Lists and Tuples
Today’s Topics

• Sequences
• Introduction to Lists
• List Slicing
• Finding Items in Lists with the in Operator
• List Methods and Useful Built-in Functions
• Copying Lists
• Processing Lists
• Two-Dimensional Lists
• Tuples
• Plotting List Data with the `matplotlib` Package
Lists

• We’ve learned about lists already. We now talk about it in more detail, and adds some new things as well.
Sequences

• **Sequence**: an object that contains **multiple items** of data. For instance:
  
  • my_list = [6, 78, 9] is an example of a sequence.
    
    • The distinctive name of the this sequence is **list**.
    
    • So list is a type of sequence.
    
    • The items are stored in sequence one after another.

• Python provides different types of sequences, including **lists** and **tuples**.
  
  • The difference between these is that:
    
    • a list is **mutable**
    
    • a tuple is **immutable**
Lists

- **List**: an object that contains multiple data items separated by a comma.
  - An data item in a list is called an **Element**.
  - **Format**: \( list = [item1, item2, \text{ etc.}] \)
  - A list can hold items of different types.
    - \( my\_list = [7, \text{ “Ted”}, [56, 78]] \)
      - Contains three elements of type int, str and list.
  - **print** function can be used to display an entire list.
  - **list()** function can convert certain types of objects to lists.
  - For instance, to convert a tuple into a list.
The Repetition Operator and Iterating over a List

• **Repetition operator**: makes multiple copies of a list and joins them together
  - The `*` symbol is a repetition operator when applied to a sequence and an integer.
    - Sequence is left operand, number is right
  - General format: `list * n`

• You can iterate over a list using a `for` loop
  - Format: `for x in list:`
Indexing

- **Index**: a number specifying the position of an element in a list
  - Enables access to individual element in list
  - Index of first element in the list is 0, second element is 1, and n’th element is n-1
  - Negative indexes identify positions relative to the end of the list
    - The index -1 identifies the last element, -2 identifies the next to last element, etc.
The `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: `size = len(my_list)`
  • Returns the number of elements in the list, so the index of last element is `len(list)-1`
  • Can be used to prevent an `IndexError` exception when iterating over a list with a loop.
    • `for i in range(len(my_list)):`
Lists AreMutable

- Mutable sequence: the items in the sequence can be changed
  - Lists are mutable, and so their elements can be changed
- An expression such as
  - `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
Concatenating Lists

- **Concatenate**: join two things together.
- The `+` operator can be used to concatenate two lists.
  - Cannot concatenate a list with another data type, such as a number.
- The `+=` augmented assignment operator can also be used to concatenate lists.
List Slicing

my_list = [5, 10, 15, 20, 25, 30]

• **Slice**: a **span of items** that are taken from a sequence.
  ▫ List slicing format: `list[start : end]`
  ▫ Span is a list containing **copies of elements** from `start` up to, but not including, `end`.
    
    ```
    my_list[2:6]  
    ```
    [15, 20, 25]
    
    • If `start` not specified, 0 is used for start index.
    
    ```
    my_list[ :6]  
    ```
    [5, 10, 15, 20, 25, 30]
    
    • If `end` not specified, `len(list)` is used for end index.
    
    ```
    my_list[2: ]  
    ```
    [15, 20, 25, 30]

• Slicing expressions can include **negative indexes** relative to end of list.
Finding Items in Lists with the \texttt{in} Operator

• You can use the \texttt{in} operator to determine whether an item is contained in a list
  ▫ General format: \texttt{item in list}
  ▫ Returns \texttt{True} if the item is in the list, or \texttt{False} if it is not in the list.

• Similarly you can use the \texttt{not in} operator to determine whether an item is not in a list.
List Methods and Useful Built-in Functions

\[
\text{my_list} = [5, 10, 15, 20, 25, 30]
\]

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

  \[
  \text{my_list.append}(35)
  \]

  \[
  \text{my_list} = [5, 10, 15, 20, 25, 30, 35]
  \]

- **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

  \[
  \text{my_list.index}(35)
  \]

  \[
  6
  \]

List Methods and Useful Built-in Functions (cont’d.)

my_list = [5, 10, 15, 20, 25, 30, 35]

- **insert(index, item)**: used to insert item at position index in the list.

  ```python
  my_list.insert(4, 'abc')
  my_list = [5, 10, 15, 20, 'abc', 25, 30, 35]
  ```

- **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.

  ```python
  my_list.reverse()
  ```
List Methods and Useful Built-in Functions (cont’d.)

• **del statement**: removes an element from a specific index in a list
  ▫ General format: `del list[index]`

• **min and max functions**: built-in functions that returns the item that has the lowest or highest value in a sequence.
  ▫ The sequence is passed as an argument.
Copying Lists

- To make a copy of a list you must copy each element of the list
  - Two methods to do this:
    1. Creating a new empty list and using a `for` loop to add a copy of each element from the original list to the new list.
    2. Creating a new empty list and **concatenating** the old list to the new empty list.
Processing Lists

• List elements can be used in calculations.

• To calculate total of numeric values in a list use loop with accumulator variable.

• To average numeric values in a list:
  ▫ Calculate total of the values
  ▫ Divide total of the values by \texttt{len(list)}

• List can be passed as an argument to a function.
Two-Dimensional Lists

- Two-dimensional list: a list that contains other lists as its elements
  - Also known as nested list.
  - Common to think of two-dimensional lists as having rows and columns (table).
  - Useful for working with multiple sets of data.
- To process data in a two-dimensional list use two indexes.
- Typically use nested loops to process.

```python
list = [['Joe', '301', 'M'], ['Kim', '302', 'F'], ['Li', '303', 'M'], ['Mi', '304', 'F']]
for i in range(4):
    for j in range(3)
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>301</td>
<td>M</td>
</tr>
<tr>
<td>Kim</td>
<td>302</td>
<td>F</td>
</tr>
<tr>
<td>Li</td>
<td>303</td>
<td>M</td>
</tr>
<tr>
<td>Mi</td>
<td>304</td>
<td>F</td>
</tr>
</tbody>
</table>
Tuples

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

```python
country_tuple = ("Canada", "America", "Mexico")
print(country_tuple)  # ('Canada', 'America', 'Mexico')
```

- Tuples support operations as lists.
  - Subscript indexing for retrieving elements: `country_tuple[0]` → 'Canada'
  - Methods such as `index`: `country_tuple.index('America')` → 1
  - Built in functions such as `len`, `min`, `max`; slicing expressions, and `in`, `+`, and `*` operators.

```python
for c in country_tuple:
    print(c)
for i in range(len(country_tuple)):
    print(country_tuple[i])
```

Tuples (cont’d.)

• **Note**: If you want to create a tuple with **just one element**, you must write a trailing comma after the element’s value: `my_tuple = (1, )`

• Tuples do not support the methods:
  - `append`
  - `remove`
  - `insert`
  - `reverse`
  - `sort`
Tuples (cont’d.)

• Advantages for using tuples over lists:
  ▫ Processing tuples is faster than processing lists.
    • So a preferred choice when processing large data.
  ▫ Tuples are safe. (Cannot be changed accidently or otherwise.)

  • `list()` function: converts tuple to list: \( \text{tuple((1, 2, 3))} \rightarrow [1, 2, 3] \)
  • `tuple()` function: converts list to tuple: \( \text{tuple([1, 2, 3])} \rightarrow (1, 2, 3) \)
Canvas Post – Due on Monday by 11:59pm

- Stores the following table data as a **list of lists** and post it on the Canvas. The list name is **contacts**. (Points: 0.25 - for a correct list.)

<table>
<thead>
<tr>
<th>Store</th>
<th>Contact</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfreds Futterkiste</td>
<td>Maria Anders</td>
<td>Germany</td>
</tr>
<tr>
<td>Centro comercial Moctezuma</td>
<td>Francisco Chang</td>
<td>Mexico</td>
</tr>
<tr>
<td>Ernst Handel</td>
<td>Roland Mendel</td>
<td>Austria</td>
</tr>
<tr>
<td>Island Trading</td>
<td>Helen Bennett</td>
<td>UK</td>
</tr>
<tr>
<td>Laughing Bacchus Winecellars</td>
<td>Yoshi Tannamuri</td>
<td>Canada</td>
</tr>
<tr>
<td>Magazzini Alimentari Riuniti</td>
<td>Giovanni Rovelli</td>
<td>Italy</td>
</tr>
</tbody>
</table>
Questions?