LẬP TRÌNH PYTHON

Iteration And Iterables

hungdn@ptit.edu.vn

Nội dung

- Comprehensions
- Iteration & Iterators

1. Comprehensions

1. Kỹ thuật Comprehension

- **Cú pháp ngắn gọn để mô tả một list, set, hoặc dict**
- **Dễ đọc và hiểu**
- **Gần với ngôn ngữ tự nhiên**
- **Cú pháp:**

```
|| expr(item) for item in iterable if condition ||
```

- **Trong đó:**
  - `item` – bắt buộc – là đối tượng/biến/giá trị trong dữ liệu kiểu tập hợp/chuỗi
  - `iterable` – bắt buộc – là dữ liệu kiểu tập hợp/chuỗi cho phép lặp
  - `expr(item)` – không bắt buộc – có thể là một biểu thức hợp lệ bất kỳ
  - `if condition` – không bắt buộc – là điều kiện để lọc các phần tử

Example 1.1: List

- **Danh sách số lẻ nhỏ hơn 10**

```
# for version
>>> iters = range(10)
>>> odds = []
>>> for item in iters:
...     if item % 2 == 1:
...
odds.append(item)
...
>>> odds
[1, 3, 5, 7, 9]
```

- **List Example**
  - []

```
# Comprehension Version
>>> >>> iters = range(10)
>>> output = [item for item in iters if item % 2 == 1]
>>> output
[1, 3, 5, 7, 9]
```

Example 1.1: Set

- **Số lẻ nhỏ hơn 10**

```
# for version
>>> iters = range(10)
>>> odds = []
>>> for item in iters:
...     if item % 2 == 1:
...
odds.append(item)
...
>>> odds
[1, 3, 5, 7, 9]
```

- **Set Example**
  - {}

```
# Comprehension Version
>>> >>> iters = range(10)
>>> output = {item for item in iters if item % 2 == 1}
>>> output
{1, 3, 5, 7, 9}
```

1. (Cont) Dict

- Kỹ thuật Comprehension không làm việc trực tiếp với dữ liệu dict
- Sử dụng dict.items() để lấy tập key và value từ dữ liệu dict

- Dict Example
  - {key:value}

```
# Comprehension Version
>>> import os, glob
>>> from pprint import pprint as pp
>>> files = {os.path.realpath(p): os.stat(p).st_size for p
                                          in glob.glob("*.*")}
>>> pp(files)
>>> files_py = {name:length for name, length in files.items() if
str(name).endswith('.py')}
>>> files_big = {name:length for name, length in files.items() if
length > 500}
```

1. (Cont) Complex Expressions

- Các biểu thức Comprehension có thể phức tạp
- Để ngăn chặn sự khó đọc/hiểu của các biểu thức phức tạp -> Nên chia biểu thức phức tạp thành các hàm riêng biệt để dễ đọc

```
>>> from math import sqrt
>>> from pprint import pprint as pp
>>> def is_prime(x):
...     if x < 2:
...         return False
...     for i in range(2,
int(sqrt(x)) + 1):
...         if x % i == 0:
...             return False
...     return True
...
>>> [x for x in range(101) if
is_prime(x)]
```

Example 1.3: Filter

```
>>> from math import sqrt
>>> from pprint import pprint as pp
>>> def is_prime(x):
...     if x < 2:
...         return False
...     for i in range(2, int(sqrt(x)) + 1):
...         if x % i == 0:
...             return False
...     return True
...
>>> [x for x in range(101) if is_prime(x)]
>>> prime_square_divisors = {x*x: (1, x, x*x) for x in range(20) if
is_prime(x)}
>>> pp(prime_square_divisors)
```

2. iterable & iterator

2. Iteration Protocols

iterable

- Các kiểu dữ liệu tập hợp/chuỗi (iterable) có thể truyền vào hàm iter() để sinh ra một đối tượng iterator

Iterator

- Các đối tượng iterator có thể được truyền vào hàm next() để lấy ra giá trị tiếp theo trong chuỗi (sequence)

```
>>> iterable = ["Spring", "Summer",
"Autumn","Winter"]
>>> iterator = iter(iterable)
>>> next(iterator)
'Spring'
>>> next(iterator)
'Summer'
>>> next(iterator)
'Autumn'
>>> next(iterator)
'Winter'
>>> next(iterator)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

2. (Cont.) Generator

- Các iterables được định nghĩa sử dụng hàm

Đặc điểm:

- Lazy evaluation: kỹ thuật cho phép giá trị của hàm chỉ được thực hiện khi hàm được gọi
- Cho phép làm việc hiệu quả với loại dữ liệu không giới hạn
- Có thể xử lý như luồng (stream) phần tử và kết hợp thành các luồng (pipelines) phức tạp hơn

Cách tạo Generator

- C1: Sử dụng hàm generator
- C2: Sử dụng biểu thức generator (generator expression) – Được giới thiệu từ PEP 289 , còn được gọi với tên gọi là hàm generator ẩn danh (anonymous generator function)

2. (Cont.) Generator Function

Từ khóa Yield

- **Hàm Generator phải chứa ít nhất một lệnh yield**
- **Hàm Generator có thể chứa cả câu lệnh return**

Một số ứng dụng:

- **Đọc dữ liệu Sensor (IoT)**
- **Chuỗi toán học**
- **File lớn**

```
# Generator function version
>>> def all_odd():
...     n = 1
...     while True:
...             yield n
...             n += 2
...
>>> for odd in all_odd():
...     print(odd)
```

Example 2.1 Generator Function

- **Số fibonaci thứ N**

```
# Non version
>>> def fib_non(n):
...     f2, f1, i = 1, 1, 1
...     while(i < n):
...             f2, f1, i = f2
+ f1, f2, i+1
...     return f2
...
>>> for i in range(10):
...     print(fib_non(i))
...
```

```
# Generator function version
(finite)
>>> def fibo(n):
...     f2, f1, i = 1, 0, 1
...     yield f2
...     while(i < n):
...             f2, f1, i = f2
+ f1, f2, i+1
...             yield f2
...
>>> for f in fibo(10):
...     print(f)
```

Example 2.1 Generator Function

- **Số fibonaci thứ N**

```
# Generator function version
(infinite)
>>> def fibo():
...     f2, f1 = 1, 0
...     yield f2
...     While True:
...             f2, f1 = f2 +
f1, f2
...             yield f2
...
>>> for f in fibo():
...     print(f)
```

Example 2.1: Generator Expression

- Tương tự như cú pháp của list comprehension nhưng sử dụng ngoặc tròn (()) thay vì ngoặc vuông ([])

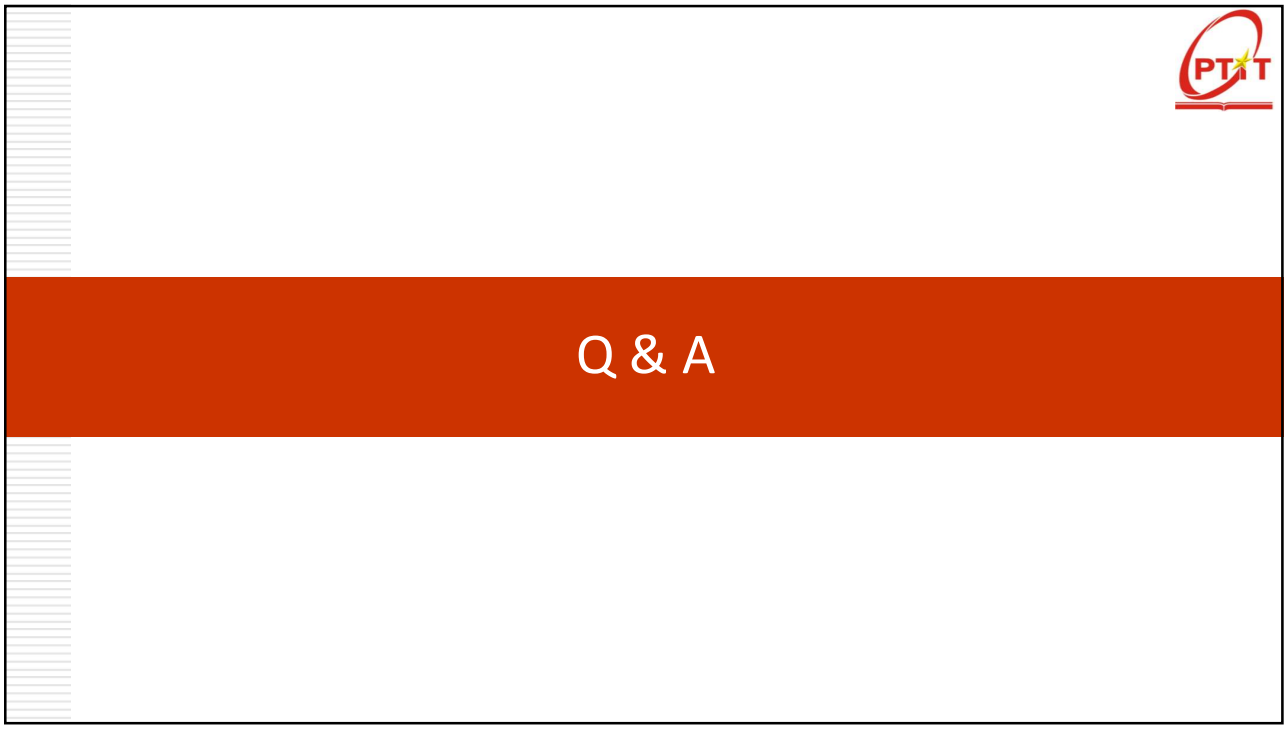
```
>>> list = [1,2,3,4,5]
>>> [x**2 for x in list]
[1, 4, 9, 16, 25]
>>> (x**2 for x in list)
<generator object <genexpr> at 0x0000020F28FEF1B0>
```

Tóm tắt

- **Kỹ thuật Comprehension**
- **Generator Function/Expression**

Bài tập

- **Các bài tập liên quan tới dữ liệu tập hợp/chuỗi sử dụng kỹ thuật Comprehension**
- **Sử dụng Generator Function và Generator Expresion cho các bài tập**
- **Python cung cấp nhiều thư viện để làm việc với iterator. Ví dụ: itertools**
  - isslice()
  - any()
  - all()
- **Tạo danh bạ sinh viên {studentCode: studentName} từ file students.txt**
- **Sử dụng Generator để đọc file .csv lớn và demo**


Q & A