速成教育
SPEED UP EDUCATION

CSCA08

Introduction of Programming I

导师： **Vincent**

UTSC Week 05 | 2025/2/13

List 列表

一个 collection，可以装个很多不同的 item，用 `[]` 去表示

```
>>> L = [1, ["vc", []], None]

>>>
'c'
```

有顺序的，所以和 **string** 一样，也有 **index** 和 **slicing** 的功能

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> names[1: 4]
['Alice', 'Bob', 'Peter']
```

也和 **string** 一样 可以用 **+** 或者 *，创建一个**「新」**的 list

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> others = ['Anya', 'Zac']

>>> new = names + others
>>> names
['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> others
['Anya', 'Zac']
>>> new
['Vincent', 'Alice', 'Bob', 'Peter', 'Carol', 'Anya', 'Zac']

>>> new2 = others * 2
>>> others
['Anya', 'Zac']
>>> new2
['Anya', 'Zac', 'Anya', 'Zac']
```

List 是 **Mutable object** (可被改变的)

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> names[1] = 'Anya'
>>> names
['Vincent', 'Anya', 'Bob', 'Peter', 'Carol']
```

常用的 List methods

因为 List 是 mutable 所以 以下所有的 method 都是直接 **modify(改变)** 原本的 list

并**不会**像 string methods 那样 返回一个新的 list, 而是 什么都不会返回

Python 为了告诉你 什么都没有 (Nothing), 这些 method 都会返回 None

L.append(x)

- 把 x 加到 L 的最后面 (直接 modify L), 返回 None

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> x = names.append('Anya')
>>> names
['Vincent', 'Alice', 'Bob', 'Peter', 'Carol', 'Anya']
>>> x
None
```

L.insert(i, x)

- 把 x 加到 L 插入到 index i 的位置 (直接 modify L) 返回 None
- i 一定要是 int

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> names.insert(2, 'Anya')
>>> names
['Vincent', 'Alice', 'Anya', 'Bob', 'Peter', 'Carol']
```

L.extend(L2)

- 把 L2 合并 在 L 的后面 (直接 modify L) 返回 None
- L2 也要是一个 list

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> others = ['Anya', 'Zac']
>>> names.extend(others)
>>> names
['Vincent', 'Alice', 'Bob', 'Peter', 'Carol', 'Anya', 'Zac']
>>> others
['Anya', 'Zac']

>>> names.append(others)
['Vincent', 'Alice', 'Bob', 'Peter', 'Carol', 'Anya', 'Zac', ['Anya', 'Zac']]
```

L.pop(i)

- 把 L 在 index i 位置 的 item 删掉 (直接 modify L) **返回 被删掉的** item
- i 一定要是 int 并且 0 <= i < len(L) 不然会有 **IndexError**
- 如果 i 不给, 默认 删掉**最后一个** index

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> x = names.pop(3)
>>> names
['Vincent', 'Alice', 'Bob', 'Carol']
>>> x
'Peter'

>>> y = names.pop()
>>> names
['Vincent', 'Alice', 'Bob']
>>> y
'Carol'

>>> names.pop(99)
IndexError: Index out of range
```

L.remove(x)

- 把 x 从 L 里删掉 (直接 modify L) 返回 None
- 如果 x 不存在 则报错: **ValueError**

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> x = names.remove('Alice')
>>> names
['Vincent', 'Bob', 'Peter', 'Carol']
>>> x
None

>>> names.remove('Anya')
ValueError: 'Anya' not in list
```

L.index(x)

- 返回 x 在 L 从左边 第一次出现的 index
- 如果 x 不在, 报错: **ValueError**

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> i = names.index('Bob')
>>> i
2

>>> names.index ('Anya')
ValueError: 'Anya' not in list
```

`L.count(x)`

- 返回 `x` 在 `L` 里出现几次

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol', 'Alice', 'alice']
>>> names.count('Alice')
2
```

`L.sort()`

- 把 `L` 从小到大排列好 (直接 modify `L`) 返回 `None`
- 如果要 从大到小, 可以多写一个 参数: `L.sort(reverse=True)`

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> names.sort()
>>> names
['Alice', 'Bob', 'Carol', 'Peter', 'Vincent']
>>> names.sort(reverse=True)
['Vincent', 'Peter', 'Carol', 'Bob', 'Alice']
```

Common Mistakes

```
>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> names = names.append('Anya')
>>> names[1]

>>> names = ['Vincent', 'Alice', 'Bob', 'Peter', 'Carol']
>>> names = names.sort()
>>> names[1]
```

Memory Model

Consider the following code,

```
>>> x = 3
>>> y = x
>>> s = 'csc108'
>>> L = [3, s, x]
>>> L
[3, 'csc108', 3]
```

Variables	Memory
x -> id0	id0: 3
y -> id0	id1: 'csc108'
s -> id1	id2: [id0, id1, id0]
L -> id2	

```
>>> L[0] = 5
>>> L
```

```
>>> s[3] = '2'
TypeError
```

Variables	Memory
x -> id0	id0: 3
y -> id0	id1: 'csc108' id4
s -> id1	
L -> id2	id2: [~~id0~~ id3, id1, id0]
	id3: 5
	id4: '2'

Immutable Type v.s. Mutable Type

```
>>> s1 = 'a'
>>> s2 = 'a'
>>> L1 = ['a', 'b']
>>> L2 = ['a', 'b']
```

Variables	Memory
s1 -> id0	id0: 'a'
s2 -> id0	id1: 'b'
L1 -> **id2**	id2: [id0, id1]
L2 -> **id3**	id3: [id0, id1]

Alias

Consider the following code, and the memory model

```
>>> L1 = [4, 5, 6]
>>> L2 = [4, 5, 6]
>>> L3 = L2
>>> L1 == L2
True
>>> L1 is L2
False
>>> L2 is L3
True
>>> L3.append('vc')
>>> L2
[1, 2, 3, 'vc']
>>> L3
[1, 2, 3, 'vc']
```

知识点

`==` 比的是 same value?
`is` 比的是 same object? same id?
`Alias` 是當兩個 variable 其實是同一個 object
例子中，`L2` 和 `L3` 的地址是一樣的，they refer to the same address
可通過所有的 alias (reference variables) 去改變那個 object

Variables	Memory
L1 -> id3 L2 -> id4 L3 -> id4	id0: 4 id1: 5 id2: 6 id3: [id0, id1, id2] id4: [id0, id1, id2, id5] id5: 'vc'

```
>>> def function1(x):
...     x = 'vc'

>>> function1(L1)
>>>  L1
```

Variables	Memory
L1 -> id3 L2 -> id4 L3 -> id4 function1 -> id6 x -> ~~id3~~ id5	id0: 4 id1: 5 id2: 6 id3: [id0, id1, id2] id4: [id0, id1, id2, id5] id5: 'vc' id6: None

```
>>> def function2(x):
...     x[0] = 'vc'

>>> function2(L1)
>>> L1
```

Variables	Memory
L1 -> id3 L2 -> id4 L3 -> id4 function2 -> id6 x -> id3	id0: 4 id1: 5 id2: 6 id3: [~~id0~~ id5, id1, id2] id4: [id0, id1, id2, id5] id5: 'vc' id6: None

Tuple

一种 collection, 类似 `list` 可以装很多 item, 用 `()` 表示

- 有顺序的, 所以和 `string`, `list` 一样, 也有 `index` 和 `slicing` 的功能
- 但是 `tuple` 是 `Immutable type` (不可被改变！！)
- 因此, `tuple` 没有任何 `method`
- 大多数时候, 可以不需要写 `()`, 两个 value 中间用 `,` 隔开 就默认是 `tuple`
- 如果要创建一个 长度为 1 的 `tuple` 必须加上一个`,` 在后面, Ex: `(3, )`

```
>>> names = ('Vincent', 'Alice', 'Bob')
>>> names[0]
'Vincent'
>>> names[2]
'Bob'

>>> names[0] = 'Anya'
TypeError: 'tuple' object does not support item assignment
```

通常会用 `tuple` 去实现 `parallel assignment` (同时赋值)

```
>>> x, y = 1, 2
>>> x
1
>>> y
2

>>> s = 'vincent,tse,18'
>>> first_name, last_name, age = s.split(',')

>>> first_name
'vincent'
>>> last_name
'tse'
>>> age
'18'
```

其他用法

```
data = [['vincent', 'tse'], ['anya', 'zhang'], ...]
for first_name, last_name in data:
    # do something ...
```

```
def func():
    a = ...
    b = ...
    return a, b

x, y = func()
```

题型: 找最大(小)

```
def find_longest(names: list[str]) -> name:
    """Return the longest name in names.

    Precondition: len(names) > 0

    >>> find_longest(['Alice', 'Bob', 'Vincent', 'Coral']
    'Vincent'
    """
```

```
def find_smallest(L: list[int]) -> int
    """Return the index of smallest number in L

    Precondition:
        len(L) > 0

    >>> find_smallest([7, 9, 4, 8, 15, 3, 20])
    5
    """
```

题型: Modify List (改变列表)

```
def square_it(num_list: list[int]) -> list[int]:
    '''Return a copy of num_list but with every number squared

    >>> square_it([2, 3, 4])
    [4, 9, 16]
    '''
```

注意！ Docstring 和 example 的細節！！

```
def square_it(num_list: List[int]) -> None:
    '''Modify num_list so that every number is squared.

    >>> l = [2, 3, 4]
    >>> square_it(l)
    >>> l == [4, 9, 16]
    True
    '''
```

题型: 删减 List (必须要先 make copy)

```
def remove_even(L: list[int]) -> None:
    """Remove all the even number from L.

    >>> L = [1, 2, 2, 4, 3, 6, 8, 7]
    >>> remove_even(L)
    >>> L == [1, 3, 7]
    True
```