Python Lists and Tuples

Distributed Network and Systems Laboratory,
Chonnam National University
Presenter : Shivani Kolekar

Recap

- ✓ **Introduction to loops (for and while loops)**
- ✓ **Loop control statements (break and continue)**
- ✓ **Writing programs with loops**

Today's Goals

- ✓ **Introduction to List**
- ✓ **Types of Elements**
- ✓ **Accessing and Modifying List Elements**
- ✓ **Introduction to Other Data Structures (Tuples etc.)**
- ✓ **Q & A**

Sequence

- ✓ Sequence: an object that contains multiple items of data
- ✓ The items are stored in sequence one after another
- ✓ Python provides different types of sequences, including lists and tuples
- ✓ The difference → a list is mutable and a tuple is immutable

```
>>> sequence = [2, 3, 5, 7, 11, 13] # a list
```

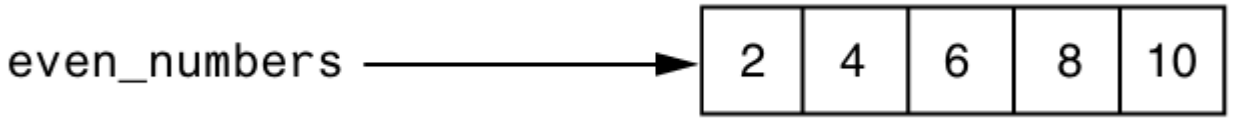
Introduction to lists

- A **list** is an object that contains multiple data items.
- They're dynamic data structures, meaning that items may be added to them or removed from them.
- A list is an object that contains multiple data items. Each item that is stored in a list is called an element.

A list of integers

even_numbers = [2, 4, 6, 8, 10] ⇦ statement that creates a list of integers

- ✓ The items that are enclosed in brackets and separated by commas are the list elements.
- ✓ After this statement executes, the variable even_numbers will reference the list, as shown in Figure,



List of strings

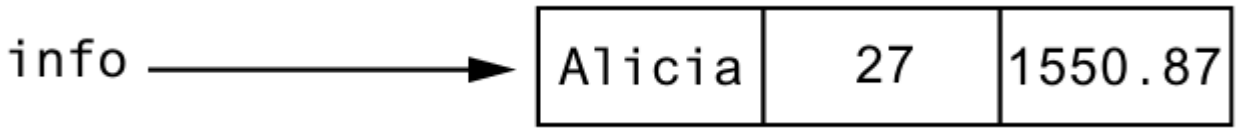
```
names = ['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
```

- ✓ This statement creates a list of five strings. After the statement executes, the name variable will reference the list as shown in Figure,

List holding different types

```
info = ['Alicia', 27, 1550.87]
```

- ✓ This statement creates a list containing a string, an integer, and a floating-point number.
- ✓ After the statement executes, the info variable will reference the list as shown in Figure,



list() function

- ✓ Python also has a built-in list() function that can convert certain types of objects to lists

- ✓ For example, lets convert the range function's iterable object to a list:

list_example1.py

```
numbers = list(range(5))
print(numbers)
```

```
[0, 1, 2, 3, 4]
```

Repetition Operator

✓ Previously, we learned that the * symbol multiplies two numbers

However,

- when the operand on the left side of the * symbol is a sequence (such as a list) and the operand on the right side is an integer, it becomes the repetition operator.

- **list * n** **#** *list is a list and n is number of copies to make*

Repetition Operator

✓ Simple examples to repeat a list using Repetiition operator

Repetition_operator (python console)

```
1   >>> numbers = [0] * 5 Enter
2   >>> print(numbers) Enter
3   [0, 0, 0, 0, 0]
4   >>>
```

List_repeat.py

```
numbers = [1, 2, 3] * 3
print(numbers)

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

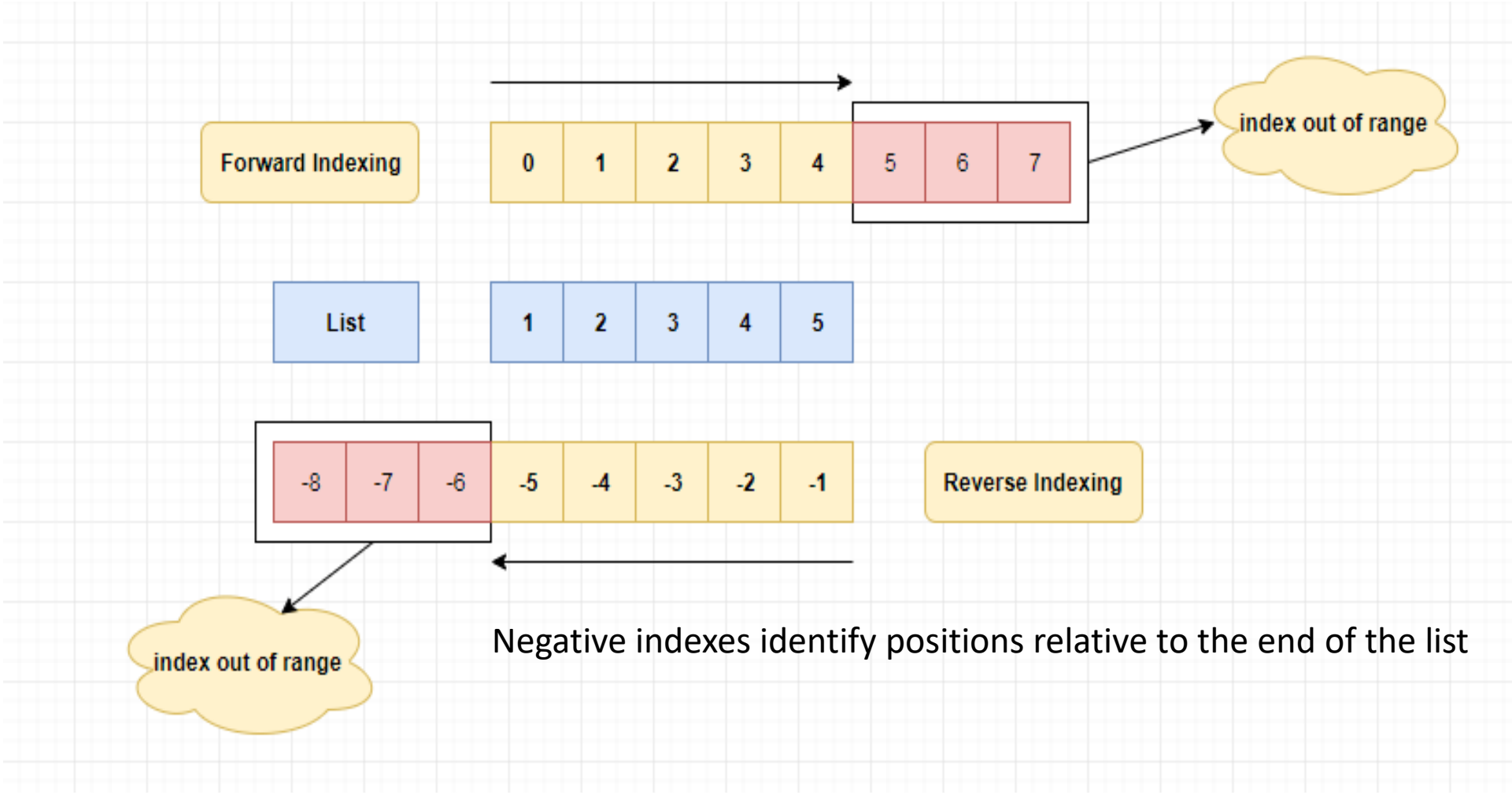
Indexing

- ✓ Enables access to individual element in list
- ✓ Index of first element in the list is 0, and n'th element is n-1
- ✓ Negative indexes identify positions relative to the end of the list

Try this code snippet and observe the result:

```
my_list = [10, 20, 30, 40]
print(my_list[0], my_list[1], my_list[2], my_list[-3])
```

Indexing -Forward, Reverse



Negative indexes identify positions relative to the end of the list

len function ()

- An IndexError exception is raised if an invalid index is used
- len function: returns the length of a sequence such as a list

Example:

```
my_list = [10, 20, 30, 40]
size = len(my_list)
print(size)

4
```

size = len(*my_list*)

✓ Returns the number of elements in the list,

✓ so the index of last element is len(list)-1

Lists Are Mutable

- ✓ Lists are mutable, and so their elements can be changed

- ✓ **list[1] = new_value** can be used to assign a new value to a list element
- ✓ Must use a valid index to prevent raising of an IndexError exception

List_mutability.py

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
#
numbers[0] = 99         #changing the value of index 0
numbers[2] = 'python'   #changing the value of index 2
numbers[4] = -46        #changing the value of index 4

print(numbers)
```

```
[1, 2, 3, 4, 5]
[99, 2, 'python', 4, -46]
```

Concatenating Lists

- ✓ To concatenate means to join two things together.
- ✓ You can use the + operator to concatenation.

Conatenating_list.py

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list3 = list1 + list2

print(list3)
```

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

Using += operator for concatenation

```
Python Console
>>> girl_names = ['Joanne', 'Karen', 'Lori']
>>> girl_names += ['Monika', 'Rachel']
>>> print(girl_names)
['Joanne', 'Karen', 'Lori', 'Monika', 'Rachel']
```

List Slicing

- Slice: a span of items that are taken from a sequence

  List slicing format:

  list[start : end]

- start → index of the first element in the slice, and
- end → index marking the end of the slice.
- The expression returns a list containing a copy of the elements from start up to (but not including) end.

List_Slicing.py

```
#slicing the list of days

days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday', 'Saturday']

print('Original list:', days)

mid_days = days[2:5]              #list[start : end]
print('Sliced list:', mid_days)
```

Output →

```
Original list: ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
Sliced list: ['Tuesday', 'Wednesday', 'Thursday']
```

Sequence Operations

✓ Lists are sequences and inherit various functions from sequences.

Function	Description
x in s	x is in sequence s
x not in s	x is not in sequence s
s1 + s2	concatenates two sequences
s * n	repeat sequence s n times
s[i]	ith element of sequence (0-based)
s[i:j]	slice of sequence s from i to j-1
len(s)	number of elements in s
min(s)	minimum element of s
max(s)	maximum element of s
sum(s)	sum of elements in s
for loop	traverse elements of sequence
<, <=, >, >=	compares two sequences
==, !=	compares two sequences

Calling Functions on Lists

- ✓ Try following list functions in interactive mode
- ✓ Take a screenshot

```
Python Console>>>
>>> Lst = [1, 2, 3, 4, 5]
>>> len(Lst)
5
>>> min(Lst)      # assumes elements are comparable
1
>>> max(Lst)      # assumes elements are comparable
5
>>> sum(Lst)      # assumes summing makes sense
15
>>> Lst2 = [1, 2, "blue"]
>>> sum(Lst2)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> min(Lst2)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'

>>>
```

List_grades_example.py

```
#calculating average scores

scores = [88,76,94,90,54,67,
          78,89,84,74,93,95,
          77,76,77,51,63,68,
          93,87,89,64,79,96]

average = sum(scores) / len(scores)

print(average)
```

```
79.25
```

←Output

List Methods and useful Built-in Functions

- **append(*item*):** used to add items to a list
- *item* is appended to the end of the existing list

- **index(*item*): used to determine where an item is located in a list**
  - Returns the index of the first element in the list containing item
  - Raises ValueError exception if *item* not in the list

List Methods and useful Built-in Functions

- **insert(index, item):** used to insert item at position index in the list
- **sort():** used to sort the elements of the list in ascending order
- **remove(item):** removes the first occurrence of item in the list
- **reverse():** reverses the order of the elements in the list

List Methods and useful Built-in Functions

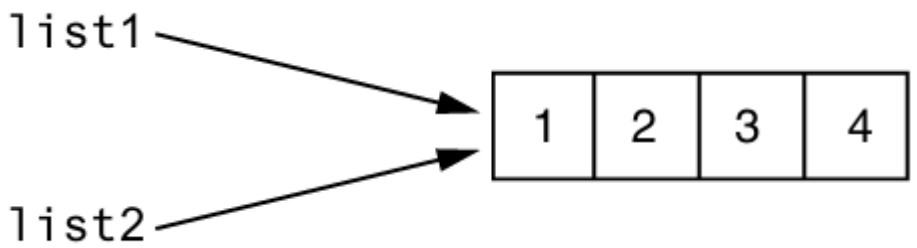
Method	Description
t.append(x)	add x to the end of t
t.count(x)	number of times x appears in t
t.extend(l1)	append elements of l1 to t
t.index(x)	index of first occurence of x in t
t.insert(i, x)	insert x into t at position i
t.pop()	remove and return the last element of t
t.pop(i)	remove and return the ith element of t
t.remove(x)	remove the first occurence of x from t
t.reverse()	reverse the elements of t
t.sort()	order the elements of t

List Examples for practice

```
>>> Lst1 = [1,2,3]
>>> Lst1.append(4)  #add 4 to the end of Lst1
>>> Lst1
[1, 2, 3, 4]
>>> Lst1.count(4)   #count occurrences of 4 in list
1
>>> Lst2 = [5,6,7]
>>> Lst1.extend(Lst2)  #add elements of Lst2 to Lst1
>>> Lst1
[1, 2, 3, 4, 5, 6, 7]
>>> Lst1.index(5)      #where does 5 occur in Lst1 ? (index)
4
>>> Lst1.insert(0,0)   #add 0 at the start of Lst1
>>> Lst1
[0, 1, 2, 3, 4, 5, 6, 7]
>>> Lst1.insert(3,'a')  #add 'a' at index 3 of Lst1 (list heterogenity)
>>> Lst1
[0, 1, 2, 'a', 3, 4, 5, 6, 7]
>>> Lst1.remove('a')    #elements from list can be removed
>>> Lst1
[0, 1, 2, 3, 4, 5, 6, 7]
>>> Lst1.pop()          #remove and return last element
7
>>> Lst1
[0, 1, 2, 3, 4, 5, 6]
>>> Lst1.reverse()      #reverse the order of elements in Lst1
>>> Lst1
[6, 5, 4, 3, 2, 1, 0]
>>> Lst1.sort()         #sorting is possible when elements are comparable
>>> Lst1
[0, 1, 2, 3, 4, 5, 6]
```

Copying Lists

✓ To make a copy of a list, you must copy the list's elements.




```
1   >>> list1 = [1, 2, 3, 4] Enter
2   >>> list2 = list1 Enter
3   >>> print(list1) Enter
4   [1, 2, 3, 4]
5   >>> print(list2) Enter
6   [1, 2, 3, 4]
7   >>> list1[0] = 99 Enter
8   >>> print(list1) Enter
9   [99, 2, 3, 4]
10  >>> print(list2) Enter
11  [99, 2, 3, 4]
12  >>>
```

Tuples

Tuples

- **Tuple**: an immutable sequence
  - Very similar to a list
  - Once it is created it cannot be changed
  - Format: `tuple_name = (item1, item2)`

  - Tuples support operations as lists
    1. Methods such as `index`
    2. Built in functions such as `len, min, max`
    3. Slicing expressions
    4. The `+` and `*` operators

- **Tuples do not support the methods:**
  append,remove,insert,reverse,sort

Tuples

- **Advantages for using tuples over lists:**
  - Processing tuples is faster than processing lists
  - Tuples are safe
  - Some operations in Python require use of tuples

- list() function: converts tuple to list
- tuple() function: converts list to tuple

```
>>> mytuple = (1,2,3)
>>> mytuple
(1, 2, 3)
>>> mylist = list(mytuple)
>>> mylist
[1, 2, 3]
>>> newlist = [4,5,6]
>>> newlist
[4, 5, 6]
>>> newtuple = tuple(newlist)
>>> newtuple
(4, 5, 6)
```

Lets Code!