Assignment 10

Dictionary

LILLEBAELT ACADEMY OF
PROFESSIONAL HIGHER EDUCATION

Author
Martin Grønholdt
mart80c7@edu.eal.dk

Sunday 29 January 2017

Table of Contents

Introduction

The program in this hand-in is is an example of using dictionaries in Python.

All files for this hand in are available at:
https://github.com/deadbok/eal_programming/tree/master/ass10

Error handling

All programs that requests user input, handle bad input by asking the user, to use only the correct data type. All programs using file I/O will show an error message if something goes wrong during file access.

Generally all error conditions lead to an error message where after the program exits or continues depending on the severity of the error.

8. Name and Email Addresses

This programs acts as a database for E-mail addresses. I has a simple menu, using text mode, listing all possible operations. The user then enters a corresponding number to select an operation. The program uses a few OOP principles the control and UI layer is kept in the App class, while data manipulation is kept in the Data class.

prog8.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 8 "Name and Email Addresses"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-01-29)

Program to organise name and email addresses.
"""
from app import App


def main():
    """
    Program main entry point.
    """
    # Instantiate the application class.
    app = App('test.db')
    # Start the application main loop.
    app.run()


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

app.py

```
# -*- coding: utf-8 -*-
"""
Name: Program 8 "Name and Email Addresses"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-01-29)

Program control and ui code.
"""
from data import Data


class App:
    """
    This class takes care of the overall control and ui of the application.
    """
    # The key assignments for the menu.
    QUIT = 0
```

```
    ADD = 1
    EDIT = 2
    DEL = 3
    SEARCH = 4
    LIST = 5
    SELECT = 6

    def __init__(self, db_file_name):
        """
        Create the application class.

        :param db_file_name: The name of the file to pickle the data to and
                             from.
        """
        # File name to save the data to.
        self.db_file_name = db_file_name
        # The object that holds the actual data.
        self.data = Data(db_file_name)
        # List of menu items, their numbers, and methods.
        self.menu_items = [
            (self.QUIT, ' {}: {}'.format(self.QUIT, 'Quit'),
             None),
            (self.ADD, ' {}: {}'.format(self.ADD, 'Add an entry'),
             self.add),
            (self.EDIT, ' {}: {}'.format(self.EDIT, 'Edit an entry'),
             self.edit),
            (self.DEL, ' {}: {}'.format(self.DEL, 'Delete an entry'),
             self.delete),
            (self.SEARCH, ' {}: {}'.format(self.SEARCH, 'Search for an entry'),
             self.search),
            (self.LIST, ' {}: {}'.format(self.LIST, 'List all entries'),
             self.printAll),
            (self.SELECT, ' {}: {}'.format(self.SELECT, 'Select an entry'),
             self.select)]
        # Currently no entry is selected.
        self.selected = None
        self.op = -1

    def menu(self):
        """
        Print the main menu.
        """
        # Print header and selection name.
        print('\nMain menu', end='')
        if self.selected is None:
            print(' (no entry selected):')
        else:
            print(' ({} selected): '.format(self.selected))

        # Print menu entries.
        for entry in self.menu_items:
            print(entry[1])

        # Print the data of the selected entry.
        if self.selected is not None:
            print('\nSelected: ')
            self.printEntry(self.selected, False)
```

```
    def selectOp(self):
        """
        Present a menu and validate the user selection.
        """
        # Show the menu.
        self.menu()
        # Get the user choice.
        choice = input('\nSelect operation: ')
        # Blank line.
        print('')

        # Validate the user input.
        try:
            self.op = int(choice)
            # Check if the choice is within range.
            if (self.op < 0) or (self.op > len(self.menu_items)):
                raise ValueError
        except ValueError:
            print('Wrong input, please try again')
            # Call recursively, program dies if you do enough wrong choices,
            # because of this.
            self.selectOp()

    def add(self):
        """
        Add entry.
        """
        # Get the name of the new entry.
        name = input('Input the name of the new entry: ')
        # Check if the key exists
        if self.data.isEntry(name):
            print('There is an existing entry under that name' +
                  ', you can only edit it.')
        else:
            # The key is OK, get the e-mail address.
            email = input('Input the e-mail address of the new entry: ')
            # Add the data.
            self.data.add(name, email)
            print('{} added'.format(name))
        # Be nice and select the entry.
        self.selected = name

    def edit(self):
        """
        Edit an entry.
        """
        # Check if a valid entry is selected.
        if self.selected is None:
            print('An entry must be selected')
        elif self.selected not in self.data.entries().keys():
            print('ERROR: Name "{}" not found'.format(self.selected))
            self.selected = None
        else:
            # Get new name.
            new_name = input(
                'Input the new name of the entry [{}]: '.format(self.selected))
            # Chose default on empty.
            if len(new_name) < 1:
                new_name = self.selected
```

```
            # Get new e-mail.
            email = input(
                'Input the new e-mail address of the entry [{}]: '.format(
                    self.data.entries()[self.selected]))
            # Chose default on empty.
            if len(email) < 1:
                email = self.data.entries()[self.selected]

            # Apply the changed data.
            self.data.edit(self.selected, new_name, email)
            # Select the item.
            self.selected = new_name
            print('{} changed'.format(self.selected))

    def delete(self):
        """
        Delete an entry.
        """
        # Check if a valid entry is selected.
        if self.selected is None:
            print('An entry must be selected')
        elif self.selected not in self.data.entries().keys():
            print('ERROR: Name "{}" not found'.format(self.selected))
            self.selected = None
        else:
            # Yep. Delete it.
            self.data.delete(self.selected)
            print('{} deleted'.format(self.selected))
            # Un-select.
            self.selected = None

    def search(self):
        """
        Search for an entry by a string.
        :return:
        """
        # Check if there are any entries.
        if self.data.n_entries():
            # Print some help.
            print('Supported search patterns:\n' +
                  ' *\tmatches everything\n' +
                  ' ?\t matches any single character\n' +
                  ' [seq]\tmatches any character in seq\n' +
                  '[!seq]\tmatches any character not in seq\n')
            # Get the search string from the user.
            search_str = input('Input search string: ')
            # Search the data.
            self.data.search(search_str)

            # Print the number of entries found.
            if self.data.n_search_results() == 1:
                print('Found {} entry'.format(self.data.n_search_results()))
            else:
                print('Found {} entries'.format(self.data.n_search_results()))
            # Select item if it there is only one.
            if self.data.n_search_results() == 1:
                self.selected = self.data.search_results()[0]
            else:
                print('Use a more specific search string.')
        else:
            print('No entries')
```

```
def select(self):
    """
    Select an entry.
    """
    # Are there any entries?
    if self.data.n_entries():
        # De-select currently selected.
        self.selected = None
        # Do a search.
        self.search()
        # If none is selected print the result of the search.
        if self.selected is None:
            self.printSearch()
    else:
        print('No entries')

def printEntry(self, name, selection=True):
    """
    Print a named entry.

    :param name: The name of the entry.
    :param selection: Highlight selection.
    """
    # Check the key.
    if name not in self.data.entries().keys():
        print('ERROR: Name "{}" not found'.format(name))
    else:
        # Print and highlight selection.
        if selection and self.selected == name:
            print('* Name: {:20}'.format(name) + ' - ' +
                  'E-mail: {:>20} *'.format(self.data.email(name)))
        else:
            print('  Name: {:20}'.format(name) + ' - ' +
                  'E-mail: {:>20}'.format(self.data.email(name)))

def printAll(self):
    """
    Print all entries.
    """
    if len(self.data.entries()) != 0:
        print('\nAll entries:')
        # Print all entries.
        for name in self.data.entries().keys():
            self.printEntry(name)
    else:
        # Tell that there are no entries.
        print('No entries')
```

```
    def printSearch(self):
        """
        Print all entries from the current search result.
        """
        # Result counter.
        i = 1

        # Print results.
        if self.data.n_search_results() == 0:
            print('No search results')
        else:
            for name in self.data.search_results():
                print(' {:d}; '.format(i), end='')
                self.printEntry(name)
                i += 1

    def save(self):
        """
        Save the data.
        """
        print('Saving data to ' + self.db_file_name)
        self.data.save()

    def load(self):
        """
        Load the data.
        """
        if self.data.load():
            print('Data loaded from ' + self.db_file_name)

    def run(self):
        """
        Load the data and enter the main loop. Save the data on exit.

        """
        # Load data.
        self.load()
        # No selection.
        self.data.selected = None

        # Let the user select an operation.
        self.selectOp()

        # Run until quit is selected.
        while self.op != 0:
            # Run the selected operation.
            self.menu_items[self.op][2]()
            # Let the user select an operation.
            self.selectOp()

        # Save the data.
        self.save()

        print('Bye.')
```

data.py

```
# -*- coding: utf-8 -*-
"""
Name: Program 8 "Name and Email Addresses"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-01-29)

Data handling code.
"""
import fnmatch
import pickle
import os.path


class Data:
    """
    This class does all data handling. It stores names and associated e-mail
    addresses.
    """

    def __init__(self, file_name):
        """
        Construct a data object for
        :param file_name: The name and path of the file used to store the data.
        """
        # The dictionary holding the data.
        self.data = dict()
        # Data file name
        self.file_name = file_name
        # List of last search results.
        self.data_search_results = list()

    def save(self):
        """
        Save the data using pickle. Complain and exit on error.
        """
        try:
            # Open the file for binary write access.
            with open(self.file_name, 'wb') as data_file:
                # Pickle the data to the file.
                pickle.dump(self.data, data_file)
        except IOError as ex:
            # Complain when something goes wrong with the file access.
            print('Exception: {}'.format(str(ex)))
            print('Error saving data.')
            exit(1)
```

```
    def load(self):
        """
        Load data from a file using pickle. Complain and exit on error.
        :return: True on success, False otherwise,
        """
        try:
            # Check if there is a file.
            if os.path.isfile(self.file_name):
                # Open the file.
                with open(self.file_name, 'rb') as data_file:
                    # Un-pickle the data
                    self.data = pickle.load(data_file)
                return True

            # Tell that load did not succeed, but the error was not fatal.
            return False
        except IOError as ex:
            # Complain when something goes wrong with the file access.
            print('Exception: {}'.format(str(ex)))
            print('Error saving data.')
            exit(1)

    def isEntry(self, name, error_not_found=False):
        """
        Check if name is in the data.

        :param name: The exact name to look for.
        :param error_not_found: Show an error if name is not found, is set to
                                True.
        :return: True if found, False otherwise.
        """
        # Default to not found.
        ret = False
        # Look for the key.
        if name in self.data.keys():
            ret = True
        else:
            # Print error if not found, and asked to.
            if error_not_found is True:
                print('ERROR: Name "{}" not found'.format(name))
        # Return result.
        return ret

    def n_entries(self):
        """
        Return the number of entries in the data.

        :return: Number of entries in the data.
        """
        return len(self.data)

    def entries(self):
        """
        Return a dictionary with all entries in the data.
        :return:
        """
        return self.data
```

```
    def add(self, name, email):
        """
        Add an entry to the data.

        :param name: Name to add.
        :param email:  E-mail to add.
        """
        # Check if the key is correct.
        if name is None:
            print('ERROR: Name can not be empty')
        elif len(name) < 1:
            print('ERROR: Name can not be empty')
        else:
            # Add/overwrite if it is.
            self.data[name] = email

    def edit(self, old_name, new_name, email):
        """
        Change an entry.

        :param old_name: The original name of the entry.
        :param new_name: the new name of the entry, None to keep the old.
        :param email: Email address to set for the entry.
        """
        # Keep the old name if no new is given
        if new_name is None:
            new_name = old_name

        # Add/Change the entry.
        self.add(new_name, email)

        # Delete the old entry if this is a new key.
        if old_name != new_name:
            self.delete(old_name)

    def delete(self, name):
        """
        Delete an entry.

        :param name: Name of the entry to delete.
        """
        # Check if the key is correct.
        if name is None:
            print('ERROR: Name can not be empty')
        elif len(name) < 1:
            print('ERROR: Name can not be empty')
        else:
            # Delete the entry if okay.
            del self.data[name]
```

```
def email(self, name):
        """
        Return the e-mail address of an entry.

        :param name: Name of the entry.
        :return: E-mail address or empty string if not found.
        """
        # Default to empty string.
        email = ''

        # Do some validation.
        if name is None:
            print('ERROR: Name can not be empty')
        elif len(name) < 1:
            print('ERROR: Name can not be empty')
        elif len(self.data) < 1:
            print('ERROR: No entries')
        else:
            # The key checks out.
            email = self.data[name]
        # Return the e-mail address if found.
        return email

    def search(self, search_str):
        """
        Find a string in the data entries using fnmatch.

        :param search_str: String to search for including wildcards.
        :return: List of names that matches the search string.
        """
        # Use a set to keep the results so they only appear once.
        names = set()

        # Run through all entries.
        for name, email in self.data.items():
            # Search in both names and email addresses using fnmatch.
            if fnmatch.fnmatch(name, search_str):
                names.add(name)
            if fnmatch.fnmatch(email, search_str):
                names.add(name)

        # Convert the set to a list and return.
        self.data_search_results = list(names)

    def n_search_results(self):
        """
        Return the number of results of the last search.

        :return: Number of result of the last search.
        """
        return len(self.data_search_results)

    def search_results(self):
        """
        Return a list of names matching the last search.

        :return: List of names matching the last search.
        """
        return self.data_search_results
```

Result

The program uses starts with a menu that uses numbers to select the desired operation.

```
Data loaded from test.db

Main menu (no entry selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Select operation:
```

Initial menu screen.

```
Select operation: 1

Input the name of the new entry: Homer Simpson
Input the e-mail address of the new entry: chunkylover53@aol.com
Homer Simpson added

Main menu (Homer Simpson selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Selected:
 Name: Homer Simpson        - E-mail:  chunkylover53@aol.com

Select operation:
```

Adding a new entry.

“Edit an entry” and “Delete an entry” requires an active selection, selected using the search or select operations.

```
Select operation: 2

An entry must be selected

Main menu (no entry selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Select operation:
```

Trying to edit, with no entry selected.

```
Select operation: 6

Supported search patterns:
 * matches everything
 ? matches any single character
 [seq] matches any character in seq
 [!seq] matches any character not in seq

Input search string: Martin*
Found 1 entry

Main menu (Martin Bo selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Selected:
 Name: Martin Bo            - E-mail:        test@test.net

Select operation: 2

Input the new name of the entry [Martin Bo]: Jens Hansen
Input the new e-mail address of the entry [test@test.net]:
Jens Hansen changed

Main menu (Jens Hansen selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Selected:
 Name: Jens Hansen          - E-mail:        test@test.net

Select operation:
```

Selecting and entry before editing it.

```
Select operation: 3

Jens Hansen deleted

Main menu (no entry selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Select operation: 5


All entries:
 Name: Homer Simpson        - E-mail:  chunkylover53@aol.com
 Name: Bill Gates           - E-mail:  billg@microsoft.com
 Name: Richard Stallman     - E-mail:          rms@gnu.org

Main menu (no entry selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Select operation:
```

Deleting the selected entry, and listing the remaining ones.

The search function will select the resulting item if the match is unique.

```
Select operation: 4

Supported search patterns:
 * matches everything
 ? matches any single character
 [seq]  matches any character in seq
 [!seq]  matches any character not in seq

Input search string: *bill*
Found 1 entry

Main menu (Bill Gates selected):
 0: Quit
 1: Add an entry
 2: Edit an entry
 3: Delete an entry
 4: Search for an entry
 5: List all entries
 6: Select an entry

Selected:
 Name: Bill Gates           - E-mail:  billg@microsoft.com

Select operation:
```

Search for an entry, item is selected.

When the program is closed, the data is saved to a file.

```
Select operation: 0

Saving data to test.db
Bye.
oblivion@gangster-martin:/mnt/data/Dokumenter/Skole/It-
et/2016/programming/Assignment 10$ python3 prog8.py
Data loaded from test.db

Main menu (no entry selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Select operation: 5


All entries:
 Name: Richard Stallman     - E-mail:          rms@gnu.org
 Name: Homer Simpson        - E-mail: chunkylover53@aol.com
 Name: Bill Gates           - E-mail:  billg@microsoft.com

Main menu (no entry selected):
0: Quit
1: Add an entry
2: Edit an entry
3: Delete an entry
4: Search for an entry
5: List all entries
6: Select an entry

Select operation:
```

Saving the file as the program closes, and loading the data as it opens.

Conclusion

This assignment ended up using a lot of Python features. The interface is reminiscent of the CP/M-86 systems that was used in Danish schools in the 80's as well as the Poly Pascal compiler.