List, Tuples

Khoá học: Python căn bản

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài “Hàm và module”

Tóm tắt lại các phần đã học từ bài “Hàm và module”

Mục tiêu

- Trình bày được khái niệm List
- Mô tả được cú pháp khai báo và sử dụng List
- Mô tả được cách sử dụng vòng lặp for/in để duyệt List
- Khai báo và sử dụng được List
- Sử dụng for-in để làm việc với List
- Trình bày được khái niệm Tuples
- Mô tả được cú pháp khai báo và sử dụng Tuples
- Khai báo và sử dụng được Tuples

Thảo luận

Mảng là gì

List và Mảng trong Python

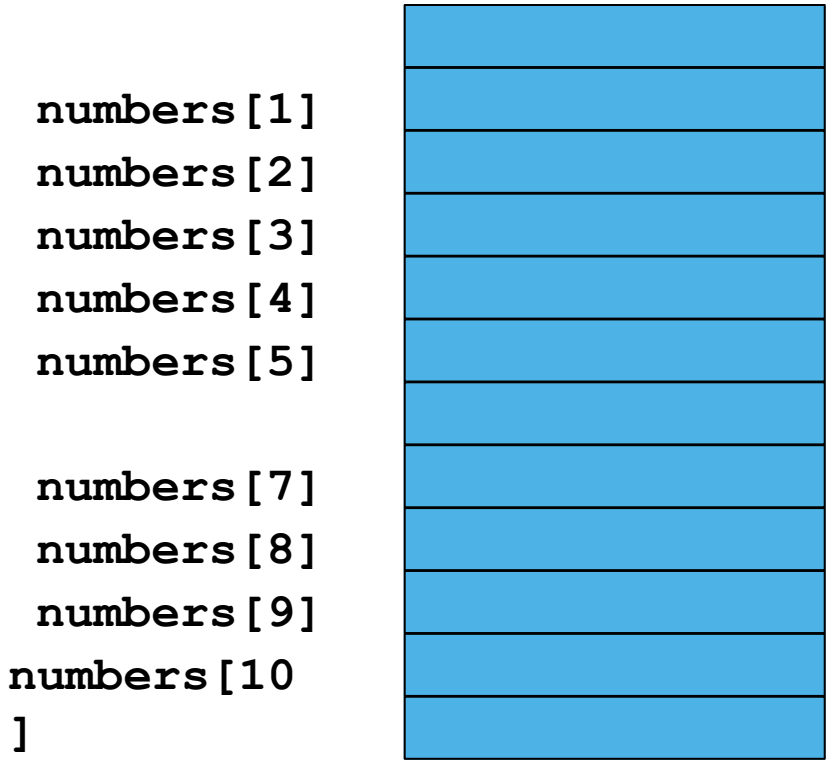
Các thao tác với List

Lưu trữ nhiều dữ liệu

- Trong các chương trình máy tính, có những trường hợp chúng ta phải lưu trữ rất nhiều dữ liệu
- Chẳng hạn, cần lưu trữ danh sách tên của hàng trăm sinh viên, cùng điểm thi của từng người
- Việc khai báo hằng trăm biến để lưu trữ các dữ liệu kiểu này rất mất thời gian và không thực tế
- Python và phần lớn các ngôn ngữ khác hỗ trợ một cấu trúc dữ liệu là mảng (danh sách) để giải quyết các tình huống này

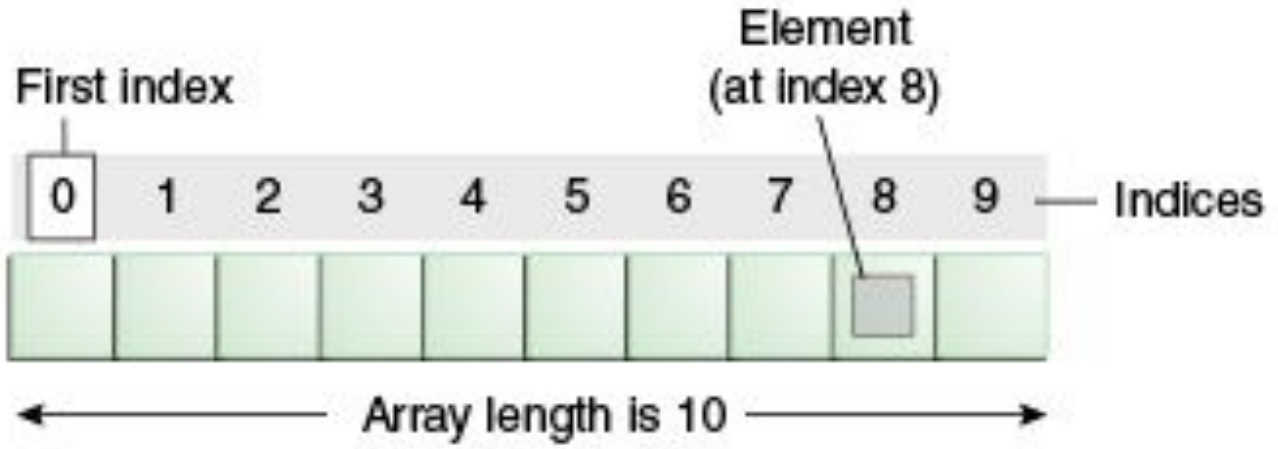
Mảng là gì

- Mảng là một loại biến đặc biệt, có thể lưu được nhiều giá trị thay vì chỉ một giá trị như các biến thông thường
- Mỗi giá trị trong mảng được gọi là một phần tử
- Các phần tử được lưu trữ ở các vị trí kế tiếp nhau trong bộ nhớ
- Chẳng hạn, mảng numbers chứa 12 phần tử là các số nguyên



Các khái niệm của mảng

- Tên mảng: Tuân thủ theo quy tắc đặt tên của biến
- Phần tử: Các giá trị được lưu trữ trong mảng
- Chỉ số: Vị trí của các phần tử trong mảng. Chỉ số bắt đầu từ 0.
- Độ dài: Số lượng các phần tử của mảng



List và Mảng

- Trong Python list là một dạng dữ liệu cho phép lưu trữ nhiều kiểu dữ liệu khác nhau và truy xuất các phần tử bên trong nó thông qua vị trí của phần tử đó trong list
- List có thể được coi là một ***mảng tuần tự*** như trong các ngôn ngữ khác
- List không nhất thiết phải đồng nhất, điều này khiến nó trở thành một công cụ mạnh mẽ nhất trong Python.
- Một list đơn có thể bao gồm các loại Datatypes như Integers, Strings cũng như Objects.
- List có thể thay đổi được ngay cả sau khi được tạo.

Khai báo List

- Cú pháp:

```
array_name = [item1, item2, ...];
```

- Ví dụ:

```
array_name = ["Saab", "Volvo", "BMW"];
```

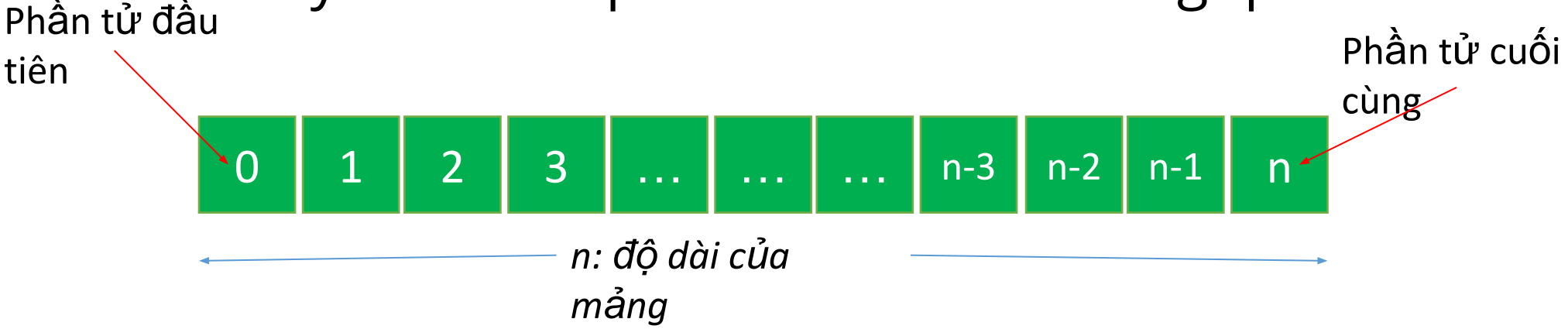
```
listNumber = [1, 2, 3, 4, 5];
```

Demo

Khai báo List

Phần tử và chỉ số của List

- Mỗi phần tử (item) được xác định bằng một số thứ tự còn gọi là chỉ số (index) duy nhất trong List
- Chỉ số là một số nguyên dương
- Chỉ số của phần tử đầu tiên là 0
- Chỉ số của phần tử cuối cùng là n – 1, trong đó n là độ dài của List
- Có thể truy xuất đến phần tử của List thông qua chỉ số



Truy xuất phần tử trong List

- Mỗi phần tử trong List được thao tác giống như một biến
- Truy xuất các phần tử thông qua chỉ số đặt trong dấu []
- Ví dụ, gán giá trị cho một phần tử đầu tiên của List

```
cars[0] = "Opel";
```

- Ví dụ, lấy giá trị của một phần tử đầu tiên của List

```
name = cars[0];
```

Vòng lặp for-in

- Vòng lặp for-in (còn gọi là *enhanced for*) được sử dụng để duyệt qua các phần tử của một collection, chẳng hạn như mảng, danh sách…
- Cú pháp:

```
for element in collection:
    statement(s)
```

Trong đó:

- element: Biến đại diện lần lượt cho từng phần tử của collection trong mỗi lần lặp
- collection: đối tượng cần lặp

for-in: Ví dụ

- Duyệt qua các phần tử của một List:

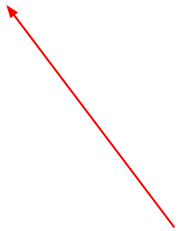
```
array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

for element in array:
    print("element = " + array[element]
```

Các thao tác với List

Độ dài của List

- Hàm *len* trả về độ dài của mảng
- Ví dụ:

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
print(len(fruits));
```



Có giá trị là 4

Thêm phần tử vào cuối List: append()

- Phương thức append() cho phép thêm phần tử vào cuối mảng
- Ví dụ:

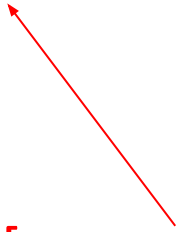
```
fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.append("Lemon");
```

Phần tử mới được thêm vào tại vị trí 5 (index 4)

Nối các phần tử

- Sử dụng toán tử + để nối 2 list với nhau
- Ví dụ:

```
fruits1 = ["Banana", "Orange"];
fruits2 = ["Apple", "Mango"];

print(fruits1 + fruits2);
```



```
["Banana", "Orange", "Apple", "Mango"]
```

Xoá phần tử trong List: del, remove(), pop()

- Có thể sử dụng từ khoá del hoặc phương thức remove() hoặc pop() để xoá một phần tử trong List
- Ví dụ:

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
del fruits[1]

print(fruits);
```

```
["Banana", "Apple", "Mango"]
```

Demo

Một số phương thức thao tác với List

Tuples

Tuples là gì

Tạo Tuple

Các thao tác với Tuples

Tuples là gì

- Tuple tương tự như List, sự khác biệt là không thể thay đổi các phần tử của một Tuple trừ khi phần tử đó là một mảng
- Dùng Tuple cho các kiểu dữ liệu không đồng nhất
- Lặp qua Tuple nhanh hơn so với List
- Tuple được sử dụng làm key cho Dictionary trong khi List thì không thể

Cú pháp tạo Tuple

- Cách 1

  tupleName = (element1, element2, ...)

  Trong đó:

  element1, elements2 là các phần tử trong tuple

- Cách 2:

  tupleName = element1, elements2, ...

Cú pháp tạo Tuple: Ví dụ

```
myTuple = (1, 2, "blog", 4.6)
print(myTuple)
```

Kết quả:

```
1, 2, blog, 4.6
```

Truy xuất các phần tử: bằng chỉ số

```
# Danh sách Tuple
myTuple = ('c','o','d','e','g','y','m')

# Lấy phần tử đầu tiên trong tuple
print(myTuple[0])
# Output: 'c'

# Lấy phần tử thứ 7 trong tuple
print(myTuple[5])
# Output: 'y'
```

```
# Mỗi phần tử của tuple là một mảng
hoặc 1 tuple khác
sourceTuple = ("WOW", [8, 4, 6],
(1, 2, 3))

# Lấy phần tử thứ 2 của phần tử thứ
nhất
print(sourceTuple[0][3])
# Output: 'O'

# Lấy phần tử thứ 2 của phần tử thứ
2
print(sourceTuple[1][1])
# Output: 4
```

Truy xuất các phần tử: Số chỉ mục âm

Python cho phép lập chỉ mục số âm

- chỉ số -1 đề cập đến phần tử cuối cùng
- -2 cho phần tử cuối cùng thứ 2
- ...

```
myTuple = ('c','o','d','e','g','y', 'm')
# Phần tử cuối cùng
print(myTuple[-1])
# Output: 'm'

# Phần tử thứ 6 tính từ cuối lên
print(myTuple[-6])
# Output: 'o'
```

Truy xuất các phần tử: bằng Slicing

- Chúng ta có thể truy cập vào một loạt các phần tử trong một tuple bằng cách sử dụng toán tử slicing ":"

  Cú pháp:

  ```
  [begin:end]
  ```

  Trong đó

  - `end` là biên, tức sẽ ko lấy phần tử `end` mà lấy từ `begin` đến `end - 1`

Truy xuất các phần tử: bằng Slicing

Ví dụ

```
myTuple = ('c','o','d','e','g','y','m','2','1')

# Lấy phần tử thứ 2 đến thứ 4
print(myTuple[1:4])
# Output: ('o', 'd', 'e')

# Phần tử đầu tiên đến thứ hai (tức thứ 7 tính từ sau tới)
print(myTuple[:-7])
# Output: ('c', 'o')

# Phần tử thứ 8 đến cuối
print(myTuple[7:])
# Output: ('2', '1')

# Lấy toàn bộ phần tử
print(myTuple[:])
# Output: ('c', 'o', 'd', 'e, 'g', 'y', 'm', '2', '1')
```

Thêm/Sửa/Xoá phần tử trong Tuple

- Cập nhật (thêm, sửa, xóa) các phần tử của Tuple
  - Bước 1: Convert Tuple sang List
  - Bước 2: Tiến hành cập nhật trên List
  - Bước 3: Convert ngược lại List đã cập nhật về Tuple

Unpacking

- Packing

```
fruits = ("apple", "banana", "cherry")
print(fruits)
```

- Unpack

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

Unpacking: Toán tử *

- Mở rộng tuple bằng toán tử *

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
['cherry', 'strawberry',
'raspberry']
```

Hàm count()

- Trả về số lần xuất hiện của một giá trị trong Tuple

```
fruits = [1, 4, 2, 9, 7, 8, 9, 3, 1]
x = fruits.count(9)
print(x)
```

⟹ 2

Hàm index()

- Tìm kiếm một giá trị nào đó trong tuple và trả về chỉ số của phần tử đó, nếu giá trị tìm kiếm không có trong tuple thì phát sinh lỗi

```
fruits = [4, 55, 64, 32, 16, 32]
x = fruits.index(32)
print(x)
```

⟹ 3

Tóm tắt bài học

- Python và phần lớn các ngôn ngữ khác hỗ trợ một cấu trúc dữ liệu là mảng (danh sách) để giải quyết việc lưu trữ tập dữ liệu
- Mảng là một loại biến đặc biệt, có thể lưu được nhiều giá trị thay vì chỉ một giá trị như các biến thông thường
- List có thể được coi là một ***mảng tuần tự*** như trong các ngôn ngữ khác

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo