Chương 3: Danh sách

Lập trình với Python

ThS. Đinh Xuân Trường

truongdx@ptit.edu.vn

Posts and Telecommunications
Institute of Technology
Faculty of Information Technology 1

CNTT1
Học viện Công nghệ Bưu chính Viễn thông

August 15, 2021

Mục tiêu

1. Nắm được khái niệm danh sách và các phần tử trong danh sách
2. Hiểu được cách tổ chức danh sách, duyệt danh sách và làm việc với các phần tử trong danh sách
3. Áp dụng làm việc với danh sách

Nội dung

Trong Python:

- Ngoặc vuông ([ ]) chỉ định một danh sách
- Các phần tử trong danh sách được phân tách bởi dấu phẩy (,).

$$name\_list = [element\_1, element\_2,..., element\_n]$$

Ví dụ danh sách chứa các loại xe đạp khác nhau:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
```

Để truy cập một phần tử trong danh sách:

- Hãy viết tên của danh sách theo sau là chỉ mục.
- Chỉ mục của phần tử được đặt trong dấu ngoặc vuông.

name_list[index_i]

Ví dụ lấy chiếc xe đạp đầu tiên trong danh sách bicycles:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
```

```
Trek
```

- Python chỉ trả về phần tử đó mà không có dấu ngoặc vuông.

Để truy cập một phần tử trong danh sách:

- Chỉ mục danh sách bắt đầu từ 0.
- Để truy cập phần tử thứ tư trong danh sách, chỉ mục là 3.

```
bicycles = ['trek', 'cannondale',
'redline', 'specialized']
print(bicycles[1])
print(bicycles[3])
```

```
cannondale
specialized
```

- Để truy cập phần tử cuối cùng trong danh sách, dùng chỉ mục -1

```
bicycles = ['trek', 'cannondale',
'redline', 'specialized']
print(bicycles[-1])
```

```
specialized
```

Có thể sử dụng các giá trị riêng lẻ từ một danh sách giống như cách ta làm với bất kỳ biến nào khác.

- Ví dụ: ta có thể sử dụng f-string để tạo một thông báo dựa trên giá trị từ một danh sách.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
message = f"My first bicycle was a {bicycles[0].title()}."
print(message)
```

```
My first bicycle was a Trek.
```

Bài tập vận dụng

Hãy viết các chương trình ngắn để có trải nghiệm trực tiếp với danh sách của Python.

1. **Names**: Lưu tên một vài người bạn vào trong một danh sách gọi là names. In ra tên mỗi người bằng cách truy cập phần tử trong danh sách.
2. **Greetings**: Bắt đầu với danh sách trong Bài tập 3-1, nhưng thay vì chỉ in tên của từng người, hãy in một thông điệp cho họ. Nội dung của mỗi thông điệp phải giống nhau, nhưng mỗi thông điệp phải được cá nhân hóa với tên của người đó.
3. **Your Own List**: Hãy nghĩ về phương tiện đi lại yêu thích của bạn, chẳng hạn như xe máy hoặc ô tô và lập danh sách lưu trữ một số ví dụ. Sử dụng danh sách để in một loạt các câu về những mục này, chẳng hạn như "Tôi muốn sở hữu một chiếc xe máy Honda".

Thêm, sửa, xóa các phần tử

- Hầu hết các danh sách tạo sẽ là động
- Ta sẽ xây dựng một danh sách và sau đó thêm và xóa các phần tử khỏi nó khi chương trình chạy

Ví dụ, ta có thể tạo một trò chơi trong đó người chơi phải bắn quái vật trên bầu trời.

- Lưu trữ nhóm nhân vật ban đầu trong một danh sách
- Xóa một nhân vật trong danh sách mỗi khi nó bị bắn hạ
- Xây dựng một danh sách, sau đó thêm và xóa các phần tử khi chương trình chạy
- Mỗi lần một nhân vật mới xuất hiện trên màn hình thêm nó vào danh sách
- Danh sách quái vật sẽ tăng lên và giảm đi trong suốt quá trình

Để thay đổi một phần tử:

- Sử dụng tên của danh sách theo sau bằng chỉ mục của phần tử mong muốn thay đổi
- Cung cấp giá trị thay đổi

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles[0] = 'ducati'
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

- Thêm phần tử vào cuối danh sách: dùng phương thức append()

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles.append('ducati')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha', 'suzuki', 'ducati']
```

- Dùng append() để tạo danh sách động

```
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
```

- Thêm một phần tử mới ở bất kỳ vị trí nào trong danh sách bằng cách sử dụng phương thức insert()

```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(0, 'ducati')
print(motorcycles)

['ducati', 'honda', 'yamaha', 'suzuki']
```

- Phương thức insert() tạo ra một khoảng trống tại giá trị 0 và lưu giá trị 'ducati' tại vị trí đó. Hành động này dịch chuyển tất cả các phần tử còn lại một vị trí sang phải.

- Xóa phần tử sử dụng lệnh del khi biết vị trí của phần tử cần xóa trong danh sách

```
motorcycles = ['honda', 'yamaha', 'suzuki']      ['honda', 'yamaha', 'suzuki']
print(motorcycles)                               ['yamaha', 'suzuki']
del motorcycles[0]
print(motorcycles)
```

```
motorcycles = ['honda', 'yamaha', 'suzuki']      ['honda', 'yamaha', 'suzuki']
print(motorcycles)                               ['honda', 'suzuki']
del motorcycles[1]
print(motorcycles)
```

- Trong cả hai ví dụ trên, chúng ta không thể truy cập vào được phần tử bị xóa khỏi danh sách sau khi sử dụng lệnh del

- Phương thức pop() loại bỏ mục cuối cùng trong danh sách
- Cho phép ta làm việc với mục đó sau khi loại bỏ nó

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
popped_motorcycle = motorcycles.pop()
print(motorcycles)
print(popped_motorcycle)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

- Sử dụng pop() để xóa một mục khỏi bất kỳ vị trí nào trong danh sách bằng cách bao gồm chỉ mục của mục mà ta muốn xóa:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
first_owned = motorcycles.pop(0)
print(f"The first motorcycle I owned was a {first_owned.title()}.")
```

```
The first motorcycle I owned was a Honda.
```

- Khi ta muốn xóa một mục khỏi danh sách và không sử dụng mục đó theo bất kỳ cách nào, hãy sử dụng câu lệnh del;
- Nếu ta muốn sử dụng phần tử khi ta xóa nó, hãy sử dụng phương thức pop().

- Nếu ta chỉ biết giá trị của phần tử muốn xóa, ta có thể sử dụng phương thức remove().

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
too_expensive = 'ducati'
motorcycles.remove(too_expensive)
print(motorcycles)
print(f"\nA {too_expensive.title()} is too expensive for me.")
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
A Ducati is too expensive for me.
```

Bài tập vận dụng

Hãy viết các chương trình ngắn để có trải nghiệm trực tiếp với danh sách của Python.

1. **Guest List**: Nếu bạn có thể mời ai đó đi ăn tối, bạn sẽ mời ai? Lập danh sách bao gồm ít nhất ba người bạn muốn mời đi ăn tối. Sau đó, sử dụng danh sách để in tin nhắn cho từng người, mời họ đi ăn tối.
2. **Changing Guest List**: Bạn vừa nghe nói rằng một trong những khách của bạn không thể thực hiện bữa tối, vì vậy bạn cần gửi một lời mời mới. Bạn sẽ phải nghĩ đến người khác để mời.
    - Bắt đầu với chương trình từ Bài tập 1. Thêm một lệnh print () vào cuối chương trình ghi rõ tên của khách không thể tham gia
    - Sửa đổi danh sách, thay thế tên của khách không thể tham gia vào danh sách bằng tên của người mới mà bạn đang mời.
    - In bộ thư mời thứ hai, cho mỗi người vẫn còn trong danh sách của bạn.

Bài tập vận dụng (cont.)

3. **More Guests**:Bạn vừa tìm thấy một bàn ăn tối lớn hơn, vì vậy hiện có nhiều không gian hơn. Hãy nghĩ đến việc mời thêm ba vị khách đến ăn tối.
   - Bắt đầu với chương trình của bạn từ Bài tập 1-2 hoặc Bài tập 3-5. Thêm một cuộc gọi print () vào cuối chương trình của bạn để thông báo cho mọi người rằng bạn đã tìm thấy một bàn ăn tối lớn hơn.
   - Sử dụng insert () để thêm một khách mới vào đầu danh sách
   - Sử dụng insert () để thêm một khách mới vào giữa danh sách
   - Sử dụng append () để thêm một khách mới vào cuối danh sách
   - In một bộ thư mời mới, một bộ cho mỗi người trong danh sách
4. **Shrinking Guest List**: Bạn vừa phát hiện ra rằng bàn ăn tối mới của bạn sẽ không đến kịp cho bữa tối và bạn chỉ có chỗ cho hai khách.
   - Bắt đầu với chương trình từ Bài tập 3. Thêm một dòng mới in thông báo rằng bạn chỉ có thể mời hai người ăn tối.

Bài tập vận dụng (cont.)

- Sử dụng pop () để xóa từng khách khỏi danh sách của bạn cho đến khi chỉ còn lại hai tên trong danh sách của bạn. Mỗi khi bạn bật tên khỏi danh sách của mình, hãy in một tin nhắn cho người đó để họ biết rằng bạn rất tiếc vì không thể mời họ đi ăn tối.
- In tin nhắn cho từng người trong số hai người vẫn còn trong danh sách của bạn, cho họ biết họ vẫn được mời.
- Sử dụng del để xóa hai tên cuối cùng khỏi danh sách của bạn, để bạn có một danh sách trống. In danh sách của bạn để đảm bảo rằng bạn thực sự có một danh sách trống ở cuối chương trình của mình

5. **No even numbers**:Tạo một danh sách gồm 10 chữ số sau đó loại bỏ toàn bộ chữ số chẵn. Thử nghiệm với một danh sách toàn số chẵn.

- Thông thường, danh sách sẽ được tạo theo một thứ tự không thể biết trước, bởi vì không thể kiểm soát thứ tự mà người dùng cung cấp dữ liệu của họ.
- Cần trình bầy thông tin theo một trật tự nào đó.
- Cần giữ nguyên thứ tự ban đầu của dữ liệu, đôi khi lại cần thay đổi thứ tự theo lúc đầu.
- Python cung cấp một số cách khác nhau để thay đổi thứ tự tùy theo tình huống

- Phương thức sort() của Python giúp sắp xếp danh sách:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort()
print(cars)
```

```
['audi', 'bmw', 'subaru', 'toyota']
```

- Sắp xếp danh sách này theo thứ tự bảng chữ cái ngược lại bằng cách chuyển đổi số reverse = True vào phương thức sort()

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True)
print(cars)
```

```
['toyota', 'subaru', 'bmw', 'audi']
```

- Hàm sorted() cho phép hiển thị danh sách theo một thứ tự cụ thể nhưng không ảnh hưởng đến thứ tự thực tế của danh sách.

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print("Here is the original list:")
print(cars)
print("\nHere is the sorted list:")
print(sorted(cars))
print("\nHere is the original list again:")
print(cars)
```

```
Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']
Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']
Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

- Hàm sorted() cũng có thể chấp nhận đối số reverse = True nếu ta muốn hiển thị danh sách theo thứ tự bảng chữ cái ngược lại.

- Để đảo ngược thứ tự ban đầu của danh sách, ta có thể sử dụng phương thức reverse()

```
cars = ['bmw', 'audi', 'toyota', 'subaru']

print(cars)

cars.reverse()

print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']

['subaru', 'toyota', 'audi', 'bmw']
```

- Lưu ý rằng phương thức reverse() không sắp xếp theo thứ tự alphabet, nó chỉ đơn thuần là đảo ngược thứ tự của các phần tử trong danh sách hiện tại

- Tìm độ dài của danh sách bằng cách sử dụng hàm len().

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> len(cars)
4
```

- Chú ý: Python đếm các mục trong danh sách bắt đầu bằng một, vì vậy ta sẽ không gặp phải bất kỳ lỗi nào khi xác định độ dài của danh sách

Bài tập vận dụng

1. **Seeing the World**: Hãy nghĩ về ít nhất năm địa điểm trên thế giới mà bạn muốn đến.
   - Lưu trữ các địa điểm trong một danh sách. Đảm bảo danh sách không theo thứ tự bảng chữ cái.
   - In ra danh sách vừa tạo
   - Sử dụng sorted () để in danh sách của bạn theo thứ tự bảng chữ cái mà không sửa đổi danh sách thực.
   - Chứng tỏ rằng danh sách vẫn theo thứ tự ban đầu bằng cách in nó.
   - Sử dụng sorted () để in danh sách của bạn theo thứ tự ngược bảng chữ cái mà không làm thay đổi thứ tự của danh sách ban đầu và chứng mình điều đó bằng cách in hai danh sách.
   - Sử dụng reverse () để thay đổi thứ tự danh sách của bạn. In danh sách để hiển thị rằng thứ tự của nó đã thay đổi.
   - Sử dụng reverse () để thay đổi lại thứ tự danh sách của bạn. In danh sách để hiển thị danh sách trở lại thứ tự ban đầu.

Bài tập vận dụng (cont.)

- Sử dụng sort () để thay đổi danh sách của bạn để danh sách được lưu trữ theo thứ tự bảng chữ cái. In danh sách để hiển thị rằng thứ tự của nó đã được thay đổi.
- Sử dụng sort () để thay đổi danh sách của bạn để danh sách được lưu trữ theo thứ tự bảng chữ cái ngược lại. In danh sách để hiển thị rằng thứ tự của nó đã thay đổi.

2. **Dinner Guests**: Tiếp tục với bài Dinner Guest sử dụng len () để in thông báo cho biết số người mà bạn đang mời đi ăn tối.

Tránh lỗi chỉ mục

Giả sử danh sách có 3 phần tử, ta yêu cầu in ra phần tử thứ tư, Python sẽ thông báo lỗi chỉ mục:

- Tìm độ dài của danh sách bằng cách sử dụng hàm len().

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[3])

Traceback (most recent call last):
File "motorcycles.py", line 2, in <module>
print(motorcycles[3])
IndexError: list index out of range
```

- Nếu lỗi chỉ mục xảy ra trong chương trình, hãy thử điều chỉnh chỉ mục đang được yêu cầu một đơn vị.

Bất cứ khi nào ta muốn truy cập phần tử cuối cùng trong danh sách, hãy sử dụng chỉ mục (-1)

- Tìm độ dài của danh sách bằng cách sử dụng hàm len().

Tổng kết chương

Nội dung trong chương

- Định nghĩa danh sách
- Thêm, sửa và xóa các phần tử trong danh sách
- Tổ chức danh sách
- Tránh lỗi chỉ mục

Tiếp theo

- Lặp qua toàn bộ danh sách, lập danh sách số, làm việc với một phần của danh sách và kiểu dữ liệu Tuple

Bài tập vận dụng

1. **Random List**: Viết chương trình thực hiện các yêu cầu sau:
   - Xóa phần tử cuối cùng của danh sách
   - Thêm một giá trị bất kỳ vào vị trí thứ 4 của danh sách
   - Thay đổi giá trị của phần tử thứ nhất bằng "Python"
2. **Swap Position**: Viết chương trình hoán đổi phần tử đầu tiên và vị trí cuối cùng của danh sách.
3. **Sum & Average**: Cho một danh sách gồm các phần tử có giá trị bất kỳ. Tính tổng và trung bình cộng của các phần tử có giá trị là các số có trong danh sách.
4. **Squared Numbers** Tính bình phương các phần tử trong danh sách và sắp xếp theo thứ tự giảm dần.
5. **Unique Value** Tính số lượng các giá trị duy nhất có trong danh sách. Ví dụ: array = [1, "a", 34, "a", "b", 1, "c"]. Kết quả: 5

Lặp qua toàn bộ danh sách

- Một tác vụ quan trọng mà chương trình thường phải làm là chạy qua toàn bộ các phần tử trong danh sách và thực hiện các việc giống nhau.
- Khi ta muốn thực hiện cùng một hành động với mọi mục trong danh sách, ta có thể sử dụng vòng lặp *for* của Python.

```
1 cars = ['Audi', 'Ford', 'Mazda']
2 cars
```

```
['Audi', 'Ford', 'Mazda']
```

```
1 # loop over cars list
2
3 for car in cars:
4     print(car)
5
```

```
Audi
Ford
Mazda
```

Lặp qua toàn bộ danh sách

- Lặp qua toàn bộ danh sách có nhiều trường hợp cần quan tâm đến chỉ mục của các phần tử.
- Sử dụng vòng lặp *for* với hàm *enumerate()*.

```
1  # we need the index -> enumerate()
2
3  for index, car in enumerate(cars):
4      print("{0} - {1}".format(index, car))
5
```

```
0 - Audi
1 - Ford
2 - Mazda
```

- Bất kỳ dòng lệnh nào sau vòng lặp *for* không được thụt lề đều được thực thi một lần mà không lặp lại.
- Trong Python, các khối lệnh sẽ được thụt lề theo các cấp độ khác nhau và cùng khối lệnh sẽ cùng cấp độ thụt lề.

```python
def two_levels_sum(p_list):

    summation = 0

    for item in p_list:

        # type check -> list
        if type(item) == list:
            # item is a list -> loop over its elements
            for inner_item in item:
                if str(inner_item).isdigit():
                    summation += inner_item

        # type check -> int
        elif str(item).isdigit():
            summation += item

    return summation
```

Cần tránh một số lỗi khi sử dụng vòng lặp *for*:

- Quên thụt lề
- Quên thụt lề các dòng bổ sung
- Thụt lề không cần thiết
- Quên dấu hai chấm ':'

```
1 # loop over cars list
2
3 for car in cars
4     print(car)
5
```

```
File "<ipython-input-9-13199f7922f8>", line 3
  for car in cars
                ^
SyntaxError: invalid syntax
```

Lập danh sách số

Danh sách số là danh sách lưu trữ tập hợp số, Python cung cấp hàm *range()* với ba tham số:

$$range\_list = range(begin, end, step)$$

- begin: Phần tử bắt đầu của danh sách
- end: Phần tử cuối
- step: khoảng cách giữa hai phần tử liên tiếp

```
1 range(1, 10, 2)
```

```
range(1, 10, 2)
```

```
1 for i in range(1, 10, 2):
2     print(i)
```

```
1
3
5
7
9
```

Python cung cấp các hàm thống kê như trả về các giá trị tối thiểu, tối đa và tổng của một danh sách số:

```
>>> digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> min(digits)
0
>>> max(digits)
9
>>> sum(digits)
45
```

Comprehension là một cách làm đơn giản và gọn lại các vòng lặp khi hiểu được bản chất của việc xử lý phần tử trong danh sách.
Ví dụ: Tạo một danh sách các số bình phương từ 1 đến 10:

```python
# classical -> loop

squares = []

for i in range(1, 11):
    squares.append(i**2)

print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```python
# comprehension

squares_comp = [i**2 for i in range(1, 11)]

print(squares_comp)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Làm việc với một phần của danh sách hay trong Python gọi là lát cắt (slice):

$$slice\_list = name\_list[index\_begin : index\_end]$$

```
1 my_list = ['a', 'b', 'c', 'd', 'e', 'f']
2 print(my_list)
3
4 # we want elements from index 1 to 4
5
6 my_list[1:4]
```

```
['a', 'b', 'c', 'd', 'e', 'f']

['b', 'c', 'd']
```

```
1 # ommit 0 for start
2
3 my_list[:4]
```

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

- Nếu ta bỏ qua chỉ mục đầu tiên trong một slice, Python sẽ tự động bắt đầu slice ở đầu danh sách.

- Nếu muốn tất cả các phần tử từ thứ 3 đến cuối cùng, ta có thể bắt đầu với 2 và bỏ qua chỉ mục thứ hai.

```
1 # ommit for end
2
3 my_list[2:]
```

```
['c', 'd', 'e', 'f']
```

- Chỉ mục âm để xuất các giá trị cuối trong danh sách.

```
1 # ommit for end
2
3 my_list[-2:]
```

```
['e', 'f']
```

Để sao chép một danh sách, chúng ta có thể tạo một phần bao gồm toàn bộ danh sách gốc bằng cách bỏ qua chỉ mục đầu tiên và chỉ mục thứ hai ([:]).

```
1 # get all elements of list
2
3 my_list[::]
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

```
1 my_list_copy = my_list[::]
2 my_list_copy
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

Một số trường hợp khi lấy các chỉ mục chẵn, lẻ trong danh sách hoặc đảo ngược danh sách:

```
1 # get the elements with even index
2 # 0, 2, 4, ...
3
4 my_list[::2]
```

```
['a', 'c', 'e']
```

```
1 # get the elements with odd index
2 # 1, 3, 5 ...
3
4 my_list[1::2]
```

```
['b', 'd', 'f']
```

```
1 # copy the reverse of the list
2
3 my_list_copy_reverse = my_list[::-1]
4 my_list_copy_reverse
```

```
['f', 'e', 'd', 'c', 'b', 'a']
```

Tuple

Tuple là kiểu dữ liệu dạng danh sách mà các phần tử của nó không thể thay đổi được.
Python đề cập đến các giá trị không thể thay đổi dưới dạng bất biến, và một danh sách không thay đổi được gọi là một tuple.

- Ngoặc đơn ( ) chỉ định một danh sách kiểu tuple
- Các phần tử trong danh sách được phân tách bởi dấu phẩy (,).

$$name\_list = (element\_1, element\_2, ..., element\_n)$$

```
1 t2 = (1, 2, 4, 6, 8, 20)
2 t2
```

```
(1, 2, 4, 6, 8, 20)
```

```
1 type(t2)
```

```
tuple
```

Tuple

Các phép toán trên Tuple để truy cập đến các phần tử trong tuple tương tự như với danh sách (list):

- Lặp qua các phần tử dùng vòng lặp for

```
dimensions = (200, 50)
for dimension in dimensions:
    print(dimension)
```

```
200
50
```

chúng ta không thể sửa đổi tuple, nhưng ta có thể chỉ định một giá trị mới cho một biến đại diện cho một tuple:

- Lặp qua các phần tử dùng vòng lặp for

```
dimensions = (200, 50)
print("Original dimensions:")
for dimension in dimensions:
    print(dimension)
    dimensions = (400, 100)
print("\nModified dimensions:")
for dimension in dimensions:
    print(dimension)
```

```
Original dimensions:
200
50

Modified dimensions:
400
100
```

Tổng kết chương

Nội dung trong chương

- Lặp qua toàn bộ danh sách
- Lập danh sách số
- Làm việc với một phần của danh sách
- Tuple

Tiếp theo

- Câu lệnh điều kiện If