Beyond the Textbook (Zelle 3e - Chapter 11)

Data Collections

The Python `list` Is Versatile

- Lists have a reliable order.
- That order can be changed:
  - By sorting.
  - By reversing.
  - By manual re-arrangement.
- Items may be added and removed.
- Items may be replaced with other items.
- Items may be of same type or of different types.

Some list Class Methods

Method	Description
list.append()	Add an item to the end of the list.
list.insert()	Insert an item at a given position.
list.pop()	Remove an item at a given position, and return it.
list.remove()	Remove an item with a particular value from the list.
list.clear()	Remove all items from the list.

More list Class Methods

Method	Description
list.index()	Returns the index of the first item that matches a particular value.
list.count()	Returns the number of times a particular value occurs in the list.
list.sort()	Sorts the items in the list in place.
list.reverse()	Reverses the order of the items in the list in place.
list.copy()	Returns a shallow copy of the list.

Changing `list` Order

- Sort in-place using the `sort()` **method**.
- Create a new sorted list using the `sorted()` **function**.
- Reverse order in-place using `reverse()` **method**.
- Create a reversed list **iterator** using the `reversed()` **function**.
- Create a new reversed list using `list(reversed())`.
- Re-ordering lists in-place avoids the overhead of new list creation.
- Creating a new re-ordered list preserves the state of the original object.

Sorting Details

- Sort key provided by function passed as `key=` keyword parameter.
- Order can be changed to **descending** with `reverse=True` keyword parameter.

Using a Custom Data Holder Class For Sorting

- Sort fields of `list` objects are specified using **index numbers**.
- Sort fields of `tuple` objects are specified using **index numbers**.
- Sort fields of objects created with **custom Python classes** can be specified using **field names**.
- Using these field names can lead to fewer coding errors greater code readability.
- See:
    - *_05_using_custom_data_holder_class_for_sorting.py*

The Python `set`

- The Python `list` holds a collection of **values**.
- The Python `dictionary` holds a collection of **key-value pairs**.
- The Python `set` holds a collections of **keys**.
- Like the `dictionary`, the `set` does not have a reliable order.
- Sets have features that support reasoning about set membership that are powerful and **beyond the scope of this course**.
- In this course, we cover two key features of sets:
    - Searching for keys in sets is substantially faster than searching for values in lists.
    - Sets do not allow duplicate keys.

Searching For a Key in a `set`

- Python sets are implemented with **hash tables**.
- So, searching for a key in a `set` has performance similar to searching for a key in a `dictionary` -- very fast.
- For any appreciable number of items, searching a `set` is substantially faster than searching a `list`.
- See:
  - *_30_searching_for_a_key_in_a_set.py*

Use a Python `set` to Ignore or Remove Duplicates

- By definition, sets do not have duplicate members.
- Items are added to a `set` using the `add()` method.
- Duplicate keys that are added are simply ignored.
- You can remove duplicate items from a `list` by converting it to a `set` and then back to a `list`. This does not preserve the order of the original list.

Extra Python Features (Syntactic Sugar)

See https://en.wikipedia.org/wiki/Syntactic_sugar

The Python `lambda`

- Any `lambda` can be re-coded as a regular function and the name of that function can be used in place of the `lambda`.
- **Syntax**: `lambda parameter1, parameter2...: expression`
- **Examples**:
  - `lambda student: student.calculate_gpa()`
  - `lambda state: "In-State" if state == "IL" else "Out-of-State"`
  - `lambda salary: salary <= 50000`
- See discussion of `lambda` in How to Use Python Lambda Functions

Some Python `lambda` Use Cases

- Shorter syntax for providing functions to tools that use the **inversion of control** pattern (**Hollywood Principle**).
- Specifying sort keys:
  - Sort key function for `sort()` and `sorted()`.
- Shorter syntax for providing functions to tools that **filter** or **transform** when using data science tools:
  - `apply()` in **pandas**.
  - `filter()` in **PySpark**.
- See:
  - *_40_expressing_sort_keys_using_lambdas.py*

The Python List Comprehension

- A Python **list comprehension** is a shorthand tool for creating a `list`.
- A **list comprehension** is a **list maker**.
- **Syntax**: newlist = [*expression* for *item* in *iterable* if *condition* == True]
- See discussion of list comprehension in Python - List Comprehension
- Using a list comprehension is an alternative to creating and empty list and then populating it with a `For-In` loop.
- You can think of the comprehension as a shorthand syntax for that longer approach.
- See:
  - *_50_creating_lists_with_and_without_list_comprehensions.py*

Last Revised 2022-10-02