第二讲
列表
Lecture 2
Lists

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

定义
Definition

- 列表特定元素的集合，但与集合不同的是，列表中元素是按特定顺序排列的（通常是创建时的顺序）。列表的例子如字母表，0 到 9 的数字列表。与其它语言中列表的概念不同，Python中列表的元素不必同质，即可以将任何类型的元素放入列表中，并且列表中的元素也不必以任何特定方式相关。因为列表通常包含多个元素，因此列表名称通常为复数形式。在 Python 中，创建列表可以用方括号 ([]) ，列表中的各个元素 列表以逗号分隔。当将其它类型如元组转换为列表时，也会显式调用函数list。

  A list is a collection of items in a particular order. Examples of lists includes alphabet of English letters, the digits from 0–9. The elements in the same list are not necessary to be homogenous, so the items put into a list can be any type and don’t have to be related in any particular way. Because a list usually contains more than one element, it’s good to make the name of the list plural. In Python, square brackets ([]) indicate a list, and individual elements in the list are separated by commas. When explicitly converts other types into list, it can invoke the function list() in an obvious way.

定义
Definition

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> letters
['a', 'b', 'c', 'd', 'e']
>>> letters = list(('a', 'b', 'c', 'd', 'e'))
>>> letters
['a', 'b', 'c', 'd', 'e']
>>> letters[0]
'a'
>>> letters[0].upper()
'A'
>>> objects = [123, "abc", ['e', 'd']]
>>> objects
[123, 'abc', ['e', 'd']]
>>>
```

- 列表是元素的有序集合，因此可以元素的位置或索引来访问列表中的任何元素。 访问列表中的元素通过列表的名称后跟用方括号括起来的元素的索引来实现。

Since lists are ordered collections, so to access any element in a list can be via the position, or index, of the item desired. To access an element in a list, write the name of the list followed by the index of the item enclosed in square brackets.

- 对于Python，列表中的第一项为位置 0，而不是位置1，这意味着索引由0开始，不是1。

Python considers the first item in a list to be at position 0, not position 1, which means index Positions Start at 0, Not 1.

列表操作
Operations of Lists

```
>>> example_list = ["foo", "bar", "foobar"]
>>> example_list
['foo', 'bar', 'foobar']
>>> example_list[2] = "blarblar"
>>> example_list
['foo', 'bar', 'blarblar']
>>>
```

- Python中的列表基本都是动态使用的，这意味着初始构建一个列表，然后在程序运行时添加和删除其中的元素。

  In most cases lists created are used in a dynamic way, meaning one builds a list and then adds and removes elements from it in the course of the program runs.

- 和其他语言类似，访问列表中的元素返回的是左值。因此修改元素的语法类似于访问列表中元素的语法，即列表名称后跟要更改的元素的索引，然后赋成该元素期望的新值。

  Just as other languages, left value is returned when accessing the element in the list. Hence, the syntax for modifying an element is similar to the syntax for accessing an element in a list. To change an element, use the name of the list followed by the index of the element to be changed, and then assign the new value that the item is to have.

列表操作
Operations of Lists

- 在实际应用中，可能出于多种原因需要将新元素添加到列表中。将新元素添加到列表的最简单方法是调用append()函数将项目附加到列表末尾。

  In application, it might desire to add a new element to a list for many reasons. The simplest way to add a new element to the end of the list is by invoking append().

- 某种情况下，添加的元素较多且位于另一个列表，则可以调用extend()函数将待添加的列表与原列表合并。

  In some cases, the elements to be added are in another list, you can call the extend() function to merge the list with the original one.

```
>>> original_list = ["foo", "bar", "blarblar"]
>>> original_list
['foo', 'bar', 'blarblar']
>>> original_list.append("blarblar")
>>> original_list
['foo', 'bar', 'blarblar', 'blarblar']
>>> another_list = ["this", "is", "a", "test"]
>>> original_list.extend(another_list)
>>> original_list
['foo', 'bar', 'blarblar', 'blarblar', 'this', 'is', 'a', 'test']
```

列表操作
Operations of Lists

- 当需要在列表中指定位置添加新元素时，可以使用 insert() 方法。此时，需要指定新元素的索引和值。

  You can add a new element at any position in your list by using the insert() method. You do this by specifying the index of the new element and the value of the new item.

- 同样，在程序中有从列表中删除一个或一组元素的需求。在Python中，可以根据元素在列表中的位置或元素的值删除一个元素。

  Often, there are scenarios that an item or a set of items need to be removed from a list. In Python, one can remove an item according to its position in the list or according to its value.

- 如果知道要从列表中删除的元素的位置，则可以使用 del 语句；但有时还希望在将元素从列表中删除后还能加以利用，此时可以用pop()函数。

  If one knows the position of the item you want to remove from a list, you can use the del statement. Sometimes you'll want to use the value of an item after you remove it from a list, in this case pop() can be utilized.

列表操作
Operations of Lists

```
>>> examples = ["foo", "bar", "foobar"]
>>> examples
['foo', 'bar', 'foobar']
>>> examples.insert(0, "blarblar")
>>> examples
['blarblar', 'foo', 'bar', 'foobar']
>>> del examples[0]
>>> examples
['foo', 'bar', 'foobar']
>>> examples.pop()
'foobar'
>>> examples
['foo', 'bar']
>>> item = examples.pop(0)
>>> assert(item == "foo")
>>> examples
['bar']
```

- 不加指明时，pop()仅从列表末尾弹出元素，需要从特定位置弹出元素时，需要将索引用作调用pop()时的参数。

  Without specification, pop() only pops elements from the end of the list, and when it is needed to pop an element from a specific position, pass the index as an argument when calling pop().

列表操作
Operations of Lists

```
>>> examples = ["foo", "bar", "foobar"]
>>> examples
['foo', 'bar', 'foobar']
>>> examples.remove("foo")
>>> examples
['bar', 'foobar']
>>> examples = ["foo", "bar", "foobar", "foobar"]
>>> examples
['foo', 'bar', 'foobar', 'foobar']
>>> examples.remove("foobar")
>>> examples
['foo', 'bar', 'foobar']
```

- 有时可能不知道要从列表中删除的元素的位置，但如果知道要删除的元素的值，则可以使用 remove() 方法。

  Sometimes it won't be possible to know the position of the value to be removed from a list. If one only knows the value of the item to be removed, remove() method caters to the need.

- remove() 方法仅删除指定的值的第一次出现。如果该值有可能在列表中出现多次，则需要使用循环来确保删除所有出现的值。

  The remove() method deletes only the first occurrence of the value you specify. If there's a possibility the value appears more than once in the list, you'll need to use a loop to make sure all occurrences of the value are removed.

列表操作
Operations of Lists

- 通常，列表将以不可预测的顺序创建，但可能基于某种考虑，希望列表以特定顺序显示信息。在某些情况下，可能想要保留列表的原始顺序，而有时需要更改原始顺序。Python提供了多种不同的方式来组织列表，具体取决于具体情况。

  Often, lists will be created in an unpredictable order. But for some reason, one will frequently want to present the elements in a particular order. In some circumstances, one wants to preserve the original order of the original list, and other cases one wants to change the original order. Python provides a number of different ways to organize the lists, depending on the situation.

- 如果要对列表进行永久性排序，则使用sort()方法；要维持原列表不变，仅返回临时排序的列表，可以使用 sorted() 函数。

  The sort() method can be used to permanently sort a list. However, to maintain the original list and only return a temporarily sorted list, the sorted() function can be used.

```
>>> cars = ["bmw", "audi", "toyota", "subaru"]
>>> cars
['bmw', 'audi', 'toyota', 'subaru']
>>> cars.sort()
>>> cars
['audi', 'bmw', 'subaru', 'toyota']
>>> cars = ["bmw", "audi", "toyota", "subaru"]
>>> sorted(cars)
['audi', 'bmw', 'subaru', 'toyota']
```

列表操作
Operations of Lists

- 当将列表按默认顺序反向排序时，则可以通过在调用sort()方法时传递参数reverse=True实现；注意，当仅将列表元素反转时，则需要调用reverse()实现。

  By passing the parameter reverse=True when calling the sort() method, the list is sorted in the reverse default order. Note that if the list elements are simply reversed, just call reverse() function.

- 当需要获得列表长度时，可以使用 len() 函数。

  One can quickly find the length of a list by using the len() function..

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

列表操作
Operations of Lists

- 当需要遍历列表中的所有条目，对每个条目执行相同的操作时，可以使用 Python 的 for 循环。

  If it is needed to run through all entries in a list, performing the same operation with each item in the list, one can use Python's for loop.

- 如果需要生成特定范围的数字列表，可以使用Python的range() 函数。

  Python's range() function can be used to generate a list of numbers in a specific range.

```
>>> cars = ["bmw", "audi", "toyota", "subaru"]
>>> for car in cars:
...     print(car)
...
bmw
audi
toyota
subaru
>>> evens = list(range(0, 10, 2))
>>> evens
[0, 2, 4, 6, 8]
```

列表解析
List Comprehension

- 当我们基于一个列表生成一个新列表时，一种方法是根据过滤条件，利用for循环生成新的列表，另外一种方法是使用列表推到，即在一行代码中生成新的列表。列表推导是将 for 循环和新元素的创建组合到一行中，并自动附加每个新元素。

  To generate a new list based on an existing list, one way is to use the for loop iterating over the existing list simultaneously applying the filtering condition. Another way is to use list comprehension, which allows one to generate this same list in just one line of code. A list comprehension combines the for loop and the creation of new elements into one line, and automatically appends each new element.

```
>>> evens = list(range(0, 10, 2))
>>> evens
[0, 2, 4, 6, 8]
>>> integers = list(range(0, 10))
>>> evens = [e for e in integers if (e & 1 == 0)]
>>> evens
[0, 2, 4, 6, 8]
```

切片
Slice

- 在某些情况下，可能需要访问和处理列表中的某个范围的元素。访问列表中单个元素的方式，称为索引；访问列表中的特定范围元素的方式，称为切片。

  In some circumstance, one needs to process items in a specific range of the list. To access the single element in a list is by using index, while to access the elements in a range is by using slice.

- 要生成切片，需要指定使用的第一个和最后一个元素的索引。注意，Python中的区间为左闭右开区间，因此，实际指定时，须在要访问的最后一个元素的索引基础上加1。例如，要输出列表中的前三个元素，须将索引 指定为0到3，此时将返回元素 0、1 和 2。

  To make a slice, one needs to specify the index of the first and last elements to work with. In Python, the interval or range is left-closed-right-open, which means to specify the range, the last index actually indicates Python stops one item before the it. For example, To output the first three elements in a list, one will request indices 0 through 3, resulting elements 0, 1, and 2 returned.

切片
Slice

- 通常使用的切片是以正向方式建立的，即从前向后访问或处理列表中的若干元素。例如，访问列表从第二个位置开始的连续3个元素，其切片为1:4

  Generally, slice is generated in a forward style, aka, to access or process elements from front to back. For instance, to process three consecutive elements starting from the second position in the list, the slice should be written as 1:4.

- 当构成切片的索引中，第一个索引省略时，默认是从第一个位置即0开始；当省略最后一个索引时，默认到列表的结束，类似于C++标准模板库的哨兵元素位置。

  If the first index is omitted in the indices constituting the slice, the slice is treated as starting from the first position of the list, or 0. If the next index is omitted, the slice extends to the end of the list, comparable to the position of pivot element in C++ STL.

```
>>> planets=["mercury", "venus", "earth", "mars", "jupiter", "saturn", "uranus", "neptune"]
>>> planets[1:4]
['venus', 'earth', 'mars']
>>> planets[:5]
['mercury', 'venus', 'earth', 'mars', 'jupiter']
>>> planets[5:]
['saturn', 'uranus', 'neptune']
```

切片
Slice

- 列表的最后一个位置方便起见，也可以用−1来表示。

  The last position of the list can also be explicitly referenced by −1.

- 实际上，构成切片的除了起始与终止索引外，还有一个步长参数，如果不写，则默认为1。

  Actually, besides the indices which constitutes of the slice, there is a third step parameters, which is by default 1 if omitted.

```
>>> planets
['mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune']
>>> planets[-1]
'neptune'
>>> planets[4:-1]
['jupiter', 'saturn', 'uranus']
>>> planets[::2]
['mercury', 'earth', 'jupiter', 'uranus']
>>> planets[1::2]
['venus', 'mars', 'saturn', 'neptune']
```

切片
Slice

- 实际上，Python中对列表的索引可以是负整数，表示从后到前访问。例如， −1表示最后一个位置， −2表示倒数第二个位置，以此类推。

  In fact, indices into lists in Python can be negative integers, meaning back-to-front access. For example, −1 is the last position, −2 is the second-to-last position, and so on.

- 除了索引，构成切片的步长亦可以为负。当为负数时，表示由后向前索引。

  Besides the indices, the step in the slice can also be negative. In the negative instance, it means traversing the list in a back-to-front manner.

```
>>> planets
['mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune']
>>> planets[-1]
'neptune'
>>> planets[-2]
'uranus'
>>> planets[-1::-1]
['neptune', 'uranus', 'saturn', 'jupiter', 'mars', 'earth', 'venus', 'mercury']
>>> planets[-1::-2]
['neptune', 'saturn', 'mars', 'venus']
>>> planets[-2::-2]
['uranus', 'jupiter', 'earth', 'mercury']
```

列表拷贝
List Copying

- 在 Python 中，一切都被视为对象处理的，对象都具有以下三个属性：
  - 标识—是指对象在计算机内存中引用的地址。
  - 类型—是指创建的对象的类型，例如整数、列表、字符串等。
  - 值—是指对象存储的值。例如List=[1,2,3] 将保存数字 1,2 和 3。

  虽然 ID 和 Type 一经创建就无法更改，但特定类型对象的值可以更改。

  In Python, everything is treated as an object. Every object has these three attributes:

  - Identity – This refers to the address that the object refers to in the computer's memory.
  - Type – This refers to the kind of object that is created. For example- integer, list, string etc.
  - Value – This refers to the value stored by the object. For example – List=[1,2,3] would hold the numbers 1,2 and 3

  While ID and Type cannot be changed once it's created, values can be changed for Mutable objects.

列表拷贝
List Copying

- 可变性是对Python对象的内部状态（值）能否改变的一种描述。简而言之，内部状态可以改变的对象是可变的，一旦创建就不允许对其进行任何更改的对象是不可变对象。

  Mutable is a way to depict that the internal state of the object is changeable or not. Put it in a simple way: An object whose internal state can be changed is mutable. On the other hand, immutable doesn't allow any change in the object once it has been created.

- Python中的可变对象包括：列表，集合，字典，及用户定义的类（取决于用户定义的特征）；不可变的对象有：数字（整数、有理数、浮点数、小数、复数和布尔数），字符串，元组，特定集合，及用户定义的类（取决于用户定义的特征）。

  Objects that are mutable includes lists, sets, dictionaries and user-defined classes (It purely depends upon the user to define the characteristics). Objects of built-in type that are immutable are: numbers (integer, rational, float, decimal, complex & booleans), strings, tuples, frozen Sets and user-defined classes (It purely depends upon the user to define the characteristics)

列表拷贝
List Copying

- 在Python中，赋值操作 = 实际上并不是创建副本，它只是对同一对象的引用。但是由于Python对于可变对象与不可变对象的处理方式不同，会导致赋值后的变量行为的差异。

  Python never makes a copy with the assignment =, it just refers to the same object. However, since Python handles mutable and immutable objects in different ways, this causes different behaviors for variables after assignment.

- 以B = A为例，若A为不可变对象，则当更新A时，实际上是将A与新的对象关联，所以此时不影响B指向的原来的对象；但当A为可变对象时，由于对象允许改变，则经由A更新时，B亦能同时反映出此种变化。。

  Considering B = A, if A is immutable, updating of A is essentially creating a new object and associating with the label A, which impacts nothing of B. However, if A is mutable, which means the object associated with label A permits modification of itself, so any update of it via A is simultaneously reflected by displaying B.

列表拷贝
List Copying

```
>>> a = [1, 2, 3]
>>> b = a
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
>>> b
[1, 2, 3, 4]
>>> import copy
>>> c = copy.copy(a)
>>> c
[1, 2, 3, 4]
>>> a.append(5)
>>> a
[1, 2, 3, 4, 5]
>>> b
[1, 2, 3, 4, 5]
>>> c
[1, 2, 3, 4]
```

- 基于上面讲述，我们知道赋值操作 = 实际上对列表来说，完全没有进行拷贝。

  Based on the previous knowledge, we know that assignment = preforms no copy.

- 因此，当需要拷贝可变对象时，Python提供了特定的包和函数以完成此功能。

  Python provides specific package and functions for copying mutable objects.

```
>>> a = [[1, 2], [3, 4]]
>>> b = copy.copy(a)
>>> c = copy.deepcopy(a)
>>> b
[[1, 2], [3, 4]]
>>> c
[[1, 2], [3, 4]]
>>> a.append([5, 6])
>>> a
[[1, 2], [3, 4], [5, 6]]
>>> b
[[1, 2], [3, 4]]
>>> c
[[1, 2], [3, 4]]
>>> a[2].append(7)
>>> a
[[1, 2], [3, 4], [5, 6, 7]]
>>> b
[[1, 2], [3, 4]]
>>> c
[[1, 2], [3, 4]]
>>> a[1].append(8)
>>> a
[[1, 2], [3, 4, 8], [5, 6, 7]]
>>> b
[[1, 2], [3, 4, 8]]
>>> c
[[1, 2], [3, 4]]
>>>
```

列表拷贝
List Copying

- 上面的方法似乎解决了赋值操作的不足，但实际上，上面的方法仅从表面上解决了问题，其所进行的拷贝称为浅拷贝。其直接原因是，如果是嵌套列表，那么这些对象的行为可能与预期不同。因此，必要时可进行深拷贝。

  The method showed above seems solve the insufficiency of assignment, however, it just solves the problem superficially, and the corresponding copying method is called shallow copy. The obvious reason is for nested list, its behavior might be different from the original expectation. Therefore, deep copy needs to be performed if necessary.

列表拷贝
List Copying

```
>>> a = [[1, 2], [3, 4]]
>>> b = a[:]
>>> b
[[1, 2], [3, 4]]
>>> a.append([5, 6])
>>> a
[[1, 2], [3, 4], [5, 6]]
>>> b
[[1, 2], [3, 4]]
>>> a[0].append(7)
>>> a
[[1, 2, 7], [3, 4], [5, 6]]
>>> b
[[1, 2, 7], [3, 4]]
```

- 由于切片操作会返回新的列表，则进行拷贝的另一种思路是利用切片操作。但这种方法对于嵌套列表来说，依然是浅拷贝。

  Since slicing a list returns a new object, another way for copying a list is to utilize slicing operation. But this method is still performing shallow copying for nested list.