Last modification: January 27, 2021

CS4341: Intro to AI
C-Term 2020/2021
Homework 1

Disclaimer

It might happen that you will find solutions, either partial or complete, on the web, on sites such as StackOverflow. Feel free to use these websites, but

1. Remember that we *know* that these websites exist.
2. We have tools to compare your code and detect when it looks suspiciously identical to someone else's.
3. If you copy-paste code you don't understand, you're missing the point of this homework and the reason why it was given.

Deliverables

The main deliverable of this homework is a zip file containing three files: `ex1.py`, `ex2.py` and `ex3.py`. The files must be stored in a directory called `LastNameFirstNameInitial`. For example, if your name is `Donald J. Duck`, you'll submit a zip file that contains:

```
DuckDonaldJ/
    ex1.py
    ex2.py
    ex3.py
```

The zip file name must be called `CS4341_HW1_LastnameFirstnameInitial.zip`, where `Lastname`, `Firstname Initial` are *your* last name, first name, and initial. For example: `CS4341_HW1_DuckDonaldJ.zip`.

About the code itself:

- The names of the functions in the code skeletons provided below are mandatory — these functions must be implemented and named as indicated. You can write extra functions, if you think it's a good idea.
- The files you decide to submit contain only the requested code and nothing else. For example, remove any `print()` statement you added for your own debugging and remove any testing code you might have.
- Be sure to comment your code thoroughly and add docstrings in your definitions of extra functions and classes.
- All the code must be written with a resonably recent version of Python 3. Any version after 3.6 would be OK.

The reason why we want you follow these guidelines strictly is that, if you do, we'll be able to make the grading process almost completely automatic, and you'll get feedback from us much faster.

Exercise 1: Loops and Arrays (30 points)

Connect Four is a turn-based two-player game in which you win if you can string four of your tokens in a horizontal, vertical, or diagonal sequence. Complete this Python class which, given a board configuration, calculates the outcome of a board configuration. Fill in this template:

```
class ConnectFour:

    def __init__(self, board, w, h):
        """Class constructor"""
        # Board data
        self.board = board
        # Board width
        self.w = w
        # Board height
        self.h = h

    def isLineAt(self, x, y, dx, dy):
        """Return True if a line of identical tokens exists starting at (x,y)
           in direction (dx,dy)"""
        # Your code here

    def isAnyLineAt(self, x, y):
        """Return True if a line of identical symbols exists starting at (x,y)
           in any direction"""
        return (self.isLineAt(x, y, 1, 0) or # Horizontal
                self.isLineAt(x, y, 0, 1) or # Vertical
                self.isLineAt(x, y, 1, 1) or # Diagonal up
                self.isLineAt(x, y, 1, -1)) # Diagonal down

    def getOutcome(self):
        """Returns the winner of the game: 1 for Player 1, 2 for Player 2, and
           0 for no winner"""
        # Your code here, use isAnyLineAt()

    def printOutcome(self):
        """Prints the winner of the game"""
        o = self.getOutcome()
        if o == 0:
            print("No winner")
        else:
            print("Player", o, " won")
```

Example output:

```
W = 7
H = 6

BOARD1 = [
    [0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0]
]
c = ConnectFour(BOARD1, W, H)
c.printOutcome()
```

```
# No winner

BOARD2 = [
    [1,1,1,1,2,1,0],
    [0,2,2,2,0,0,0],
    [0,0,2,0,0,0,0],
    [0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0]
]
c = ConnectFour(BOARD2, W, H)
c.printOutcome()
# Player 1 won
```

Note that the board is stored upside down with respect to what is normally seen in the original game.

Exercise 2: Recursion (40 points)

Write a *recursive* function that checks whether two strings passed as input are one the reverse of the other. Fill in this template.

```
def isreverse(s1, s2):
    # Your code here
```

Example output:

```
print("<empty>, <empty> -> ", isreverse("", ""))         # True
print("a, a -> ",             isreverse("a", "a"))       # True
print("ab, ba -> ",           isreverse("ab", "ba"))     # True
print("abc, cba -> ",         isreverse("abc", "cba"))   # True
print("abcd, cba -> ",        isreverse("abcd", "cba")) # False
```

Exercise 3: Files and Data Structures (30 points)

This exercise asks you to write a program that computes the Jaccard index of two text files. This coefficient is used in Natural Language Processing as a crude estimate of the similarity of two text files.

To calculate the Jaccard index, also known as "intersection over union", you first parse each file and produce a *set* of tokens for each of them (`words1` and `words2`). Note: it is a *set*, not a list, because we do not care about duplicates within a specific file.

Then, you calculate the index with

$$\frac{|\texttt{intersection(words1,words2)}|}{|\texttt{union(words1,words2)}|} \tag{1}$$

where `intersection(words1,words2)` is a set containing the words in common between the two texts, and `union(words1,words2)` is the union of the two sets. Finally, the $|\cdot|$ operator indicates the number of elements in a set (i.e., `len()` in Python).

To test your code, you'll need some text dump. Project Gutenberg (https://www.gutenberg.org/) has thousands of books in various formats, including plain text. For example:

- Alice in Wonderland
- Through the looking glass

Fill in this template:

```
def wordset(fname):
    """Returns the set of words corresponding to the given file"""
    # Your code here

def jaccard(fname1, fname2):
    """Calculate Jaccard index"""
    # Your code here - call wordset()
```

Example output:

```
FNAME1 = "alice_ascii.txt"
FNAME2 = "glass_ascii.txt"
print("Jaccard index between", FNAME1, "and", FNAME2, jaccard(FNAME1, FNAME2))
# Jaccard index between alice_ascii.txt and glass_ascii.txt 0.4730329208498716
```

Implementation Notes

Converting a text file from UTF8 to ASCII

The books in Project Gutenberg are encoded in UTF8, which is more cumbersome to parse than plain ASCII. Since the point of the exercise is not to make parsing difficult, but to review data structures and some Python syntax, focus your text parsing on ASCII text.

You can either take text that is already in ASCII format, or, under Linux and Mac, you can convert UTF8 to ASCII with this command typed in the terminal (the `$` indicates the Bash command prompt, it's not part of the command itself):

```
$ iconv -f UTF-8 -t ASCII -c alice_utf8.txt > alice_ascii.txt
```

The command above assumes that the text you want to convert is stored in `alice_utf8.txt`. The option `-c` ignores any character in the UTF8 file that cannot be converted to ASCII. The conversion won't be perfect, but it's not a big deal.

Constructing the word list of an ASCII file

To construct the word list from an ASCII file, filter out anything that is not an alphabetic character. A fast and simple way to achieve this is to employ a regular expression, as explained in this StackOverflow answer. The functions `str.lower()` and `str.split()` will also be useful.