex)

```
>>> import numpy as np
      A = np.array([[1, 2, 3], [3, 4, 5]])
      print(A)
      A = np.array([[1.1, 2, 3], [3, 4, 5]]) # mảng số thực
      print(A)
      A = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex)
# mảng số phức
     print(A)
```

NumPy

Kết quả:

```
>>> [[1 2 3]
     [3 4 5]]
    [[1.1 2. 3. ]
     [3. 4. 5. ]]
    [[1.+0.j 2.+0.j 3.+0.j]
     [3.+0.j 4.+0.j 5.+0.j]]
```

NumPy

Mảng giá trị mặc định (0 và 1)

```
>>> import numpy as np
    # Mọi phần tử đều là 0
    A = np.zeros( (2, 3) )
    print(A)
    # Output:
    [[0. 0. 0.]
     [0. 0. 0.]]
    # Mọi phần tử đều là 1
    B = np.ones( (1, 5) )
    print(B)
    # Output: [[1 1 1 1 1]]
```

NumPy

Sử dụng arange() và shape()

```
>>> import numpy as np
    A = np.arange(4)
    print('A =', A)
    B = np.arange(12).reshape(2, 6)
    print('B =', B)
>>> # Output:
    A = [0 1 2 3]
    B = [[ 0 1 2 3  4  5]
         [ 6 7 8 9 10 11]]
```

Các phép toán với ma trận

3 phép toán cơ bản thường được sử dụng là cộng ma trận, nhân ma trận và chuyển vị ma trận.
Các phép toán sử dụng nested list và thư viện NumPy.

Cộng 2 ma trận

cộng từng phần tử tương ứng của 2 ma trận cùng cấp với nhau

```
>>> import numpy as np
    A = np.array([[2, 4], [5, -6]])
    B = np.array([[9, -3], [3, 6]])
    C = A + B
    print(C)
Output: [[11  1]
         [ 8  0]]
```

Các phép toán với ma trận

Nhân 2 ma trận
tổng của tích từng phần tử của hàng với cột tương ứng.
Chú ý: chỉ nhân ma trận khi số cột của ma trận A bằng với số hàng của ma trận B. Ví dụ ma trận [A]mp nhân [B]pn, kết quả sẽ là ma trận [AB]mn.

```
>>> import numpy as np
    A = np.array([[3, 6, 7], [5, -3, 0]])
    B = np.array([[1, 1], [2, 1], [3, -3]])
    C = A.dot(B)
    print(C)
Output: [[ 36 -12]
         [ -1 2]]
```

Các phép toán với ma trận

Chuyển vị ma trận
phép biến cột thành dòng và dòng thành cột của ma trận.

```
>>> import numpy as np
    A = np.array([[1, 1], [2, 1], [3, -3]])
    print(A.transpose())
#Output:
    [[ 1 2 3]
    [ 1 1 -3]]
```

Xuất phần tử của ma trận

Xuất phần tử bằng NumPy tương tự như list. Với mảng một chiều:

```
>>> import numpy as np
      A = np.array([12, 14, 16, 18, 20])
      print("A[0] =", A[0]) # phần tử đầu tiên
      print("A[2] =", A[2]) # phần tử thứ 3
      print("A[-1] =", A[-1]) # phần tử cuối cùng
Output:
      A[0] = 12
      A[2] = 16
      A[-1] = 20
```

Xuất phần tử của ma trận

mảng hai chiều:

```
>>> import numpy as np
      A = np.array([[1, 4, 5, 12],
                    [-5, 8, 9, 0],
                    [-6, 7, 11, 19]])
      print("A[0][0] =", A[0][0]) # Phần tử đầu tiên hàng 1
      print("A[1][2] =", A[1][2]) # Phần tử thứ 3 hàng thứ 2
      print("A[-1][-1] =", A[-1][-1]) # Phần tử cuối của hàng
cuối
# Output: A[0][0] = 1
            A[1][2] = 9
            A[-1][-1] = 19
```

Xuất các dòng của ma trận

```
>>> import numpy as np
    A = np.array([[1, 4, 5, 12],
                  [-2, 8, 6, 14],
                  [-1, 5, 10, 22]])
    print("A[0] =", A[0]) # Dòng đầu tiên
    print("A[2] =", A[2]) # Dòng thứ 3
    print("A[-1] =", A[-1]) # Dòng cuối (dòng thứ 3)
Output:
    A[0] = [1, 4, 5, 12]
    A[2] = [-1, 5, 10, 22]
    A[-1] = [-1, 5, 10, 22]
```

Xuất các cột của ma trận

```
>>> import numpy as np
    A = np.array([[1, 4, 5, 12],
                  [-2, 8, 6, 14],
                  [-1, 5, 10, 22]])
    print("A[:,0] =",A[:,0]) # Cột đầu tiên
    print("A[:,3] =", A[:,3]) # Cột thứ 4
    print("A[:,-1] =", A[:,-1]) # Cột cuối (Cột thứ 4)
Output:
    A[:,0] = [ 1 -2 -1]
    A[:,3] = [12 14 22]
    A[:,-1] = [12 14 22]
```

Cắt Ma trận

Cắt mảng một chiều trong NumPy tương tự như list.

```
>>> import numpy as np
    A = np.array([1, 3, 5, 7, 9, 7, 5])
    # Cắt phần tử từ 3 đến 5
    print(A[2:5])  # Output: [5, 7, 9]
    # Cắt phần tử từ 1 đến 4
    print(A[:-5])   # Output: [1, 3]
    # Phần tử thứ 6 trở đi
    print(A[5:])    # Output:[7, 5]
    # Cắt cả mảng
    print(A[:])     # Output:[1, 3, 5, 7, 9, 7, 5]
    # Đổi chiều mảng
    print(A[::-1])  # Output:[5, 7, 9, 7, 5, 3, 1]
```

Cắt Ma trận

```
>>> import numpy as np
    A = np.array([[1, 4, 5, 12, 14],
                  [-5, 8, 9, 0, 17],
                  [-6, 7, 11, 19, 21]])
    print(A[:2, :4]) # 2 hàng, 4 cột
# Output:
    [[ 1 4 5 12]
     [-5 8 9 0]]
    print(A[:1,]) # hàng đầu tiên, tất cả cột
# Output:  [[ 1 4 5 12 14]]
    print(A[:,2]) # tất cả các hàng, cột 2
# Output: [ 5 9 11]
```

NumPy

Sử dụng thư viện NumPy thay vì nested list làm cho các phép toán với ma trận dễ dàng hơn rất nhiều. Nên tìm hiểu và học cách sử dụng thư viện NumPy thật kĩ, đặc biệt khi sử dụng Python để áp dụng cho việc tính toán khoa học hay phân tích dữ liệu.

Ma trận

- Nhân ma trận: hàm **numpy.dot()**

```
>>> import numpy.matlib
    import numpy as np
    a = np.array([[1,2],[3,4]])
    b = np.array([[11,12],[13,14]])
    np.dot(a,b)
```

Kết quả :

```
>>> [[37 40]
     [85 92]]
```

Ma trận

Determinant (Định thức): Hàm numpy.linalg.det ()

```
>>> import numpy as np
    a = np.array([[1,2], [3,4]])
    print np.linalg.det(a)
```

Kết quả :

```
>>> -2.0
>>> import numpy as np
    b = np.array([[6,1,1], [4, -2, 5], [2,8,7]])
    # print b
    print np.linalg.det(b)
    # print 6*(-2*7 - 5*8) - 1*(4*7 - 5*2) + 1*(4*8 - -2*2)
```

Kết quả ;

```
>>> -306.0
```

Ma trận

Ma trận nghịch đảo: hàm numpy.linalg.inv ()

```
>>> import numpy as np
    x = np.array([[1,2],[3,4]])
    y = np.linalg.inv(x)
    print x
    print y
    print np.dot(x,y)
```

Kết quả :

```
>>> [[1 2]
    [3 4]]
    [[-2. 1. ]
    [1.5 -0.5]]
    [[ 1.00000000e+00 1.11022302e-16]
    [ 0.00000000e+00 1.00000000e+00]]
```

Đồ thị

Nhập thư viện matplotlib

```
>>>import matplotlib.pyplot as plt
    plt.plot([1, 2, 3, 4, 10]) # matplotlib inline
    plt.show()
```

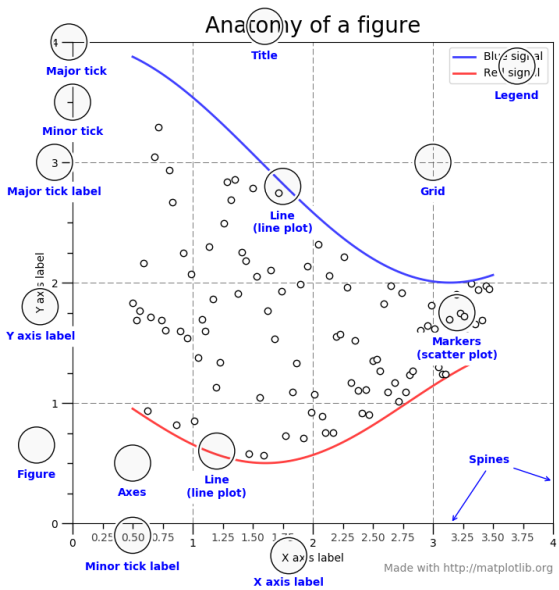
Phương thức plot() có 3 tham số:

```
>>> plot(x, y, format)
```

- Tham số x là danh sách các tọa độ trục x
- Tham số y là danh sách các tọa độ trục y
- format định dạng đồ thị

Khi đưa vào một List thì mặc định đó là danh sách tọa độ trục y và định dạng mặc định là vẽ đường thẳng giữa các điểm. Ví dụ: plt.plot([1, 2, 3, 4, 10])

Đồ thị



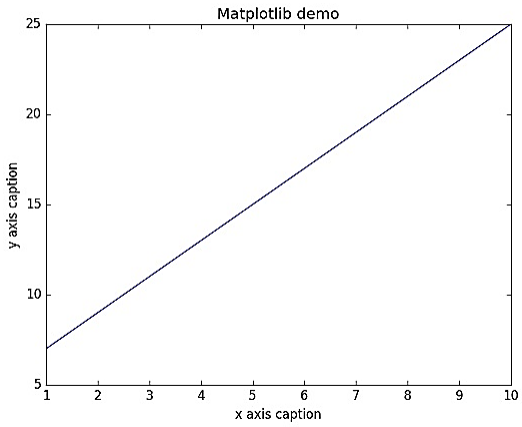
Đồ thị

Vẽ dữ liệu 2D: hàm **pyplot ()** trong thư viện matplotlib

```
>>> import numpy as np
    from matplotlib import pyplot as plt
    x = np.arange(1,11)
    y = 2 * x + 5
    plt.title("Matplotlib demo")
    plt.xlabel("x axis caption")
    plt.ylabel("y axis caption")
    plt.plot(x,y)
    plt.show()
```

Đồ thị

Kết quả



Đồ thị

Các kiểu hiển thị khi thêm ký tự định dạng vào hàm plot (). Các ký tự định dạng được sử dụng:

1. **'-'** : Kiểu đường liền nét
2. '-': Kiểu đường đứt nét
3. '-.' : Kiểu đường dấu gạch ngang
4. ':': Kiểu đường chấm
5. '.' : Điểm đánh dấu
6. ',': Điểm đánh dấu pixel
7. 'o': Điểm đánh dấu vòng tròn
8. 'v': Điểm đánh dấu tam giác_dưới
9. **'^'**: Điểm đánh dấu tam giác_trên
10. **'<' :** Điểm đánh dấu tam giác_trái
11. **'>' :** Điểm đánh dấu tam giác_phải

Đồ thị

Bảng chữ viết tắt màu:

Ký tự	Màu
‘b’	Blue
‘g’	Green
‘r’	Red
‘c’	Cyan
‘m’	Magenta
‘y’	Yellow
‘k’	Black
‘w’	White

Hiển thị vòng tròn đại diện cho các điểm, có thể sử dụng “ob” làm chuỗi định dạng trong hàm plot ().

Đồ thị

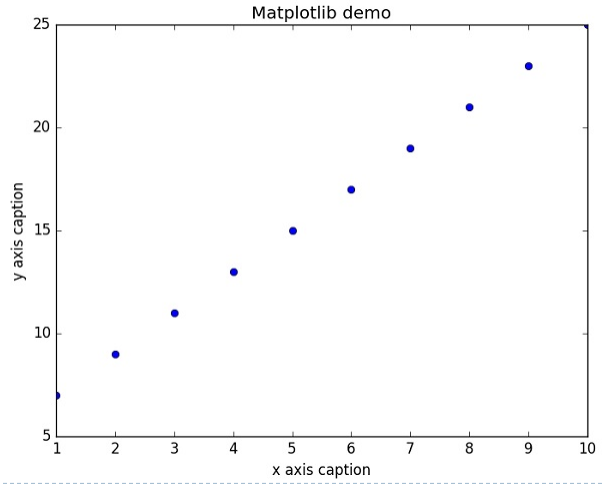
Một số định dạng thường dùng:
'r*--' các điểm hình ngôi sao màu đỏ, đường nối các điểm dạng --.
'bD-.' các điểm hình kim cương màu xanh dương, đường nối các điểm dạng -.
'g^-' các điểm hình tam giác hướng lên màu xanh lá, đường nối các điểm dạng -.
Nếu không muốn các điểm nối với nhau, có thể bỏ định dạng đường thẳng đi, ví dụ 'go-' sẽ thành 'go'

Đồ thị

```
# Đồ thị dạng điểm màu xanh
>>> import numpy as np
    from matplotlib import pyplot as plt
    x = np.arange(1,11)
    y = 2 * x + 5
    plt.title("Matplotlib demo")
    plt.xlabel("x axis caption")
    plt.ylabel("y axis caption")
    plt.plot(x,y,"ob")
    plt.show()
```

Đồ thị

Kết quả:



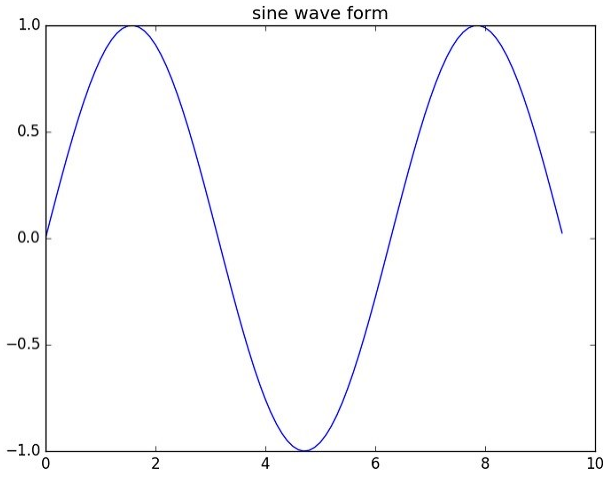
Đồ thị

Đồ thị hình sin

```
>>> import numpy as np
    import matplotlib.pyplot as plt
    # Compute the x and y coordinates for points on a
sine curve
    x = np.arange(0, 3 * np.pi, 0.1)
    y = np.sin(x)
    plt.title("sine wave form")
    # Plot the points using matplotlib
    plt.plot(x, y)
    plt.show()
```

Đồ thị

Kết quả:



Đồ thị

subplot()

Hàm subplot () cho phép vẽ đồ thị khác nhau trong cùng một hình.

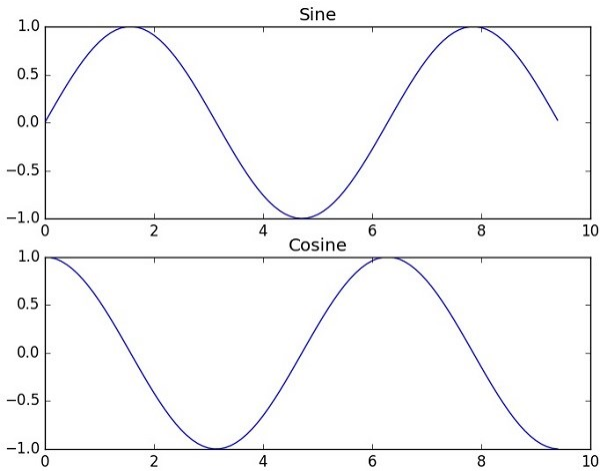
```
>>> import numpy as np
    import matplotlib.pyplot as plt
    # Compute the x and y coordinates for points on
sine and cosine curves
    x = np.arange(0, 3 * np.pi, 0.1)
    y_sin = np.sin(x)
    y_cos = np.cos(x)
```

Đồ thị

```
# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
>>> plt.subplot(2, 1, 1)
        # Make the first plot
        plt.plot(x, y_sin)
        plt.title('Sine')
        # Set the second subplot as active, and make the
second plot.
        plt.subplot(2, 1, 2)
        plt.plot(x, y_cos)
        plt.title('Cosine')
        # Show the figure.
        plt.show()
```

Đồ thị

Kết quả:



Đồ thị

Thêm nhãn cho từng tập điểm với tham số thứ 4 trong plot().
Hiển thị ghi chú các thành phần trong đồ thị với phương thức legend().
Hiển thị nhãn các trục tọa độ x, y với xlabel() và ylabel().

```
>>>plt.plot([0,1, 2, 3, 4], [1, 2, 3, 4, 10], 'go-', label= 'Python')
    plt.plot([0, 1, 2, 3, 4], [10, 4, 3, 2, 1], 'ro-', label= 'C#')
    plt.plot([2.5, 2.5, 2.5, 1.5, 0.5], [1, 3, 5, 7, 10], 'bo-', label= 'Java')
    plt.title('Vẽ đồ thị trong Python với Matplotlib')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend(loc='best')
    plt.show()
```

Đồ thị

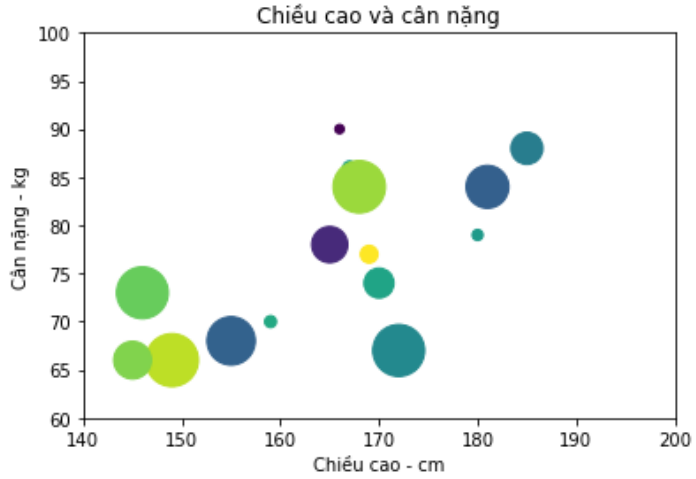
Sự khác biệt giữa plot() và scatter():
- plot() không có khả năng thay đổi màu và kích thước điểm trong tập hợp điểm ban đầu nhưng scatter() lại có thể.
- plot() có thể vẽ các đường nối hai điểm liên tiếp, scatter() thì không.

Ví dụ dưới đây vẽ ra các điểm trên đồ thị với dữ liệu về chiều cao và cân nặng, mỗi điểm có màu ngẫu nhiên và có kích thước cũng ngẫu nhiên.

Đồ thị

```
>>> height = np.array([167, 170, 149, 165, 155, 180,
166, 146, 159, 185, 145, 168, 172, 181, 169])
      weight = np.array([86, 74, 66, 78, 68, 79, 90, 73,
70, 88, 66, 84, 67, 84, 77])
      colors = np.random.rand(15)
      area = (30 * * np.random.rand(15)) ** 2
      plt.xlim(140, 200)
      plt.ylim(60,100)
      plt.scatter(height,weight,s =area,c =colors)
      plt.title("Chiều cao và cân nặng")
      plt.xlabel("Chiều cao - cm")
      plt.ylabel("Cân nặng - kg")
      plt.show()
```

Đồ thị



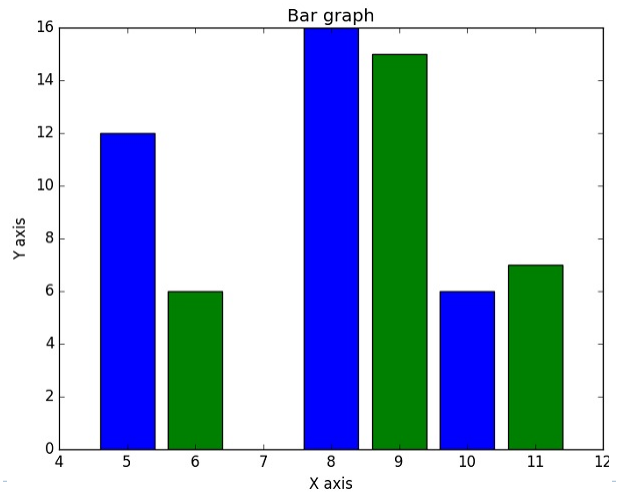
Đồ thị

bar() pyplot submodule cung cấp hàm bar () để tạo đồ thị hình cột.

```
>>> from matplotlib import pyplot as plt
    x = [5,8,10]
    y = [12,16,6]
    x2 = [6,9,11]
    y2 = [6,15,7]
    plt.bar(x, y, align = 'center')
    plt.bar(x2, y2, color = 'g', align = 'center')
    plt.title('Bar graph')
    plt.ylabel('Y axis')
    plt.xlabel('X axis')
    plt.show()
```

Đồ thị

Kết quả:



Đồ thị

Hàm numpy.histogram() có hai tham số là mảng và bins. Các phần tử kế tiếp nhau trong mảng đóng vai trò là ranh giới của mỗi bins.

```
>>> import numpy as np
    a = np.array([22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5, 79, 31, 27])
    np.histogram(a,bins = [0,20,40,60,80,100])
    hist,bins = np.histogram(a,bins = [0, 20, 40, 60, 80, 100])
    print hist
    print bins
```

Kết quả: [3 4 5 2 1]
[0 20 40 60 80 100]

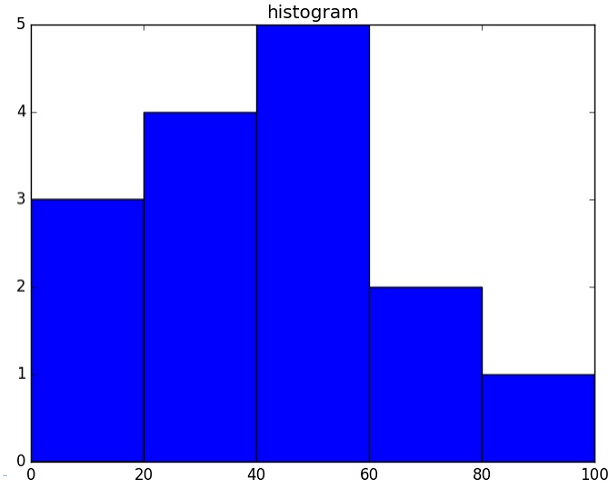
Đồ thị

Hàm plt() của submodule pyplot nhận mảng chứa dữ liệu và mảng bin làm tham số và chuyển đổi thành biểu đồ.

```
>>> from matplotlib import pyplot as plt
    import numpy as np
    a = np.array([22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5, 79, 31, 27])
    plt.hist(a, bins = [0,20,40,60,80,100])
    plt.title("histogram")
    plt.show()
```

Đồ thị

Kết quả:



Lưu đồ thị

Lưu biểu đồ thành .png, cú pháp

```
>>> plt.savefig('sampleFileName.png')
```

lưu ở chế độ toàn màn hình:

```
>>> figure = plt.gcf() # get current figure
    figure.set_size_inches(32, 18) # set figure's size manually to your full screen (32x18)
    plt.savefig('filename.png', bbox_inches='tight') # bbox_inches removes extra white spaces
```

Thêm dpi (Độ phân giải tính bằng điểm trên inch)

```
>>> plt.savefig('filename.png', bbox_inches='tight', dpi=100)
```

Lưu đồ thị

```
>>> #Show full screen
    mng = plt.get_current_fig_manager()
    mng.full_screen_toggle()
```

Đầy đủ

```
>>> fig = plt.figure()
    fig.imshow(image)
    ...
    plt.figure(fig.number)
    mng = plt.get_current_fig_manager()
    mng.full_screen_toggle()
    fig.show()
    fig.savefig('figure.png')
    mng.full_screen_toggle()
```

Giao diện GUI (Graphical User Interface)

Lập trình giao diện GUI bằng Tkinter trong Python

```
>>> #Thêm thư viện tkinter
    from tkinter import *
    #Tạo một cửa sổ mới
    window = Tk()
    #Thêm tiêu đề cho cửa sổ
    window.title('Welcome to Lac Hong')
    #Đặt kích thước của cửa sổ
    window.geometry('350x200')
    #Mainloop để hiển thị cửa sổ
    window.mainloop()
```

Label

Dùng để hiển thị văn bản hoặc hình ảnh.

Cú pháp

>>> w = Label(master, option, …)

Tham số

- master - Tên của cửa sổ chứa nhãn.
- options - Đây là danh sách một số tùy chọn thường dùng cho nhãn, tùy chọn này có thể sử dụng cặp từ khóa và giá trị.

Label

TT	Tùy chọn và giải thích
1	anchor Tùy chọn này dùng để căn lề của văn bản, mặc định anchor=CENTER, nghĩa là văn bản sẽ căn giữa.
2	bg Màu nền của nhãn (label).
3	bitmap Đặt tùy chọn này bằng một bitmap hoặc đối tượng hình ảnh và nhãn sẽ hiển thị đồ họa đó.
4	bd Quy định kích thước của đường viền (border) bao quanh nhãn, mặc định là 2 pixels.
5	cursor Dùng để thiết lập loại con trỏ chuột (arrow, dot,…).
6	font Nếu đang hiển thị văn bản trong nhãn này (với tùy chọn văn bản hoặc văn bản có thể thay đổi, tùy chọn phông chữ chỉ định loại phông chữ mà văn bản sẽ được hiển thị.

Label

7	fg Nếu đang hiển thị văn bản hoặc một bitmap trong nhãn này, tùy chọn này chỉ định màu của văn bản. Nếu đang hiển thị một bitmap, đây là màu sẽ xuất hiện tại vị trí của các bit 1 trong bitmap.
8	height Quy định độ cao của nhãn.
9	image Dùng để hiển thị ảnh tĩnh trong nhãn.
10	justify Quy định nhiều dòng của văn bản sẽ canh lề LEFT, CENTER (mặc định) hoặc RIGHT.
11	padx Thiết lập khoảng trống phía trước và sau (padding) của văn bản, mặc định là 1.
12	pady Thiết lập khoảng trống phía trên và phía dưới (padding) của văn bản, mặc định là 1.
13	relief Chỉ định sự xuất hiện của đường viền quanh nhãn, mặc định là FLAT; có nhiều giá trị khác.

Label

14	text Dùng để hiển thị nội dung của một hoặc nhiều dòng văn bản chứa trong nhãn. Sử dụng quy ước (“\n”) để ngắt dòng.
15	textvariable Để bổ sung văn bản được hiển thị trong tiện ích nhãn thành một biến điều khiển của lớp StringVar, hãy đặt tùy chọn này cho biến đó.
16	underline Gạch chân chữ viết cho một số ký tự, bắt đầu đếm từ 0, có thể thiết lập n. Mặc định underline=-1, nghĩa là không gạch chân chữ viết.
17	width Quy định chiều rộng của nhãn (đơn vị số ký tự), nếu tùy chọn này không thiết lập thì nhãn sẽ thay đổi kích thước cho phù hợp với số lượng ký tự của văn bản mà nó chứa.
18	wraplength Thiết lập số lượng ký tự cho mỗi dòng, mặc định bằng 0 nghĩa là dòng sẽ tự động ngắt (chỉ gồm có duy nhất một dòng).

Label

```
>>> from tkinter import *
    window = Tk()
    window.title("Welcome to Lac Hong")
    #Thêm label có nội dung Hello, font chữ
    lbl = Label(window, text="Hello", font=("Arial Bold",
50))
    #Xác định vị trí của label
    lbl.grid(column=0, row=0)
    window.mainloop()
```

Button

Nút bấm hiển thị văn bản hoặc hình ảnh, thiết lập sự kiện khi click vào nút bấm như: tự động gọi hàm.

Cú pháp

```
>>> w = Button( master, option = value, ... )
```

Tham số

- master - Tên của cửa sổ chứa nút bấm.
- options - Đây là danh sách một số tùy chọn thường dùng cho nút bấm, tùy chọn này có thể sử dụng cặp từ khóa và giá trị.

Button

TT	Tùy chọn và giải thích
1	activebackground Màu nền của nút bấm khi con trỏ chuột ở phía trên nút bấm
2	activeforeground Màu chữ của nút bấm khi con trỏ chuột ở phía trên nút bấm
3	bd Độ rộng của đường biên của nút bấm, mặc định là 2.
4	bg Màu nền của nút bấm.
5	command Hàm hoặc phương thức sẽ được gọi khi click vào nút bấm.
6	fg Màu chữ của nút bấm.
7	font Phông chữ của văn bản trên nút bấm.
8	height Độ cao của nút bấm (số dòng) hoặc pixels đối với hình ảnh.

Button

TT	Tùy chọn và giải thích
9	highlightcolor Màu nổi bật khi nút bấm có focus.
10	image Hình ảnh sẽ hiển thị trên nút bấm thay vì văn bản.
11	justify Quy định nhiều dòng của văn bản sẽ canh lề LEFT, CENTER (mặc định) hoặc RIGHT.
12	padx Thiết lập khoảng trống phía trước và sau (padding) của văn bản, mặc định là 1.
13	pady Thiết lập khoảng trống phía trên và phía dưới (padding) của văn bản, mặc định là 1.
14	relief Chỉ định sự xuất hiện của đường viền quanh nút bấm. Một số giá trị như: SUNKEN, RAISED, GROOVE và RIDGE.

Button

15	state Thiết lập thành DISABLED để chuyển sang nút bấm màu xám (bị ẩn) và không phản hồi. Có giá trị là ACTIVE khi con trỏ chuột ở trên nút bấm. Mặc định là NORMAL.
16	underline Mặc định là -1, nghĩa là văn bản trên nút bấm sẽ không gạch chân. Nếu không âm, ký tự văn bản tương ứng sẽ được gạch chân.
17	width Độ rộng của nút bấm là số ký tự (nếu hiển thị văn bản) hoặc pixels (nếu hiển thị hình ảnh).
18	wraplength Thiết lập số lượng ký tự cho mỗi dòng, mặc định bằng 0 nghĩa là dòng sẽ tự động ngắt (chỉ gồm có duy nhất một dòng).

Button

Phương thức

TT	Phương thức và giải thích
1	flash() Làm cho nút nhấp nháy nhiều lần giữa màu đang hoạt động và màu bình thường. Để nút ở trạng thái ban đầu. Bỏ qua nếu nút bị tắt.
2	invoke() Gọi lệnh gọi lại của nút và trả về những gì mà hàm đó trả về. Không có tác dụng nếu nút bị tắt hoặc không có lệnh gọi lại.

Button

```
>>> from tkinter import *
    window = Tk()
    window.title("Welcome to Lac Hong")
    window.geometry('350x200')
    lbl = Label(window, text="Hello")
    lbl.grid(column=0, row=0)
    #Thêm một nút nhấn Click Me
    btn = Button(window, text="Click Me",
bg="orange", fg="red")
    #Thiết lập vị trí của nút nhấn có màu nền và màu
chữ
    btn.grid(column=1, row=0)
    window.mainloop()
```

Button

Xử lý sự kiện khi nhấn nút Click Me

```
>>> from tkinter import *
    window = Tk()
    window.title("Welcome to Lac Hong")
    window.geometry('350x200')
    lbl = Label(window, text="Hello")
    lbl.grid(column=0, row=0)
    #Hàm khi nút được nhấn
    def clicked():
        lbl.configure(text="Button was clicked !!")
    #Gọi hàm clicked khi nút được nhấn
    btn = Button(window, text="Click Me",
command=clicked)
    btn.grid(column=1, row=0)
    window.mainloop()
```

Button

Không nhất thiết tên hàm để xử lý sự kiện click là clicked.

```
>>> # !/usr/bin/python3
    from tkinter import *
    from tkinter import messagebox
    top = Tk()
    top.geometry("100x100")
    def helloCallBack():
        msg = messagebox.showinfo( "Hello Python",
"Hello World")
    B = Button(top, text = "Hello", command =
helloCallBack)
    B.place(x = 50,y = 50)
    top.mainloop()
```

Textbox

```
>>> from tkinter import *
    window = Tk()
    window.title("Welcome to Lac Hong")
    window.geometry('350x200')
    lbl = Label(window, text="Hello")
    lbl.grid(column=0, row=0)
    #Tạo một Textbox
    txt = Entry(window,width=10)
    #Vị trí xuất hiện của Textbox
    txt.grid(column=1, row=0)
    #Đặt vị trí con trỏ tại Textbox
    txt.focus()
```

Textbox

```
>>> #Hàm xử lý khi nút được nhấn
    def clicked():
        res = "Welcome to " + txt.get()
        lbl.configure(text= res)
    btn = Button(window, text="Click Me",
command=clicked)
    btn.grid(column=2, row=0)
    #Để tắt chức năng nhập của Textbox bằng state
    #txt = Entry(window,width=10, state='disabled')
    window.mainloop()
```

Combobox

```
>>> from tkinter import *
    from tkinter.ttk import *
    window = Tk()
    window.title("Welcome to Lac Hong")
    window.geometry('350x200')
    #Tạo hộp chọn Combobox
    combo = Combobox(window)
    #Các giá trị của hộp chọn
    combo['values']= (1, 2, 3, 4, 5, "Text")
    #Thiết lập giá trị được chọn
    combo.current(1) #set the selected item
    combo.grid(column=0, row=0)
    #Lấy giá trị của hộp chọn bằng combo.get()
    window.mainloop()
```

Checkbox

```
>>> from tkinter import *
    from tkinter.ttk import *
    window = Tk()
    window.title("Welcome to Lac Hong")
    window.geometry('350x200')
    #Thiết lập trạng thái của Checkbox
    chk_state = BooleanVar()
    chk_state.set(True) #set check state
    #Tạo Checkbox có trạng thái đã tích chọn
    chk = Checkbutton(window, text='Choose',
var=chk_state)
    chk.grid(column=0, row=0)
    window.mainloop()
```

Radio

```
>>> from tkinter import *
    from tkinter.ttk import *
    window = Tk()
    window.title("Welcome to Lac Hong")
    selected = IntVar()
    rad1 = Radiobutton(window,text='First', value=1,
variable=selected)
    rad2 = Radiobutton(window,text='Second',
value=2, variable=selected)
    rad3 = Radiobutton(window,text='Third', value=3,
variable=selected)
```

Radio

```
>>> def clicked():
        print(selected.get())
    btn = Button(window, text="Click Me",
command=clicked)
    rad1.grid(column=0, row=0)
    rad2.grid(column=1, row=0)
    rad3.grid(column=2, row=0)
    btn.grid(column=3, row=0)
    window.mainloop()
```

ScrolledText

```
>>> from tkinter import *
    from tkinter import scrolledtext
    window = Tk()
    window.title("Welcome to Lac Hong")
    window.geometry('350x200')
    txt = scrolledtext.ScrolledText(window, width=40,
height=10)
    txt.grid(column=0,row=0)
    window.mainloop()
```

Messagebox

Hộp thoại thông báo

```
>>> from tkinter import *
    from tkinter import messagebox
    window = Tk()
    window.title("Welcome to Lac Hong")
    window.geometry('350x200')
    def clicked():
        messagebox.showinfo('Message title',
'Message content')
    btn = Button(window,text='Click here',
command=clicked)
    btn.grid(column=0,row=0)
    window.mainloop
```

Messagebox

Hộp thoại cảnh báo và lỗi

```
>>> #Hộp thoại cảnh báo
    messagebox.showwarning('Message title',
'Message content')
>>> #Hộp thoại báo lỗi
    messagebox.showerror('Message title', 'Message
content')
```

Messagebox

Hộp thoại câu hỏi

```
>>> from tkinter import messagebox
      res = messagebox.askquestion('Message
title','Message content')
      res = messagebox.askyesno('Message
title','Message content')
      res = messagebox.askyesnocancel('Message
title','Message content')
      res = messagebox.askokcancel('Message
title','Message content')
      res = messagebox.askretrycancel('Message
title','Message content')
```

SpinBox

```
>>> from tkinter import *
      window = Tk()
      window.title("Welcome to Lac Hong")
      window.geometry('350x200')
     #Tạo spinbox có chiều rộng 5, giá trị từ 0 đến 100
      spin = Spinbox(window, from_=0, to=100,
width=5)
      spin.grid(column=0,row=0)
      window.mainloop()
```

SpinBox

Liệt kê các giá trị của spinbox

```
>>> #spinbox chỉ gồm có 3 giá trị là 3, 5 và 10
>>> spin = Spinbox(window, values=(3, 5, 10),
width=5)
```

Đặt giá trị mặc định cho spinbox

```
>>> var =IntVar()
      #Đặt giá trị mặc định là 25
      var.set(25)
      spin = Spinbox(window, from_=0, to=100,
width=5, textvariable=var)
```

Chọn tệp và thư mục

```
>>> from tkinter import filedialog
      file = filedialog.askopenfilename(filetypes = (("Text
files","*.txt"),("all files","*.*")))
      dir = filedialog.askdirectory()
```

Chỉ định thư mục ban đầu cho hộp thoại tệp bằng cách chỉ định initialdir

```
>>> from tkinter import filedialog
      from os import path
      file = filedialog.askopenfilename(initialdir=
path.dirname(__file__))
```

Menu

```
>>> from tkinter import *
    from tkinter import Menu
    window = Tk()
    window.title("Welcome to Lac Hong")
    menu = Menu(window)
    new_item = Menu(menu)
    new_item.add_command(label='New')
    new_item.add_separator()
    new_item.add_command(label='Edit')
    menu.add_cascade(label='File', menu=new_item)
    window.config(menu=menu)
    window.mainloop()
```

Menu

Sự kiện khi người dùng click chọn một bảng chọn nào đó:

#Thực hiện hàm clicked khi người dùng chọn New

```
>>> new_item.add_command(label='New',
command=clicked)
```

Tabs

Để tạo một điều khiển tab, có các bước.
- Đầu tiên, tạo một điều khiển tab bằng cách sử dụng lớp Notebook .
- Tạo một tab bằng cách sử dụng Frame lớp.
- Thêm tab đó vào điều khiển tab.
- Đóng gói điều khiển tab để nó hiển thị trong cửa sổ.

Tabs

```
>>> from tkinter import *
    from tkinter import ttk
    window = Tk()
    window.title("Welcome to Lac Hong")
    tab_control = ttk.Notebook(window)
    tab1 = ttk.Frame(tab_control)
    tab_control.add(tab1, text='First')
    tab_control.pack(expand=1, fill='both')
    window.mainloop
```

Tabs

Thêm Widgets vào Notebook
Sau khi tạo các tab, có thể đặt các widget bên trong các tab này bằng cách gán thuộc tính cha cho tab mong muốn.

```
>>> from tkinter import *
    from tkinter import ttk
    window = Tk()
    window.title("Welcome to Lac Hong")
    tab_control = ttk.Notebook(window)
    tab1 = ttk.Frame(tab_control)
    tab2 = ttk.Frame(tab_control)
    tab_control.add(tab1, text='First')
    tab_control.add(tab2, text='Second')
```

Tabs

```
>>> lbl1 = Label(tab1, text= 'label1')
    lbl1.grid(column=0, row=0)
    lbl2 = Label(tab2, text= 'label2')
    lbl2.grid(column=0, row=0)
    tab_control.pack(expand=1, fill='both')
    window.mainloop()
```

Thêm khoảng cách cho các widget (Padding)
Thêm các khoảng cách trước và sau của mỗi tab bằng cách sử dụng các thuộc tính padx và pady .

```
>>> lbl1 = Label(tab1, text= 'label1', padx=5, pady=5)
```