Regular Expressions and Search

ISTA 350 Hw1, Due Thursday 1/30/2020 at 11:59 pm

Introduction. This homework will introduce you to regular expressions and two types of search: linear and binary. Later we will examine the advantages of each. For now, know that linear search always works, but binary search requires a sorted sequence.

Domestic and national security is dependent on the integrity of our currency. Thus, tracking of counterfeit bills and legitimate dollars associated with criminal activity is a criminal justice system priority, involving local, state, and federal agencies. Examples of situations resulting in bills of interest with serial numbers known to law enforcement include recovery of counterfeit bills (more fakes with the same serial number may be in circulation), bills included in ransom payments but not recovered, loss by theft of batches of new bills with consecutive serial numbers in bank robberies or armored car heists, random recording of bills by banks for security purposes later lost in robberies, and bills distributed by law enforcement officers in undercover operations such as drug buys. Bills can come into the possession of the criminal justice system in many ways including recovery at crime scenes, seizure in drug busts, and from so-called 'reverse buys' where undercover agents sell drugs or other contraband. Law enforcement must keep track of bills of interest and bills recovered and be able to compare the two. For this assignment, you will implement a class that does this.

For this assignment, we will concern ourselves only with bills of denomination $5 or more – no $1 or $2 bills, which have one less character in their serial numbers. Such bills come in denominations of 5, 10, 20, 50, and 100. A legitimate bill is uniquely identified by the combination of its denomination and serial number. Serial numbers have 11 characters. The first is a capital letter between 'A' and 'M', and represents the 'series', or year of last major design change (the smallest such change results from the appointment of a new Secretary of the Treasury). The second character is a capital letter between 'A' and 'L', representing the Federal Reserve Bank from which the note was issued. This is followed by 8 digits. However, the number 00000000 is not allowed. The last character is a capital letter representing the 'run'. Whenever all possible serial numbers with a given last letter are used up, the run character is incremented. The letter 'O' is not used as run indicator because of its resemblance to 0, and 'Z' is reserved for special printings. Thus, for every denomination, there are 13 * 12 * 99999999 * 24 = 370 billion possible serial numbers. That's a lot of potential bills.

Instructions. Create a module named `hw1.py`. Below is the spec for six methods. Implement them and upload your module to the appropriate D2L Assignments folder.

Testing. Download `hw1_test.py` and the auxiliary files and put them in the same folder as your `hw1.py` module. Run it from the command line to see your current correctness score. Each of the methods except `init` is worth 14% of your correctness score. `init` is worth 28%, 14% for your regular expression, 14% for the rest of it. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, `ISTA 350 Hw1`, and a brief summary of the module. Each function must

contain a docstring. Each function docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

Resources.
https://docs.python.org/3/tutorial/classes.html
https://docs.python.org/3.7/howto/regex.html
https://docs.python.org/3.7/library/re.html#module-re
https://www.vipinajayakumar.com/parsing-text-with-python/
https://pythex.org/
http://www.pyregex.com/

Method specifications for class `WatchList`.

`class WatchList:`

`init`: This magic method takes a filename that defaults to the empty string. Initialize an instance variable called `bills` to a dictionary that maps each of the five denominations of interest, represented as strings (i.e. `'5'`, `'10'`, etc.), to an empty list. If a filename was passed in, each line of the file will represent a bill that we want to add to our watch list dictionary and will be in the format `'<serial_number> <denomination>\n'`. Look at one of the bill files in a text editor to see specific examples. Append the serial number for each bill in the file to the appropriate list in the dictionary. A Boolean instance variable called `is_sorted` indicates whether or not the lists in the dictionary are sorted. Assume that the bill files are not sorted. Finally, an instance variable called `validator` holds a compiled regular expression that will be used to check for valid serial numbers (see the **Introduction** above for the rules governing serial numbers).

`insert`: This instance method takes a string representing a bill in the format `'<serial_number> <denomination>'`. If the `bills` dictionary is sorted and the bill is not already in the dictionary, insert the new serial number into the appropriate list maintaining that list in sorted order. If the lists in the dictionary are not sorted and the bill is a new serial number, append it to the appropriate list. You may not use the `sort` method, the `sorted` function, or the `sort_bills` method described below. **Later, we will discuss in-depth why sorting in this method is a bad idea, but what do you think the explanation might be?**

`sort_bills`: This instance method sorts the lists in the dictionary.

`linear_search`: This instance method takes a string representing a bill in the format `'<serial_number> <denomination>'`. Return `True` if it represents a bill in the dictionary, `False` otherwise. Hint: what is Python's built in search operator? You may use it.

`binary_search`: This instance method takes a string representing a bill in the format `'<serial_number> <denomination>'`. Going through a sequence element-by-element searching for an item (the target of a search is called the key, in this case a serial number) is called linear or sequential search. If the sequence is sorted, there is a much faster technique called binary search. This method assumes that the lists in the dictionary are sorted. If the key is in the dictionary, return `True`, otherwise `False`. Here is some pseudocode for this method:

```
initialize variables containing your low and high indices
while low is <= high:
      check the position midway between them
      if it's the key, return True
      if the item in the array is larger than the key:
            change high appropriately
      else:
            change low appropriately
return False
```

`check_bills`: This instance method takes a filename and a Boolean that defaults to `False`. The file contains a list of bills in the format `'<serial_number> <denomination>\n'` that we wish to check against our watch list. The method returns a list of bills in the format `'<serial_number> <denomination>'`. If the second argument is `True`, then you have to use binary search. Go through the file line-by-line. If a bill is on the watch list, append it to the list of bad bills that you are building. It is also a bad bill if it has an invalid serial number. Return the list of bad bills.

In this class we will be concerned with the efficiency of our code. It is faster to decide only once which search to use, instead of repeating that decision in a loop. Functions and methods are objects in Python (everything in Python is an object), so we can assign them to variables. The following line of code will allow you to choose your search and save that choice:

```
search = self.binary_search if self.is_sorted else self.linear_search
```

Now you can call `search` just like any other function.