8. Kiểu dữ liệu tuần tự

String có thể được xem như là tuần tự các kí tự. Có thể xem lại bàikiểu dữ liệu và biến để để hiểu sâu hơn.

Nhắc lại việc đánh chỉ mục (indexing) trong string. Các kiểu dữ liệu mà chúng tôi sẽ trình bày bên dưới sẽ sử dụng cách đánh chỉ mục này.

h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Kiểu “list” trong Python

Danh sách (list) là một loại dữ liệu linh hoạt nhất trong Python. Nó có thể được viết như một danh sách các giá trị được phân cách bởi dấu phẩy giữa các dấu ngoặc vuông. Danh sách (list) có mối quan hệ với các mảng của các ngôn ngữ lập trình khác như C, C ++ hoặc Java, nhưng list trong Python linh hoạt hơn nhiều so với các mảng "cổ điển". Ví dụ các mục trong danh sách không nhất thiết phải có cùng kiểu dữ liệu. Hơn nữa danh sách (list) có thể mở rộng khi chương trình đang chạy, trong khi trong C kích thước của một mảng đã được cố định tại thời gian biên dịch.

Ví dụ: languages = ["Python", "C", "C++", "Java", "Perl"]

Nếu một danh sách (list) lại là một phần tử của một một danh sách (list) khác, thì ta gọi nó là sublist. Tính chất như một list thông thường.

Ví dụ:

```
languages = ["Python", "C", "C++", "Java", "Perl"]

certificates = ['Master','CCNA']

cv = [languages, certificates]

print cv


Output:

[['Python', 'C', 'C++', 'Java', 'Perl'], ['Master', 'CCNA']]
```

Cập nhật hoặc xóa phần tử trong một “list”.

Chúng ta có thể thay đổi giá trị của một hoặc một vài phần tử trong “list” cho trước bằng phép gán giá trị mới tương ứng với index của phần tử đó trong “list”, hoặc có thể thêm 1 phần tử mới vào list, sử dụng hàm append(obj) như ví dụ sau:

```
>>> list = ['Python', 'Java', 1000, 2000]

>>> print list[1]

Java

>>> list[1] = 'C++'

>>> print list

['Python', 'C++', 1000, 2000]

>>> list.append(3000)

>>> print list

['Python', 'C++', 1000, 2000, 3000]
```

Chúng ta có thể xóa một phần tử trong list sử dụng cậu lệnh del nếu bạn biết vị trí (index) của phần tử đó hoặc sử dụng hàm remove(obj) nếu bạn chỉ biết giá trị của phần tử.

```
>>> list = ['Python', 'Java', 1000, 2000, 3000]

>>> del list[4]

>>> print list

['Python', 'C++', 1000, 2000]

>>> list.remove('C++')

>>> print list

['Python', 1000, 2000]
```

Kiểu “Tuples” trong Python

“Tuples” là một danh sách (list) không có khả năng thay đổi (immutable list), tức là một tuple không thể thay đổi bằng bất kỳ cách nào khi nó đã được tạo ra. Một tuple được định nghĩa tương tự như các danh sách, ngoại trừ tập hợp các phần tử được đặt trong dấu ngoặc đơn “()” thay vì dấu ngoặc vuông “[]”. Các quy tắc cho các chỉ mục giống như đối với các danh sách (list). Khi một tuple đã được tạo ra, bạn không thể thêm các phần tử vào hoặc loại bỏ các phần tử ra khỏi tuple vừa tạo ra.

Lợi ích của tuple ở đâu?

Tuple nhanh hơn danh sách (list).

Nếu bạn biết rằng một số dữ liệu không cần phải thay đổi, bạn nên sử dụng tuple thay vì danh sách (list) vì điều này bảo vệ dữ liệu của bạn chống lại các thay đổi ngẫu nhiên.

Tuple có thể được sử dụng như là key trong dictionary sẽ được nhắc trong phần tiếp theo, trong khi các danh sách (list) thì không thể.

Ví dụ dưới đây chỉ ra cách xác định một tuple và cách truy cập vào một tuple. Hơn nữa chúng ta có thể thấy rằng chúng ta gây ra một lỗi, nếu chúng ta cố gắng gán một giá trị mới cho một phần tử của một tuple:

```
tuple = ("Python", "C", "C++", "Java", "Perl")
tuple[1] = "PHP"

Output:

Traceback (most recent call last):

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

TypeError: 'tuple' object does not support item assignment
```

Chú ý:

Khi bạn khai báo một tập các phần tử phân tách nhau bởi dấu phảy nhưng chúng không được đặt trong bất kỳ kí tự định danh cho một kiểu dữ liệu cụ thể (ví dụ [] cho kiểu “list”, () cho kiểu “Tuple”…thì Python mặc định tập các phần tử đã khai báo là kiểu “tuple”

```
>>> ls = 'a', 'b', 1, 2

>>> print ls

('a', 'b', 1, 2)
```

Khai báo 1 tuple rỗng

```
>>> tup1 = ()

>>> tup1

()
```

Khai báo 1 tuple chỉ có một phần tử, bạn phải thêm dấu phảy sau phần tử

```

>>> tup1 = (50,)

>>> tup1

(50,)

>>> tup1 = (5)

>>> tup1

5
```

Kĩ thuật slicing (Kĩ thuật quan trọng trong xử lý dữ liệu)

Trong nhiều ngôn ngữ lập trình, nó có thể khá khó khăn để cắt một phần của một chuỗi và thậm chí khó khăn hơn nếu bạn muốn địa chỉ một "subarray". Python làm điều này dễ dàng bằng kĩ thuật gọi là slicing. Slicing thường được biết đến như là substring hoặc substr.

Khi bạn muốn trích xuất một phần của một chuỗi, hoặc một phần của một danh sách, bạn nên nghĩ ngay đến kĩ thuật slicing trong Python. Cú pháp đơn giản. Trên thực tế nó có vẻ giống như truy cập vào một phần tử đơn với một chỉ mục, nhưng thay vì chỉ một con số chúng ta có nhiều hơn, cách nhau bởi một dấu hai chấm ":". Chúng ta cần truyền là chỉ mục đại diện cho điểm bắt đầu và một chỉ mục cuối để chỉ rõ điểm kết thúc, trong nhiều trường hợp một hoặc cả hai có thể bị thiếu và chúng sẽ nhận giá trị mặc định. Tốt nhất nên nghiên cứu phương thức hoạt động của

slice bằng cách xem ví dụ:

```
languages = ["Python", "C", "C++", "Java", "Perl"]
print "languages[2:4]=", languages[2:4]
print "languages[:len(languages)] = ",languages[:len(languages)]
print "languages[:-1] = ",languages[:-1]

Output:

languages[2:4]= ['C++', 'Java']

languages[:len(languages)] = ['Python', 'C', 'C++', 'Java', 'Perl']

languages[:-1] = ['Python', 'C', 'C++', 'Java']
```

Slicing làm việc cả với 3 đối số. Cú pháp như sau, nếu s là kiểu dữ liệu tuần tự thì nó hoạt động như sau: s[begin:end:step]. Ví dụ:

```
languages = ["Python", "C", "C++", "Java", "Perl"]
print languages[:4:2]

Output:

['Python', 'C++']
```

Một số toán tử hay phương thức thường dùng khác trong dữ liệu tuần tự.

Hàm len() xác định chiều dài của một danh sách (list), một chuỗi hoặc tuple

```
len([['Python', 'C', 'C++', 'Java', 'Perl'], ['Master', 'CCNA']])

Output:

2
```

Tạo list mới từ nhiều list nhỏ qua toán tử ‘+’

```
lang1 = ['Python', 'C']
lang2 = ['C++', 'Java', 'Perl']
lang1 + lang2

Output

 ['Python', 'C', 'C++', 'Java', 'Perl']
```

Kiểm tra một phần tử có tồn tại trong list/tuple hay không, ta dùng toán tử ‘in’

```
abc = ["a","b","c","d","e"]
'b' in abc

Output:

True
```

Lập gấp n lần một list/tuple, ta dùng toán tử ‘*’. Ví dụ

```
abc = (1,2,3,4,5)
abc *3

Output:
```

```
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

Thận trọng với cách sử dụng toán tử ‘*’

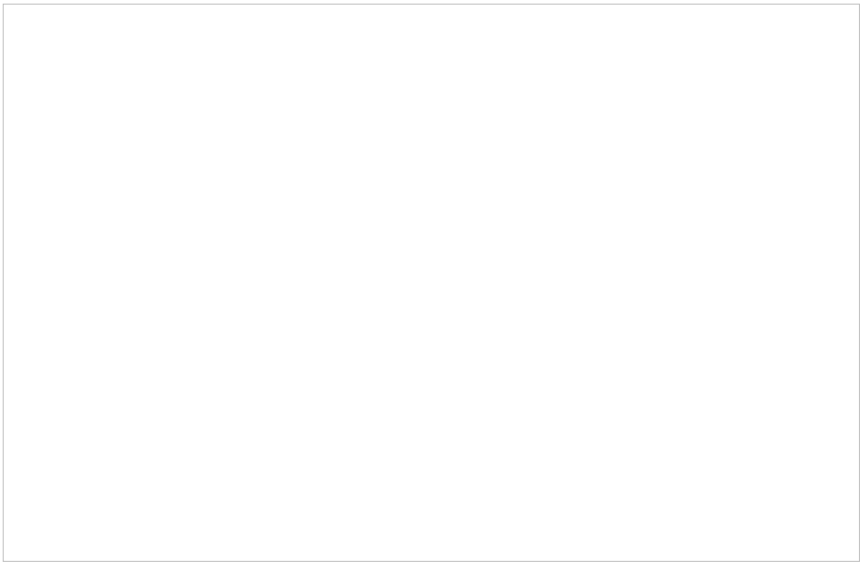
Ví dụ sau miêu tả cách hoạt động của toán tử ‘*’

```
x = ["a","b","c"]
y = [x] * 4
print "init, y= ", y
y[0][0] = "p"
print "changed, y= ", y

Output:

init, y= [['a', 'b', 'c'], ['a', 'b', 'c'], ['a', 'b', 'c'], ['a', 'b', 'c']]

changed, y= [['p', 'b', 'c'], ['p', 'b', 'c'], ['p', 'b', 'c'], ['p', 'b', 'c']]
```



Kết Luận

Trong bài này chúng tôi đã giới thiệu với các bạn thêm 2 kiểu dữ liệu mới trong Python là “list” và “tuple”. Các bạn cần nắm được cách khai báo cũng như các phương thức thường dùng với hai kiểu dữ liệu này. Trái với “list” thì “Tuple” lại là một danh sách không có khả năng cập nhật hay thay đổi giá trị của các phần tử trong nó, nhưng điều đó lại đem lại khả năng xử lý nhanh hơn cho “tuple”. Kĩ thuật slicing cũng được chúng tôi đề cập để giúp bạn có thể linh hoạt hơn trong việc xử lý dữ liệu trong Python.

Trong bài tới, chúng tôi sẽ tiếp tục giới thiệu đến các bạn thêm 2 kiểu dữ liệu nữa là set và Frozenset trong Python.