BỘ GIÁO DỤC VÀ ĐÀO TẠO



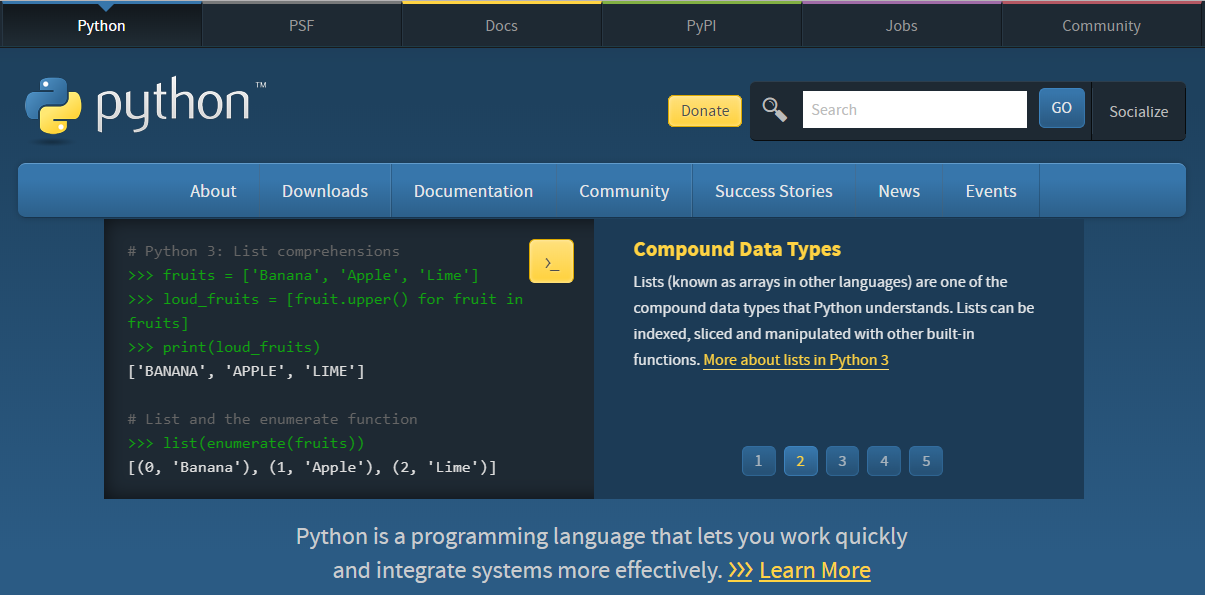
**AN TOÀN HỆ ĐIỀU HÀNH VÀ NGÔN NGỮ LẬP TRÌNH**

# LẬP TRÌNH VỚI PYTHON VÀ ASSEMBLY

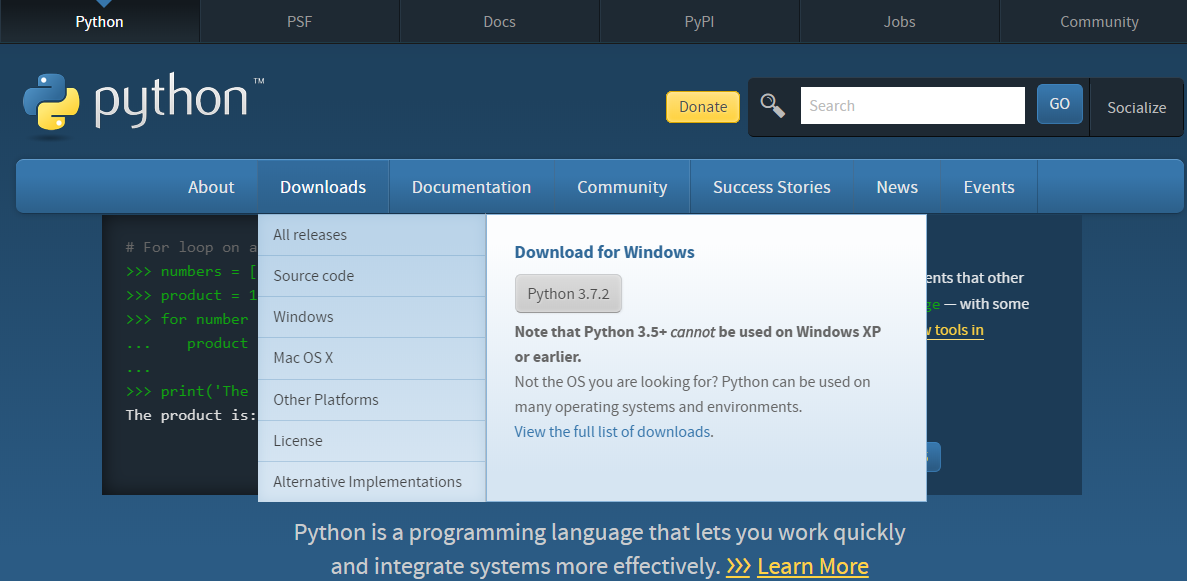
## Lập trình Python

### Cài đặt môi trường Python

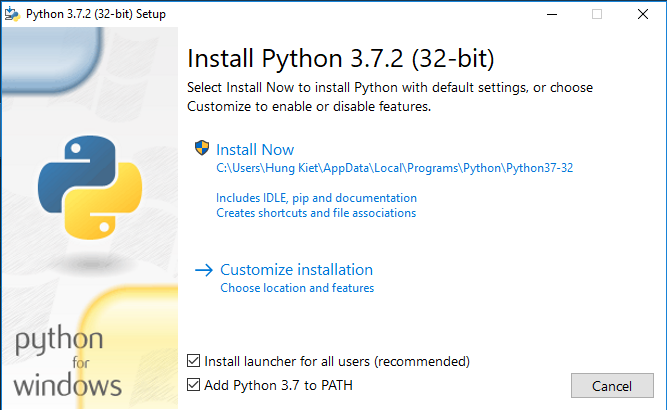
Bước 1: Truy cập vào trang chủ của python: <https://www.python.org/>



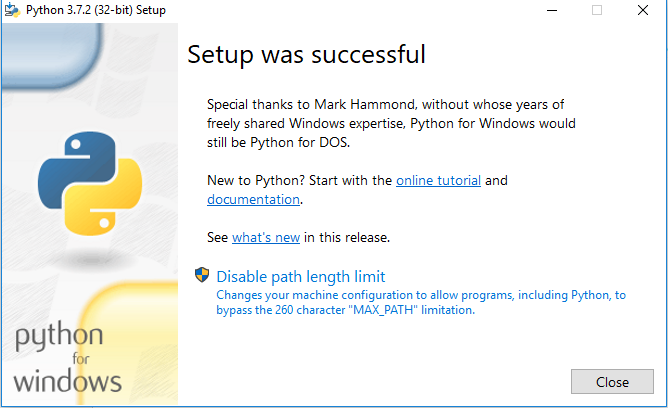
Bước 2: Chọn phiên bản ở mục Downloads và nhập chọn Python 3.7.2 để download



**Bước 3:**Sau khi download hoàn tất. Chúng ta nhấn chọn chạy file python-3.6.1.exe để bắt đầu tiến trình cài đặt. Tick vào ô Add Python 3.7 to PATH và chọn Install Now

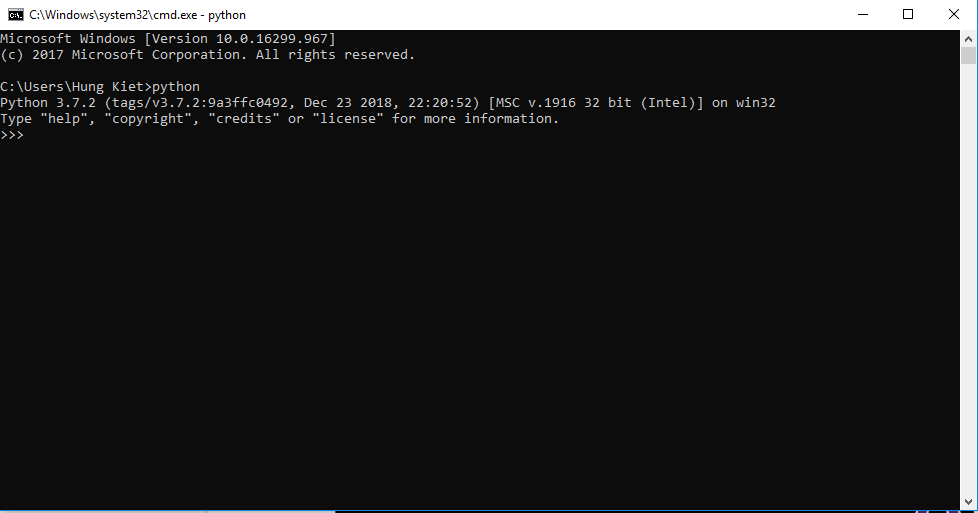


Bước 4: Khi cửa sổ hiển thị Setup was successful là ta đã cài đặt thành công môi trường Python > Close.



Bước 6:Tiếp đến, ta mở Command Prompt (CMD)để kiểm tra.

Trong cửa sổ Command Prompt, bạn gõ python > Enterđể kiểm tra



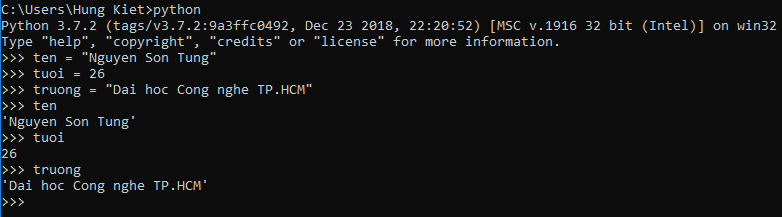
Nếu bạn hiển thị ra được shell để tương tác với Python như hình ở trên có nghĩa là phần cài đặt đã hoàn tất

### Kiểu dữ liệu trong python

* Khởi tạo 1 biến trong python

Cú pháp: <Tên biến> = <giá trị của biến>

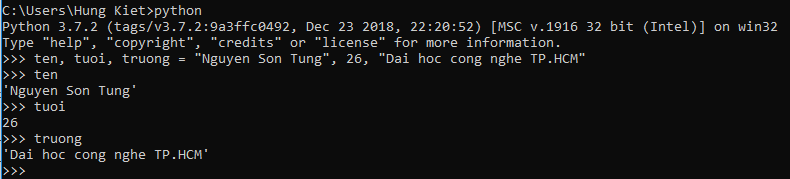
Ví dụ: đoạn code sau khai báo 3 biến:



* Khởi tạo nhiều biến trong python

Cú pháp: <tên biến thứ nhất>, <tên biến thứ hai>, ..,<tên tên biến thứ n> = <giá trị biến thứ nhất>, <giá trị biến thứ hai>, .., <giá trị biến thứ n>

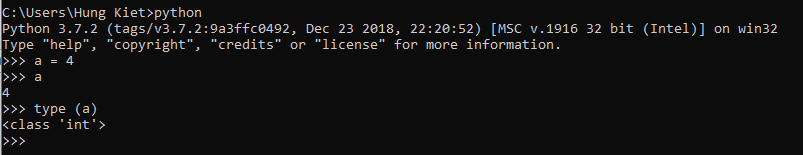
Ví dụ:



### Một số kiểu dữ liệu cơ bản trên python

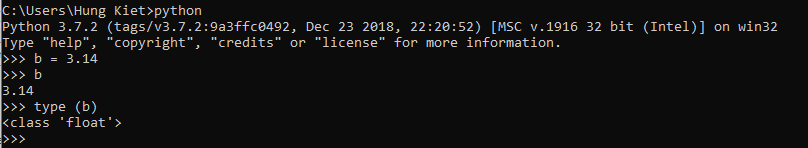
* Kiểu số nguyên

Ví dụ: gán giá trị cho biến a là 5 và xuất ra kiểu dữ liệu của biến a



* Số thực

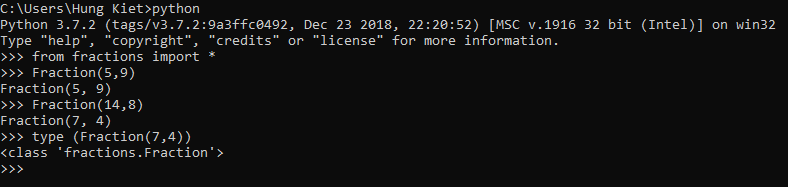
Ví dụ: gán giá trị cho biến b là 3.14 và xuất ra kiểu dữ liệu của biến a



* Phân số

Tạo 1 phân số: để tạo 1 phân số trong python chúng ta sử dụng hàm Fraction cú pháp như sau Fraction(<Tử\_số>, <Mẫu\_số>)

Ví dụ: nhâp phân số 5/9, 14/8, 3/9

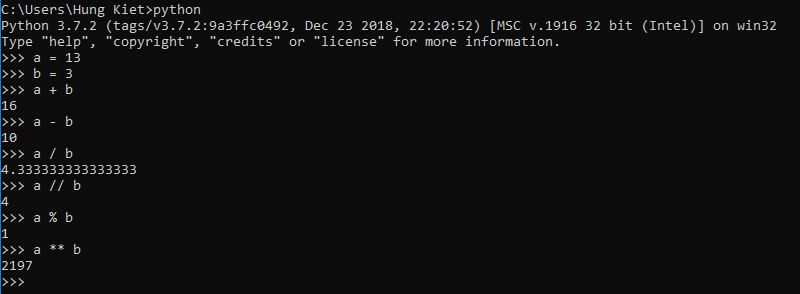


* Một số toán tử cơ bản với dữ liệu số trên python

Bảng một số biểu thức toán học với kiểu dữ liệu số trên python:

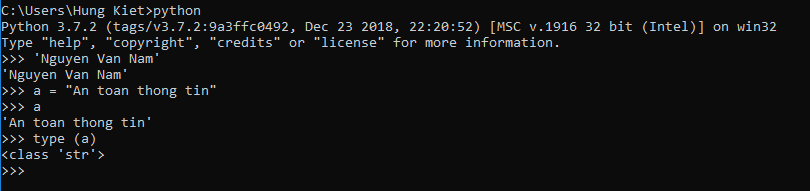
|  |  |
| --- | --- |
| Biểu thức | Giải thích |
| A + B | Tổng của A và B |
| A – B | Hiệu của A và B |
| A \* B | Tích của A và B |
| A / B | Thương của A và B (kết quả luôn là một số thực) |
| A // B | Thương nguyên của A và B ( kết quả luôn luôn nhỏ hơn hoặc bằng X / Y) |
| A % B | Dư của thương A và B |
| A \*\* B | Lũy thừa mũ B với cơ số A |

Ví dụ: cho 2 biến A và B lần lượt có giá trị là 13 và 3. Thực hiện các biểu thức toán học trên



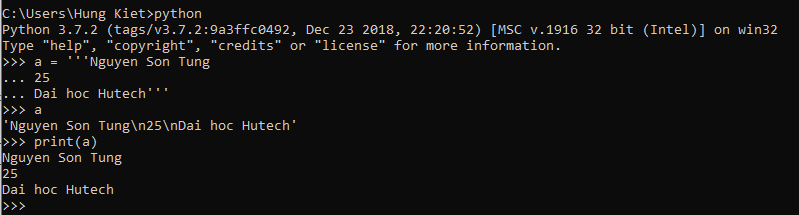
### Kiểu dữ liệu chuỗi trong python

Trong python chuỗi thường được đặt trong dấu ‘ ’ hoặc dấu “ ”.

Ví dụ: 

Chuỗi trên nhiều dòng với ‘’’ và “”” trong python

Ví dụ: in ra 3 dòng Nguyen Son Tung, 25, Dai hoc Hutech



Khái niệm Escape Sequence: là một ký tự đặc biệt trong python, bắt đầu với một dấu /

Một số Escape Sequence cần lưu ý:

|  |  |
| --- | --- |
| Ký hiệu | Giải thích |
| \b | Đưa con trỏ về lại một khoảng trắng |
| \n | Đưa con trỏ xuống dòng |
| \t | in ra 1 tab |
| \’ | In ra ký tự ‘ |
| \” | In ra ký tự “ |
| \\ | In ra ký tự \ |

#### Chuỗi trần

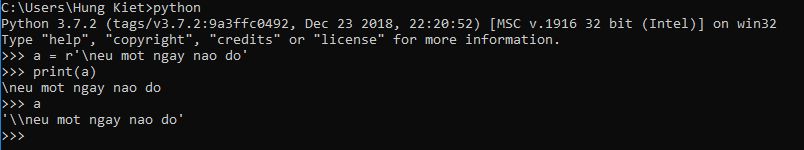
Chuỗi trần trong python giúp chúng ta thao tác với các đường dẫn trên hệ điều hành Windows. Ví dụ như các đường dẫn có dạng

Ổ-Đĩa:\\Thư\_Mục\\Thư\_Mục

Nếu tên thư mục bắt đầu bằng các chữ cái như t, n, a, ... và kết hợp với kí tự \. Thì kết quả sẽ không được như chúng ta mong muốn. Nên trường hợp này chúng ta sẽ sử dụng một dạng chuỗi, gọi là chuỗi trần. Chuỗi trần sẽ không quan tâm đến những nội dung bên trong chuỗi (Escape Sequence)

Cú pháp: r’nội dung chuỗi’

ví dụ:

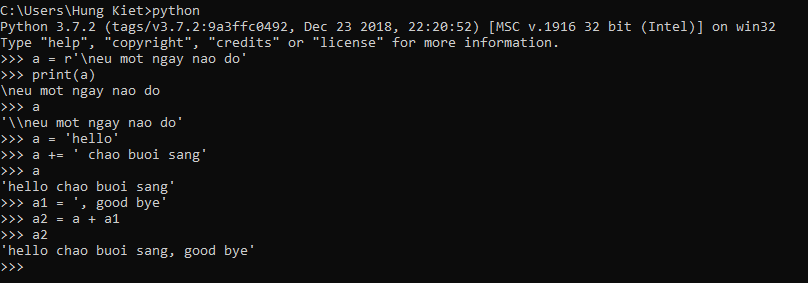


#### Một số toán tử với chuỗi

* Toán tử Cộng(+) : Đây là một toán tử sử dụng để nối các chuỗi

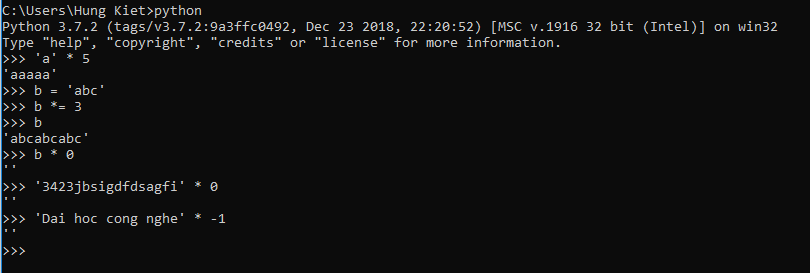
Cú pháp: A + B (A và B là chuỗi)

Ví dụ:



* Toán tử Nhân (\*): toán tử này sẽ lặp lại số lần chuỗi như người dùng mong muốn

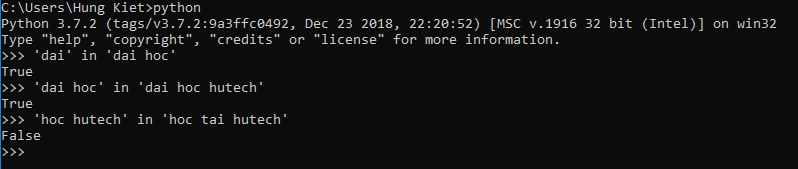
Cú pháp: A \* N (A là một chuỗi, N là một số nguyên)



* Toán tử IN: toán tử này sẽ trả về kết quả là True hoặc False.

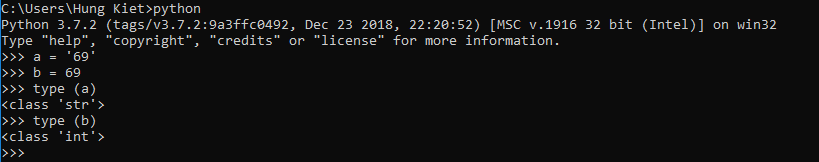
Cú pháp: s in A (Với s và A là chuỗi)

Kết quả sẽ là True nếu chuỗi s xuất hiện trong chuỗi A, còn ngược lại là False

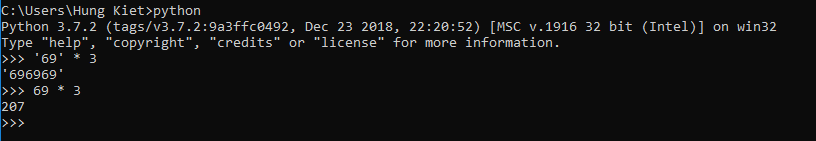


* Ép kiểu dữ liệu

Ví dụ: giá trị của 2 biến a và b ở dưới đây khác nhau



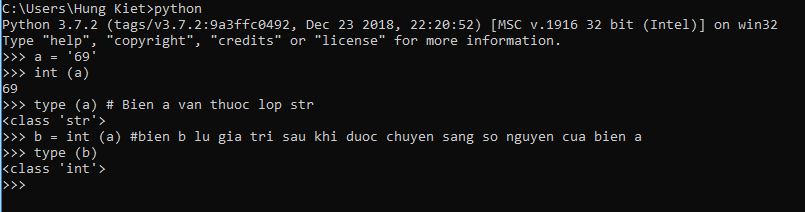
Một ví dụ khác cho chúng ta thấy sự khác nhau giữa hai biểu thức sau



Một số bất kỳ được đặt trong dấu nháy đơn hoặc kép sẽ được xem như là một ký tự, nếu như chúng ta muốn tính toán, chúng ta phải ép kiểu dữ liệu sang kiểu số. Ví dụ ở đây theo trường hợp là số nguyên

Cú pháp: int(giá trị)

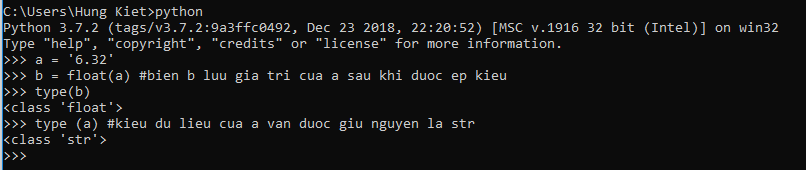
Ví dụ:



Tương tự như vậy chúng ta cũng có thể chuyển kiểu dữ liệu str sang số thực

Cú pháp: float(giá trị)

Ví dụ:

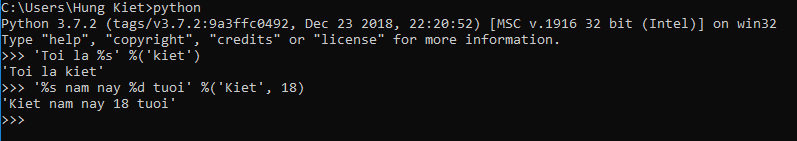


#### Định dạng chuỗi bằng toán tử %

Kiểu định dạng này rất quen thuộc bởi vì chúng ta đã từng tiếp xúc với ngôn ngữ C.

Cú pháp: <chuỗi>%(giá trị thứ 1, giá trị thứ 2, ..giá trị thứ n)

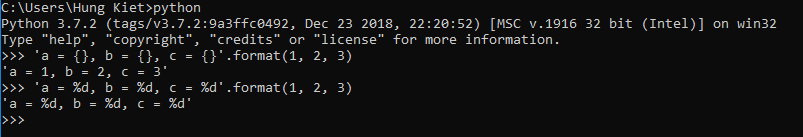
Ví dụ:



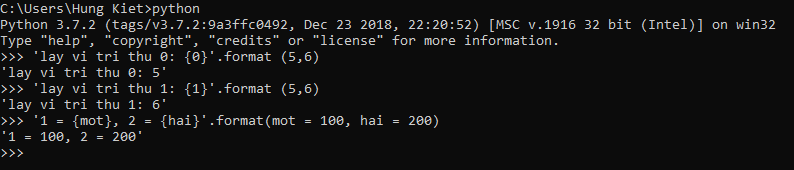
#### Định dạng bằng phương thức format

Phương thức format cho phép chúng ta thay thế các phẩn tử với nhau thông qua định dạng vị trí

Ví dụ:



Để hiểu rõ hơn chúng ta xem ví dụ này:



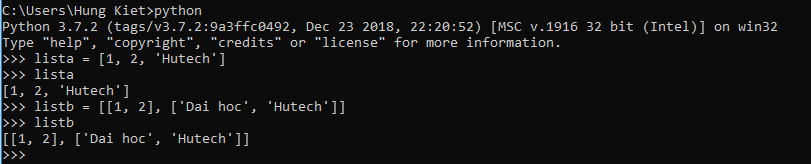
### Kiểu dữ liệu list trong python

* List là một container thường xuyên được sử dụng trong python. Một list gồm các yêu tố sau:
* Được giới hạn bởi cặp ngoặc [], tất cả những gì nằm trong đó là những phần tử của List
* Các phần tử của List được ngăn cách nhau bởi dấu phẩy ( ,).
* List có thể chứa mọi giá trị, đối tượng trong python. Và có thể chứa chính nó

Cách khởi tạo List

Cú pháp: [<giá trị thứ nhất>, <giá trị thứ hai>, <giá trị thứ ba> ... <giá trị thứ n>]

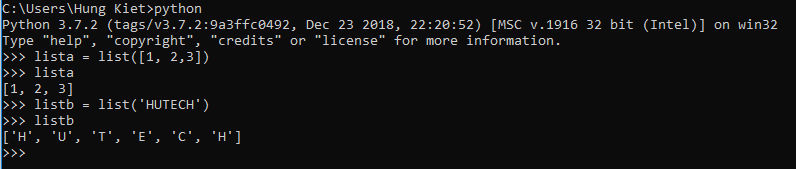
Ví dụ:



* Sử dụng constructor List

Cú pháp: list(iterable)

Ví dụ:

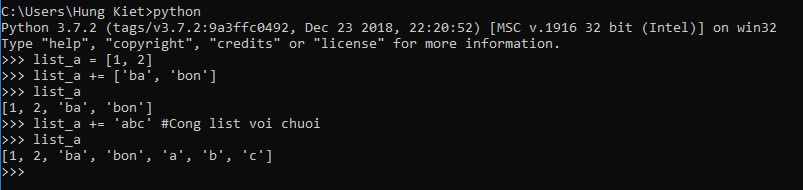


#### Một số toán tử với List trong python

Các toán tử List tương tự với kiểu dữ liệu chuỗi trong python.

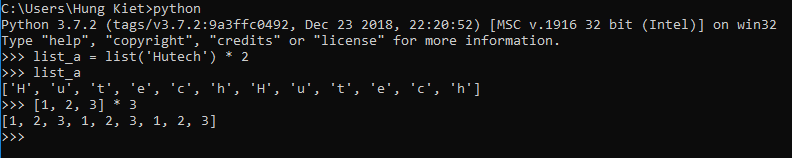
* Toán tử Cộng (+)

Ví dụ:



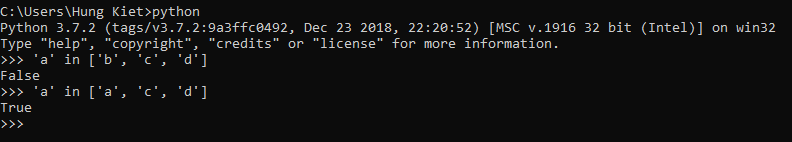
Toán tử Nhân (\*)

Ví dụ:



* Toán tử in

Ví dụ:



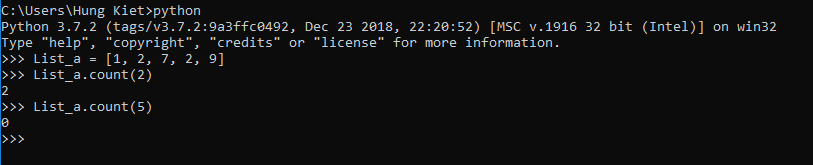
#### Các phương thức tiện ích của kiểu dữ liệu list

* Phương thức Count

Cú pháp: <List>.count(subject)

Công dụng: phương thức này trả về 1 số nguyên là số lần xuất hiện của subject trong chuỗi

Ví dụ:



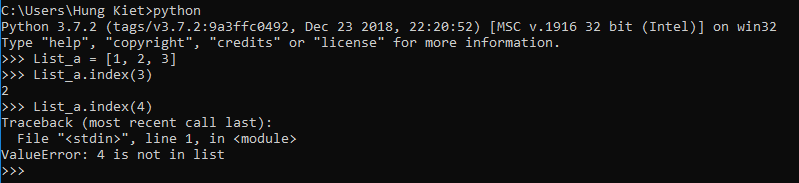
* Phương thức index

Cú pháp:

<List>.index(subject)

Công dụng: Trả về một số nguyên, là vị trí đầu tiên của sub khi dò từ trái sang phải trong list. Nếu subject không có trong list thì sẽ báo lỗi

Ví dụ:

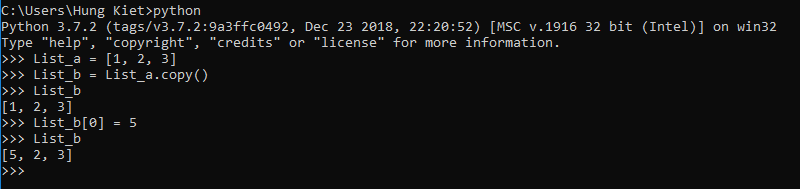


* Phương thức copy

<List>.copy()

Công dụng: trả về một List tương tự với List[:]

Ví dụ:

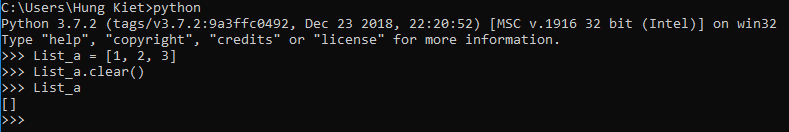


* Phương thức clear:

Cú pháp: <List>.clear()

Công dụng: xóa mọi phần tử có trong List

Ví dụ:

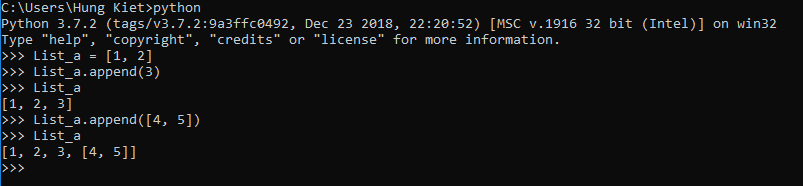


* Phương thức append

Cú pháp: <List>.append(x)

Công dụng: thêm 1 phân tử vào cuối List

Ví dụ:

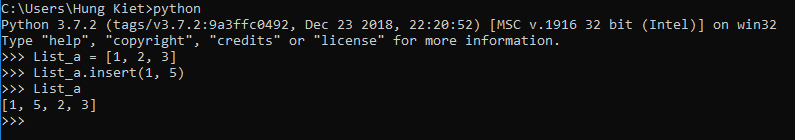


* Phương thức insert

Cú pháp: <List>.insert(i, x)

Công cụ: chèn 1 phần tử x vào vị trí i trong List:

Ví dụ:



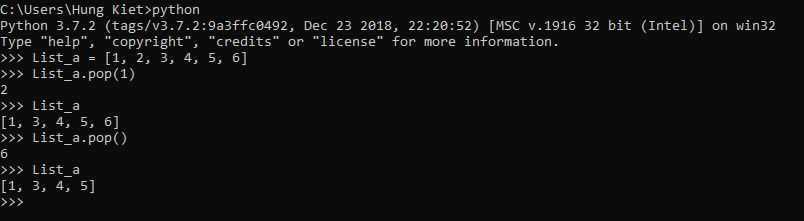
* Phương thức pop

Cú pháp:

<List>.pop([i])

Công dụng: bỏ đi phần tử thứ i trong List và trả về giá trị tại vị trí i. Nếu i không được khai báo, phương thức này sẽ tự bỏ đi phần tử cuối cùng của List và trả về giá trị đó

Ví dụ:

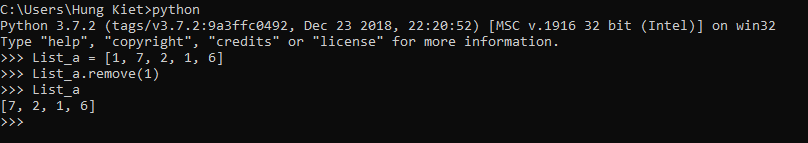


* Phương thức remove

Cú pháp: <List>.remove(x)

Bỏ đi phần tử có giá trị x đầu tiên trong List. Nếu giá trị x không có trong List phương thức sẽ báo lỗi

Ví dụ:

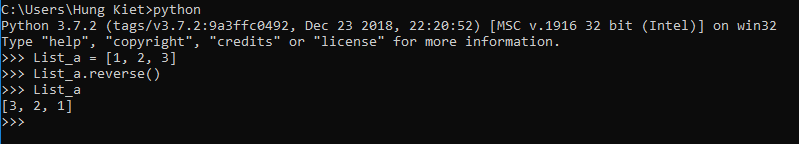


* Phương thức reverse

Cú pháp: <List>.reverse()

Công dụng: đảo ngược các phẩn tử ở trong List

Ví dụ:

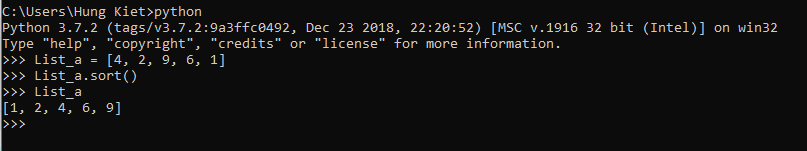


* Phương thức sort

Cú pháp: <List>.sort([func])

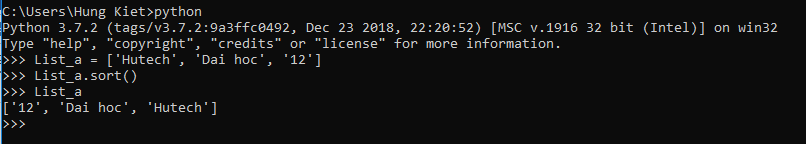
Công dụng: sắp xếp các phần tử từ bé đến lớn bằng cách so sánh trực tiếp

Ví dụ:



Phương thức này còn có thể so sánh cả chuỗi và List và mọi phương thức khác

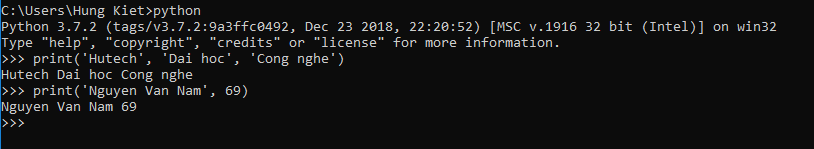
Ví dụ:



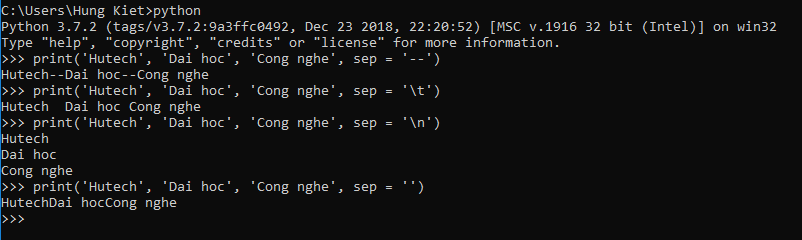
### Hàm nhập xuất trong python

Hàm xuất trong python: cấu trúc: print(**\*objects, sep=' ', end='\n', file=sys.stdout, flush=False**)

hàm xuất của python có thể truyền argument và hàm print với số lượng bất kỳ.

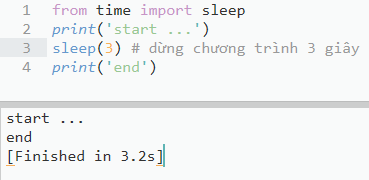
Ví dụ: 

Sep (separate – chia ra): giá trị mặc định của các parameter là một khoảng trắng. Chúng ta có thể dùng sep để tùy chỉnh giữa khoảng cách các parameter

Ví dụ: 

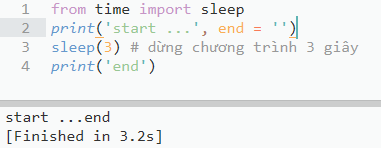
End (kết thúc bằng)

Mặc định ở trong ngôn ngữ lập trình python hoặc các ngôn ngữ khác (C, C++, java ...) mỗi lần print chúng sẽ tự xuống dòng. Đó là nhờ parameter end, nó sẽ tự thêm một ký tự newline (\n). ở trong python chúng ta có thể thay đổi giá trị parameter này.

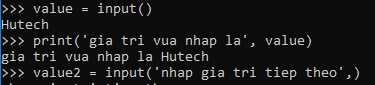
Ví dụ: 

Khi chạy chương trình, sau khi xuất hiện dòng start thì sau đó chương trình dừng 3 giây mới xuất dòng end. Và chúng ta không thay đổi parameter end nên hàm print vẫn xuống hàng bình thường

Chúng ta tiến hành thay đổi parameter end và xem kết quả

Ví dụ: 

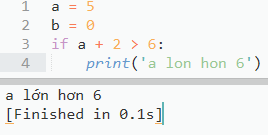
* Hàm nhập trong python (input)



Cấu trúc if:

Cấu trúc: if expression:

#if-block

Ví dụ: 

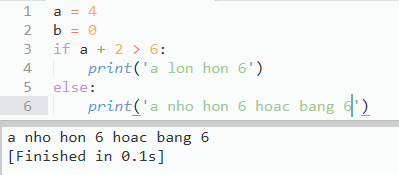
Cấu trúc if – else:

**if**expression**:**

    # If-block

**else:**

    # else-block

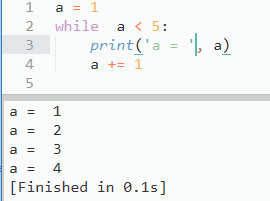
Ví dụ: 

### Cấu trúc vòng lặp while

**Cấu trúc:** while expression:

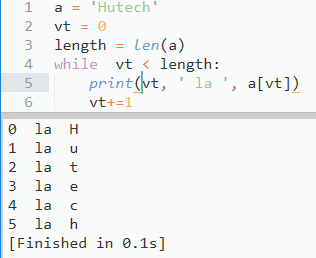
     # while-block

Ví dụ:



Sử dụng vòng lặp while để xử lý chuỗi

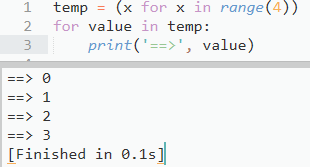
Ví dụ:



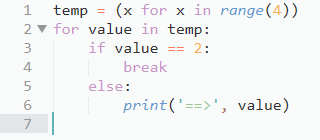
### Cấu trúc vòng lặp for:

**for**variable\_1, variable\_2, .. variable\_n in sequence:

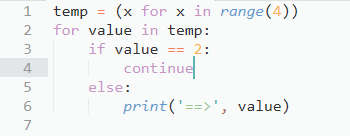
    # for-block

Ví dụ:

Câu lệnh break, continue trong vòng lặp for

Ví dụ: 

Ví dụ về continue trong vòng lặp for



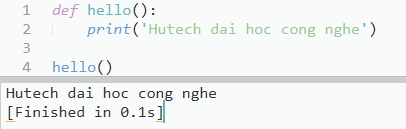
### Hàm trong python

Cách khai báo hàm trong python:

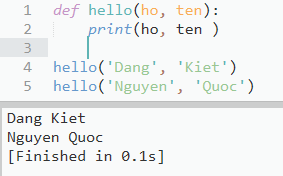
def <function\_name>(parameter\_1, parameter\_2, .., parameter\_n):

    function-block

Ví dụ về khai báo hàm và gọi hàm trong python:



Parameter và argument



# Bài tập thực hành

Bài tập 1:

Viết chương trình tìm tất cả các số chia hết cho 7 nhưng không phải bội số của 5, nằm trong đoạn 2000 và 3200 (tính cả 2000 và 3200). Các số thu được sẽ được in thành chuỗi trên một dòng, cách nhau bằng dấu phẩy.

Gợi ý:

Sử dụng range(#begin, #end)

Bài tập 2: Viết chương trình chấp nhận một chuỗi số, phân tách bằng dấu phẩy từ giao diện điều khiển, tạo ra một danh sách và một tuple chứa mọi số.

Ví dụ: Đầu vào được cung cấp là 34,59,55,32,18,98 thì đầu ra là:

['34', '59', '55', '32', '18', '98']  
('34', '59', '55', '32', '18', '98')

Bài tập 3: Viết một chương trình tính giá trị bình phương của một số.

Gợi ý:

Sử dụng toán tử \*\*.

Bài tập 4: Viết một chương trình chấp nhận đầu vào là chuỗi các số nhị phân 4 chữ số, phân tách bởi dấu phẩy, kiểm tra xem chúng có chia hết cho 5 không. Sau đó in các số chia hết cho 5 thành dãy phân tách bởi dấu phẩy.

Ví dụ đầu vào là: 0100,0011,1010,1001

Đầu ra sẽ là: 1010

Bài tập 5: Viết một chương trình chấp nhận đầu vào là một câu, đếm chữ hoa, chữ thường.

Giả sử đầu vào là: Dang Hung Kiet

Thì đầu ra là:

Chữ hoa: 3

Chữ thường: 10

Bài tập 6: Sử dụng một danh sách để lọc các số lẻ từ danh sách được người dùng nhập vào.

Giả sử đầu vào là: 1,2,3,4,5,6,7,8,9 thì đầu ra phải là: 1,3,5,7,9

## Lập trình Assembly

* Một số lệnh Assembly cơ sở

**Cú pháp lệnh:**

Một lệnh hợp ngữ đầy đủ gồm bốn thành phần sau đây:

**[Nhãn lệnh:]         <Tên lệnh>        [Các toán hạng]       [;Lời giải thích]**

Trong đó:

* <Nhãn lệnh>: là một dãy các kí tự đứng trước câu lệnh (kết thúc bởi hai dấu chấm (:) ), nó được chỉ định thay thế cho địa chỉ của câu lệnh trong các đoạn lệnh lặp, rẽ nhánh,... Do đó, nó chỉ được sử dụng khi cần
* Trong một chương trình hợp ngữ không thể có hai nhãn lệnh trùng tên, tên của các nhãn cũng không thể trùng với tên của các thủ tục trong chương trình
* <Tên lệnh>: là một trong các lệnh thuộc tập lẹnh hợp ngữ của vi xử lý trên máy tính thực hiện lệnh này.
* Lệnh hợp ngữ không phân biệt chữ hoa hay chữ thường. Trong chương trình hợp ngữ mỗi dòng chỉ có thể chứa một lệnh và mỗi lệnh phải được đặt trên một dòng.
* **[Các toán hạng]**: Là đối tượng mà lệnh tác động vào. Một lệnh hợp ngữ của Intel 8088/8086 có thể không có toán hạng, có một toán hạng, hoặc có hai toán hạng. Nếu có hai toán hạng thì toán hạng đứng trước gọi là [Toán hạng đích], toán hạng đứng sau gọi là [Toán hạng nguồn]. [Toán hạng đích] không thể là một hằng số.
* Một số lệnh hợp ngữ của các Intel 80286/80386/... có thể có đến 3 toán hạng, trong trường hợp này cũng chỉ có một [Toán hạng đích].
* **[;Lời giải thích]**: Chỉ có tác dụng với người viết và người đọc chương trình, nó không có ý nghĩa với trình biên dịch, tức là, không được dịch sang mã máy. Lời giải thích thường được sử dụng để làm rõ ý nghĩa của câu lệnh (khi người viết thấy cần). Lời giải thích phải nằm sau dấu chấm phảy (**;**).

Ví dụ: Mov AX, BX ; đặt giá trị thanh ghi BX vào thanh ghi AX

Trong đó: ***Lenh\_VD*:**Trong trường hợp này dãy kí tự Lenh\_VD được sử dụng làm nhãn lệnh cho lệnh Mov.

**Mov:**Là tên lệnh.

**AX và BX:**Là các toán hạng (đích và nguồn). Trong trường hợp này toán hạng là các thanh ghi đa năng 16 bít.

***“đặt giá trị thanh ghi BX vào thanh ghi AX”:****L*à lời giải thích cho lệnh này. Trong thực tế lời giải thích thường là tiếng Việt không dấu.

* **Lệnh mov (move):**

Cú pháp lệnh: Mov      [Toán hạng đích], [Toán hạng nguồn]

**Trong đó:**

[Toán hạng đích]: Có thể là thanh ghi (8 bít hay 16 bít), ô nhớ (chính xác hơn là địa chỉ của một ô nhớ) hay một biến nào đó. [Toán hạng đích] không thể là hằng số.

[Toán hạng nguồn]: Có thể là hằng số, biến, thanh ghi, ô nhớ (chính xác hơn là địa chỉ của một ô nhớ) nào đó.

Tác dụng: Lấy nội dung (giá trị) của [Toán hạng nguồn] đặt vào [Toán hạng đích]. Nội dung của [Toán hạng nguồn] không bị thay đổi.

Ví dụ: Mov        Ax, 5                ; Ax ß 5: đặt giá trị 5 vào thành ghi Ax

Mov        Dl, ‘A’              ; Dl = 41h: đặt mã ASCII của ‘A’ vào thanh ghi Dl

* Các lệnh Inc – Dec – Add và Sub

Cú pháp lệnh:

Inc        [Toán hạng đích]

**Add       [**Toán hạng đích**],[**Toán hạng nguồn**]**

**Dec       [**Toán hạng đích**]**

**Sub       [**Toán hạng đích**],[**Toán hạng nguồn**]**

**Trong đó:**[Toán hạng đích], [Toán hạng nguồn]: tương tự lệnh Mov.

**Tác dụng:**

Lệnh **Inc (Inc**rement**)**: làm tăng giá trị của [Toán hạng đích] lên 1 đơn vị.

Lệnh **Dec (Dec**rement**)**: làm giảm giá trị của [Toán hạng đích] xuống 1 đơn vị.

Lệnh **Add (Add**ition**)**: lấy giá trị/nội dung của [Toán hạng nguồn] cộng vào giá trị/nội dung của [Toán hạng đích], kết quả này đặt vào lại [Toán hạng đích].

Lệnh **Sub (Sub**tract**)**: lấy giá trị/nội dung của [Toán hạng đich] trừ đi giá trị/nội dung của [Toán hạng nguồn], kết quả này đặt vào lại [Toán hạng đích].

Ví dụ:   Mov        Ax, 121             ; đặt giá trị 121 vào thanh ghi Ax

Mov        Bx, 223             ; đặt giá trị 232 vào thanh ghi Bx

                    Inc           Ax                    ; Ax = Ax + 1: tăng Ax lên 1 đơn vị (Ax = 122)

                    Dec         Bx                    ; Bx = Bx + 1: giảm Bx xuống 1 đơn vị (Bx = 222)

                    Sub         Ax, Bx              ; Ax = Ax – Bx : Ax = -100

                    Add         Ax, 120            ; Ax = Ax + 120 : Ax = 20

                    Mov        Cx, Ax              ; Cx= Ax : Cx = 20

Dãy lệnh trên, đặt giá trị cuối cùng của thanh ghi Ax vào thanh ghi Cx (Cx = 20).

* Lệnh ngắt trong chương trình Int

Cú pháp:

Int <n>

**Trong đó:**Trong đó <n> là số hiệu ngắt của một ngắt mềm cần gọi. Tức là, n có thể là một trong các giá trị từ 0 đến 255, đó chính là số hiệu ngắt của 256 ngắt mềm mà BIOS và MSDOS cung cấp.

* Lệnh so sánh CMP (compare): cú pháp

**Cú pháp:      Cmp**     **[**Toán hạng đích**]**, **[**Toán hạng nguồn**]**

Trong đó: [Toán hạng đích], [Toán hạng nguồn] có thể là hằng, biến, thanh ghi hay ô nhớ. [Toán hạng đích] không thể là hằng số. [Toán hạng đích] và [Toán hạng nguồn] không thể đồng thời là ô nhớ.

Tác dụng: Lệnh Cmp (Compare) được sử dụng để so sánh giá trị/nội dung của [Toán hạng đích] so với [Toán hạng nguồn]. Tương tự như lệnh Sub, nó lấy [Toán hạng đích] trừ đi [Toán hạng nguồn] nhưng kết quả không làm thay đổi [Toán hạng đích] mà chỉ làm thay đổi giá trị của một số cờ hiệu: CF, ZF, OF,.

Kết quả so sánh của lệnh Cmp là: [Toán hạng đích] > [Toán hạng nguồn]; [Toán hạng đích] ≥[Toán hạng nguồn]; [Toán hạng đích] < [Toán hạng nguồn]; [Toán hạng đích] ≤[Toán hạng nguồn]; [Toán hạng đích] = [Toán hạng nguồn]; [Toán hạng đích] ≠ [Toán hạng nguồn];... mỗi kết quả sẽ tác động (0 →1, 1→0) đến một cờ tương ứng cụ thể nào đó.

Do đó, để biết được kết quả so sánh chương trình phải sử dụng các lệnh kiểm tra cờ (đó là cá lệnh nhảy), và chúng phải được đặt ngay sau lệnh so sánh. Như vậy lệnh Cmp sẽ không có ý nghĩa khi nó đứng độc lập.

Có thể nói ngược lại, lệnh Cmp được sử dụng để cung cấp điều kiện nhảy (thay đổi giá trị các cờ) cho các lệnh nhảy có điều kiện.

Ví dụ 1:

Cmp        Ax, Bx                        ; so sánh giá tị thanh ghi Ax với Bx

Cmp        Ax, 20                        ; so sánh giá trị thanh ghi Ax với 20

Cmp        Ax, [SI]                      ; so sánh Ax với nội dung ô nhớ được chỉ bởi SI

* Các lệnh nhảy
* Lệnh nhảy không điều kiện:

**Cú pháp:   Jmp**       **<**Vị trí đích**>**

**Trong đó:**<Vị trí đích> có thể là nhãn của một lệnh, tên của một thủ tục hoặc có thể là một thanh ghi, một ô nhớ (đã được định nghĩa) nào đó. <Vị trí đích> cũng có thể là một biến nào đó, giá trị của nó thường là địa chỉ của một ô nhớ trong đoạn Code.

**Tác dụng:**Khi gặp lệnh này chương trình chuyển điều khiển (nhảy đến) đến thực hiện lệnh sau <Nhãn đích> mà không phụ thuộc vào bất kỳ điều kiện nào.

Cơ chế thực hiện của lệnh Jmp là thay đổi nội dung của cặp thanh ghi con trỏ lệnh CS:IP. CS:IP mới sẽ là địa chỉ của lệnh sau <Nhãn đích> trong bộ nhớ.

Lệnh Jmp có 3 dạng: Short, Near và Far. Đối với dạng Short và Far thì chỉ có thanh ghi IP bị thay đổi khi lệnh thực hiện, ngược lại, với dạng Far, khi lệnh thực hiện thì cả thanh ghi IP và thanh ghi đoạn CS đều bị thay đổi. Hay nói cách khác: Đối với dạng Short và Near thì lệnh Jmp và <Vị trí đích> phải nằm trong cùng Segment nhớ, ngược lại, với dạng Far thì lệnh Jmp và <Vị trí đích> có thể nằm ở các Segment nhớ khác nhau.

**Ví dụ 1:**

                    Start**:**

                          Jmp               Main

                          TieuDe         DB      ‘Khoa CNTT – DHHUTECH’

                    Main       **Proc**

                  ..................

                    Main       **Endp**

**End**            Start

* Lệnh nhảy có điều kiện

Cú pháp chung:           <Lệnh nhảy có điều kiện>        <Vị trí đích>

Trong đó: <Vị trí đích>: Tương tự như lệnh Jmp.

Tác dụng: Khi gặp một lệnh nhảy có điều kiện, đầu tiên chương trình sẽ kiểm tra điều kiện nhảy của nó, nếu thỏa mãn thì sẽ nhảy đến thực hiện lệnh ở <Vị trí đích>, nếu không thì bỏ qua không thực hiện lệnh nhảy này.

Điều kiện nhảy của các lệnh nhảy này chính là sự thay đổi giá trị của các cờ hiệu, do đó để tạo điều kiện nhảy cho một lệnh nhảy xác định thì chương trình phải làm thay đổi giá trị của cờ hiệu tương ứng với nó. Chương trình thường dùng các lệnh địch bít, quay bít, so sánh,... để làm thay đổi giá trị các cờ hiệu để tạo điều kiện nhảy cho các lệnh nhay. Cách đơn giản nhất là sử dụng lệnh Cmp ngay trước lệnh nhảy.

Sau đây là các lệnh nhảy có điều kiện với dữ liệu có dấu:

Lệnh JG:             Nhảy nếu [Đích] > [Nguồn] ; (SF = 0F và ZF = 0)

Lệnh JL:             Nhảy nếu [Đích] < [Nguồn] ; (SF <> 0F)

Lệnh JGE:          Nhảy nếu [Đích] ≥[Nguồn] ; (SF = 0F)

Lệnh JLE:           Nhảy nếu [Đích] ≤[Nguồn] ; (CF <> 0F và ZF = 1)

 ... <còn nhiều lệnh khác>

Trong đó: [Đích] và [Nguồn] chính là hai toán hạng: [Toán hạng đích] và [Toán hạng nguồn] trong lệnh Cmp đứng ngay trước lệnh nhảy. Tức là, chương trình sử dụng lệnh Cmp để tạo điều kiện nhảy cho các lệnh này. Cụ thể: lệnh nhảy có thực hiện được hay không (có chuyển điều khiển đến <Vị trí đích> hay không) phụ thuộc vào giá trị của [Đích] và [Nguồn] trong lệnh Cmp đứng ngay trước nó.

Với việc sử dụng lệnh Cmp để tạo điều kiện nhảy cho các lệnh nhảy thì ta không cần quan tâm đến các cờ điều kiện nhảy của chúng.

Sau đây là các lệnh nhảy có điều kiện với dữ liệu không dấu:

Lệnh JA:            Nhảy nếu [Đích] > [Nguồn] ; (CF = 0 và ZF = 0)

Lệnh JB:              Nhảy nếu [Đích] < [Nguồn] ; (CF = 0)

Lệnh JNA:          Nhảy nếu [Đích] không lớn hơn [Nguồn]; (CF =1 or ZF =1)

Lệnh JNB:           Nhảy nếu [Đích] không nhỏ hơn [Nguồn] ; (CF = 0)

Các lệnh nhảy với dữ liệu có dấu có thể áp dụng với các dữ liệu không dấu.

Sau đây là các lệnh nhảy có điều kiện dùng chung:

Lệnh JC:              Nhảy nếu cờ CF = 1

Lệnh JNC:           Nhảy nếu cờ CF = 0

Lệnh JZ:              Nhảy nếu cờ ZF = 1

Lệnh JNZ:           Nhảy nếu cờ ZF = 0

Lệnh JE:              Nhảy nếu [Đích] = [Nguồn]; Tương tự JZ; (ZF = 1)

Lệnh JNE:           Nhảy nếu [Đích] ≠[Nguồn]; Tương tự JNZ; (ZF = 0)

... <còn nhiều lệnh khác> [2 - 150]

Với các lệnh này, chương trình thường sử dụng các lệnh dịch bít hoặc lệnh quay bít để tạo điều kiện nhảy nó.

Ví dụ 1a: Dãy lệnh sau đây thực hiện việc gán giá trị cho thanh ghi Cx dựa vào giá trị của thanh ghi Ax và Dx:

                    Mov        Ax, 12

                    Mov        Dx, 25

                    ;-------------------

                    Cmp        Ax, Dx                      ; Ax ? Bx

                    Jg            Nhan1                       ; nếu Ax > Dx

                    Jle           Nhan2

               Nhan1:

                    Mov        Cx, Ax

                    Jmp         Tiep\_Tuc

               Nhan2:

                    Mov        Cx, Dx

                    Jmp        Tiep\_Tuc

                Tiep\_Tuc:

                    Mov        Bx, Cx

                    ;------------------------

Có thể thấy, ở đây không cần dùng lệnh Jle    Nhan2, vì nếu Ax không lớn hơn Dx thì chắc chẵn nó sẽ nhỏ hơn hoặc bằng Dx. Ngoài ra cũng không cần dùng lệnh Jmp     Tiep\_Tuc sau nhãn Nhan2, vì việc chuyển đến lệnh sau nhãn Tiep\_Tuc ở đây là tất nhiên. Vì thế đoạn lệnh trên có thể được viết rút gọn như trong Ví dụ 1b sau đây.

Ví dụ 1b: Dãy lệnh sau đây là trường hợp rút gọn của dãy lệnh trên:

                    Mov        Ax, 12

                    Mov        Dx, 25

                    ;------------------

                    Cmp        Ax, Dx

                    Jg            Nhan1                       ; nếu Ax > Dx

                    Mov        Cx, Dx                        ; khi Ax ≤Dx

                    Jmp        Tiep\_Tuc

               Nhan1:

                    Mov        Cx, Ax                        ; khi Ax > Dx

                Tiep\_Tuc:

                    Mov        Bx, Cx

                    ;------------------------

Trong cả hai ví dụ trên: khi kết thúc, Bx =  Cx = Dx = 25. Nhưng nếu cho  
Ax = 120 (Ax > Bx) thì Bx = Cx = Ax = 120.

* Các lệnh dịch bit

Các lệnh dịch bít là các lệnh làm cho các bít trong một thanh ghi bị dịch về bên trái (lệnh ShR) hoặc về bên phải (lệnh ShL) một hoặc nhiều bít. Lệnh quay bít làm cho các bít trong một thanh ghi quay theo “chiều trái” hoặc theo “chiều phải” một hoặc nhiều bít. Thông thường các bít được dịch hay được quay đều lần lượt được đưa qua cờ CF. Do đo, các lệnh dịch bít và quay bít thường được sử dụng để kiểm tra giá trị bít (= 0 hay = 1) của các bít trong thanh ghi.

Hợp ngữ cung cấp hai dạng lệnh quay bít, quay không qua cờ CF (lệnh RoL và RoL) và quay có qua cờ CF (lệnh RcL và RcR).

**Cú pháp:**

**Shr      [**Toán hạng đích**], <**n**>**

**Shl      [**Toán hạng đích**], <**n**>**

**Rcr     [**Toán hạng đích**], <**n**>**

**Rcl      [**Toán hạng đích**], <**n**>**

**Trong đó:** [Toán hạng đích] là một thanh ghi 8 bít hoặc 16 bít. <n> là số bít cần dịch, nếu <n> = 1 thì chỉ định trực tiếp trong câu lệnh, nếu <n> lớn hơn 1 phải chỉ định thông qua thanh ghi CL.

**Tác dụng:**

Lệnh **ShR** (**S**hift Logical **R**ight): Dịch chuyển các bít trong thanh ghi [Toán hạng đích] sang phải một hoặc nhiều bít. Các bít được dịch lần lượt được đưa vào cờ CF, cờ CF sẽ chứa bít của lần dịch cuối cùng. Sau khi dịch các bít bị khuyết (ở bên đối diện) sẽ được thay bằng các bít có trị 0. Tức là, với thanh ghi 8 bít thì sau 8 lần dịch nó sẽ nhận được một dãy 8 bít = 0, tương tự với thanh ghi 16 bít thì sau 16 lần dịch nó sẽ nhận được một dãy 16 bít = 0. Nếu thanh ghi AL = **01**001001 thì sau khi bị dịch về bên trái 2 bít nó sẽ như sau: AL = 001001**00**, khi đó CF = 1.

Lệnh **ShL** (**S**hift Logical **L**eft): Tương tự như lệnh ShR nhưng các bít được dịch về phía bên trái.

Lệnh **RCR** (**R**otate through **C**arry **R**ight): Tương tự như lệnh ShR, nhưng bít được dịch sẽ được đặt vào lại bít bị khuyết ở bên đối diện. Tức là, với thanh ghi 8 bít thì sau 8 lần dịch nó sẽ nhận lại dãy bít ban đầu, tương tự với thanh ghi 16 bít thì sau 16 lần dịch nó sẽ nhận lại dãy bít ban đầu. Nếu thanh ghi AL = **01**001001 thì sau khi bị quay về bên trái 2 bít nó sẽ như sau: AL = 001001**01**, khi đó CF = 1.

Lệnh **RCL** (**R**otate through **C**arry **L**eft): Tương tự như lệnh RCR nhưng các bít được quay về phía bên trái.

Ví dụ 1:

Shr          Al, 1

Mov        Cl, 2

             Shr          Al, CL

             Shl          Bl, CL

              Rcl          AL, CL

Rcr          Dl, 1

# Bài tập thực hành

Bài tập 1: Viết chương trình hiện ra hai câu "Chao mung ban den voi Assembly" "Assembly that de!". Mỗi câu trên một dòng.

Bài tập 2: Viết chương trình yêu cầu nhập một ký tự và xuất ra màn hình ký tự vừa nhập

Bài tập 3: Viết chương trình nhập vào một ký tự. Xuất ra màn hình ký tự đứng trước và ký tự đứng sau

Bài tập 4: Viết chương trình nhập vào một chuỗi. In ra màn hình chuỗi thường, chuỗi in. Dùng chương con

Bài tập 5: Lập chương trình nhập vào 2 số kiểu word,in ra màn hình tổng 2 số vừa nhập