

BÀI GIẢNG

# CƠ SỞ LẬP TRÌNH

CHƯƠNG 5.

## DỮ LIỆU KIỂU LIST

---

NGUYỄN THÀNH THỦY

BỘ MÔN TIN HỌC QUẢN LÝ

TRƯỜNG ĐẠI HỌC KINH TẾ, ĐẠI HỌC ĐÀ NẴNG

THUYNT@DUE.EDU.VN

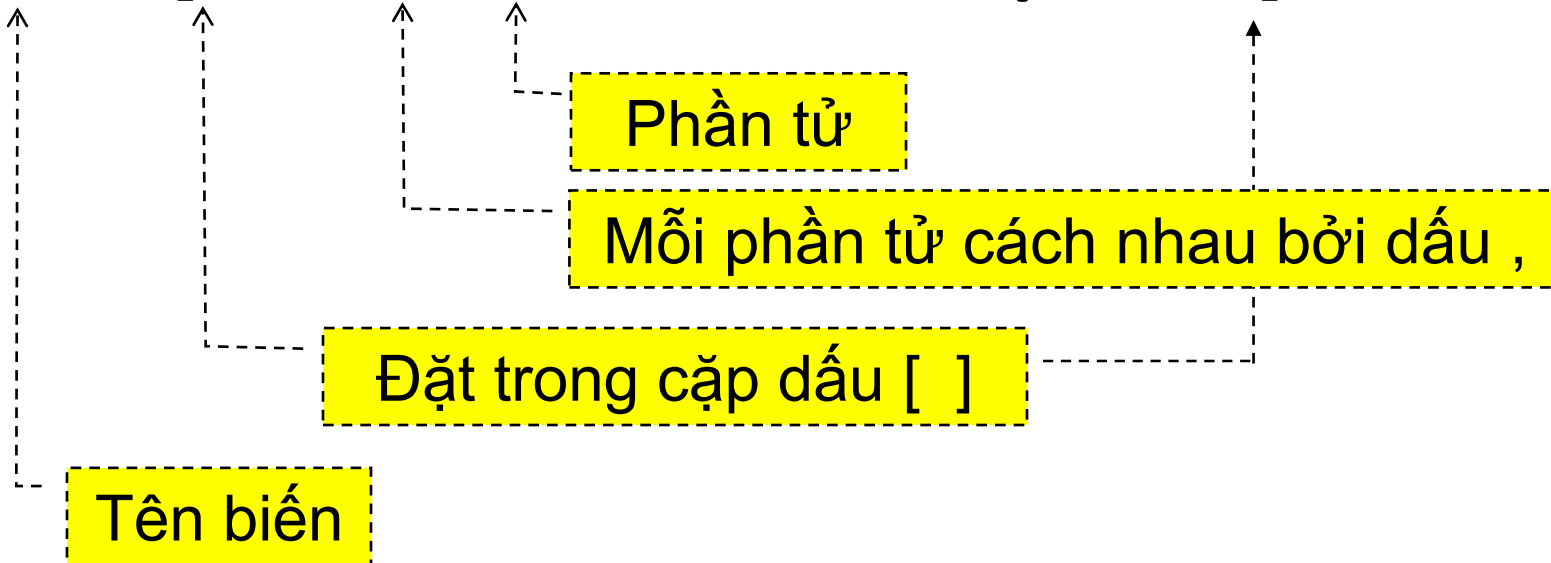
# NỘI DUNG

- Cấu trúc dữ liệu kiểu danh sách – List
- Khởi tạo List
- Truy xuất các phần tử trong List
- Cập nhật giá trị cho phần tử trong List
- Thao tác trên List
- Các phương thức của List
- Sao chép List

# CẤU TRÚC DỮ LIỆU KIỂU LIST - DANH SÁCH

- ❑ List là một giá trị có thể chứa nhiều kiểu giá trị trong một tập hợp có thứ tự.
- ❑ Một biến kiểu List có thể lưu trữ được đồng thời nhiều giá trị (có thể khác nhau)

```
spam = ["cat", "bat", "rat", "elephant"]
```



# CẤU TRÚC DỮ LIỆU KIỂU LIST - DANH SÁCH

## ❑ Ví dụ

---

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant']
>>> ['hello', 3.1415, True, None, 42]
['hello', 3.1415, True, None, 42]
❶ >>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam
['cat', 'bat', 'rat', 'elephant']
```

---

- ❑ ❶ biến **spam** được gán cho một giá trị kiểu List, trong list chứa nhiều giá trị khác nhau;
- ❑ **List rỗng** là list không có phần tử nào
  - Ví dụ: `spam = [ ]`

# KHỞI TẠO LIST

## ❑ Sử dụng phép gán

- Ví dụ:

```
numbers = [1, 2, 3, 4, 5]
```

## ❑ Sử dụng cấu trúc for

- Ví dụ:

```
numbers = [item for item in range(1,6)]  
print(numbers)
```



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

# KHỞI TẠO LIST

## ❑ Sử dụng cấu trúc for

### ■ Ví dụ:

```
matrix = [ [x,x+1,x+2] for x in range(1,10,3) ]  
print(matrix)
```

➔ [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

# KHỞI TẠO LIST

## ❑ Phương pháp constructor List

- Ví dụ:

```
char=list("Python")  
print(char)
```



```
['P', 'y', 't', 'h', 'o', 'n']
```

# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Số chỉ mục (index):

- Có thể truy xuất đến các phần tử trong List thông qua số chỉ mục (index);
- Index bắt đầu từ số 0 (zero), phần tử thứ n trong List sẽ có index là (n-1);
- Sử dụng tên biến và số index để truy xuất;

```
spam = ["cat", "bat", "rat", "elephant"]
```

```
      ↗       ↗       ↗       ↗  
spam[0] spam[1] spam[2] spam[3]
```



# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Số chỉ mục (index):

### ■ Ví dụ:

---

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
>>> spam[1]
'bat'
>>> spam[2]
'rat'
>>> spam[3]
'elephant'
```

# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Số chỉ mục (index):

- Ví dụ: số index vượt quá giới hạn số phần tử

---

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[10000]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    spam[10000]
IndexError: list index out of range
```

---

# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Số chỉ mục (index):

- Ví dụ: số index phải là số nguyên

---

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1]
'bat'
>>> spam[1.0]
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    spam[1.0]
TypeError: list indices must be integers, not float
>>> spam[int(1.0)]
'bat'
```

---

# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Số chỉ mục (index):

- Ví dụ: List trong List

---

Lists can also contain other list values. The values in these lists of lists can be accessed using multiple indexes, like so:

---

```
>>> spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
>>> spam[0]
['cat', 'bat']
>>> spam[0][1]
'bat'
>>> spam[1][4]
50
```

---

# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Số chỉ mục (index):

### ■ Ví dụ: List trong List

```
>>> matrix=[[1,2,3],[4,5,6],[7,8,9]]
```

```
>>> matrix
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
>>> matrix[0]
```

(1) ???

```
>>> matrix[1]
```

(2) ???

```
>>> matrix[2]
```

(3) ???

```
>>> matrix[0][0]
```

(4) ???

```
>>> matrix[0][1]
```

(5) ???

```
>>> matrix[0][2]
```

(6) ???

1	2	3
4	5	6
7	8	9



# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Số chỉ mục âm (*Negative indexes*):

- **Index=-1** sẽ tham chiếu đến phần tử cuối trong List, **Index=-2** là phần tử áp cuối, ...

---

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
>>> spam[-3]
'bat'
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + ' .'
'The elephant is afraid of the bat.'
```

---

# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Số chỉ mục âm (*Negative indexes*):

### ■ Ví dụ:

```
>>> sinhvien=["An","Binh","Minh","Lan","Ngoc"]
```

```
>>> sinhvien[-1]
```

(1) ???

```
>>> sinhvien[-3]
```

(2) ???

```
>>> sinhvien[0]
```

(3) ???



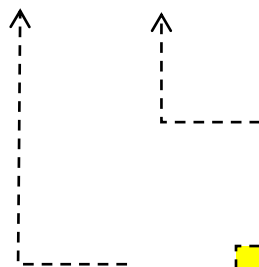
# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Truy xuất tập con trong List

- **Slice:** khung trượt (cửa sổ trượt) cho phép truy xuất được đồng thời nhiều phần tử trong list

`spam[2]` is a list with an index (one integer).

`spam[1:4]` is a list with a slice (two integers).



Phần tử cuối trong tập con có index=3

Phần tử đầu trong tập con có index=1



# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Truy xuất tập con trong List

### ■ Ví dụ:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam[0:4] ←----- Tập con từ index=0 → index=3
```

```
['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam[1:3] ←----- Tập con từ index=1 → index=2
```

```
['bat', 'rat']
```

```
>>> spam[0:-1] ←----- Tập con từ index=0 → phần tử áp cuối
```

```
['cat', 'bat', 'rat']
```

# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Truy xuất tập con trong List

### ■ Ví dụ:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam[:2] <----- Tập con từ index=0 → index=1
```

```
['cat', 'bat']
```

```
>>> spam[1:] <----- Tập con từ index=1 → cuối List spam
```

```
['bat', 'rat', 'elephant']
```

```
>>> spam[:] <----- Tập con là toàn bộ List spam
```

```
['cat', 'bat', 'rat', 'elephant']
```

# TRUY XUẤT CÁC PHẦN TỬ TRONG LIST

## ❑ Truy xuất tập con trong List

### ■ Ví dụ:

```
>>> spam=[1,2,3,4,5,6]
```

```
>>> spam[2]
```

(1) ???

```
>>> spam[1:4]
```

(2) ???

```
>>> spam[(3)? ]
```

```
[3, 4, 5]
```

```
>>> spam[(4)? ]
```

```
[1, 2, 3]
```



# CẬP NHẬT GIÁ TRỊ CHO PHẦN TỬ TRONG LIST

## ❑ Sử dụng phép gán và số Index

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark'
>>> spam
['cat', 'aardvark', 'rat', 'elephant']
>>> spam[2] = spam[1]
>>> spam
['cat', 'aardvark', 'aardvark', 'elephant']
>>> spam[-1] = 12345
>>> spam
['cat', 'aardvark', 'aardvark', 12345]
```

# THAO TÁC TRÊN LIST

## ❑ Toán tử +

```
numbers = [1, 2]
```

```
print(numbers)
```

```
numbers = numbers + [3, 4]
```

```
print(numbers)
```

```
numbers = numbers + numbers
```

```
print(numbers)
```



```
[1, 2]
```

```
[1, 2, 3, 4]
```

```
[1, 2, 3, 4, 1, 2, 3, 4]
```

# THAO TÁC TRÊN LIST

## ❑ Toán tử \*

```
numbers = [1, 2]
```

```
print(numbers)
```

```
numbers = numbers*2
```

```
print(numbers)
```

```
numbers = numbers + numbers
```

```
print(numbers)
```



```
[1, 2]
```

```
[1, 2, 1, 2]
```

```
[1, 2, 1, 2, 1, 2, 1, 2]
```

# THAO TÁC TRÊN LIST

## ❑ Toán tử **in** và **not**

- Toán tử **in** và **not** giúp xác định một giá trị có tồn tại trong một List hay không;
- Biểu thức trả về **True** (có) hoặc **False** (không);

---

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True
```

---

# THAO TÁC TRÊN LIST

## ❑ Toán tử **in** và **not**

- Ví dụ: Nhập vào tên của một thú cưng và kiểm tra tên đó có trong danh sách hay không

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']  
print('Enter a pet name:')  
name = input()  
if name not in myPets:  
    print('I do not have a pet named ' + name)  
else:  
    print(name + ' is my pet.')
```



Enter a pet name:  
Pooka  
Pooka is my pet.

Enter a pet name:  
Gau iu  
I do not have a pet named Gau iu



# THAO TÁC TRÊN LIST

## ❑ Hàm `len()`

- Trả về chiều dài của List

```
>>> spam=[1, 2, 3, 4, 5, 6]
>>> len(spam)
6
>>> matrix=[[1,2,3],[4,5,6],[7,8,9]]
>>> len(matrix)
3
>>> len(matrix[0])
3
>>> matrix=[[1,2,3],[4,5,6]]
>>> len(matrix)
2
>>> len(matrix[0])
3
```

# THAO TÁC TRÊN LIST

## ❑ Hàm `max()`

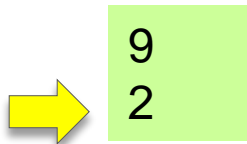
- Trả về phần tử có giá trị lớn nhất trong List

## ❑ Hàm `min()`

- Trả về phần tử có giá trị bé nhất trong List

Ví dụ:

```
numbers = [5, 3, 8, 2, 9]  
print(max(numbers))  
print(min(numbers))
```



9  
2

# THAO TÁC TRÊN LIST

## ❑ Xóa phần tử trong List với hàm `del()`

### ■ Ví dụ:

```
spam=[1,2,3,4,5]
```

```
del(spam[2])
```

```
print(spam)
```

```
del(spam[1])
```

```
print(spam)
```



```
[1, 2, 4, 5]
```

```
[1, 4, 5]
```

# THAO TÁC TRÊN LIST

## ❑ Xóa phần tử trong List với hàm del()

### ■ Ví dụ:

```
students=["An","Binh","Lan","Thanh","Minh"]  
del(students[2])  
print(students)
```

```
del(students[1])  
del(students[2])  
print(students)
```



```
??? (1)  
??? (2)
```



# THAO TÁC TRÊN LIST

## ❑ Duyệt các phần tử trong List

### ■ Ví dụ:

```
students=["An","Binh","Lan","Thanh","Minh"]
```

*#Cách 1*

```
for x in students:  
    print(x, end=", ")
```

*#Cách 2*

```
for x in range(len(students)):  
    print(students[x], end=", ")
```



An, Binh, Lan, Thanh, Minh,

# THAO TÁC TRÊN LIST

## ❑ Bổ sung phần tử cho List

### ■ Ví dụ:

```
catNames = [ ]  
while True:  
    print('Enter the name of cat ' + str(len(catNames) + 1)  
          + ' (Or enter nothing to stop.):')  
    name = input()  
    if name == "":  
        break  
    catNames = catNames + [name] # list concatenation  
  
print('The cat names are:')  
for name in catNames:  
    print(' ' + name)
```

Khởi tạo List rỗng

Biến **name** đặt trong [ ]

Phương pháp thêm phần tử bằng toán tử cộng ( + )

## BÀI TẬP

Viết chương trình nhập từ bàn phím **10** số nguyên, lưu trữ vào 1 List; và nhập một số nguyên **x**.

**a.** Tìm tất cả các phần tử có giá trị bằng **5** và thay bằng **x**. In kết quả lên màn hình;

**b.** Xóa tất cả các phần tử có giá trị bằng **x** xuất hiện trong tập hợp trên.

# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ Phương thức (method)

- Một phương thức có chức năng như **một hàm được định nghĩa sẵn**;
- Cho phép **xử lý dữ liệu trên đối tượng** tương ứng;
- Mỗi kiểu dữ liệu sẽ **có tập phương thức riêng**;

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']
```

```
myPets.
```

copy(self)	list
pop(self, __index)	list
sort(self, key, reverse)	list
append(self, __object)	list
clear(self)	list
count(self, __value)	list
extend(self, __iterable)	list
index(self, __value, __...	list
insert(self, __index, __...	list
remove(self, __value)	list
reverse(self)	list

Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards [Next Tip](#)



# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ `index()` method

- Trả về **index** của một **phần tử** được tìm thấy trong **List**;
- Trong trường hợp x không tồn tại trong List sẽ bị lỗi → cần sử dụng cú pháp: **if x in L** để kiểm tra trước khi gọi phương thức `index()`.
- **Ví dụ:**

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']  
namePet="Pooka"  
if namePet in myPets:  
    print("Index is", myPets.index(namePet))  
else:  
    print("Not Found!")
```



Index is 1

# CÁC PHƯƠNG THỨC CỦA LIST

## □ *append()*, *insert()* method

- **append(x)** thêm phần tử **x** vào cuối List;

```
>>> spam = ['cat', 'dog', 'bat']  
>>> spam.append('moose')  
>>> spam  
['cat', 'dog', 'bat', 'moose']
```

- **insert(i,x)** chèn **x** vào vị trí **i** hiện có trong List;

```
>>> spam = ['cat', 'dog', 'bat']  
>>> spam.insert(1, 'chicken')  
>>> spam  
['cat', 'chicken', 'dog', 'bat']
```

# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *append()*, *insert()* method

```
names = [1, 2, 3, 4, 5, 6]  
names.append(6)  
print(names)  
names.insert(0, 7)  
print(names)  
names.insert(-1, 8)  
print(names)
```



```
??? (1)  
??? (2)  
??? (3)
```



# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *remove()* method

- **remove(x)**: xóa phần tử **x** đầu tiên tìm thấy trong List;
- Trong trường hợp x không tồn tại trong List sẽ bị lỗi → cần sử dụng cú pháp: **if x in L** để kiểm tra trước khi gọi phương thức **remove()**.

■ **Ví dụ:**

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']  
>>> spam.remove('bat')  
>>> spam  
['cat', 'rat', 'elephant']
```

■ **Ví dụ:**

```
>>> spam = ['cat', 'bat', 'rat', 'cat', 'hat', 'cat']  
>>> spam.remove('cat')  
>>> spam  
['bat', 'rat', 'cat', 'hat', 'cat']
```

# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *remove()* method

- Phân biệt giữa hàm **del()** và phương thức **remove()**
  - Hàm **del()** xóa một phần tử khi **biết index**
  - Hàm **remove()** xóa một phần tử khi **biết giá trị**
- Ví dụ: xóa **phần tử thứ 2** trong danh sách spam

```
spam = ['cat', 'bat', 'rat', 'elephant']
```

*#Cách 1*

```
del(spam[1])
```

*#Cách 2*

```
spam.remove('bat')
```



```
['cat', 'rat', 'elephant']
```

# CÁC PHƯƠNG THỨC CỦA LIST

## □ *sort()* method

- Cho phép sắp xếp các phần tử trong List có thứ tự;
- Ví dụ:

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

# CÁC PHƯƠNG THỨC CỦA LIST

## ❏ *sort()* method

- Mặc định sắp xếp **tăng dần**, thêm tham số **reverse=True** để sắp xếp **giảm dần**;
- Ví dụ:

```
numbers = [1, 2, 3, 4, 5]
```

```
numbers.sort() # or: numbers.sort(reverse=False)  
print(numbers)
```

```
numbers.sort(reverse=True)  
print(numbers)
```



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

# CÁC PHƯƠNG THỨC CỦA LIST

## □ *sort()* method

### ■ Lưu ý:

- Hàm **sort()** chỉ thực hiện được khi các phần tử có cùng kiểu dữ liệu;

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

- Dữ liệu kiểu chuỗi sẽ được sắp xếp theo thứ tự alphabet

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```



# CÁC PHƯƠNG THỨC CỦA LIST

## □ *sort()* method

### ■ Lưu ý:

- Hàm **sort()** chỉ thực hiện được khi các phần tử có cùng kiểu dữ liệu;

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

```
>>> spam = [1, 3, 2, 4, 'Alice', 'Bob']
>>> spam.sort()
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    spam.sort()
TypeError: unorderable types: str() < int()
```

# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *reverse()* method

- Thực hiện đảo ngược thứ tự các phần tử trong List
- Ví dụ:

```
L=["Binh", "An", "Nam", "Anh", "A"]  
L.sort()  
print(L)  
  
L.reverse()  
print(L)
```



```
['A', 'An', 'Anh', 'Binh', 'Nam']  
['Nam', 'Binh', 'Anh', 'An', 'A']
```

# CÁC PHƯƠNG THỨC CỦA LIST

## ❏ *clear()* method

- Thực hiện xóa tất cả các phần tử trong List
- Ví dụ:

```
numbers = [1, 2, 3, 4, 5]  
print(numbers)  
  
numbers.clear()  
print(numbers)
```



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

# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *count(x)* method

- Thực hiện đếm số phần tử **x** xuất hiện trong List
- Ví dụ:

```
numbers = [1, 2, 3, 2, 5]  
print(numbers.count(2))  
print(numbers.count(5))  
print(numbers.count(10))
```



2  
1  
0

# CÁC PHƯƠNG THỨC CỦA LIST

## ❏ *copy()* method

- Thực hiện tạo ra một bản sao mới của List
- Ví dụ:

```
numbers1 = [1, 2, 3, 4, 5]  
numbers2=numbers1.copy()  
print(numbers1)  
print(numbers2)
```



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

# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *copy()* method

```
names = ["An", "Nam", "Binh", "Ngoc"]  
x=names.copy()  
print(x)  
print(x.count("Nam"))
```



??? (1)  
??? (2)



# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *pop(i)* method

- Thực hiện **xóa và lấy ra giá trị của phần tử** có số chỉ mục **i** trong List.
- Nếu tham số **i** để trống thì mặc định là **lấy phần tử cuối** trong List.
- **Ví dụ:**

```
numbers = [1, 2, 3, 4, 5]  
x=numbers.pop(2)  
print(numbers)  
print(x)
```



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

```
numbers = [1, 2, 3, 4, 5]  
x=numbers.pop()  
print(numbers)  
print(x)
```



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

# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *pop(i)* method

```
names = ["An", "Nam", "Binh", "Ngoc"]  
x1=names.pop(0)  
x2=names.pop(-1)  
print(x1)  
print(x2)  
print(names)
```



??? (1)  
??? (2)  
??? (3)





# CÁC PHƯƠNG THỨC CỦA LIST

## ❑ *pop(i)* method

```
names = ["An", "Nam", "Binh", "Ngoc"]  
names.remove("An")  
del(names[0])  
x=names.pop(-2)  
print(x)  
print(names)
```



??? (1)  
??? (2)



# SAO CHÉP LIST

## ❑ Phép tham chiếu (References)

- Ví dụ: phép gán trong biến đơn

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

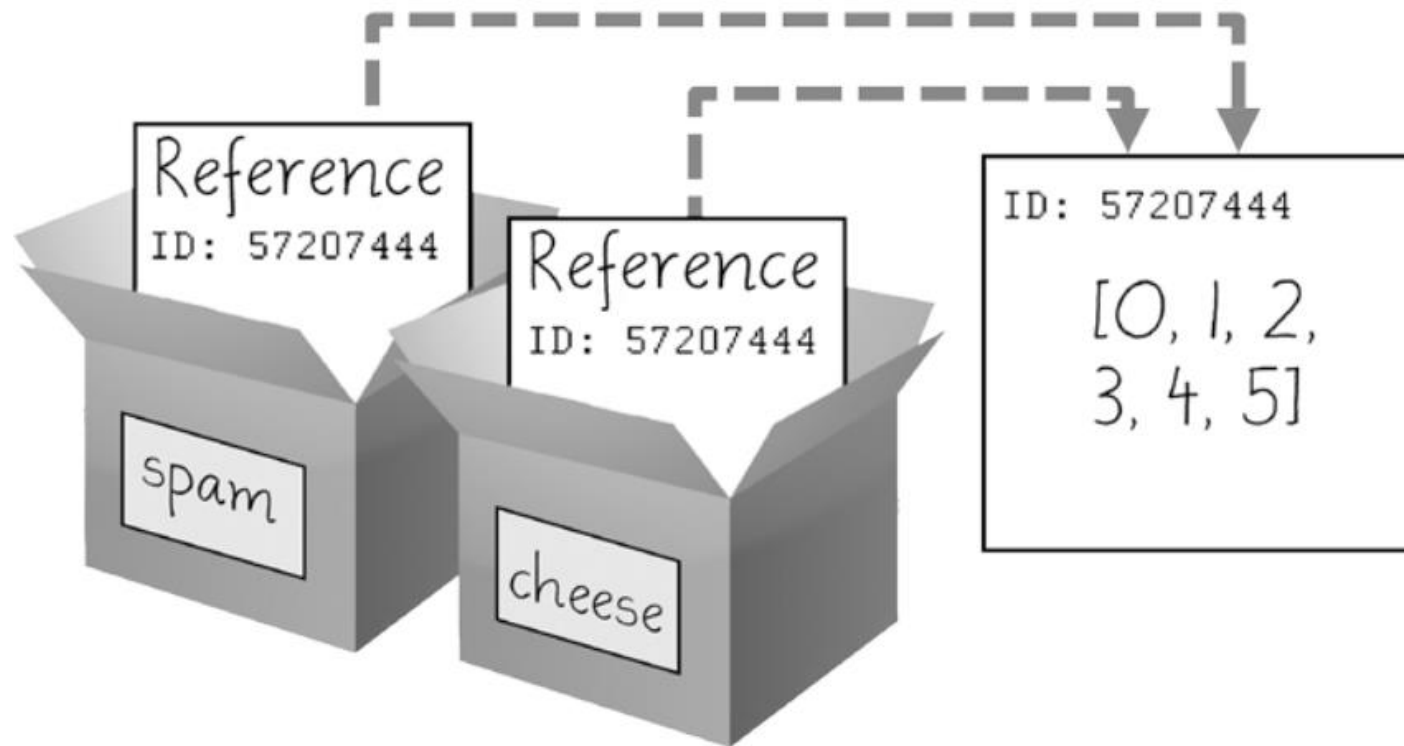
- Ví dụ: phép gán trong list

```
❶ >>> spam = [0, 1, 2, 3, 4, 5]
❷ >>> cheese = spam
❸ >>> cheese[1] = 'Hello!'
>>> spam
[0, 'Hello!', 2, 3, 4, 5]
>>> cheese
[0, 'Hello!', 2, 3, 4, 5]
```



# SAO CHÉP LIST

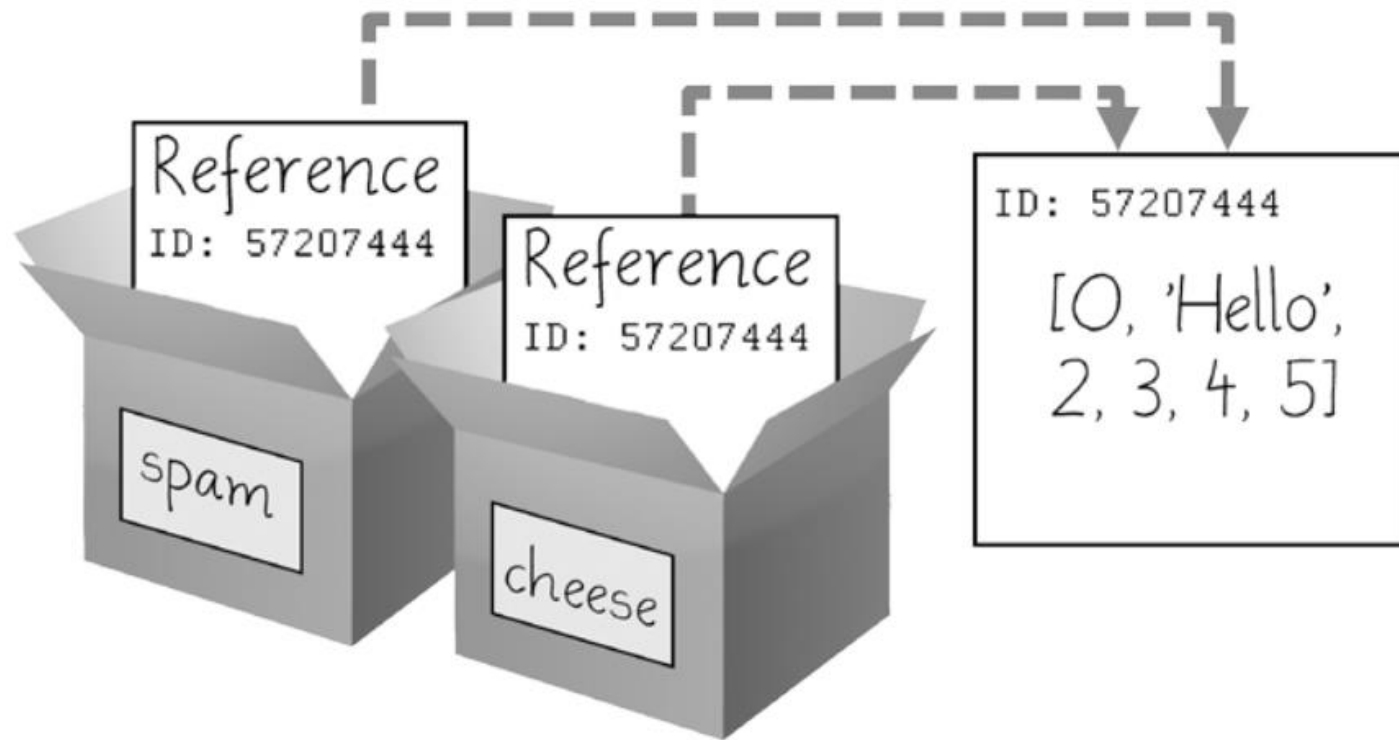
## ❑ Phép tham chiếu (References)



*Figure 4-5: spam = cheese copies the reference, not the list.*

# SAO CHÉP LIST

## ❑ Phép tham chiếu (References)



*Figure 4-6: `cheese[1] = 'Hello!'` modifies the list that both variables refer to.*

# SAO CHÉP LIST

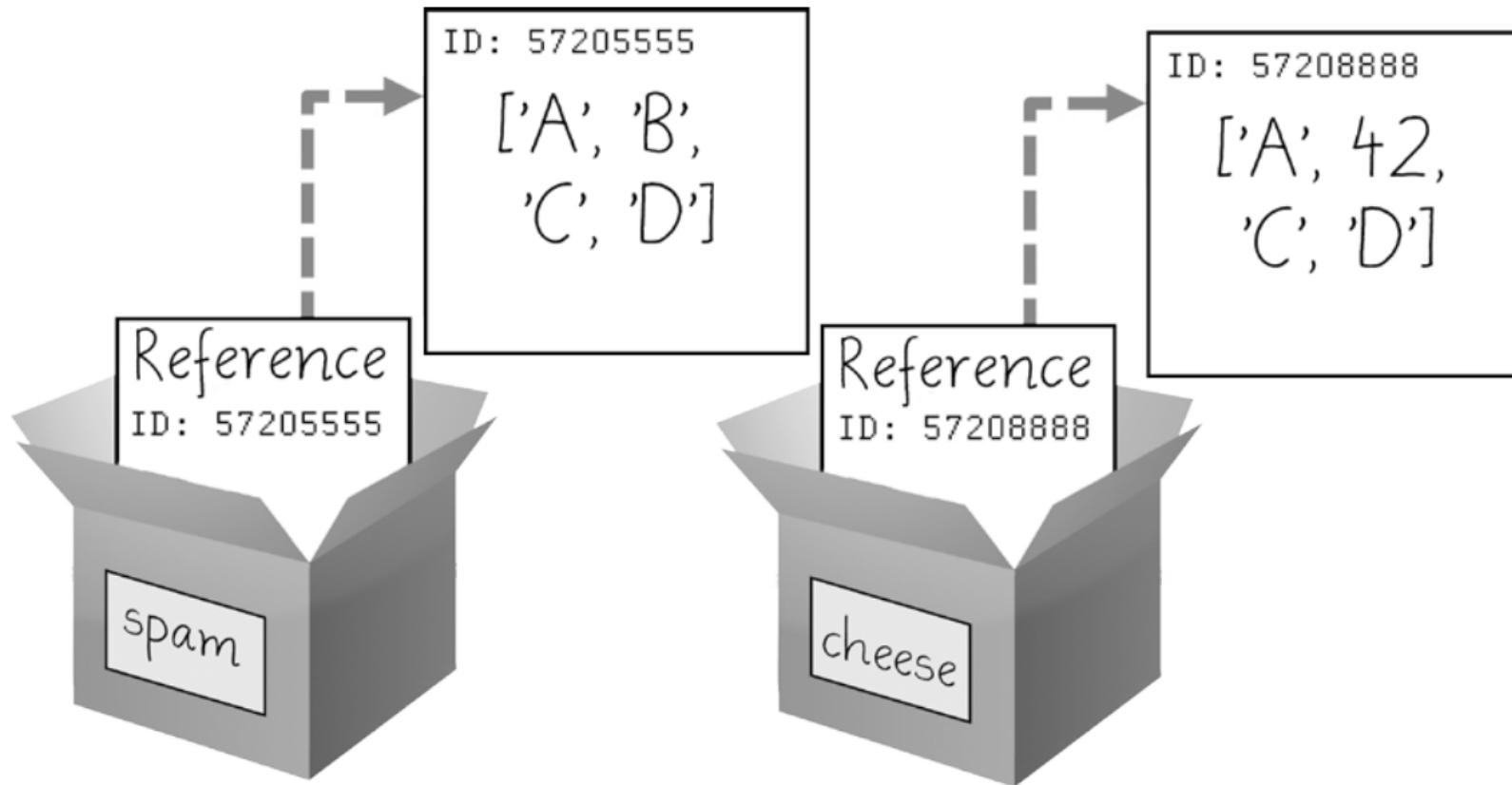
## ❑ Phép tham biến (variables refer)

- Không sử dụng phép gán để tạo bản sao cho List
- Sử dụng hàm **copy** trong thư viện **copy** để tạo bản sao List
- Ví dụ:

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> cheese = copy.copy(spam)
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

# SAO CHÉP LIST

## ❑ Phép tham biến (variables refer)



*Figure 4-7: `cheese = copy.copy(spam)` creates a second list that can be modified independently of the first.*

# SAO CHÉP LIST

## ❑ Phép tham biến (variables refer)

### ■ Ví dụ:

```
import copy
numbers1 = [1, 2, 3, 4, 5]
numbers2 = copy.copy(numbers1)
print(numbers2)

numbers3 = numbers1
numbers3[2] = 6
print(numbers3)
```



```
??? (1)
??? (2)
```



# BÀI TẬP ÔN TẬP

**Xây dựng hàm, thực hiện các yêu cầu sau:**

**Bài 1.** Hàm **add(L, x, k)** thêm phần tử **x** vào List **L** tại vị trí **k**, nếu **k** lớn hơn số phần tử của **L** thì thêm **x** vào cuối **L**;

**Bài 2.** Hàm **search(L, x)** tìm **x** trong List **L**, nếu tìm thấy thì trả về **index** của **x** trong **L**, còn lại trả về **None**;

**Bài 3.** Hàm **delete(L, x)** xóa tất cả phần tử có giá trị bằng **x** trong List **L**;

**Bài 4.** Hàm **count(L)** trả về số lượng phần tử trong List **L**; (tương tự hàm **len()**)

**Bài 5.** Hàm **update(L, x, y)** tìm **x** trong List **L** và thay thế bằng **y**;

*Lưu ý: không sử dụng các hàm có chức năng tương tự có sẵn trong python*