Searching and Sorting (1)

DEMONSTRATE SONG.PY, SONG_APP.PY, ALBUM.PY, AND ALBUM_APP.PY

```
from date import *

class Song:
    '''
    A Song class that stores title, singer, and date of the song
    '''
    def __init__(self, title, singer, date):
        self.title  = title
        self.singer = singer
        self.date   = date

    def __str__(self):
        return 'Title: ' + self.title + '\n' \
               + 'Singer: ' + self.singer + '\n' \
               + 'Date: ' + str(self.date)
```

```
from song import *

# version 2

class Album:
    '''
    An Album class that stores a collection of songs along with other
    use methods.
    '''
    def __init__(self, sList):
        self.songs = []
        for s in sList:
            self.songs += [s]

    def __str__(self):
        retStr = '++++++++\n'
        for i in range(len(self.songs)):
            retStr += str(i+1) + ':' + str(self.songs[i])
            retStr += '\n++++++++\n'
        return retStr

    def __len__(self):
        return len(self.songs)

    def __add__(self, newSong):
        self.songs += [newSong]
        return self
```

```
'''
Top 10 all time movie songs from
http://www.billboard.com/articles/list/5922814/top-50-movie-songs-of-all-time
(title, singer, peak date)
1. "You Light Up My Life", Debby Boone, 10/15/1977
2. "Endless Love", Diana Ross and Lionel Riche, 8/15/1981
3. "(Everything I Do) I Do It For You", Bryan Adams, 7/27/1991
4. "The Theme From 'A Summer Place'", Percy Faith and his orchstra, 2/27/1960
5. "How Deep Is Your Love", Bee Gees, 12/24/1977
6. "Eye Of The Tiger", Survivor, 7/24/1982
7. "Flashdance ... What A Feeling", Irene Cara, 5/28/1983
8. "Night Fever", Bee Gees, 3/18/1978
9. "I Will Always Love You", Whitney Houston, 11/28/1992
10. "End Of The Road", Boyz II Men, 8/15/1992
'''
from song import *
from album import *


def createAlbum():
    s1 = Song("You Light Up My Life", "Debby Boone", Date(10,15,1977))
    s2 = Song("Endless Love", "Diana Ross and Lionel Riche", Date(8,15,1981))
    s3 = Song("(Everything I Do) I Do It For You", "Bryan Adams", Date(7,27,1991
    s = []
    s += [s1]
    s += [s2]
    s += [s3]
    album = Album(s)
    print('The top three all time movie songs\n----------')
    print(album)

    return album
```

How to test functions?

- We just wrote the function createAlbum()
- We can test the function in one of two ways
  - Run the Python module by clicking “Run|Module” from within the IDLE, then execute the command

    >>>createAlbum()

    at the Python shell prompt
  - Alternatively, we can write the function call directly in the file, “album = createAlbum()”
  - If we have multiple functions to test, we can write a main() function that contains these function calls.

The main() function for album app

```
def main():

    album = createAlbum()
    print(album)
    #testAdd(album)
    #album = testSorting(album)
    #testSearch(album)

main()
```

Review some of the OOP features

- The len() function
  - len(album) comes from the definition of the __len__() function in the Album class
- The += operator comes from the __iadd__() function in the album class
- The operation of album = album + song comes from the __add__() function in the album class
- The 'in' operator comes from the __contains__() function in the album class

Function code

```
def __len__(self):
    return len(self.songs)

def __add__(self, newSong):
    self.songs += [newSong]
    return self

def __iadd__(self, newSong):
    self.songs += [newSong]
    return self

def __contains__(self, aSong):
    return aSong in self.songs
```

With above definitions, we can try the testAdd() function in album_app.py

Search

In our Album class, we developed a search by title method. The search starts at the very beginning and goes through the entire list, until we either find the song, or reach the end of the list without finding a match. Called sequential search.

```
def searchByTitle(self, titleToSearch ):
    found = False
    i = 0
    while found == False and i < len(self):
        if self.songs[i].title.lower() == titleToSearch.lower():
            found = True
        else:
            i += 1

    if found == True:
        return self.songs[i]
    else:
        return None
```

What if the titles are alphabetically ordered …?

"(Everything I Do) I Do It For You", Bryan Adams, 7/27/1991
"End Of The Road", Boyz II Men, 8/15/1992
"Endless Love", Diana Ross and Lionel Riche, 8/15/1981
"Eye Of The Tiger", Survivor, 7/24/1982
"Flashdance … What A Feeling", Irene Cara, 5/28/1983
"How Deep Is Your Love", Bee Gees, 12/24/1977
"I Will Always Love You", Whitney Houston, 11/28/1992
"Night Fever", Bee Gees, 3/18/1978
"The Theme From 'A Summer Place'", Percy Faith and his orchstra, 2/27/1960
"You Light Up My Life", Debby Boone, 10/15/1977

If we found the current title is 'I Will Always Love You' and we are looking for 'How Deep Is Your Love,' do we need to continue? (Assume the song in red is not in the list.)

The answer is NO, we know the titles are SORTED, if we saw 'I …' but not 'How …', we'd know the song 'How …' is not in the list!

Revised search

Let's revise our search method based on the above observation …

```
def searchByTitle(self, titleToSearch ):  # assume titles are sorted
    found = False
    i = 0
    while found == False and i < len(self):
        if self.songs[i].title.lower() == titleToSearch.lower():
            found = True
        elif self.songs[i].title.lower() > titleToSearch.lower(): # added check
            break
        else:
            i += 1

    if found == True:
        return self.songs[i]
    else:
        return None
```

What's the difference?

- In the first version (not sorted) of the search, roughly how long do we have to search to either find the one we look for, or conclude that the song is not in the list?
- n steps where n is the number of songs in the list
- In the second version (sorted), we'd know much earlier (on the average) that the song is not in the list!

Wait … we can even do better!

Do we really need to search one-by-one from the beginning?

The answer is NO. Binary search is much more faster.

```
def binarySearch(self, titleToSearch ):  # assume titles are sorted
    found = False
    left = 0
    right = len(self)
    mid = (left + right) // 2
    while found == False and left <= right:
      if self.songs[mid].title.lower() == titleToSearch.lower():
        found = True
        break    # leave the loop
      elif self.songs[mid].title.lower() > titleToSearch.lower():  # search the left half
        right = mid - 1
      else:    # search the right half
        left =  mid + 1
      mid = (left + right) // 2

    if found == True:
      return self.songs[mid]
    else:
      return None
```