LIST COMPREHENSION

Comprehension là một cách nhanh chóng để có thể tạo một cấu trúc dữ liệu từ iterable. Nó có thể kết hợp với điều kiện và vòng lặp để rút gọn cú pháp. List comprehension sẽ giúp code của bạn Pythonic hơn.

1. Cú pháp và các ví dụ cơ bản:

CÚ PHÁP

[Expression for var in iterable]

- **Expression:** Biểu thức được thực hiện mỗi khi vòng lặp thực thi.
- **Var:** Một biến là một item trong iterable.
- **Iterable:** Collections chứa các object (list, tuple, str...)

1. Cú pháp và các ví dụ cơ bản:

VÍ DỤ

EXAMPLE

Copy từ một list đã có:

```
a = ["python", "apple", "28tech"]
b = [x for x in a]
print(b)
```

OUTPUT

['python', 'apple', '28tech']

EXAMPLE

Tạo một list giá trị bình phương của các phần tử của một list đã có:

```
a = [1, 2, 3, 4]
b = [x ** 2 for x in a]
print(b)
```

OUTPUT

[1, 4, 9, 16]

1. Cú pháp và các ví dụ cơ bản:

VÍ DỤ

EXAMPLE

Tạo một list từ range:

```
a = [x ** 3 for x in range(5)]
print(a)
```

OUTPUT

[0, 1, 8, 27, 64]

EXAMPLE

Tạo một list từ str:

```
s = "28tech"
a = [x for x in s]
print(a)
```

OUTPUT

['2', '8', 't', 'e', 'c', 'h']

1. Cú pháp và các ví dụ cơ bản:

VÍ DỤ

EXAMPLE

Tạo một list trị tuyệt đối của các phần tử của một list đã có:

```
a = [1, 2, -3, 4, -5, -10]
b = [abs(x) for x in a]
print(b)
```

OUTPUT

[1, 2, 3, 4, 5, 10]

EXAMPLE

Kết hợp hàm với list comprehension:

```
def digitSum(n):
    sum = 0
    while n != 0:
        sum += n % 10
        n //= 10
    return sum

a = [123, 31, 20, 503, 114]
b = [digitSum(x) for x in a]
print(b)
```

OUTPUT

[6, 4, 2, 8, 6]

2. List comp với câu lệnh if:

Khi sử dụng list comprehension bạn có thể sử dụng mệnh đề if để lọc dữ liệu phù hợp.

CÚ PHÁP

[Expression for var in iterable if_clause]

EXAMPLE

Lọc ra các số không âm

```
a = [1, 2, 3, -5, 3, -4, 0]
b = [x for x in a if x >= 0]
print(b)
```

OUTPUT

[1, 2, 3, 3, 0]

EXAMPLE

Lọc các số chẵn từ 0 tới n:

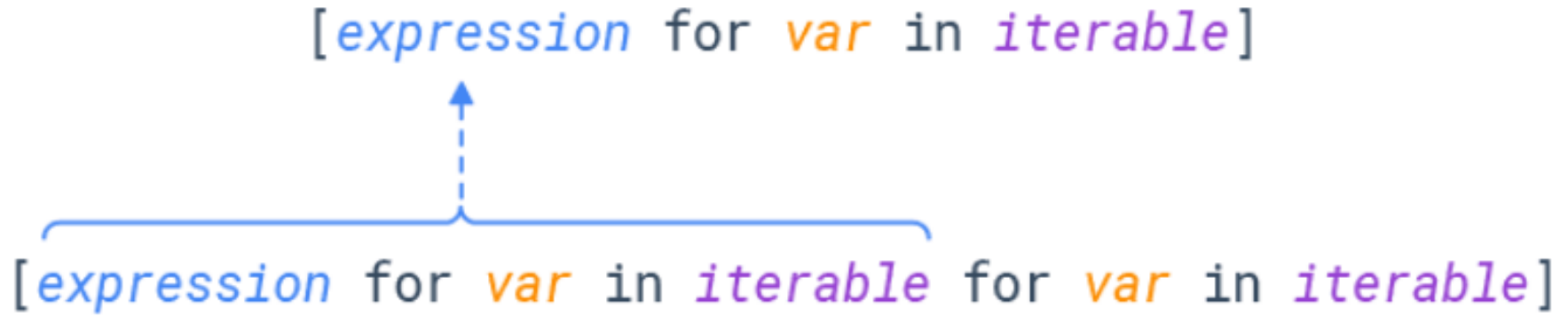
```
n = 10
a = [x for x in range(n) if x % 2 == 0]
print(a)
```

OUTPUT

[0, 2, 4, 6, 8]

3. Nested List comprehension:

Biểu thức bên trong list comp có thể là một list comp khác.



EXAMPLE

Đưa mọi phần tử trong nested list thành một list (flatten):

```
a = [[1, 2, 3], [4, 5], [6, 7, 8]]
b = [x for small_list in a for x in small_list]
print(b)
```

OUTPUT

[1, 2, 3, 4, 5, 6, 7, 8]

4. List comprehension với map + lambda:

Khi sử dụng các hàm có sẵn thì sử dụng map sẽ nhanh gọn hơn so với listcomp.

EXAMPLE

```
#Map
s = "28tech"
a = list(map(ord, s))
print(a)
#Listcomp
b = [ord(x) for x in s]
print(b)
```

OUTPUT

```
[50, 56, 116, 101, 99, 104]
[50, 56, 116, 101, 99, 104]
```

Tuy nhiên khi apply một hàm khác các hàm có sẵn thì sự kết hợp của listcomp với lambda sẽ ngắn gọn hơn so với sự kết hợp của lambda vs map.

EXAMPLE

```
a = range(5)
#Listcomp
b = [x ** 2 for x in a]
print(b)
#Map
c = list(map(lambda x : x ** 2, a))
print(c)
```

OUTPUT

```
[0, 1, 4, 9, 16]
[0, 1, 4, 9, 16]
```

5. List comprehension với filter:

Lọc ra những số chẵn bằng listcomp và filter kết hợp vs lambda.

EXAMPLE

```
a = range(10)
#listcomp
even = [x for x in a if x % 2 == 0]
print(even)
#map
c = list(filter(lambda x : x % 2 == 0, a))
print(c)
```

OUTPUT

```
[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```