Fairfield UNIVERSITY

CPSC 1101
Introduction to Computing

School of Engineering & Computing

Dept. of Computer Science & Engineering

Dr. Sidike Paheding

Class Itinerary

- Lecture
- Read Chapter 6: 164 - 183

Lists

- A **list** is a sequential collection of Python data values, where each value is identified by an index.
- **Elements** are the values that make up a list
  - thislist = ["apple", "banana", "cherry"]
  - print(thislist)

```
["apple", "banana", "cherry"]
```

  - thislist = ["apple", "banana", "cherry"]
    print(thislist[1])

```
banana
```

Sublists

- A **sublist** or **nested list** is a list within a list
  - [1, 2, 3, [1, 2, 3]]
- [] is called an **empty list**

List Accessors & Membership

- You can access elements of a list the same way a character in a string is accessed
  - alist = [3, 67, "cat", [56, 57, "dog"], [ ], 3.14, False]
  - print("String functions are available to string elements in a list:", alist[2].upper(), alist[2][0])

String functions are available to string elements in a list: CAT c

List Accessors & Membership

- **In:** returns **True** if element is member of the list otherwise it returns **False**
  - print(“teacher” in [“student”, “teacher”, “principal”])

True

List slicing

- List slice operation works just like string slicing.

The syntax for slicing a list: *mylist*[*start*:*end*:*step*]

- The first index is **the starting point** for the slice and the second number is go **up** to **but not including** that element.
- Omitting the first index (before the colon) causes the slice to start at the **beginning** of the sequence.
- Omitting the second index, the slice goes to the **end** of the sequence.

Slice a List

```
#Code that slices with the start and end arguments
numbers = [52, 54, 56, 58, 60, 62]
numbers[0:2]
 [52, 54]
numbers[:2]
 [52, 54]
numbers[4:]
 [60, 62]
alist = [3, 67, "cat", [56, 57, "dog"], [ ], 3.14, False]
print("Find sublist using negative accessor:", alist[-2: len(alist)])
Find sublist using negative accessor: [3.14, False]
```

Slice a List

```
numbers = [52, 54, 56, 58, 60, 62]

#Code that slices with the step argument
numbers[0:4:2]
 [52, 56]

 numbers[::-1]
 [62, 60, 58, 56, 54, 52]
```

List concatenation and repetition

- You can create a new list joining two existing lists together
  - print([1, 2] + [3, 4])

  [1, 2, 3, 4]
- You can also create a new list by using the repeater operator
  - print([‘go’] *4)

  ['go', 'go', 'go', 'go']

Concatenate a List

```
inventory = ["staff", "robe"]
chest = ["scroll", "pestle"]

combined = inventory + chest
print(combined)
["staff", "robe", "scroll", "pestle"]

print(inventory)
["staff", "robe"]

inventory += chest
print(inventory)
["staff", "robe", "scroll", "pestle"]
```

List Methods: del, remove, pop, clear

- The keyword **del** with slicing can be used to **delete** part of a list using an element's **index** or the entire list

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
del thislist
```

- **remove** method removes a specific element using element **value** (the first matching value).

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
```

List Methods: del, remove, pop, clear

- The **pop** method removes the specific **index** or last index if not specified. It returns the removed element from the list.

  thislist = ["apple", "banana", "cherry“, “grape”]
  print(thislist.**pop(1))** - > `banana`

  print(thislist.pop()) - > `grape`

  print(thislist) -> `['apple', 'banana', 'cherry']`

- The **clear** method empties the list

  thislist = ["apple", "banana", "cherry“, “grape”]
  print(thislist**.clear()**) -> `None`

List methods

Method	Parameters	Description
append	item	Adds a new item to the end of a list
insert	position, item	Inserts a new item at the position given
sort	none	Modifies a list to be sorted
reverse	none	Modifies a list to be in reverse order
index	item	Returns the position of first occurrence of item
count	item	Returns the number of occurrences of item

append()

- Adds an element to the end of the list.

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)

# Output: [1, 2, 3, 4]
```

insert()

- Inserts an element at a specific position.

```
my_list = [1, 2, 3]
my_list.insert(1, 'a')  # Insert 'a' at index 1
print(my_list)

# Output: [1, 'a', 2, 3]
```

sort()

- Sorts the list in ascending order (modifies the list in place).

```
my_list = [3, 1, 2]
my_list.sort()
print(my_list)

# Output: [1, 2, 3]
```

reverse()

- Reverses the elements of the list in place.

```
my_list = [1, 2, 3]
my_list.reverse()
print(my_list)

# Output: [3, 2, 1]
```

index()

- Returns the index of the **first** occurrence of a value.

```
my_list = [1, 2, 3, 2]
index_value = my_list.index(2)
print(index_value)

# Output: 1
```

count()

- Returns the number of occurrences of a value.

```
my_list = [1, 2, 2, 3]
count_value = my_list.count(2)
print(count_value)

# Output: 2
```

List Comprehension

- It offers a shorter syntax when you want to create a new list based on an existing list.

```
[expression for item in iterable if condition]
```

  - **expression**: The value or transformation you want to apply to each element.
  - **for item in iterable**: This is the loop that goes through each item in the iterable (like a list, range, or string).
  - **if condition** *(optional)*: This is an optional part where you can filter elements based on a condition.

List Comprehension

- List Comprehension with for loop

```
# Create a list of squares of numbers
from 0 to 9
squares = [x**2 for x in range(10)]
print(squares)

# Output: [0, 1, 4, 9, 16, 25, 36, 49,
64, 81]
```

List Comprehension

- List Comprehension with Condition

```
# Create a list of even numbers from 0 to 9
even_numbers = [x for x in range(10) if x % 2 == 0]
print(even_numbers)

# Output: [0, 2, 4, 6, 8]
```

Exercise 1

- Given the quote: "Bones of full fifty men lie strewn about its lair. So, brave knights, if you do doubt your courage or your strength, come no further, for death awaits you all with nasty, big, pointy teeth."
- Write a Python program that create a list with all the tokens (i.e. words) in the quote after removing "," and ".".

The split() method splits a string into a list.

Exercise 2

Create a program that builds a list of integers and calculates the mean and the median. Your solution should follow these instructions:

- Prompt the user to enter 6 integers smaller than 42.
- Read the inputs from the user and create a list. When reading the input , validate that they are smaller than 42. If they are bigger ask the user to enter another integer.
- Find the mean and the median for all the numbers in the list and print the result

Exercise 3: Bubble sort

- **Bubble sort** is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.
- Just like the movement of air bubbles in the water that rise up to the surface, each element of the array move to the end in each iteration. Therefore, it is called a bubble sort.

Exercise: Bubble sort

Suppose we are trying to sort the elements in **ascending order**.

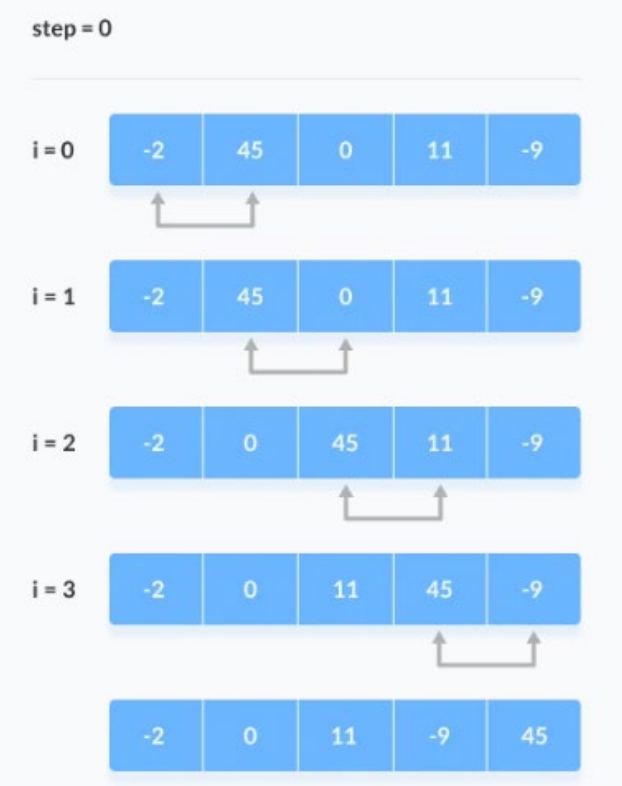
1. First Iteration (Compare and Swap)

- Starting from the first index, compare the first and the second elements.
- If the first element is greater than the second element, they are swapped.
- Now, compare the second and the third elements. Swap them if they are not in order.
- The above process goes on until the last element

Exercise: Bubble sort

Suppose we are trying to sort the elements in **ascending order**.

1. First Iteration (Compare and Swap)



Exercise: Bubble sort

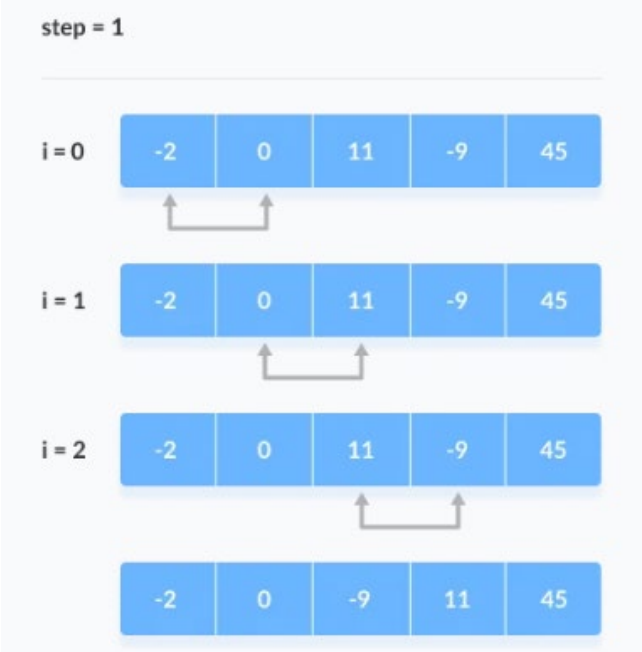
Suppose we are trying to sort the elements in **ascending order**.

2. Remaining Iteration

- The same process goes on for the remaining iterations.
- After each iteration, the largest element among the unsorted elements is placed at the end.

Exercise: Bubble sort

2. Remaining Iteration



Fairfield UNIVERSITY

CPSC 1101
Introduction to Computing

School of Engineering & Computing

Dept. of Computer Science & Engineering

Dr. Sidike Paheding