The deadline for this exercise sheet is **Monday, 14.05.2018, 08:00.**

1 Warm-Up

1.1 Key Sort

Given a list of the following structure:

```
my_lst = [
  {
    "key1" : value1,
    "key2" : value2,
    "key3" : value3
  },
  {
    "key1" : a_value,
    "key2" : b_value,
    "key3" : c_value
  },
  ...
]
```

Sort the list in such a way, that the value of `"key2"` is used to compare and sort the dictionaries inside the list. In the file `task_1.py` you can find an example dictionary as well as a sorted version to compare to when you are done.
Solution:

```
sorted(my_lst, key=lambda x: x["key2"])
```

1.2 Read & Write

In the file `task_1.txt`, is an unsorted list of numbers. Read in the list, sort it in *descending* order and write it back to the file. In the end the file should only contain the sorted list.
Solution:

```
with open("./task_1.txt", "r+") as f:
    lst_in = f.read() # read the list as string
    lst_in = lst_in[1:-1] # get rid of the brackets
    # split on the commas of the list,
    # and cast to float since we are dealing with numbers
    lst = [float(x) for x in lst_in.split(",")]
    lst.sort(reverse=True) # and sort
    f.write(str(lst)) # back in the file it goes
```

2 Powering through I/O

Let's make a game! In Hangman, one player – in our case this will be the computer – picks a random word and tells the other player(s) how many letters it has – usually displayed through underscores. For example, if the word is

`hello` we would get `_ _ _ _ _`.
The other player(s) now have to guess the word letter by letter. If a letter that is part of the word is guessed, it is revealed. To continue our example, if you would guess an **e** the computer would reveal it and the new game state would be `_ e _ _ _`. If your next guess would be an **l** the new game state would be `_ e l l _`. If you guessed the whole word, you win the game!
But there is a catch: Traditionally, at the start of the game you would draw empty gallows, and every time you guessed a letter that is *not* in the word, you would draw one more part of a man hanging – thus the name hangman (see 2). If you guess the word before the man is complete, you win! Otherwise the stick figure has come to a tragic end and you lose.
Since it might be hard to visualise the hangman on the terminal (you are very welcome to try it out though), you might want to use just a counter.

Task

Write a `hangman.py` script which implements a version of the hangman game. In the supplied **.zip** file you can find a `words.txt`, which you should use to read in the list of possible words. You are welcome to change the file though.
An example pseudocode:

```
Set number of misses
Read in possible words
Choose a word
Prepare guessword with underscores
Display the rule set
While not guessed and more than 0 misses left:
  Display current game state
  Get a guess letter as user input
  If guessed letter is in the word:
    Update the guess word
    If guessed:
      win
  Else:
    Update list of failes and misses
    If no misses left:
      Lose
```

Hints

- Remember that strings are immutable, so you can not do:

```
1 a = 'hello'
2 a[3] = 'a
```

- You can instead display the guessword as a list of underscores

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

- Whenever you have gotten an input letter, check whether it is part of the word. You can use the in keyword and the word.index(input_char) function for this.

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

  Note: This code snippet probably is not how you are going to use it. But it might point you in the right direction.

- Similarly, you can use `in` to check whether the player has won.

- For choosing a word, you could take a look at the `choice` function from the `random` module. You can read upon it here: `https://docs.python.org/3.5/library/random.html#random.choice`.
  And you can use it with the following structure:

```
import random


random.choise(my_list)
```

It might be a good idea to split your code into several smaller functions, which each perform a single task. For example, one function which checks whether the player has won, one function to print the current game state, one to read in the file, and one to pick a word, and so on. Then combine those functions to build your whole game.
Please note, that the hints are just that: hints for a possible solution to a problem. Your program can be perfectly fine without using any of the hints.

Solution:

¡¡¡¡¡¡¡ HEAD

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

The goal of the game for the player is to guess a word the computer chose at
random 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
```

```
12  updated and the player is presented with the positions of the letters
        they
13  guessed correctly.
14
15  If the guess was wrong, that means it is not part of the guess word, it
        is
16  added to the list of wrong guesses.
17
18  If the player guesses all letters before the number of wrong guesses
19  exceeds the number of allowed misses, they win. Otherwise the computer
20  wins.
21  """
22  import random
23  import string
24
25
26  MAX_MISSES = 5
27  RULES = """
28  Hello! Let's play a game of hangman!
29  I already picked a word, and you now have to guess letters.
30  But be warned, if you guess wrong more than {} times, you lose!
31  """.format(MAX_MISSES)
32
33
34  def get_art(lvl=0, width=16):
35      """
36      Returns an array of lines of a hangman ascii art.
37
38      The lines depend on the given level. The higher the level, the more
39      of the man is displayed. The Art allows for 5 levels max.
40
41      Args:
42          lvl: the level to which the man is to be drawn
43          width: what minimum width the drawing should have
44
45      Returns:
46          A list of strings, each string representing one line of the
                drawing,
47          formatted to be at least `width` wide.
48      """
49      art_top = [" _______",
50                 "|       |"]
51      if lvl < 5:
52          art_mid = ["|", "|", "|", "|"]
53          if lvl > 0:
54              art_mid[0] = "|      (_) "
55          if lvl > 1:
56              art_mid[1] = "|      /|  "
57          if lvl > 2:
58              art_mid[1] = "|      /|\ "
59          if lvl > 3:
60              art_mid[2] = "|       |  "
61              art_mid[3] = "|      /   "
62
63          art_bot = ["|____________",
64                     "/|        | |"]
65      else:
```

```
66          art_mid = [
67              "|      (-)    ",
68              "|      /|\    ",
69              "|       |     ",
70              "|      / \    "
71          ]
72          art_bot = ["|----      ---",
73                     "/|   \    | |"]
74
75      form = lambda s: "{0:{width}s}".format(s, width=width)
76
77      art = [form(s) for s in art_top] +\
78            [form(s) for s in art_mid] +\
79            [form(s) for s in art_bot]
80
81      return art
82
83
84  def read_words(file="words.txt"):
85      """
86      Reads a list of words from a file.
87
88      There needs to be one word per line, for this to work properly.
89
90      Args:
91          file: the file to read from
92
93      Returns:
94          An array of all the words in the file
95      """
96      with open(file, "r") as f:
97          return f.read().lower().splitlines()
98
99
100 def pick_word(words):
101     """
102     Chooses a random entry from the given list.
103
104     Args:
105         words: the list of words to pick a word from
106
107     Returns:
108         One word from the list `words`
109     """
110     return random.choice(words)
111
112
113 def get_guess():
114     """
115     Asks the user for an input letter until it is a valid letter.
116
117     If the input is more than one character, or is not from the ascii
118     alphabet (a-z), we ask the user again. We only deal with lowercase
119     letters.
120
121     Returns:
122         The input guess from the user
```

```
123     """
124     guess = ""
125     while (len(guess) != 1) or (not guess in string.ascii_lowercase):
126         guess = input("Which letter is your next guess? ").lower()
127
128     return guess
129
130
131 def print_game_state(turn, misses, guess_word, guesses):
132     """
133     Print the current state of the game.
134
135     It displays a short header, and the ASCII art depending on how many
136     misses there were already. It then displays the game state
            information:
137     - How many misses we made
138     - The current state of the guess word
139     - The letters that were misses
140
141     Args:
142         turn: In which turn we are
143         misses: how many misses are left
144         guess_word: the current guess word
145         guesses: the list of mistaken letters
146     """
147     # missed holds how many misses we made already
148     missed = MAX_MISSES - misses
149
150     # and empty line at the start
151     space = ""
152     header = "HANGMAN - THE GAME: Turn {}".format(turn)
153     lines = [space] + [header] + get_art(missed)
154     lines[2] += "MISSED: {} / {}".format(missed, MAX_MISSES)
155     lines[3] += "GUESS THE WORD!"
156     lines[4] += " ".join(guess_word)
157     lines[6] += "Misses so far: " + ", ".join(guesses)
158
159     # print the game state
160     for line in lines:
161         print(line)
162
163
164 def update_guess_word(word, guess_word, guess):
165     """
166     Updates the guess_word with the newly guessed letter.
167
168     By iterating over the word and comparing each character, we can
            find
169     all occurences of that letter and can replace the underscores in
            the
170     guess word for each found occurence.
171
172     E.g. if the word is 'hello' and the guess_word was ['_', 'e', '_',
            '_', '_']
173     and the guess was 'l', the result will be ['_', 'e', 'l', 'l', '_']
174
175     The function modifies the list in place.
```

```
176
177     Args:
178         word: the target word
179         guess_word: the current state of the guess word
180         guess: the guessed letter
181
182     Returns:
183         the updated state of the guess word. Though unnecessary, since
                lists
184         are passed by reference and altered directly.
185     """
186     for i, letter in enumerate(word):
187         if letter == guess:
188             guess_word[i] = letter
189
190     return guess_word
191
192
193 def print_guide():
194     """Prints the rules of the game"""
195     print(RULES)
196
197
198 def check_win(guess_word):
199     """
200     Returns True if the player has won.
201
202     The player has won if there are no underscores left to guess.
203
204     Args:
205         guess_word: the current state of the guess word
206
207     Returns:
208         True in case of win, False otherwise.
209     """
210     return not "_" in guess_word
211
212
213 def game_end(won, word):
214     """
215     Prints a message depending on whether the player has won.
216
217     Args:
218         won: Boolean whether the player has won
219         word: The target word
220     """
221     win_msg = "Congratulations!"
222     lose_msg = "Oh no! Good luck next time! The word was {}"
223
224     msg = win_msg if won else lose_msg.format(word)
225     print(msg)
226
227
228 def init_guess_word(length):
229     """
230     Returns the initial guess word state.
231
```

```
232     The guess word is initialised with underscores, one for each letter
233     of the target word.
234
235     Args:
236         length: the length of the target word
237
238     Returns:
239         The initialised guess word, a list of `length` underscores
240     """
241     return ["_"] * length
242
243
244 def init():
245     """
246     Initialises our game world.
247
248     Sets the default values for the game state variables, and then
            return
249     them as a tuple. This includes to read the words from the file,
            picking
250     a target word at random and initialising the guess word, as well as
251     printing the guide to the game.
252
253     Returns:
254         The tuple that forms the game state.
255     """
256     turn = 0
257     words = read_words()
258     the_word = pick_word(words)
259     guess_word = init_guess_word(len(the_word))
260     misses = MAX_MISSES
261     guesses = []
262
263     print_guide()
264
265     return turn, the_word, guess_word, misses, guesses
266
267
268 def game():
269     """
270     Runs the game loop.
271
272     This function puts it all together to form the whole game. It
            initialises
273     the game state values, and sets the default win state (false).
274     It then loops until either the player has won, or the player missed
             all
275     his allowed mistakes.
276     In each loop we print the current state of the game, and get a new
            guessed
277     letter. If it was a correct guess, we update the guess word and
            check
278     whether the player has won, otherwise we decrement our allowed
            mistakes.
279
280     Once the loop ends we print the game world one last time, and print
281     the end of game message.
```

```
    """
    turn, word, guess_word, misses, guesses = init()
    won = False

    while not won and misses > 0:
        print_game_state(turn, misses, guess_word, guesses)
        guess = get_guess()

        if guess in word:
            guess_word = update_guess_word(word, guess_word, guess)
            won = check_win(guess_word)
        else:
            guesses += guess
            misses -= 1

        turn += 1

    print_game_state(turn, misses, guess_word, guesses)
    game_end(won, word)


# we can continue the game until the player quits
cont = "y"
while cont == "y":
    game() # play a game!
    # on a y or Y we continue
    cont = input("Do you want to play again? (y/n): ").lower()
```

=======

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

The goal of the game for the player is to guess a word the computer chose at
random 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.
"""
```

```
22 __all__ = ['game']
23
24 import random
25 import string
26
27
28 MAX_MISSES = 5
29 RULES = (
30     "Hello! Let's play a game of hangman!\n"
31     "I already picked a word, and you now have to guess letters.\n"
32     "But be warned, if you guess wrong more than {} times, you lose!\n"
33 ).format(MAX_MISSES)
34 MSG_WIN = "Congratulations!"
35 MSG_LOSE = "Oh no! Good luck next time! The word was {}"
36
37
38 def get_art(lvl=0, width=16):
39     """
40     Returns an array of lines of a hangman ascii art.
41
42     The lines depend on the given level. The higher the level, the more
43     of the man is displayed. The Art allows for 5 levels max.
44
45     Args:
46         lvl: the level to which the man is to be drawn
47         width: what minimum width the drawing should have
48
49     Returns:
50         A list of strings, each string representing one line of the
              drawing,
51         formatted to be at least `width` wide.
52     """
53     art_top = [" -------",
54                "|       |"]
55
56     art_mid = ["|", "|", "|", "|"]
57     art_bot = ["|-----------",
58                "/|       | |"]
59
60     if lvl > 0:
61         art_mid[0] = "|     (_) "
62     if lvl > 1:
63         art_mid[1] = "|     /|  "
64     if lvl > 2:
65         art_mid[1] = "|     /|\ "
66     if lvl > 3:
67         art_mid[2] = "|      |  "
68         art_mid[3] = "|     /   "
69     if lvl > 4:
70         art_mid[3] = "|     / \   "
71         art_bot = ["|----    ---",
72                    "/|   \   | |"]
73
74     def form(s): return "{0:{width}s}".format(s, width=width)
75
76     art = [form(s) for s in art_top] +\
77           [form(s) for s in art_mid] +\
```

```
78              [form(s) for s in art_bot]
79
80      return art
81
82
83  def read_words(file="words.txt"):
84      """
85      Reads a list of words from a file.
86
87      There needs to be one word per line, for this to work properly.
88
89      Args:
90          file: the file to read from
91
92      Returns:
93          An array of all the words in the file
94      """
95      with open(file, "r") as f:
96          return f.read().lower().splitlines()
97
98
99  def pick_word(words):
100     """
101     Chooses a random entry from the given list.
102
103     Args:
104         words: the list of words to pick a word from
105
106     Returns:
107         One word from the list `words`
108     """
109     return random.choice(words)
110
111
112 def get_guess(guesses):
113     """
114     Asks the user for an input letter until it is a valid letter.
115
116     If the input is more than one character, or is not from the ascii
117     alphabet (a-z), we ask the user again. We only deal with lowercase
118     letters.
119
120     Returns:
121         The input guess from the user
122     """
123     guess = ""
124     while ((len(guess) != 1) or
125             (not guess in string.ascii_lowercase) or
126             (guess in guesses)):
127         guess = input("Which letter is your next guess? ").lower()
128
129     return guess
130
131
132 def print_game_state(turn, misses, guess_word, guesses):
133     """
134     Print the current state of the game.
```

```
135
136     It displays a short header, and the ASCII art depending on how many
137     misses there were already. It then displays the game state
            information:
138     - How many misses we made
139     - The current state of the guess word
140     - The letters that were misses
141
142     Args:
143         turn: In which turn we are
144         misses: how many misses are left
145         guess_word: the current guess word
146         guesses: the list of mistaken letters
147     """
148     # missed holds how many misses we made already
149     missed = MAX_MISSES - misses
150
151     # and empty line at the start
152     space = ""
153     header = "HANGMAN - THE GAME: Turn {}".format(turn)
154     lines = [space] + [header] + get_art(missed)
155     lines[2] += "MISSED: {} / {}".format(missed, MAX_MISSES)
156     lines[3] += "GUESS THE WORD!"
157     lines[4] += " ".join(guess_word)
158     lines[6] += "Misses so far: " + ", ".join(guesses)
159
160     # print the game state
161     for line in lines:
162         print(line)
163
164
165 def update_guess_word(word, guess_word, guess):
166     """
167     Updates the guess_word with the newly guessed letter.
168
169     By iterating over the word and comparing each character, we can
            find
170     all occurences of that letter and can replace the underscores in
            the
171     guess word for each found occurence.
172
173     E.g. if the word is 'hello' and the guess_word was ['_', 'e', '_',
            '_', '_']
174     and the guess was 'l', the result will be ['_', 'e', 'l', 'l', '_']
175
176     The function modifies the list in place.
177
178     Args:
179         word: the target word
180         guess_word: the current state of the guess word
181         guess: the guessed letter
182
183     Returns:
184         the updated state of the guess word. Though unnecessary, since
                lists
185         are passed by reference and altered directly.
186     """
```

```
187     for i, letter in enumerate(word):
188         if letter == guess:
189             guess_word[i] = letter
190 
191     return guess_word
192 
193 
194 def print_guide():
195     """Prints the rules of the game"""
196     print(RULES)
197 
198 
199 def check_win(guess_word):
200     """
201     Returns True if the player has won.
202 
203     The player has won if there are no underscores left to guess.
204 
205     Args:
206         guess_word: the current state of the guess word
207 
208     Returns:
209         True in case of win, False otherwise.
210     """
211     return not "_" in guess_word
212 
213 
214 def init():
215     """Initialises our game world.
216 
217     Sets the default values for the game state variables, and then
        return
218     them as a tuple. This includes to read the words from the file,
        picking
219     a target word at random and initialising the guess word, as well as
220     printing the guide to the game.
221 
222     Returns:
223         The tuple that forms the game state.
224     """
225     turn = 0
226     words = read_words()
227     the_word = pick_word(words)
228     guess_word = ['_'] * len(the_word)  # works because strings are
        immutable
229     misses = MAX_MISSES
230     guesses = []
231 
232     print_guide()
233 
234     return turn, the_word, guess_word, misses, guesses
235 
236 
237 def game(filename="words.txt"):
238     """
239     Runs the game loop.
240 
```

```
241     This function puts it all together to form the whole game. It
            initialises
242     the game state values, and sets the default win state (false).
243     It then loops until either the player has won, or the player missed
             all
244     his allowed mistakes.
245     In each loop we print the current state of the game, and get a new
            guessed
246     letter. If it was a correct guess, we update the guess word and
            check
247     whether the player has won, otherwise we decrement our allowed
            mistakes.
248
249     Once the loop ends we print the game world one last time, and print
250     the end of game message.
251
252     Args:
253         filename: The file to load the words for hangman from
254             Defaults to "words.txt"
255     """
256     turn, word, guess_word, misses, guesses = init()
257
258     while not check_win(guess_word) and misses > 0:
259         print_game_state(turn, misses, guess_word, guesses)
260         guess = get_guess(guesses)
261
262         if guess in word:
263             guess_word = update_guess_word(word, guess_word, guess)
264         else:
265             guesses += guess
266             misses -= 1
267
268         turn += 1
269
270     print_game_state(turn, misses, guess_word, guesses)
271
272     print(MSG_WIN if check_win(guess_word) else MSG_LOSE.format(word))
273
274
275
276 if __name__ == '__main__':
277     # we can continue the game until the player quits
278     cont = "y"
279     while cont == "y":
280         game()  # play a game!
281         # on a y or Y we continue
282         cont = input("Do you want to play again? (y/n): ").lower()
```

¿¿¿¿¿¿¿ f02e5a0dc271881fab8d9e89529da082c88d8fae

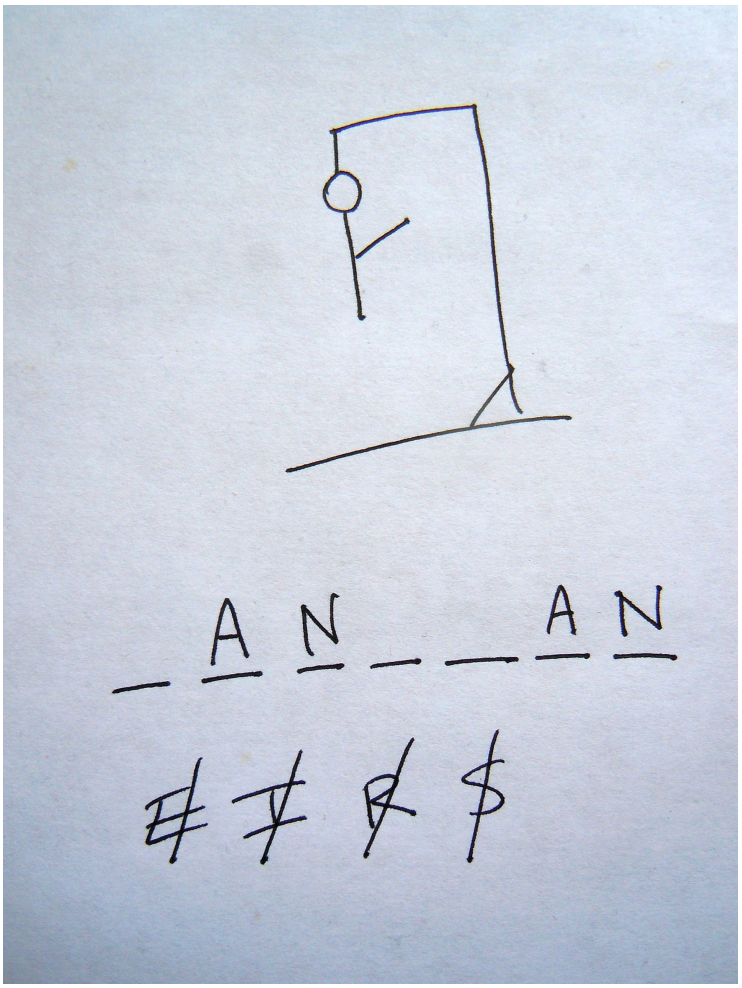

Figure 1: Example hangman game
Taken from `https://upload.wikimedia.org/wikipedia/commons/thumb/f/f4/Hangman_game.jpg/1920px-Hangman_game.jpg`