大学计算机入门课

Class #5

[教学目的]

- More on list
- And start thing about while loops.
- Getting into our first assignment!

[课程大纲]

- Part 1: more on list
- Part 2: nested list
- Part 3: more function about for loops and nested list
- Part 4: while loop
- Part 5: assignment

part 1 : More on list

- 区分我们什么时候是建了一个新的 list，什么时候是改变原来的 list
  - 首先，我们要知道所有的 list method 都是改变 list 本身

```
>>> a = [1,2,3]
>>> a.insert(2,5)
>>> a
[1, 2, 5, 3]
```

```
>>> a = [1,2,3]
>>> a.append(1)
>>> a
[1, 2, 3, 1]
```

  - 但是，简单的 addition 并不会改变 list 本身

```
>>> a = [1,2,3]
>>> a + [7]
[1, 2, 3, 7]
>>> a
[1, 2, 3]
```

所以 a + [7] 本身相当于建了一个新的 list！

  - 复制法也不会改变 list 本身

```
>>> a = [1,2,3]
>>> b = [a[0], a[1], a[2], 8]
>>> a
[1, 2, 3]
>>> b
[1, 2, 3, 8]
```

所以，在这个时候，我们也相当于建了一个新的 list！

Now， re-consider the question from exercise 4:

1. Consider this code:

```
a = [1, 0]
```

All of the following code fragments cause `a` to refer to `[1, 0, 8]`.

Circle all of the code fragment(s) that create a new list.

(a) `a.append(8)` (b) `a = a + [8]`
(c) `a.insert(len(a), 8)` (d) `a = [a[0], a[1], 8]`

Circle all of the code fragment(s) that modify the original list.

(a) `a.append(8)` (b) `a = a + [8]`
(c) `a.insert(len(a), 8)` (d) `a = [a[0], a[1], 8]`

2. Consider this code:

```
a = [1, 0, 8]
b = a.sort()
```

After the code above is executed, which of the following expressions evaluate to `True`? Circle those expression(s).

(a) `a == [1, 0, 8]` (b) `a == [0, 1, 8]`
(c) `b == [1, 0, 8]` (d) `b == [0, 1, 8]`

- aliases
  (we need to be much more careful about list)
  recall， something we do for the integer

```
>>> a = [1,2,3]
>>> b = [a[0], a[1], a[2], 8]
>>> a
    [1, 2, 3]
>>> b
    [1, 2, 3, 8]
```

also, for strings:

```
>>> a = "hello"
>>> b = a
>>> a = "lala"
>>> b
    'hello'
>>> a
    'lala'
```

所以，对于 integer 和 string —— value 来说，如果把 b 指向 a，再改变 a 后，b 不会再发生响应的改变。这事因为，b 指向了 a，但当我们改变 a 的时候，我们直接给了 a 一个新的值，而不是单纯的改变 a 原来的值！（这事因为我们说过 string 和 integer 是 immutable）

但是对于 list 而言，事情就有所不同了。

```
>>> a = [1,2,3]
>>> b = a
>>> a.append(7)
>>> a
    [1, 2, 3, 7]
>>> b
    [1, 2, 3, 7]
```

这是因为当 b 指向 a 时，它是 refer to a，并不是单单 a 的值。所以在这个时候，a，b 是 aliases，把它们想为共生体，一个自身被 modified，另一个也会改变！

```
>>> a = [1,2,3]
>>> b = a
>>> b.append(5)
>>> a
    [1, 2, 3, 5]
>>> b
    [1, 2, 3, 5]
```

但是，注意！只有本身被 modified 才会改变

因为我们刚刚说过，a = a + [8] 相当于建了一个新的 list，所以我们不算在 a 本身做改变，在这个时候

```
>>> a = [1,2,3]
>>> b = a
>>> a = a + [8]
>>> a
    [1, 2, 3, 8]
>>> b
    [1, 2, 3]
```

```
>>> a = [1,2,3]
>>> b = a
>>> b = b + [8]
>>> b
    [1, 2, 3, 8]
>>> a
    [1, 2, 3]
```

在这种情况下，就像我们对于 int / str 一样，在这个时候， aliases 被 break！

总结：当 b = a 时，a,b 叫做 aliases
当其中一个本身自己被 modified 时，两者同时改变！
但如果其中一个被赋予一个新的值时，只有一个会改变！

- forcing a copy some times
  因为 list 具有这样特殊的性质，所以有时在我们写 code 的时候我们要注意先 make a copy。
  Example：

```
def original_and_sorted(l):
    """ (list) -> None

    Print out the original list l and the sorted l line by line.

    >>> original_and_sorted([1,8,4])
    [1,8,4]
    [1,4,8]
    """
    # first try
    new = l
    new.sort()
    print(l)
    print(new)
```

如果我们这样写，当我们运行时，会发现

```
>>> original_and_sorted([1,8,4])
    [1, 4, 8]
    [1, 4, 8]
```

所以，我们一定要先 make a copy！！

```
def original_and_sorted(l):
    """ (list) -> None

    Print out the original list l and the sorted l line by line.

    >>> original_and_sorted([1,8,4])
    [1,8,4]
    [1,4,8]
    """
    new = []
    for item in l:
        new.append(item)
    new.sort()
    print(l)
    print(new)

>>> original_and_sorted([1,8,4])
    [1, 8, 4]
    [1, 4, 8]
```

part 2: Nested list.

Nested list: 顾名思义，就是一个 list 里套有另外一个 list
如：
[[1] , [2] , 5, 9]
[[1, 3, [4]] , 6]

depth of a nested list,
最多套的 list 数
1 depth 0
[1, 2, 3] depth 1
[[1] , [2] , 9] depth 2
[[1, 3, 4] , [5 , [6, [7]]] , 5] depth 3

Part 3: more on for loops

我们现在先考虑一个 function，它将一个 depth 为 2 的 list 转化成一个 depth 为 1 的 list

- the use of : isinstance()

```
>>> isinstance(1,int)
True
>>> isinstance([1,2,3], list)
True
>>> isinstance(1, list)
False
>>>
```

来判断 data type

```
def depth2_to_1(l):
    """(list) -> list

    return the list of item in l with depth 1.

    >>> depth2_to_1([[1],2,3,4,[5,6]])
    [1,2,3,4,5,6]
    """
    result = []
    for item in l:
        if isinstance(item, list):
            for i in item:
                result.append(i)
        else:
            result.append(item)
    return result
```

下面，我们考虑另外一个 function，它 return the list contains l and sorted l and the highest value in l

```
def get_highest_mark(l):
    """(list[float]) -> list

    Return the list conatins l and sorted l and the highest value in l.

    >>> get_highest_mark([79,34,90,96,82,40,27])
    [[79,34,90,96,82,40,27], [27, 34, 40, 79, 82, 90, 96], 96]
    """
    # get a copy
    copy = []
    for item in l:
        copy.append(item)
    copy.sort()
    result = []
    result.append(l)
    result.append(copy)
    result.append(copy[-1])
    return result
```

Part 4: while loop

当…. A is True

Do()

Only stop when A become False.

What if A is always true?
There is no error will occur at this time, but our code never stops!
So we must force it to stop!
Therefore, please double check your code when you are write a while loop.

How to use?

- Can do the same thing with for loop

```
>>> i = 1
>>> while i < 10:
...     print(i)
...     i += 1
...
1
2
3
4
5
6
7
8
9
>>>
```

Consider a function by given docstring

```
def find_letter_n_times(s, letter, n):
    """ (str, str, int) -> str

    Precondition: letter occurs at least n times in s

    Return the smallest substring of s starting from index 0 that contains
    n occurrences of letter.

    >>> find_letter_n_times('Computer Science', 'e', 2)
    'Computer Scie'
    """
```

complete the body part:

```
def find_letter_n_times(s, letter, n):
      """ (str, str, int) -> str

      Precondition: letter occurs at least n times in s

      Return the smallest substring of s starting from index 0 that contains
      n occurrences of letter.

      >>> find_letter_n_times('Computer Science', 'e', 2)
      'Computer Scie'
      """
      i = 0
      count = 0
      while i < len(s) and count < n:
            if s[i] == letter: # checking if current letter matches parameter
                  count = count + 1
            i = i + 1
      return s[:i]
```

another function:

```
def every_nth_character(s, n):
      """ (str, int) -> str

      Precondition: n > 0

      Return a string that contains every nth character from s, starting at index 0.

      >>> every_nth_character('Computer Science', 3)
      'CpeSee'
      """
```

```
def every_nth_character(s, n):
      """ (str, int) -> str

      Precondition: n > 0

      Return a string that contains every nth character from s, starting at index 0.

      >>> every_nth_character('Computer Science', 3)
      'CpeSee'
      """

      result = ''
      i = 0

      while i < len(s): # needs to make sure i can't be len(s)
            result = result + s[i]
            i = i + n #TODO
      return result
```

Part 5: start our first assignment!

Reading the handout carefully!