Exercise Sheet 04 – File I/O and Algorithms

Sebastian Höffner Aline Vilks

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

Submission

By the end of this sheet you will have a number of different files to submit. In Stud.IP you will have a directory for your own group, please upload them there. It is easier for you if you just archive (preferably zip) all files and upload your archive, but it is okay if you upload them one by one.

Exercise 1: Vector & matrix mathematics

We can model mathematical vectors with tuples and lists. However, as Python is not designed to do vector math with lists nor tuples, we have to define some functions ourselves. To be able to do some basic vector math, write a script `vector_math.py` which defines the following functions:

a) `def add(x, y):` Adds `x` and `y`, such that the result z follows $z_i = x_i + y_i$.
b) `def sub(x, y):` Subtracts `x` and `y`, such that the result z follows $z_i = x_i - y_i$.
c) `def dot(x, y):` Calculates the scalar (dot) product of two vectors `x` and `y`. The scalar product $< x, y >$ is defined as $< x, y >= \sum_{i=0}^{N} x_i y_i$.
d) `def pdist(x, y, p=2):` Calculates the distance between two vectors `x` and `y` over a given p-norm. It is defined as $d_p(x, y) = \sqrt[p]{\sum_{i=0}^{N} (x_i - y_i)^p}$, where $p \geq 1$.
e) `def angle(x, y):` Calculates the angle between two vectors `x` and `y`. The angle can be found by using an alternative definition of the scalar product: $< x, y >= \|x\| \, \|y\| \cos\theta$. If you solve it for θ, you find the angle between x and y.

You can assume that all vectors are of the same length.

Test your implementations using the following inputs and results. Note that it is fine to return lists or tuples, use what you feel comfortable with.

Use `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]`.

Call	Result
`add(a, b)`	`[5, 7, 9]`
`add(b, a)`	`[5, 7, 9]`
`sub(a, b)`	`[-3, -3, -3]`
`dot(c, d)`	`8.0`
`pdist(a, b)`	approx. `5.2`
`pdist(c, d, 4)`	approx. `5.6`
`angle(a, b)`	approx. `0.23`

You can try out other values as well.

Exercise 2: Mastering I/O: Hangman

Let's implement a little game. In Hangman one person (in our case the computer) picks a random word and tells us how many letters there are, e.g. for `hello` it would tell us: `_____`. Now your job is to guess the word, letter by letter. So if you would guess `e`, the computer would reveal `_e___`. If you then guessed `l`, the result would be `_ell_`. Traditionally, for each wrong letter guessed, a player would get another stroke of the little hangman (see Figure 1). Eventually the stick figure will hang – or you solve the word and win! Since it's hard to visualize the stick figure on the terminal, you might want to just use a counter.

Task

Write a file `hangman_game.py` which implements a game of hangman. In the accompanying `*.zip` archive there is a file called `hangman_words.txt` which you should use (but you can change it as much as you like) to read the list of possible words.

Example pseudocode

```
Set number of misses
Read possible words
Pick one word
Prepare guess word with underscores
```

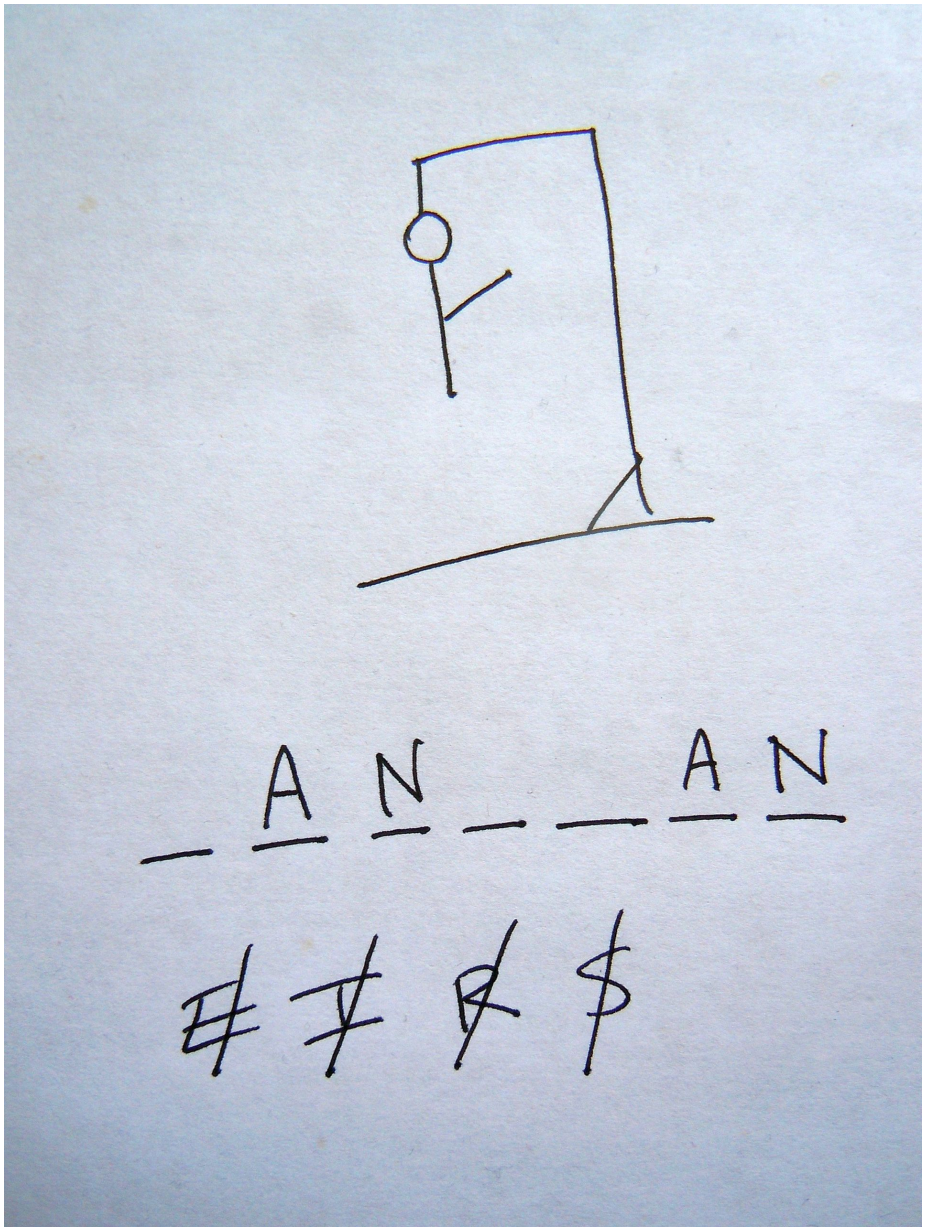

Figure 1: Example hangman game (McGeddon, Wikipedia, Public Domain)

```
Present user with the "rules"
While not guessed and more than 0 misses left:
    Present current game state
    Get letter from user input
    If letter exists in chosen word:
        Update guess word
        If guessed:
            Win
    Else:
        Update list of failures and misses
        If no misses left:
            Lose
```

Hints

Strings are immutable, so you **can not** do something like this:

```
a = 'hello'
a[3] = 'b'
```

Instead, try to represent the *guess word* as a list filled with underscores, like this:

```
word = 'hello'
guess_word = ['_', '_', '_', '_', '_']
```

Then, whenever a letter is guessed, check if it is inside the word, and if so, update the `guess_word` accordingly (this code snippet is not really useful, but remember the keyword `in`):

```
if 'l' in word:
    guess_word[word.index('l')] = 'l'
```

Similarly, to check if the game is won, you just need to see if there are still _s inside the list.

Try to split your code into several function which only do small bits, e.g. write a function which checks if you won, one which prints the current game state, one which reads in the file, etc.