**Contact Book (Nuode, 3F)**

User Interface

* Simple-to-use command line interface with that displays colors on the mac terminal

Storage of data

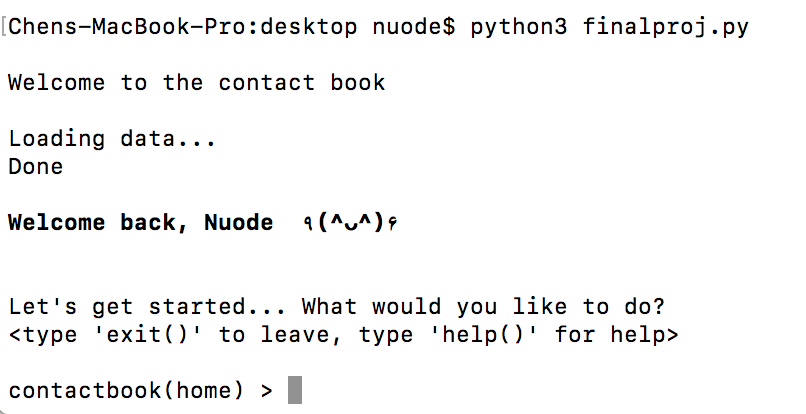
* I collected 7 attributes from the user: First name, last name, age, phone number, birthday, birthday in MMDD format and horoscope. I stored these data into my custom array [].
* I then stored each user’s array in a “master array” so it becomes a nested array.
* I created 7 AVL trees, 1 for each attribute
* I then got the index in the master array where the user array is stored
* In each of my AVL tree, I added in a new node where node’s key is user’s value for that attribute. Besides the node’s key, the node also stores a list with the index at which the user array is in the master array. Future insertions with the same node’s key will just append the user data’s index into that list.

Functions supported:

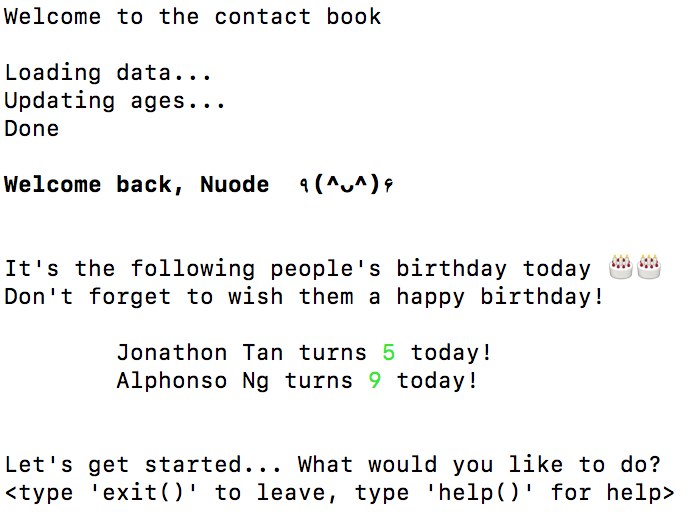
* Add (O log n)
  + Sample syntax: add()
  + I used AVL trees and an array to add. So the time complexity is O (log n) for insertion into each of my attribute trees and O (1) for appending the user’s data into the master array. I used the AVL trees as they allow me to save time due to their self-balancing and log n lookup time. For the master array, I allowed it to be unsorted and only used the append function into order to save time. In order to save space, the AVL trees only store the index at which the user’s data is in the master array so the main data is concentrated into 1 master array instead of having duplicates for each tree.
* Edit (O log n)
  + Sample syntax:
    - edit() 🡨 normal edit
    - edit(first name=Nuode) 🡨 for quick edit
  + I first searched for the user using AVL tree (O log n) and got the reference to the data in the master array, I then edited the data in the master array. And updated the attribute trees whose attribute the user changed. By using a master array to concentrate all user-data and only storing the index in master array of the data in each tree, I saved time as I only needed to edit the nodes in trees whose attributes are changed (by deleting the node with old key (O log n) and inserting (O log n) a new node with the new value as its key for the attribute) instead of editing all trees whenever the data array of the user is changed
* Delete (O log n)
  + Sample syntax:
    - delete() 🡨 normal delete
    - delete(first name=Nuode) 🡨 for quick delete
  + I deleted the node with the user’s attribute as key in each of my 7 AVL trees (O log n). In my master array, instead of deleting it normally (O N worse case) I used a stack to collect all the deleted indexes, when indexes are deleted, that slot in master array will be set to None and the index will be pushed into the stack, in following append functions of the master array, the available indexes in the stack will be used first. This creates a O (1) deletion time for master array and thus saves time.
* Search (Criteria separated by AND) O (log n) for each search criteria
  + Sample syntax:
    - search() 🡨 normal search
    - search(first name=Nuode) for quick search for 1 criteria
    - search(first name=Nuode,last name=Chen) for 2 criteria or more
  + I used AVL trees to search as they are quick and are O (log n). I am also able to search with any of the attributes as I have a tree for it.
  + For AND search criteria, I first searched with the first criteria and got a list of possible entries, I then used a for loop to check the possible entries with the other search criteria and narrow down the list. Worse case (O m \* n) where n is the number of possible entries and m is the number of criteria
* Search (Criteria separated by OR) O (log n) for each search criteria
  + Sample syntax:
    - orsearch() 🡨 normal search
    - orsearch(first name=Nuode) for quick search for 1 criteria
    - orsearch(first name=Nuode,last name=Chen) for 2 criteria or more
  + O log N to search the AVL trees for users that match the criteria
  + I used a for loop to search matching entries for each criterion, I used a set to collect these entries to ensure that there are no repeated. O(n) for set operations
* Sort (O n)
  + Sample syntax:
    - sort()
    - sort(birthday[MMDD]) 🡨 shortcut
  + I used an in order traversal for the attribute AVL tree, reverse also supported
* Max (O log n)
  + Sample syntax:
    - max()
    - max(age) 🡨 shortcut
  + Found the highest value’s node’s key in the attribute AVL tree
* Min (O log n)
  + Sample syntax:
    - min()
    - min(age) 🡨 shortcut
  + Found the lowest value’s node’s key in the attribute AVL tree
* Average (O n)
  + Sample syntax:
    - average()
    - average(age) 🡨 shortcut
  + From the tree, get an array of the all indexes of the data and did a for loop to average time. Limited usefulness as it only supports age
* Save (O n)
  + Sample syntax: exit()
  + I saved the date, master array, stack for deleted indexes, user’s own data into my file (On)
  + I converted each tree into a tuple representation and saved it (On)
* Load (O n)
  + Sample syntax:
    - load(frienddata.txt)
  + Using a plaintext file with the format [first name, last name, birthday, phone number] and separated by \n, loop through the list and add it.
* init-load (O n)
  + Sample syntax:
    - initload(secretcode)
  + From my database file, take the literal of master array, user’s own data and recreate stack, rebuild the trees from their tuple.
* Date
  + I stored the date as an 8-digit integer with YYYYMMDD and birthday in MMDD format as 3 or 4-digit integer. This allows me to treat them as numbers for my AVL tree and compare them to achieve O (log n) search for date

Additional features:

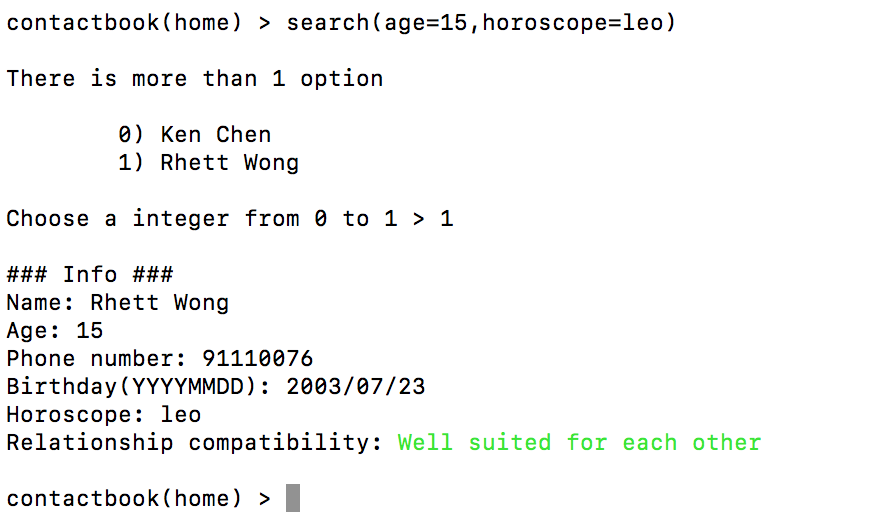
* Auto-load (need to close the program after the first time, then agree to auto load in the future on the second time, and from then on will auto load from then on)



* + Once user agrees to auto load, init load will be called on default from the database file thus contact book will be ready as soon as program starts running.
* Error catching to ensure proper format of data inputted
  + E.g. only accepts numbers
  + Date must be valid month, day, year and must be 8 digits long in YYYYMMDD
  + Phone numbers have no length limits, so 8,3 digits are accepted
* Live age updating (worse case O(n))
  + By saving the date in the database file. I am able to check if there was a gap between when it was last open and today, I am able to update the dates by searching the birthday [MMDD] attribute tree. Update the ages and change the age tree nodes accordingly
* Birthday announcement/reminder



* + Due to age updating feature, I was also able to check whether the person’s birthday was today thus creating a birthday alert message.
* Horoscope and compatibility status



* + Using predetermined chart found on the web
  + And determined horoscope by birthday.
  + Maybe help you find your future partner/best friend?
    - Don’t trust it too much.
* Size
  + Sample syntax:
    - size()
  + Return size of contact book
* Clear
  + Sample syntax:
    - clear()
  + Reset everything back to default