Week 6: Dictionaries

Contents

Important

First, `download_week06.py` and run the script in `VSCode`. You may need to confirm the download in your browser.

The script should end with `Successfully completed!`. If you encounter an error, follow the instructions in the error message. If needed, you can proceed with the exercises and seek assistance from a TA later.

The topic of this week will be to familiarize with the use of dictionaries. A dictionary can be defined as shown below:

```
person = {"first_name": "John",
        "last_name" : "Johnson",
        "age"       : 37
        }
```

Dictionaries are a fundamental data type in Python used to store data in a key-value pair format. They are a vital part of Python's core data structures, so understanding their functionality is crucial. Dictionaries offer a convenient and efficient way to access values based on their associated keys. In the provided example, we have a dictionary named "person" that includes various keys representing attributes of an individual, such as first name, last name, and age, each paired with its respective value. To retrieve values from the dictionary, you can access them like this:

```
>>> print("My name is", person["first_name"], person["last_name"], "and I am", person["age"], "years old")
My name is John Johnson and I am 37 years old
```

Skip to main content

Video 6.0: Slow tests (week5), tests that cannot be found, and a few other problems

Exercise 6.1: Text to NATO

Exercise 6.1

To get started, we will use dictionaries in order to convert letters of the alphabet to their corresponding word from the NATO alphabet. The NATO alphabet is useful for spelling words over a noisy telephone connection. You should create a function called `text_to_nato` that spells out a word (written in plain text) in the NATO alphabet. The function should accept words which contain letters in upper, lower, or mixed case, and should return the NATO "code words" exactly as written below, separated by dashes. You can assume that the input string contains only letters from A-Z (in either upper or lowercase). We also provide with the following dictionary to ease the implementation of this exercise.

```
alphabet_to_nato = {'a': 'Alpha', 'b': 'Bravo', 'c': 'Charlie', 'd': 'Delta', 'e': 'Echo',
        'f': 'Foxtrot', 'g': 'Golf', 'h': 'Hotel', 'i': 'India', 'j': 'Juliet', 'k': 'Kilo',
        'l': 'Lima', 'm': 'Mike','n': 'November', 'o': 'Oscar', 'p': 'Papa', 'q': 'Quebec',
        'r': 'Romeo', 's': 'Sierra', 't': 'Tango', 'u': 'Uniform', 'v': 'Victor',
        'w': 'Whiskey', 'x': 'Xray', 'y': 'Yankee', 'z': 'Zulu'}
```

Here is an example of how this function should work:

```
>>> text_to_nato('helLo')
'Hotel-Echo-Lima-Lima-Oscar'
>>> text_to_nato('alOha')
'Alpha-Lima-Oscar-Hotel-Alpha'
```

You can test the function by inserting your solution into the file `cp/ex06/nato.py`.

cp.ex06.nato.text_to_nato(*plaintext*)

Return the NATO version of a word separated by dashes.

Parameters:

plaintext (`str`) – The word to replace with its phrase according to the NATO alphabet.

Return type:

`str`

Returns:

The NATO representation of the input word.

Skip to main content

Exercise 6.2: Letter histogram

Exercise 6.2

Now you should write a function `letter_histogram`, which takes a `str` as input and computes a histogram of letter occurrences. Here are the specifications:

- The function should only consider 26 letters of the English alphabet.
- You should remove/disregard the following characters: Dot `.`, comma `,` and space " ".
- Uppercase letters should be counted together with their corresponding lowercase letters (use `.lower()` on the string)
- The function should return a dictionary with 26 key-value pairs in alphabetical order, such as `{'a': 132, 'b': 79, ...}`.

Tip

Tip

```
>>> letter_histogram('Bananananana')
{'a': 6, 'b': 1, 'c': 0, 'd': 0, 'e': 0, 'f': 0, 'g': 0, 'h': 0, 'i': 0, 'j': 0, 'k': 0, 'l': 0, 'm': 0, 'n': 5, 'o': 0
>>> letter_histogram('Batman')
{'a': 2, 'b': 1, 'c': 0, 'd': 0, 'e': 0, 'f': 0, 'g': 0, 'h': 0, 'i': 0, 'j': 0, 'k': 0, 'l': 0, 'm': 1, 'n': 1, 'o': 0
```

You can test the function by inserting your solution into the file `cp/ex06/letter.py`.

cp.ex06.letter.letter_histogram(*input_string*)

Return the histogram of letter occurrences.

Parameters:

input_string (`str`) – The word based on which the letter histogram is calculated.

Return type:

`dict`

Returns:

The alphabet characters as keys with their corresponding occurrences as values.

Exercise 6.3: Word histogram

Exercise 6.3

Now your task is to implement a function `word_histogram` which takes a `list` of `str` containing sentences of a text as input. The function should create a histogram of words found within the provided text. Here are the specific requirements:

- Punctuation (`.`, `,`), spaces, and capitalization should be disregarded. Everything should be converted to lowercase.
- The function should return a dictionary where each key represents a unique word, and the corresponding value is the number of occurrences of that word within the text.
- The format of the returned dictionary should look like this: `{'write': 2, 'the': 12, 'function': 7, ...}`.

```
>>> word_histogram(['This is the first sentence', 'This is the second sentence'])
{'this': 2, 'is': 2, 'the': 2, 'first': 1, 'sentence': 2, 'second': 1}
>>> word_histogram(['... to be or not to be ? ...'])
{'to': 2, 'be': 2, 'or': 1, 'not': 1}
```

You can test the function by inserting your solution into the file `cp/ex06/word_histogram.py`.

cp.ex06.word_histogram.word_histogram(*lines*)

Return the word count histogram from the input lines.

Parameters:

Skip to main content

Return type:
`dict`

Returns:
The histogram of word occurrences.

Exercise 6.4: Extracting most commonly occurring words that are not part of a list.

Exercise 6.4

Now you should create a function `extract_keyword` which takes two arguments:

- A text, which is a `list` containing sentences of a text.
- A `list` with words to ignore/disregard from the text.

The objective of this function is to return a `dict` containing the five most frequently occurring words in the text, excluding those in the ignore list, along with their respective counts.

To achieve this, you can utilize the previously implemented `word_histogram` function. You can assume that there are more than five unique words in the text that are not present in the ignore list.

```
>>> text = ["In a world full of possibilities, creativity shines brighter than anything else.",
...         "Creativity is the key to solving complex problems and driving innovation.",
...         "The mindset of creativity finds solutions where others see only obstacles.",
...         "Unlock your potential creativity, and embrace a future filled with endless opportunities."]
>>> ignore_list = ['a', 'an', 'the', 'in']
>>> extract_keyword(text, ignore_list)
{'creativity': 4, 'of': 2, 'and': 2, 'world': 1, 'full': 1}
```

Tip

You can test the function by inserting your solution into the file `cp/ex06/word_histogram.py`.

cp.ex06.word_histogram.extract_keyword(*lines*, *ignore_list*)

Return the five most frequent words that are not on the ignore list and their occurrences.

Parameters:
- **lines** (`str`) – The sentence to extract keywords from.
- **ignore_list** (`list`) – The words that should ignored.

Return type:
`dict`

Returns:
The five most frequent words in the sentence as keys with their count as values.

Exercise 6.5: Find language speakers from a group of people.

Exercise 6.5

Now, you are given a dictionary `{'Peter': ['Danish', 'English'], ...}`, which contains people's names and the languages that they speak. Write a function `get_people_by_language` which given a `str` that contains a language returns a list containing names of people speaking that language. Here is an example:

```
>>> name_languages = {'Peter': ['Danish', 'English'], 'Alice': ['English', 'French'], 'John': ['Spanish', 'English'],
>>> language = 'English'
>>> get_people_by_language(language, name_languages)
['Peter', 'Alice', 'John', 'Anna']
```

Skip to main content

You can test the function by inserting your solution into the file `cp/ex06/language.py`.

`cp.ex06.language.get_people_by_language(language, name_languages)`

Return the names of people who speak a specific language.

Parameters:

- **language** (`str`) – A string containing the desired language.
- **name_languages** (`dict`) – A dictionary containing the names of people along with their spoken languages

Return type:

`list`

Returns:

The names of people that speak the desired language.

Exercise 6.6: Truncate the values in a list of floats.

Exercise 6.6

Now, you will create a function named `truncate_values` that takes a `list` of floating-point numbers and a settings `dict` (e.g., `list = [1.5,2.5,3.4]`, `settings = {'vmin': 0, 'vmax': 1, 'normalize': True}`). The purpose of the function is to process the list of floating-point numbers, and to return the processed values.

If `normalize` is set to `True` then, before truncating the values, the list of floats should be normalized. This means that the values v in the list should be transformed with a multiplication and addition such that the new values $\tilde{v} = a \cdot v + b$ are within the range `[0,1]`, and have mininium and maximum values of `0` and `1` respectively.

Following this step, the numbers should be truncated to be within the specified `vmin` and `vmax` range. Values outside (`vmin`, `vmax`) range should be truncated, meaning that they should be replaced with `vmin` or by `vmax` if they are smaller than or greater than those values respectively.

Two examples are shown below:

```
>>> my_list = [0, 1.0, 2.0, 3.0, 4.0, 5.0]
>>> truncate_values(my_list, {'vmin': 0, 'vmax': 2, 'normalize': False})
[0, 1.0, 2.0, 2, 2, 2]
>>> truncate_values(my_list, {'vmin': 0, 'vmax': 2, 'normalize': True})
[0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
>>> truncate_values(my_list, {'vmin': 0, 'vmax': 0.7, 'normalize': True})
[0.0, 0.2, 0.4, 0.6, 0.7, 0.7]
```

You can test the function by inserting your solution into the file `cp/ex06/truncate.py`.

`cp.ex06.truncate.truncate_values(float_list, settings)`

Return a truncated list of floats given the initial list and the settings for truncating. If normalize is True, the values are first normalized to the [0,1] range and then truncated.

Parameters:

- **float_list** (`list`) – list of floats
- **settings** (`dict`) – Dictionary containing the keys vmin, vmax and normalize with their corresponding values.

Return type:

`list`

Returns:

Truncated/Normalized+Truncated values.

Exercise 6.7: Sentiment analysis

Exercise 6.7

Skip to main content

The goal of this exercise is to analyze the emotional tone of a sentence. There's a theory (or perhaps a belief) that the true emotional meaning of a sentence lies in the words used after the word `but`.

In simpler terms, the program should disregard all words preceding the word `but` and calculate the emotional content of the sentence. To compute this score, you are provided with the following `dict`, which links specific words to their corresponding sentiment scores:

```
score  = {'horrible': -5, 'awful': -5, 'bad': -4, 'terrible': -4, 'mediocre': -2, 'lousy': -3, 'poor': -3,
'unfair': -1, 'average': 0, 'wonderful': +2, 'beautiful': +3, 'good': +3,'fantastic': +4,
'excellent': +4, 'superb': +4, 'amazing': +4, 'great': +4, 'best': +5, 'brilliant': +5}
```

For example:

```
>>> sentiment_analysis('I think the food was excellent and great, but the service was horrible ')
-5
>>> sentiment_analysis('The weather forecast was horrible, but despite the bad news, our spirits remained high, knowing
1
```

In the second example, we can observe that the words `horrible`, `bad` and `best` appear within our dictionary. However only the last two (`bad` and `best`) appear after the word `but`. That so `bad` (-4) + `best` (+5), the total sentiment score of the sentence is 1.

The tests will not contain punctuation.

You can test the function by inserting your solution into the file `cp/ex06/sentimental_analysis.py`.

cp.ex06.sentiment_analysis.sentiment_analysis(*text*)

Return the sentence sentiment score, according to the rules of words scoring, as described in the text above.

Parameters:

text (`str`) – The sentence to check for sentiment scoring.

Return type:

`int`

Returns:

The total sentiment scoring of the sentence.

Exercise 6.8: Spell check

Important

Try to generate the token file for project 3. Ensure you get a total of **87 points**!

If the total is 85 points, you have a version of project 3 that is not up to date, and you should re-download and re-run the download script.

Now your task is to create a function `spell_check` which expects two inputs:

- A text containing numerous spelling mistakes.
- A dictionary that pairs misspelled words (keys) with their corrected versions (values), for example:

```
corrections = {
'apsolute': 'absolute',
'teh': 'the',
'acess': 'access',
 }
```

The goal of the `spell_check` function is to perform spell checking on the input text. Specifically, it should identify and replace words that are misspelled (keys in the dictionary) with their respective corrected versions (values from the dictionary) The code output will look like this:

```
>>> corrections = {'apsolute': 'absolute', 'teh': 'the'}
```

Note

The fo
Projec
link co
includ

Skip to main content

```
'I have an absolute understanding of this language.'
```

> **Note**
>
> The sentences used in these examples contain punctuation marks. This punctuation can cause a word to not be found in the spelling dictionary. To get full credit, your function should accurately reproduce the punctuation as well. The only permitted punctuation characters are the comma `,` and the period `.`, both of which can only be found at the end of a word.

You can test the function by inserting your solution into the file `cp/ex06/word_histogram.py`.

cp.ex06.spell_check.spell_check(*text*, *corrections*)

Return the corrected text for spelling errors according to a set of rules.

Parameters:

- **text** (`str`) – The sentence to check for spelling.
- **corrections** (`dict`) – The dictionary of wrongly spelled words and their equivalent corrected version.

Return type:

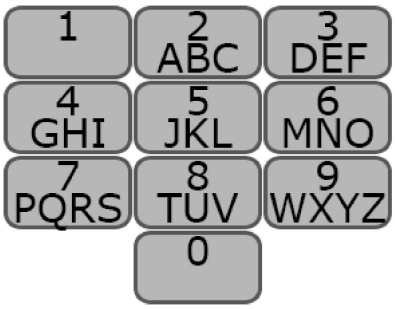
`str`

Returns:

The correctly spelled sentence.

Exercise 6.9: Multi-tap entry

Before the era of touchscreen smartphones, teenagers would master the art of composing SMS messages on a 3×4 numeric keyboard using a multi-tap entry method. The alphabet was printed under each numerical key (beginning with `2`), and the key needed to be pressed repeatedly to change between the letters. For example, pressing the key `3` twice would indicate the letter `E`. Pausing for a period of time automatically completed the current letter, as would pressing a different key.



Multi-tap keyboard

In this exercise, your task is to write a function that takes a sequence of numeric inputs and their corresponding timestamps and translates them into text. To clarify the input for your function, let's consider a concrete example using the word `PROGRAMMING`. To type this word, the user would tap the keys following this pattern: `7-pause-777-666-4-777-2-6-pause-6-444-66-4`. There are a few important things to note:

- A pause is needed between entering the letter `P` and the letter `R`, because both letters are on the same key `7`.
- However, no pause is needed when switching between keys.
- The input to your function consists of two lists.
  - A sequence of numeric inputs, for a word `PROGRAMMING` it would be a sequence `[7, 7, 7, 7, 6, 6, 6, 4, 7, 7, 7, 2, 6, 6, 4, 4, 4, 6, 6, 4]`.
  - The timestamps for each key-input in seconds.
- The pause is the event where two consecutive time stamps are more than 0.5 seconds apart.

So a time stamp list for the word `PROGRAMMING` could start with `[0.4, 1.5, 1.8, 2.0, 2.2, 2.6 ...]`

In practice, a key was used to cycle between letters, but in this exercise, you can assume that a letter is always written using the

Skip to main content

uppercase.

The following `dict` could be a useful starting point:

```
keypad_dict = {0: [' '],
    2: ['A', 'B', 'C'],
    3: ['D', 'E', 'F'],
    4: ['G', 'H', 'I'],
    5: ['J', 'K', 'L'],
    6: ['M', 'N', 'O'],
    7: ['P', 'Q', 'R', 'S'],
    8: ['T', 'U', 'V'],
    9: ['W', 'X', 'Y', 'Z']}
```

In the end a call of your function should look like this:

```
>>> keys = [7, 7, 7, 7, 6, 6, 6, 4, 7, 7, 7, 2, 6]
>>> times = [0.4, 1.5, 1.8, 2.0, 2.2, 2.6, 2.9, 3.0, 3.3, 3.6, 3.9, 4.2, 4.3]
>>> multi_tap(keys, times)
'PROGRAM'
```

You can test the function by inserting your solution into the file `cp/ex06/multi_tap.py`.

cp.ex06.multi_tap.multi_tap(*keys*, *times*)

Return the string corresponding to the multi-tap key presses.

Parameters:

- **keys** (`list`) – The list of keys pressed.
- **times** (`list`) – The list of times of when the keys were pressed.

Return type:

`str`

Returns:

The string corresponding to the key presses.