**Container. Đặt vấn đề và cách giải quyết**

Các bạn đã biết đến **BIẾN**(đã giới thiệu trong bài [BIẾN TRONG PYTHON](http://www.howkteam.vn/course/kieu-du-lieu-so-trong-python/bien-trong-python-1539)), đó là một **container**cho phép ta lưu trữ các dữ liệu và lấy ra khi cần, thay đổi khi ta cần cập nhật giá trị hoặc sửa chữa.

Nhưng, khả năng của biến vẫn bị giới hạn! Đơn giản với một ví dụ, ta cần biến **teo** lưu cho ta giá trị là chuỗi `”Teo”` là tên của Tèo, và tuổi của Tèo là số 17.

>>> teo = "Teo"

>>> teo

'Teo'

>>> teo = 17

>>> teo

17

Biến **teo** của chúng ta không thể lưu hai giá trị một lúc. Không chỉ tên và tuổi, Tèo còn rất nhiều thông tin muốn lưu vào biến **teo** nữa như ngày sinh của gấu, số lần fix bug trong một tháng, khóa học mới coi gần nhất, số lần tè dầm ở tuổi 17,…

Với năng lực của một người mới học lập trình, sáng kiến tối ưu nhất họ đưa ra là mỗi giá trị ta có một biến riêng biệt. Và đây được coi là một giải pháp hay!

Tuy nhiên vẫn ở tầm vi mô. Tèo nó tham, muốn lưu cả mấy thứ linh tinh về cô gấu dễ thương của hắn. Lúc đó, việc bạn kêu Tèo tạo ra số lượng biến để lưu trữ cũng là một điều gian nan rồi.

Đó là vì sao ta cần một thứ cũng như biến, nhưng nội công lại thâm hậu hơn biến, có khả năng lưu trữ nhiều giá trị cùng một lúc. Vì thế, Python có rất nhiều các container cho phép ta lưu trữ nhiều các giá trị, đối tượng cùng một lúc, hỗ trợ cho chúng ta trong việc truy xuất, tính toán, thay đổi (một số container trong Python không hỗ trợ việc thay đổi),…

Trong các ngôn ngữ lập trình khác, những container chứa được nhiều giá trị cùng một lúc thường được gọi là **ARRAY**(mảng).

**Giới thiệu về List trong Python**

**LIST**là một container được sử dụng rất nhiều trong các chương trình Python. Một List gồm các yếu 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 phân cách nhau ra bởi dấu phẩy (**,**).
* List có khả năng chứa mọi giá trị, đối tượng trong Python. Và bao gồm chứa chính nó! (một trường hợp hay ho Kteam sẽ giới thiệu ở phần khác).

**Ví dụ:**

>>> [1, 2, 3, 4, 5] # Một List chứa 5 số nguyên

[1, 2, 3, 4, 5]

>>> ['a', 'b', 'c', 'd'] # Một List chứa 4 chuỗi

['a', 'b', 'c', 'd']

>>> [[1, 2], [3, 4]] # Một List chứa 2 List là [1, 2] và [3, 4]

[[1, 2], [3, 4]]

>>> [1, 'one', [2, 'two']] # List chứa số nguyên, chuỗi, và List

[1, 'one', [2, 'two']]

**Cách khởi tạo List**

**Sử dụng cặp dấu ngoặc [] đặt giá trị bên trong**

**Cú pháp:**

**[<giá trị thứ nhất>, <giá trị thứ hai>, .., <giá trị thứ n – 1>, <giá trị thứ n>]**

**Ví dụ:**

>>> lst = [1,2,5,"kteam"]

>>> lst

[1, 2, 5, 'kteam']

>>> empty\_list = [] # khởi tạo list rỗng

>>> empty\_list

[]

**Sử dụng List Comprehension**

**Cú pháp**

**[Comprehension]**

**Ví dụ:**

>>> a = [kteam **for** kteam **in** range(3)]

>>> a

[0, 1, 2]

>>> another\_lst = [[n, n \* 1, n \* 2] **for** n **in** range(1, 4)]

>>> another\_lst

[[1, 1, 2], [2, 2, 4], [3, 3, 6]]

List **comprehension** là một cách khởi tạo một List rất thú vị trong Python. Do đó, rất khó để có thể nói hết các trường hợp. Vì vậy, hãy tạm gác lại kiến thức này, bạn không cần phải cố gắng hiểu nó khi chúng ta chưa gặp gỡ các vòng lặp.

**Sử dụng constructor List**

**Cú pháp:**

**list (iterable)**

|  |
| --- |
|  |

**Lưu ý:** **iterable**là một đối tượng nói chung của các container. Khái niệm này sẽ được Kteam giới thiệu ở bài sau. Đối với bạn khi theo dõi khóa học này của Kteam, bạn đã được biết hai iterable đó chính là chuỗi, và List.

**Ví dụ:**

>>> lst = list([1, 2, 3])

>>> lst

[1, 2, 3]

>>> str\_lst = list('HOWKTEAM')

>>> str\_lst

['H', 'O', 'W', 'K', 'T', 'E', 'A', 'M']

>>> list(1)

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

TypeError: 'int' object **is** **not** iterable

**Một số toán tử với List trong Python**

Các toán tử của List gần giống và tương tự với chuỗi (bạn có thể tham khảo toán tử của chuỗi ở bài [KIỂU DỮ LIỆU CHUỖI – phần 2](http://www.howkteam.vn/course/kieu-du-lieu-chuoi-trong-python--phan-1/kieu-du-lieu-chuoi-trong-python--phan-2-1542)).

**Toán tử +**

>>> lst = [1, 2]

>>> lst += ['one', 'two']

>>> lst

[1, 2, 'one', 'two']

>>> lst += 'abc' # cộng List và chuỗi

>>> lst

[1, 2, 'one', 'two', 'a', 'b', 'c']

>>> 'abc' + [1, 2] # List cộng chuỗi cho phép, chuỗi cộng List thì không.

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

TypeError: must be str, **not** list

**Toán tử \***

>>> lst = list('KTER') \* 2

>>> lst

['K', 'T', 'E', 'R', 'K', 'T', 'E', 'R']

>>> [1, 2] \* 3

[1, 2, 1, 2, 1, 2]

**Toán tử in**

>>> 'a' **in** [1, 2, 3]

**False**

>>> 'a' **in** ['a', 2, 3]

**True**

>>> 'a' **in** [['a'], 'b', 'c'] # chỉ có ['a'] thôi, không có 'a'

**False**

**Các toán tử so sánh**

Cú pháp:

**A** <**toán tử so sánh**> **B** (A và B là 2 list)

Trong Python, cách so sánh 2 list cũng giống như cách so sánh 2 chuỗi.

Hiểu một cách đơn giản, kết quả của các toán tử so sánh (trên 2 list) sẽ dựa trên việc so sánh 2 list đó. Kết quả trả về sẽ là **True** hoặc **False**.

Khi so sánh 2 list, chương trình sẽ lần lượt so sánh các phần tử có cùng vị trí trong 2 list. Nếu xuất hiện 2 giá trị khác nhau, thì kết quả của phép so sánh sẽ là kết quả khi so sánh 2 giá trị đó.

Khi so sánh đến hết một trong 2 list nhưng vẫn không có giá trị khác biệt, chương trình sẽ so sánh độ dài của 2 list và trả về kết quả tương ứng.

Do đó:

* [1, 2, 3] bằng [1, 2, 3]
* [1, 2, 3] lớn hơn [1, 2] (do [1, 2, 3] dài hơn)
* [4] > [3, 4] (do khi so sánh giá trị đầu tiên, ta có 4 > 3)

**Ví dụ:**

>>> [1, 2, 3] == [1, 2, 3]

**True**

>>> [1, 2, 3] == [1, 2]

**False**

>>> [4] > [3, 4]

**True**

>>> ['b', 'c', 'd'] < ['x', 'y', 'z']

**True**

>>> ['a'] > ['b']

**False**

**Indexing và cắt List trong Python**

Như đã đề cập, List với chuỗi giống nhau rất nhiều điểm, và phần Indexing và cắt List này hoàn toàn giống với Indexing và cắt chuỗi. (Nếu chưa biết về chuỗi bạn có thể tham khảo qua các bài về [KIỂU DỮ LIỆU CHUỖI TRONG PYTHON – Phần 1](http://www.howkteam.vn/course/kieu-du-lieu-chuoi-trong-python--phan-1/kieu-du-lieu-chuoi-trong-python--phan-2-1542))

>>> lst = [1, 2, 'a', 'b', [3, 4]]

>>> lst[0]

1

>>> lst[-1]

[3, 4]

>>> lst[3]

'b'

>>> lst[1:3]

[2, 'a']

>>> lst[:2]

[1, 2]

>>> lst[2:]

['a', 'b', [3, 4]]

>>> lst[::-1]

[[3, 4], 'b', 'a', 2, 1]

**Thay đổi nội dung List trong Python**

Như bạn đã biết, ta không thể thay đổi nội dung của chuỗi như ví dụ bên dưới

>>> s = 'math'

>>> s[1]

'a'

>>> s[1] = 'i'

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

TypeError: 'str' object does **not** support item assignment

Còn về phần List, ta có thể thay đổi nội dung của nó

>>> lst = [1, 'two', 3]

>>> lst[1]

'two'

>>> lst[1] = 2

>>> lst

[1, 2, 3]

**Ma trận**

Nghe ma trận hoành tráng thế thôi, bạn đã thấy nó rồi. Ví dụ một List chứa một List khác đấy.

>>> lst = [[1, 2, 3], [4, 5, 6]]

>>> lst

[[1, 2, 3], [4, 5, 6]]

Ta dễ dàng truy cập hai phần tử của List vừa mới khởi tạo

>>> **lst**[0]

[1, 2, 3]

>>> **lst**[-1]

[4, 5, 6]

Hai giá trị đó cũng là một List. Và lẽ dĩ nhiên, bạn có quyền truy cập đến các phần tử con của phần tử nằm trong List bạn vừa khởi tạo. Thậm chí là cắt List!

>>> lst[0][0]

1

>>> lst[0][-1]

3

>>> lst[1][1]

5

>>> lst[0][:2]

[1, 2]

>>> lst[1][:]

[4, 5, 6]

**Vấn đề cần lưu tâm khi sử dụng List**

Những lưu ý này nếu bạn không biết, chương trình của bạn có thể có output khác với bạn mong muốn.

**Không được phép gán List này qua List kia nếu không có chủ đích**

Hãy xem xét đoạn code sau đây

>>> lst = [1, 2, 3]

>>> another\_lst = lst

>>> lst

[1, 2, 3]

>>> lst = [1, 2, 3]

>>> lst

[1, 2, 3]

>>> another\_lst = lst

>>> another\_lst

[1, 2, 3]

>>> lst

[1, 2, 3]

Mọi thứ ổn, không có gì xảy ra, cho tới khi bạn thay đổi giá trị bất kì của một trong hai List đó.

>>> another\_lst[1]

2

>>> another\_lst[1] = 'Two'

>>> another\_lst

[1, 'Two', 3]

>>> lst

[1, 'Two', 3]

Chỉnh một, nhưng đổi tới hai. Lí do là vì khi bạn gán giá trị List trực tiếp như thế, bạn đang đưa hai List đó trỏ cùng vào một nơi. Nói cách khác, cùng một giá trị list, nhưng lại có đến hai cái tên (có thể có nhiều hơn 2 tùy vào cách ta gán biến).

Hãy tưởng tượng Tèo có 50 nghìn. Sau đó bạn sử dụng phép thuật của mình gán số tiền cô gấu của Tèo bằng số tiền của Tèo. Khi đó, cô gấu dễ thương của Tèo không tự nhiên mà có 50 nghìn, mà thay vào đó, bạn đã gián tiếp cho phép gấu của Tèo sử dụng số tiền của Tèo nhịn ăn mì tôm bấy lâu nay. Và vào một ngày trơi mưa không rơi, cô ấy chạy đi mua một gói Snack mất 5 nghìn và sử dụng số tiền 50 nghìn bạn vừa mới gán cho cô ấy. Hậu quả là Tèo về thấy mất đâu 5 nghìn.

Do đó, trước khi gán, bạn phải copy giá trị của List

Ta có thể chứng thực điều đó bằng cách sử dụng toán tử **is**.

**Toán tử is**

Cú pháp:

**A is B**

**Tác dụng:** Kiểm tra xem hai biến A và B có cùng trỏ đến một đối tượng hay không. Nếu một trong hai biến được gán giá trị bằng biến còn lại, thì kết quả trả về là True.

Để hiểu rõ hơn về toán tử is, cũng như là lỗi dễ mắc phải khi gán giá trị của các list cho nhau, ta cùng xem xét các ví dụ sau:

**Ví dụ 1:**

>>> a = [1, 2, 3]

>>> b = [1, 2, 3]

>>> a **is** b

**False**

>>> a[1] = 100 # Vì a và b trỏ đến 2 giá trị khác nhau, nên việc thay đổi giá trị bên trong của một biến không tác động đến biến còn lại

>>> a

[1, 100, 3]

>>> b

[1, 2, 3]

**Ví dụ 2:**

>>> a = [1, 2, 3]

>>> a = b

>>> a **is** b

**True**

>>> a[1] = 100 # Vì a và b trỏ đến cùng một giá trị list, nên việc thay đổi giá trị bên trong của một biến cũng sẽ “kéo theo” sự thay đổi của biến còn lại.

>>> a

[1, 100, 3]

>>> b

[1, 100, 3]

Kteam xin lưu ý với các bạn là:

Nếu như 2 biến cùng trỏ vào một giá trị, thì việc thay đổi giá trị bên trong của một biến sẽ tác động đến biến còn lại. Tuy nhiên, việc thay đổi toàn bộ giá trị của một biến lại không gây ảnh hưởng tới biến kia, và khi đó, 2 biến lại không cùng trỏ vào một giá trị nữa.

>>> a = [1, 2, 3]

>>> b = a

>>> a **is** b

**True**

>>> a = [1, 2, 3, 4]

>>> a **is** b

**False**

>>> a

[1, 2, 3, 4]

>>> b

[1, 2, 3]

Lưu ý khi sử dụng toán tử is:

* Đừng bao giờ sử dụng toán tử is đối với 2 biến thuộc kiểu số hoặc 2 biến thuộc kiểu chuỗi. Việc so sánh như vậy không mang lại bất kì ý nghĩa nào cả. Nếu muốn so sánh, hãy sử dụng các toán tử khác.
* Các phép so sánh giữa một biến với giá trị None luôn được thực hiện bằng toán tử is.

Để tránh gặp phải lỗi khi gán giá trị của một list cho một list khác, trước khi gán, bạn phải copy giá trị của list ban đầu

>>> lst = [1, 2, 3]

>>> lst\_copy\_1 = list(lst) # Cách 1

>>> lst\_copy\_1 **is** lst

**False**

>>>

>>> lst\_copy\_2 = lst[:] # Cách 2

>>> lst\_copy\_2 **is** lst

**False**

>>>

>>> lst\_copy\_3 = lst.copy() # Cách 3: dùng phương thức copy(), sẽ được Kteam đề cập đến trong các bài tiếp theo

>>> lst\_copy\_3 **is** lst

**False**

Thêm một trường hợp nữa bạn cần phải lưu ý, đó là lúc bạn cần copy giá trị của một ma trận

>>> lst = [[1, 2, 3], [4, 5, 6]]

>>> lst\_copy\_1 = lst[:]

>>> lst\_copy\_1[0] = 'ok'

>>> lst

[[1, 2, 3], [4, 5, 6]]

>>> lst\_copy\_1

['ok', [4, 5, 6]]

Đúng như chúng ta mong đợi. Thế nhưng…

>>> lst = [[1, 2, 3], [4, 5, 6]]

>>> lst

[[1, 2, 3], [4, 5, 6]]

>>> lst\_copy\_1 = lst[:]

>>> lst\_copy\_1[0][1] = 'Two'

>>> lst\_copy\_1

[[1, 'Two', 3], [4, 5, 6]]

>>> lst

[[1, 'Two', 3], [4, 5, 6]]

**Lưu ý:** nó chỉ sao chép các phần tử của List. Không hề sao chép các phần tử con của các phần tử nằm trong List. Do đó, nếu bạn thay đổi các phần tử trong List thì không sao, tuy nhiên nếu thay đổi phần tử con của các phần tử trong List, thì vấn đề lại xuất hiện.

>>> lst\_copy\_1 **is** lst

**False**

>>> lst\_copy\_1[0] **is** lst[0]

**True**

Đương nhiên, bạn vẫn giải quyết được, nhưng vì rườm rà khi không có vòng lặp. Do đó, chúng ta tạm dừng ở việc nhận biết. Còn phần giải quyết sẽ đợi nay mai. Khi võ công Xà Ngữ của chúng ta được nâng cao và tiếp cận với tuyệt kĩ vòng lặp.