CS112 Introduction to Python Programming

Session 03: Lists

Shengwei Hou
Ph.D., Assistant Professor
Department of Ocean Science and Engineering

Fall 2022

SUSTech Southern University of Science and Technology

明德求是 日新自强

VIRTUE | TRUTH | ADVANCE

Contents

List

- List is a container that holds a number of items
- Lists are defined by a pair of square brackets with individual elements (can be of mixed type) separated by commas:

```
a = [0, 1, 1, 2, 3, 5, 8, 13]
b = [5., "girl", 2+0j, "horse", 21]
>>> type(b)
<class 'list'>
>>> c = []     Empty list
>>> c
[]
>>> type(c)
<class 'list'>
```

List operations

- Lists can be concatenated using the + sign, and the * operator is used to create a repeated sequence of list:

```
>>> list_1 = [1, 3, 5, 7]
>>> list_2 = [2, 4, 6, 8]
>>> list_1 + list_2
[1, 3, 5, 7, 2, 4, 6, 8]
>>> list_1 * 3
[1, 3, 5, 7, 1, 3, 5, 7, 1, 3, 5, 7]
>>> list_1 == list_2
False
>>> 5 in list_1
True
>>> 5 not in list_2
True
```

Adding lists concatenates them, just as the "+" operator concatenates strings

`range()` function

- `range()` creates a uniformly spaced sequence of integers. Its general form is `range([start,] stop[, step])`:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(3,10))
[3, 4, 5, 6, 7, 8, 9]
>>> list(range(0,10,2))
[0, 2, 4, 6, 8]
>>> list(range(-5,5,2))
[-5, -3, -1, 1, 3]
>>> list(range(5,0,-1))
[5, 4, 3, 2, 1]
```

- *The second argument ends the list, but is not included in the list*
- *If the third argument is not included, it's taken to be 1*

- `range()` generates a sequence of numbers one at a time:

```
>>> range(10)
range(0, 10)
```

List indexing

- Individual elements of a list can be accessed using the variable name for the list with an integer in square brackets:

```
>>> b = [7., "girl", 2+0j, "horse", 21]
>>> b[0]
7.0
>>> b[1]
'girl'
>>> b[4]
21
```

```
>>> b[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

- The last element can also be by index -1:

```
>>> b[-1]
21
>>> b[-2]
'horse'
```

Python's index starts from zero

List slicing

```
>>> b = [7., "girls & boys", 2+0j,
3.14159, 21]

>>> b[1:4]
['girls & boys', (2+0j), 3.14159]

>>> b[2:]
[(2+0j), 3.14159, 21]

>>> b[:3]
[7.0, 'girls & boys', (2+0j)]

>>> b[:]
[7.0, 'girls & boys', (2+0j),
3.14159, 21]

>>> b[1:-1]
['girls & boys', (2+0j), 3.14159]

>>> b[::2]
[7.0, (2+0j), 21]

>>> b[1::2]
['girls & boys', 3.14159]

>>> b[::-1]
[21, 3.14159, (2+0j), 'girls & boys',
7.0]
```

`[start:end:stepsize]`

The default step size is 1

List modification

- Lists are mutable in nature as the list items can be modified after creation:

```
>>> b = [7., "girl", 2+0j, "horse", 21]

>>> b[0] = b[0] + 2
>>> b
[9.0, 'girl', (2+0j), 'horse', 21]

>>> b[1] = b[1] + " & boy"
>>> b
[9.0, 'girl & boy', (2+0j), 'horse', 21]

>>> b[3] = 2.5
>>> b
[9.0, 'girl & boy', (2+0j), 2.5, 21]
```

List modification

- When you assign an existing list variable to a new variable, an assignment (=) on lists does not make a new copy. Instead, assignment makes both the variable names point to the same list in memory:

```
>>> zoo = ['Zebra', 'Tiger', 'Lion']

>>> forest = zoo
>>> forest
['Zebra', 'Tiger', 'Lion']

>>> zoo[0] = "Fox"
>>> zoo
['Fox', 'Tiger', 'Lion']
>>> forest
['Fox', 'Tiger', 'Lion']
```

Assigning an existing list variable to a new variable does not create a new copy of the existing list items

Built-in functions for lists

Southern University of Science and Technology

```
>>> zoo = ['Zebra', 'Tiger', 'Lion']
>>> len(zoo)
3

>>> zoo_sorted = sorted(zoo)
>>> zoo_sorted
['Lion', 'Tiger', 'Zebra']
>>> zoo
['Zebra', 'Tiger', 'Lion']
```

- *The* `sorted()` *function returns a modified copy of the list while without modifying the original list*
- *The list is sorted based on the ASCII value*

```
>>> numbers = [1, 2, 3, 4, 5]
>>> sum(numbers)
15
>>> max(numbers)
5
>>> min(numbers)
1
```

List methods

```
>>> dir(list)
['__add__', '__class__', '__contains__',
'__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__',
'__imul__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'clear',
'copy', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
```

List method examples

Southern University of Science and Technology

- The `append()` adds a single item to the end of the list:

```
>>> list_1 = [1,2,3]
>>> list_1.append(1)
>>> list_1
[1, 2, 3, 1]
```

- The `count()` counts the number of times the item has occurred in the list and returns it:

```
>>> list_1.count(1)
2
```

- The `insert()` method inserts the item at the given index, shifting items to the right:

```
>>> list_1.insert(1,5)
>>> list_1
[1, 5, 2, 3, 1]
```

The list size changes dynamically whenever you add or remove the items.

List method examples

```
>>> list_1 = [1, 5, 2, 3, 1]
>>> list_1.index(5)
1
```

`index()` returns the index for the given item from the start of the list

```
>>> list_2 = list_1.copy()
>>> list_2[1] = 6
>>> list_1
[1, 5, 2, 3, 1]
>>> list_2
[1, 6, 2, 3, 1]
```

`copy()` creates a new copy of the existing list

```
>>> list_1.reverse()
>>> list_1
[1, 3, 2, 5, 1]
>>> list_1.sort()
>>> list_1
[1, 1, 2, 3, 5]
```

List method examples

- The `remove()` searches for the first instance of the given item in the list and removes it

```
>>> list_1.remove(3)
>>> list_1
[1, 1, 2, 5]
```

- The `pop()` removes and returns the item at the given index; returns the rightmost item if the index is omitted

```
>>> list_1.pop(3)
5
>>> list_1
[1, 1, 2]
>>> list_1.pop()
2
>>> list_1
[1, 1]
```

List method examples

- The `remove()` searches for the first instance of the given item in the list and removes it

```
>>> list_1.remove(3)
>>> list_1
[1, 1, 2, 5]
```

- The `pop()` removes and returns the item at the given index; returns the rightmost item if the index is omitted

```
>>> list_1.pop(3)
5
>>> list_1
[1, 1, 2]
>>> list_1.pop()
2
>>> list_1
[1, 1]
```

The `del` Statement

- `del` statement removes an item from a list based on its index
- The difference between `del` and `pop()` is that `del` does not return any value while `pop()` returns a value
- The `del` statement can also be used to remove slices from a list or clear the entire list

```
>>> a = [5, -8, 99.99, 432, 108, 213]
>>> del a[0]
>>> a
[-8, 99.99, 432, 108, 213]
>>> del a[2:4]
>>> a
[-8, 99.99, 213]
>>> del a[:]
>>> a
[]
```

```
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
NameError: name 'a' is not defined
```

Multidimensional lists

SUSTech Southern University of Science and Technology

- Multidimensional lists, or lists of lists, or nested list:

```
>>> a = [[3, 9], [8, 5], [11, 1]]
>>> a[0]
[3, 9]
>>> a[1][0]
8
>>> a[2][1]
1
>>> a[2][1] = 5
>>> a
[[3, 9], [8, 5], [11, 5]]
```

Exercises

Exercise 1

```python
fruits = ["apple", "banana", "cherry"]
```

1. Print the second item in the `fruits` list.
2. Change the value from "apple" to "kiwi", in the `fruits` list.
3. Use the `append` method to add "orange" to the `fruits` list.
4. Use the `insert` method to add "lemon" as the second item in the `fruits` list.
5. Use the `remove` method to remove "banana" from the `fruits` list.
6. Use negative indexing to print the last item in the list.
7. Use the correct syntax to print the number of items in the list.
8. Use a range of indexes to print the third and fourth item in the list.

Exercise 2

Write a Python program to print a specified list after removing the 0th, 4th and 5th elements.

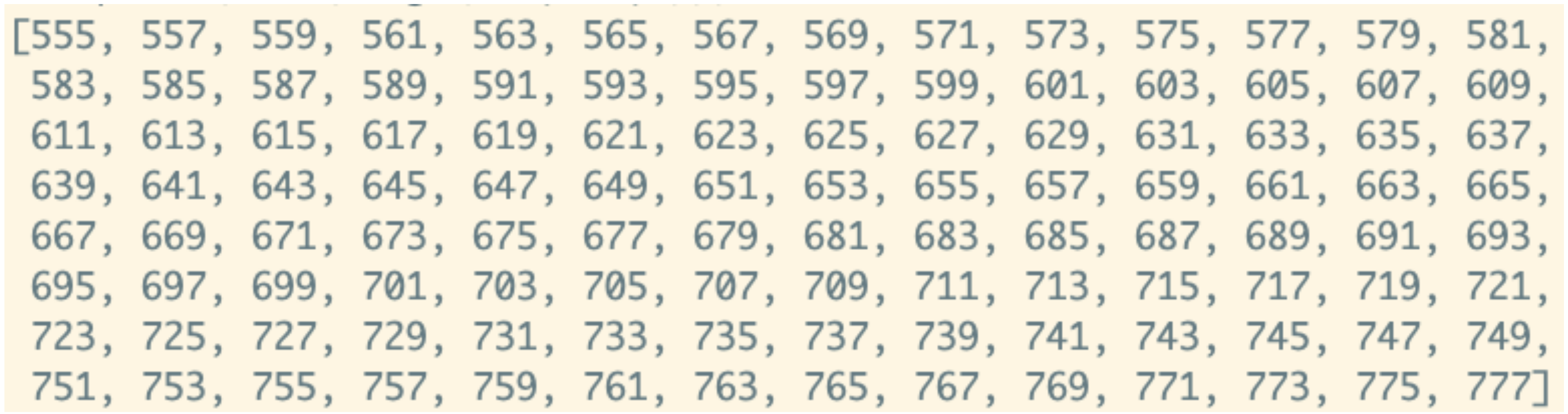
Sample List : `['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']`

Expected Output : `['Green', 'White', 'Black']`

Exercise 3

Write a Python program to print a list of all odd numbers from 555 to 777.

Expected Output :



```
[555, 557, 559, 561, 563, 565, 567, 569, 571, 573, 575, 577, 579, 581,
 583, 585, 587, 589, 591, 593, 595, 597, 599, 601, 603, 605, 607, 609,
 611, 613, 615, 617, 619, 621, 623, 625, 627, 629, 631, 633, 635, 637,
 639, 641, 643, 645, 647, 649, 651, 653, 655, 657, 659, 661, 663, 665,
 667, 669, 671, 673, 675, 677, 679, 681, 683, 685, 687, 689, 691, 693,
 695, 697, 699, 701, 703, 705, 707, 709, 711, 713, 715, 717, 719, 721,
 723, 725, 727, 729, 731, 733, 735, 737, 739, 741, 743, 745, 747, 749,
 751, 753, 755, 757, 759, 761, 763, 765, 767, 769, 771, 773, 775, 777]
```

Exercise 4

SUSTech Southern University of Science and Technology

Write a Python script named as 'id_recorder.py' to record three user entered student IDs in a list.

Expected output when running the script :

```
python id_recorder.py
Enter your ID:123
Current student IDs are ['123']
Enter your ID:234
Current student IDs are ['123', '234']
Enter your ID:345
Current student IDs are ['123', '234', '345']

Process finished with exit code 0
```