Python

Lecture

Lists & Tuples

Sequences

- 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 between these is that a list is mutable and a tuple is immutable

Introduction to Lists

- List: an object that contains multiple data items
  - Element: An item in a list
  - Format: `list = [item1, item2, etc.]`
  - Can hold items of different types
- `print` function can be used to display an entire list
- `list()` function can convert certain types of objects to lists

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

Lists Are Mutable

- 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

- 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`
    - If `start` not specified, `0` is used for start index
    - If `end` not specified, `len(list)` is used for end index
  - Slicing expressions can include a step value and negative indexes relative to end of list

Finding Items in Lists with the **`in`** Operator

- You can use the `in` operator to determine whether an item is contained in a list
  - General format: `*item* in *list*`
  - Returns `True` if the item is in the list, or `False` if it is not in the list
- Similarly you can use the `not in` operator to determine whether an item is not in a list

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 (cont’d.)

- `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 (cont'd.)

- `del` **statement**: removes an element from a specific index in a list
  - General format: `del` *`list`*`[`*`i`*`]`
- `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:
    - 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
    - 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 `len(list)`
- List can be passed as an argument to a function

Processing Lists (cont'd.)

- A function can return a reference to a list
- To save the contents of a list to a file:
  - Use the file object's `writelines` method
    - Does not automatically write `\n` at then end of each item
  - Use a `for` loop to write each element and `\n`
- To read data from a file use the file object's `readlines` method

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
  - Useful for working with multiple sets of data
- To process data in a two-dimensional list need to use two indexes
- Typically use nested loops to process

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
    - Subscript indexing for retrieving elements
    - Methods such as `index`
    - Built in functions such as `len, min, max`
    - Slicing expressions
    - The `in`, `+`, and `*` operators

Tuples (cont’d.)

- 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
  - Tuples are safe
  - Some operations in Python require use of tuples
- `list()` function: converts tuple to list
- `tuple()` function: converts list to tuple