Exercise Sheet 04 Solutions – File I/O and Algorithms

Sebastian Höffner Aline Vilks

Deadline: Mon, 1 May 2017 08:00 +0200

Exercise 1: Vector & matrix mathematics

File: `code/vector_math.py`

```
import math


def add(x, y):
    """Adds two vectors x and y.

    Args:
        x: The first summand.
        y: The right summand.

    Returns:
        The result is the vector z for which z_i = x_i + y_i holds.
    """
    result = []
    for i, x_i in enumerate(x):
        result.append(x_i + y[i])
    return result


def sub(x, y):
    """Subtracts two vectors x and y.

    Args:
        x: The minuend.
        y: The subtrahend.

    Returns:
        The result is the vector z for which z_i = x_i - y_i holds.
```

```
    """
    result = []
    for i, x_i in enumerate(x):
        result.append(x_i - y[i])
    return result


def dot(x, y):
    """Calculates the scalar product between two vectors x and y.

    The scalar product is the sum of the products of each individual elements.

    Args:
        x: The left multiplicand.
        y: The right multiplicand.

    Returns:
        The sum of all z_i for which z_i = x_i * y_i holds.
    """
    result = 0
    for i, x_i in enumerate(x):
        result = result + x_i * y[i]
    return result


def angle(x, y):
    """Calculates the angle between two vectors x and y.

    Uses the definition of the scalar product <x, y>:

        <x, y> = |x| |y| cos(alpha)

    where |x| is the norm of x.

    The function uses the distance between the two points and the origin
    respectively to calculate the respective norms.

    Args:
        x: The first vector.
        y: The second vector.

    Returns:
        The angle between the two vectors.
    """
    return math.acos(dot(x, y) / math.sqrt(dot(x, x) * dot(y, y)))
```

```
def pdist(x, y, p=2):
    """Calculates the p-distance between x and y.

    The p-distance between two points is the p-th root of the sum over all
    (x_i - y_i) ^ p.

    By default, the p=2 distance is returned, which is also known as the
    euclidean distance.

    Args:
        x: One point.
        y: Another point.

    Returns:
        The p-distance between x and y.
    """
    result = 0
    for i, x_i in enumerate(x):
        result = result + (x_i - y[i]) ** p
    return result ** (1 / p)


if __name__ == '__main__':
    a = [1, 2, 3]
    b = [4, 5, 6]
    c = [0, 1, 0, 0, 1]
    d = [1.5, 2.5, 3.5, 4.5, 5.5]

    print("1. Expected: [5, 7, 9]\nActual: {}\n".format(add(a, b)))
    print("2. Expected: [5, 7, 9]\nActual: {}\n".format(add(b, a)))
    print("3. Expected: [-3, -3, -3]\nActual: {}\n".format(sub(a, b)))
    print("4. Expected: 8.0\nActual: {}\n".format(dot(c, d)))
    print("5. Expected: ~5.2\nActual: {}\n".format(pdist(a, b)))
    print("6. Expected: ~5.6\nActual: {}\n".format(pdist(c, d, 4)))
    print("7. Expected: ~0.23\nActual: {}\n".format(angle(a, b)))
```

Output:

```
1. Expected: [5, 7, 9]
Actual: [5, 7, 9]

2. Expected: [5, 7, 9]
Actual: [5, 7, 9]
```

```
3. Expected: [-3, -3, -3]
Actual: [-3, -3, -3]

4. Expected: 8.0
Actual: 8.0

5. Expected: ~5.2
Actual: 5.196152422706632

6. Expected: ~5.6
Actual: 5.595528796309361

7. Expected: ~0.23
Actual: 0.2257261285527342
```

Exercise 2: Mastering I/O: Hangman

File: code/hangman_game.py

```
"""
This module implements the classic game hangman.

The goal of the game for the player is to guess a word the computer chose by
guessing individual letters. If one of the letters is part of the chosen word,
the computer tells the player the positions of all occurences.

The game consists of multiple rounds where the player can guess a letter
when prompted to do so.

If the guessed letter is in the word the computer chose, the game state is
updated and the player is presented with the positions of the letters they
guessed correctly.

If the guess was wrong, that means it is not part of the guess word, it is
added to the list of wrong guesses.

If the player guesses all letters before the number of wrong guesses
exceeds the number of allowed misses, they win. Otherwise the computer
wins.
"""
import random
```

```
ALLOWED_MISSES = 5
RULES = """
Hello! Let's play a game of hangman!
I already picked a word, and you now have to guess letters.
But be warned, if you guess more than {} times wrong, you lose!
""".format(ALLOWED_MISSES)


def read_possible_words(filename):
    """Reads a list of words from a file.

    Args:
        filename: The file to read.
    """
    with open(filename, 'r') as wordlist:
        return wordlist.read().splitlines()


def win(word):
    """Returns True if the players wins.

    The player wins if there are no underscores left in the word.

    Args:
        word: The word to check for underscores.

    Returns:
        True on win, False otherwise.
    """
    return '_' not in word


def loss(guesses):
    """Returns True if the player loses.

    The player loses if there are no allowed misses left.

    Args:
        guesses: The list of wrongly guessed letters.

    Returns:
        True on loss, False otherwise.
    """
    return ALLOWED_MISSES < len(guesses)
```

```
def initialize_word(wordlist):
    """Chooses a random word from the given wordlist.

    Args:
        wordlist: A list containing words to choose from.
    """
    words = read_possible_words(wordlist)
    return random.choice(words)


def initialize_guess_word(word):
    """Creates a list containing only underscores.

    The resulting list has as many underscores as word has letters. It is used
    to represent the correct guesses.

    Args:
        word: The word.

    Returns:
        A list containing an underscore per letter in word.
    """
    guess_word = []
    for letter in word:
        guess_word.append('_')
    return guess_word


def print_state(guess_word, guessed_letters):
    """Prints the game state.

    Args:
        guess_word: The current guess_word.
        guessed_letters: The letters which were guessed but wrong.
    """
    print("Word: {} Wrong guesses: {}".format(guess_word, guessed_letters))


def update_guessword(guess_word, word, guessed_letter):
    """Updates the guess word.

    Places the guessed_letter at its positions in word into the guess_word,
    i.e.  if the guess_word was ['_', 'e', '_', '_', '_'], the word 'hello' and
    the guessed_letter 'l', the result would be ['_', 'e', 'l', 'l', '_'].
```

```
    This function modifies the list in place.

    Args:
        guess_word: The list to be updated if guessed_letter is in word.
        word: The correct word.
        guessed_letter: The letter to be inserted if existing.

    Returns:
        The updated guess_word.
    """
    for i, letter in enumerate(word):
        if letter == guessed_letter:
            guess_word[i] = letter
    return guess_word


def hangman():
    """Plays a game of hangman.

    This function will initialize a game of hangman. It reads a list of words,
    chooses a random word, prints the rules and then plays the game.
    """
    word = initialize_word('04_CollectionsFileIO/code/hangman_words.txt')
    guess_word = initialize_guess_word(word)

    print(RULES)

    guessed_letters = []
    while not win(guess_word) and not loss(guessed_letters):
        print_state(guess_word, guessed_letters)
        letter = input('Next guess? ')
        if letter in word:
            guess_word = update_guessword(guess_word, word, letter)
            if win(guess_word):
                print_state(guess_word, guessed_letters)
                print('You win! It was "{}".'.format(word))
        else:
            guessed_letters.append(letter)
    if loss(guessed_letters):
        print_state(guess_word, guessed_letters)
        print('Sorry, you lose. The word was "{}".'.format(word))


if __name__ == '__main__':
    hangman()
```