Algorithmic Methods of Data Mining

Homework 2

Implementation of a search engine for recipes

Work Group :

Garcia Maria Camila

Giannone Giorgio

Sessa Vittoria

Professors:

Aris Anagnostopoulos

Ioannis Chatzigiannakis

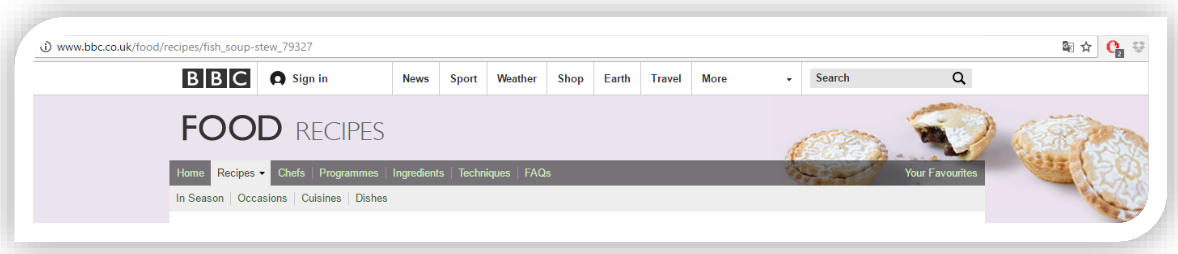
Università degli Studi di Roma “Sapienza”

Data Science Master Degree

2016/2017

Index:

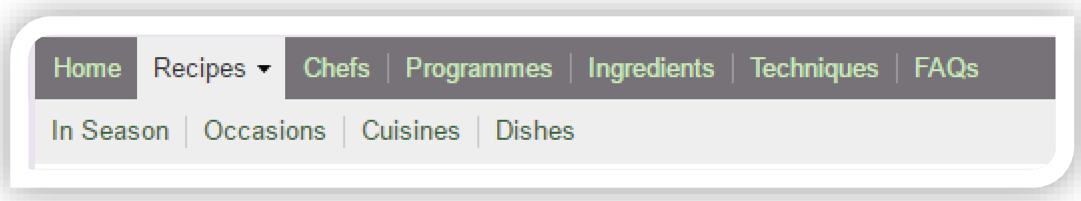
1.



1.1. http://www.bbc.co.uk/food/ is a web site with more than 10.000 recipes, each of them has the following important information:

- Title
- Author
- Preparation time
- Cooking time
- Number of people it serves
- Dietary
- Ingredients
- Method

To download the data we had to pay particular attention to the structure of the site and to the classification of the recipes disposed in different groups.

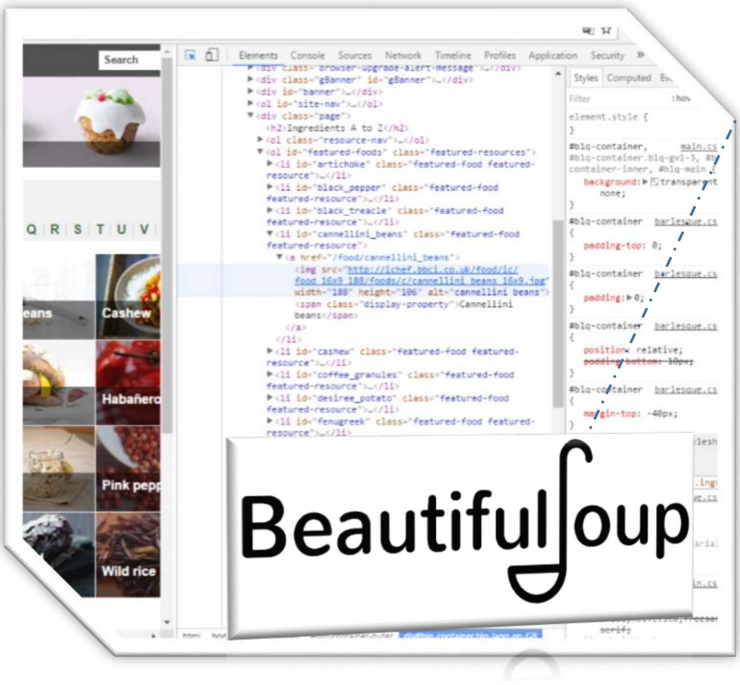


We had to decide the way in which we could gather the most amount of information, being sure of taking all the recipes(without exceptions), but in a second time we realized that the recipes are uploaded continually.

We thought about 3 different possible way in which we could reach our aim:

- Exploring the sub-labels of the main label Recipe: in season, occasions, cuisines and dishes, but this solution never reached the total amount of recipes in the website.
- Taking really basic ingredients like sugar, salt, oil, butter and watching how many recipes we could gather with just 4 requests. This method gave us more than 8.000 recipes but we didn't know how to get the ones we missed. This approach was useless if anyway at the end we have to do a search through the ingredients in alphabetical order.
- Exploring through the list of ingredients in alphabetical order, and taking all the recipes containing that particular ingredient.

The “developer tool” of chrome and the package bs4 have been really important and useful during the working progress, in fact with the first one we understood from which tag we could get a particular information from the web-site, with the second one we wrote some functions to download the data more easily and fast.



```
▶<div class="browser-upgrade-alert-message">…</div>
▶<div class="gBanner" id="gBanner">…</div>
▶<div id="banner">…</div>
▶<ol id="site-nav">…</ol>
▼<div class="page">
   <h2>Ingredients A to Z</h2>
  ▶<ol class="resource-nav">…</ol>
  ▼<ol id="featured-foods" class="featured-resources">
    ▶<li id="artichoke" class="featured-food featured-
    resource">…</li>
    ▶<li id="black_pepper" class="featured-food
    featured-resource">…</li>
    ▶<li id="black_treacle" class="featured-food
    featured-resource">…</li>
    ▼<li id="cannellini_beans" class="featured-food
    featured-resource">
      ▼<a href="/food/cannellini_beans">
         <img src="http://ichef.bbci.co.uk/food/ic/
         food_16x9_188/foods/c/cannellini_beans_16x9.jpg"
         width="188" height="106" alt="cannellini beans">
         <span class="display-property">Cannellini
         beans</span>
       </a>
     </li>
    ▶<li id="cashew" class="featured-food featured-
    resource">…</li>
    ▶<li id="coffee_granules" class="featured-food
    featured-resource">…</li>
    ▶<li id="desiree_potato" class="featured-food
    featured-resource">…</li>
    ▶<li id="fenugreek" class="featured-food featured-
    resource">…</li>
```

```
element.style {
}
#blq-container,                main.cs
#blq-container.blq-gvl-3, #b
container-inner, #blq-main {
   background:▶ transparent
      none;
}
#blq-container   barlesque.cs
{
   padding-top: 0;
}
#blq-container   barlesque.cs
{
   padding:▶0;
}
#blq-container   barlesque.cs
{
   position: relative;
   ~~padding-bottom: 10px;~~
}
#blq-container   barlesque.cs
{
   margin-top: -40px;
}
```

requests.get()

BeautifulSoup()

find_all()

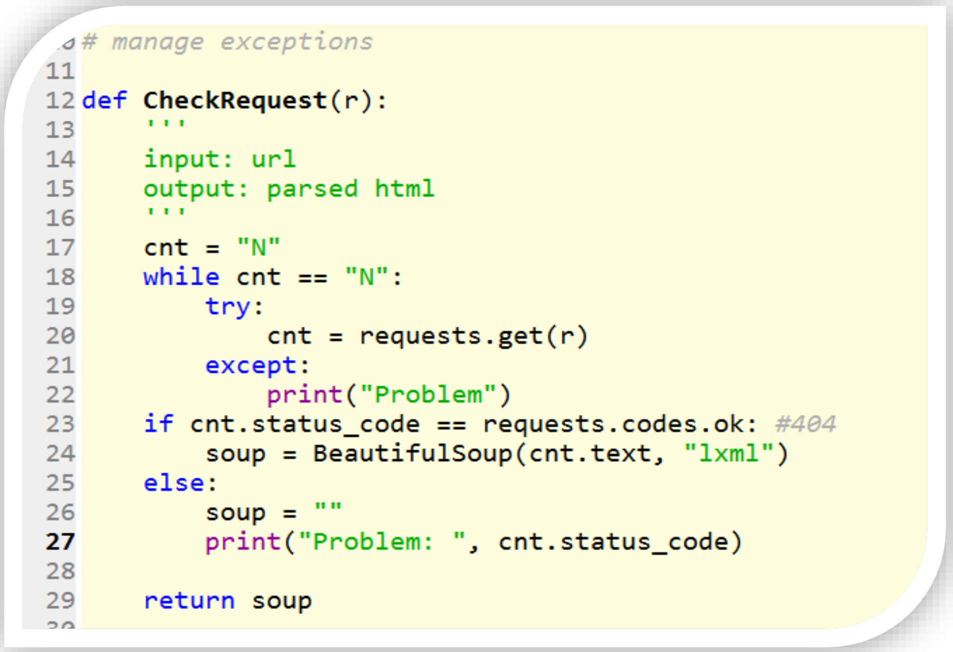
startswith()

1.2.

The python library written for download the recipes contains the following functions:

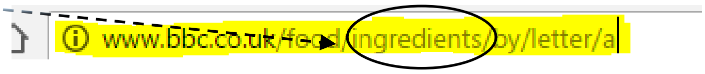
- CheckRequest
- foodsForCharacter
- MultiPage
- ExtractSearch
- RecipesTotal
- RecipesLink
- Link

CheckRequest takes as input an url and do some controls considering the status code of an htlm parser request, at the end it returns an html page.



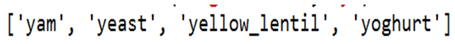
```python
# manage exceptions

def CheckRequest(r):
    '''
    input: url
    output: parsed html
    '''
    cnt = "N"
    while cnt == "N":
        try:
            cnt = requests.get(r)
        except:
            print("Problem")
    if cnt.status_code == requests.codes.ok: #404
        soup = BeautifulSoup(cnt.text, "lxml")
    else:
        soup = ""
        print("Problem: ", cnt.status_code)

    return soup
```

foods_for_character takes as input a word to put in an URL and a character which indicate the first character of each element of a list of ingredients.

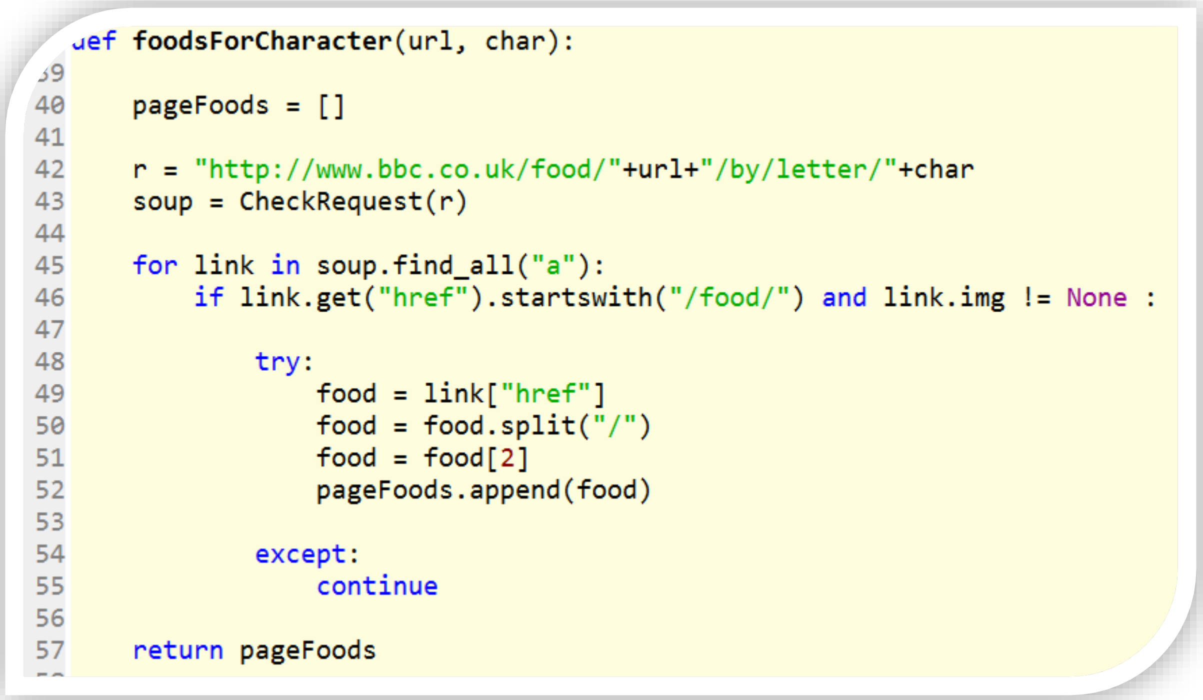


ⓘ www.bbc.co.uk/food/ingredients/by/letter/a

It returns a list of ingredients with the same first character. e.g. :



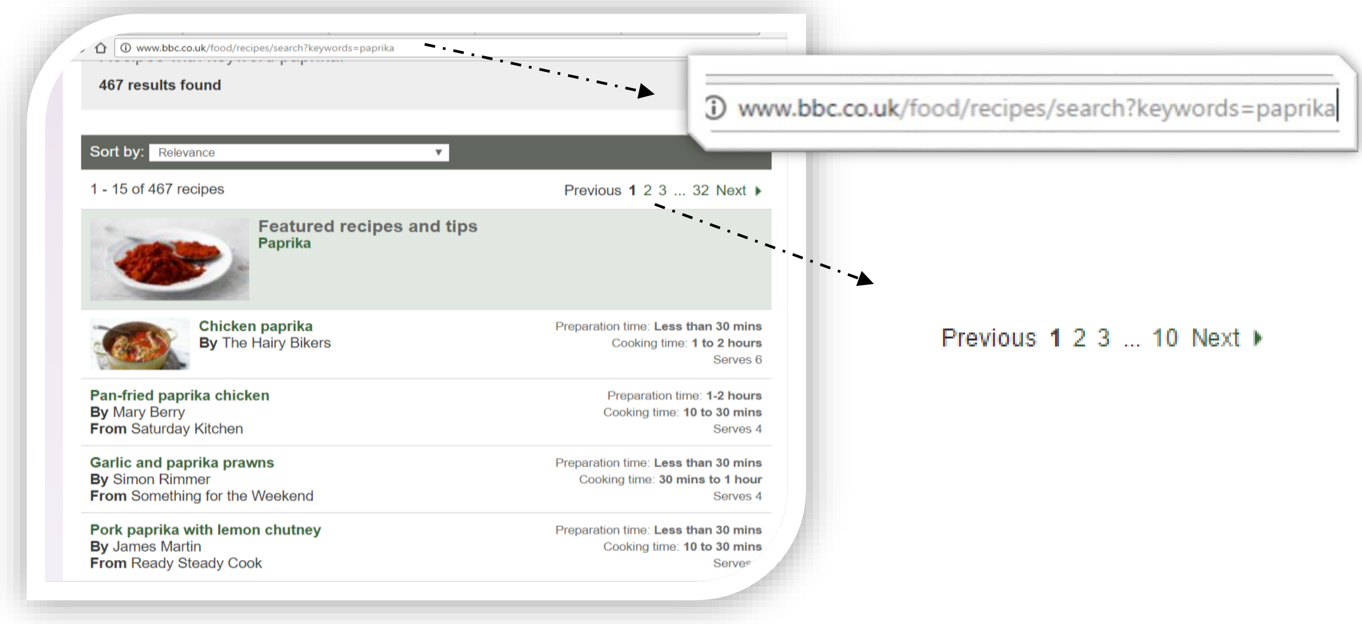
```
['yam', 'yeast', 'yellow_lentil', 'yoghurt']
```



```python
def foodsForCharacter(url, char):

    pageFoods = []

    r = "http://www.bbc.co.uk/food/"+url+"/by/letter/"+char
    soup = CheckRequest(r)

    for link in soup.find_all("a"):
        if link.get("href").startswith("/food/") and link.img != None :

            try:
                food = link["href"]
                food = food.split("/")
                food = food[2]
                pageFoods.append(food)

            except:
                continue

    return pageFoods
```

The two condition in line 46 mean that if a link in the page starts with “/food/” and if there’s an image in the page of the current link then take the portion of the url with the ingredient and add it to **pageFoods** vector.

Since we decided to consider all the ingredients grouped alphabetically, we had to handle the ingredient’s pages in which there were a lot of recipes divided in some pages. We created the next function to manage this issue.



www.bbc.co.uk/food/recipes/search?keywords=paprika

467 results found

Sort by: Relevance

1 - 15 of 467 recipes

Previous 1 2 3 ... 32 Next

Featured recipes and tips
Paprika

Chicken paprika
By The Hairy Bikers
Preparation time: Less than 30 mins
Cooking time: 1 to 2 hours
Serves 6

Pan-fried paprika chicken
By Mary Berry
From Saturday Kitchen
Preparation time: 1-2 hours
Cooking time: 10 to 30 mins
Serves 4

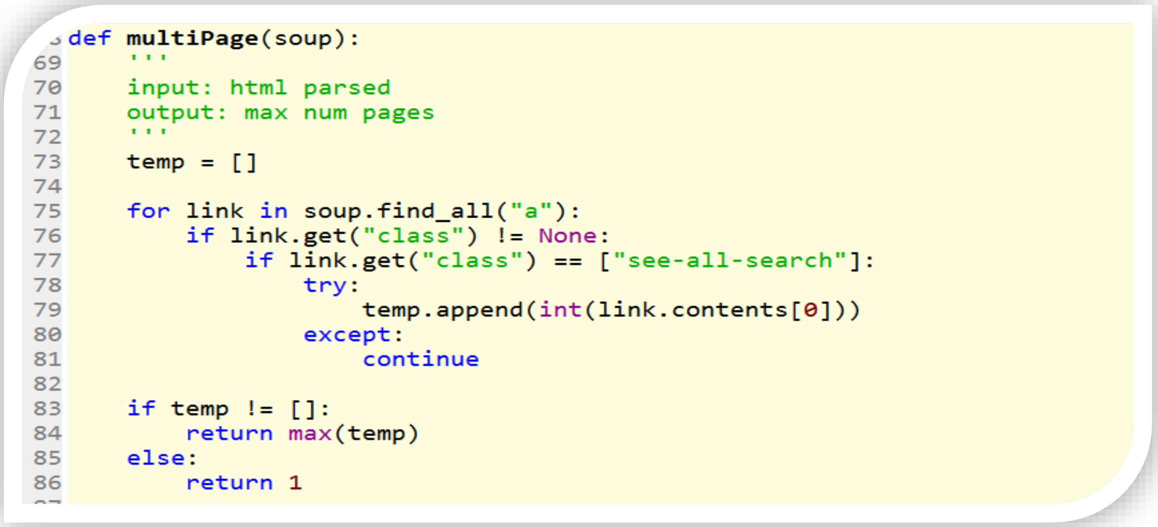
Garlic and paprika prawns
By Simon Rimmer
From Something for the Weekend
Preparation time: Less than 30 mins
Cooking time: 30 mins to 1 hour
Serves 4

Pork paprika with lemon chutney
By James Martin
From Ready Steady Cook
Preparation time: Less than 30 mins
Cooking time: 10 to 30 mins
Serves

www.bbc.co.uk/food/recipes/search?keywords=paprika

Previous 1 2 3 ... 10 Next

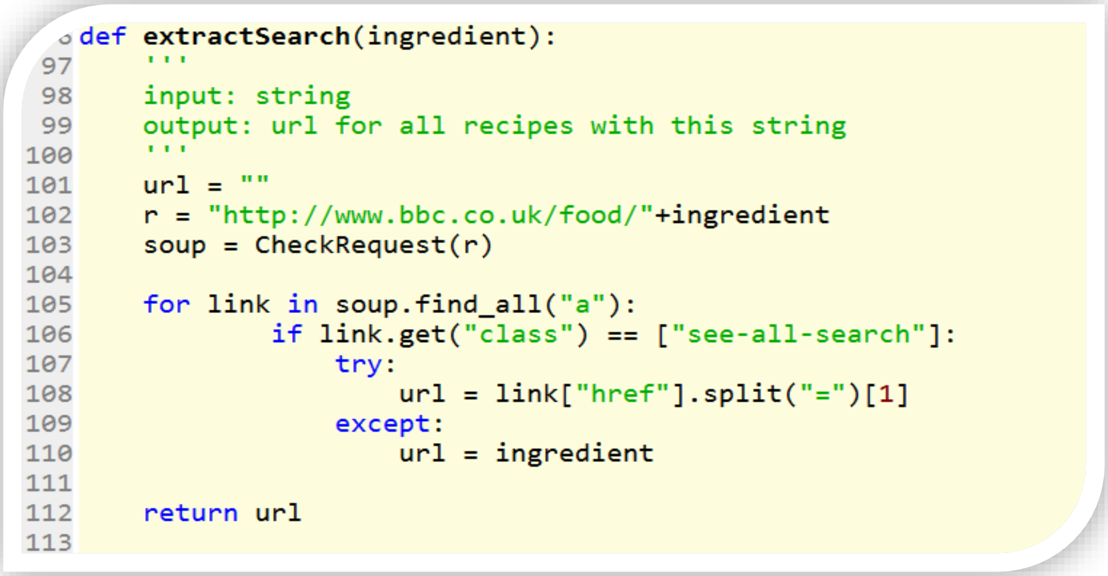
MultiPage takes as input an html parser with an url like this:



```python
def multiPage(soup):
    '''
    input: html parsed
    output: max num pages
    '''
    temp = []

    for link in soup.find_all("a"):
        if link.get("class") != None:
            if link.get("class") == ["see-all-search"]:
                try:
                    temp.append(int(link.contents[0]))
                except:
                    continue

    if temp != []:
        return max(temp)
    else:
        return 1
```

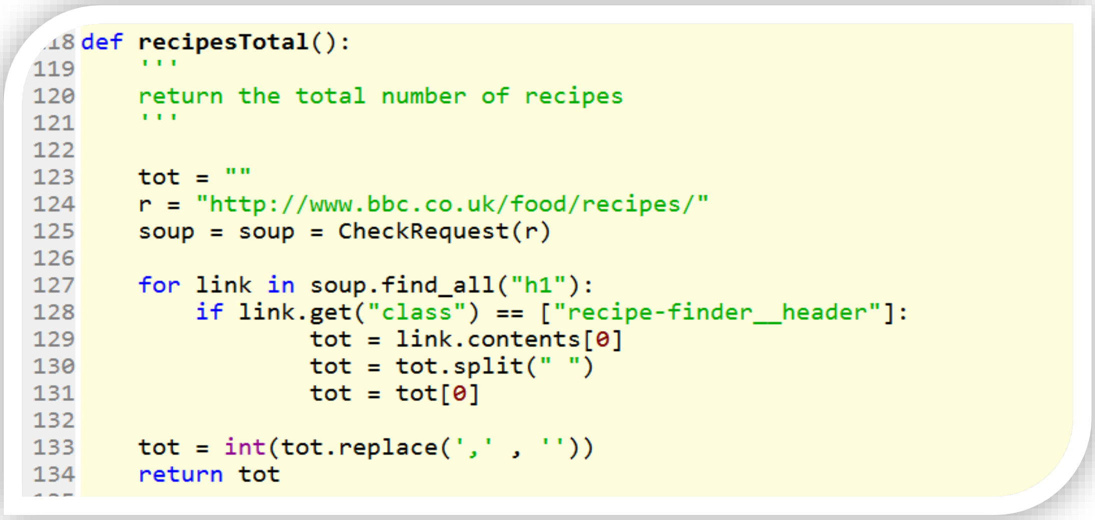
The element soup should be the result of the function BeautfiulSoup to be accepted by this function. **MulitPage** returns total number of pages containing recipes with a specific ingredient.

ExtractSearch extract the search page for a particular ingredient



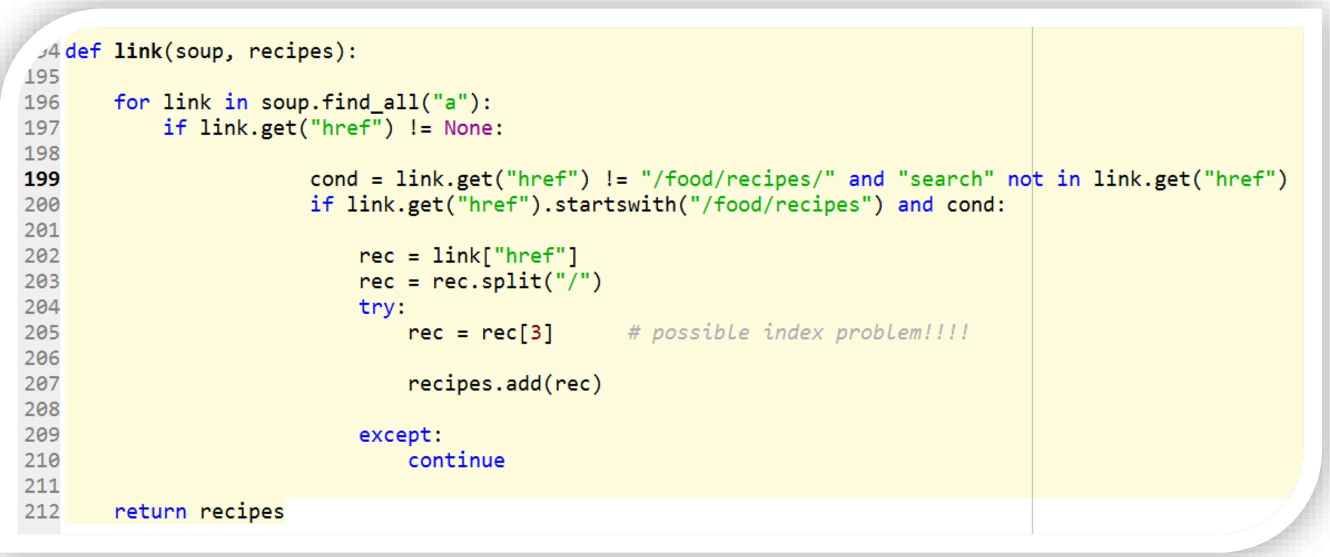
```python
def extractSearch(ingredient):
    '''
    input: string
    output: url for all recipes with this string
    '''
    url = ""
    r = "http://www.bbc.co.uk/food/"+ingredient
    soup = CheckRequest(r)

    for link in soup.find_all("a"):
            if link.get("class") == ["see-all-search"]:
                try:
                    url = link["href"].split("=")[1]
                except:
                    url = ingredient

    return url
```

RecipesTotal functions helped us to understand how many recipes we had to download and it'll be used in **recipesLink** function.



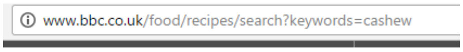
```python
def recipesTotal():
    '''
    return the total number of recipes
    '''

    tot = ""
    r = "http://www.bbc.co.uk/food/recipes/"
    soup = soup = CheckRequest(r)

    for link in soup.find_all("h1"):
        if link.get("class") == ["recipe-finder__header"]:
                tot = link.contents[0]
                tot = tot.split(" ")
                tot = tot[0]

    tot = int(tot.replace(',' , ''))
    return tot
```

Link function takes ad input an html parser given by a call of the function BeautifuSoup and a set composed by recipe titles, it returns recipes set with some added recipes titles taken from the html page taken as input.

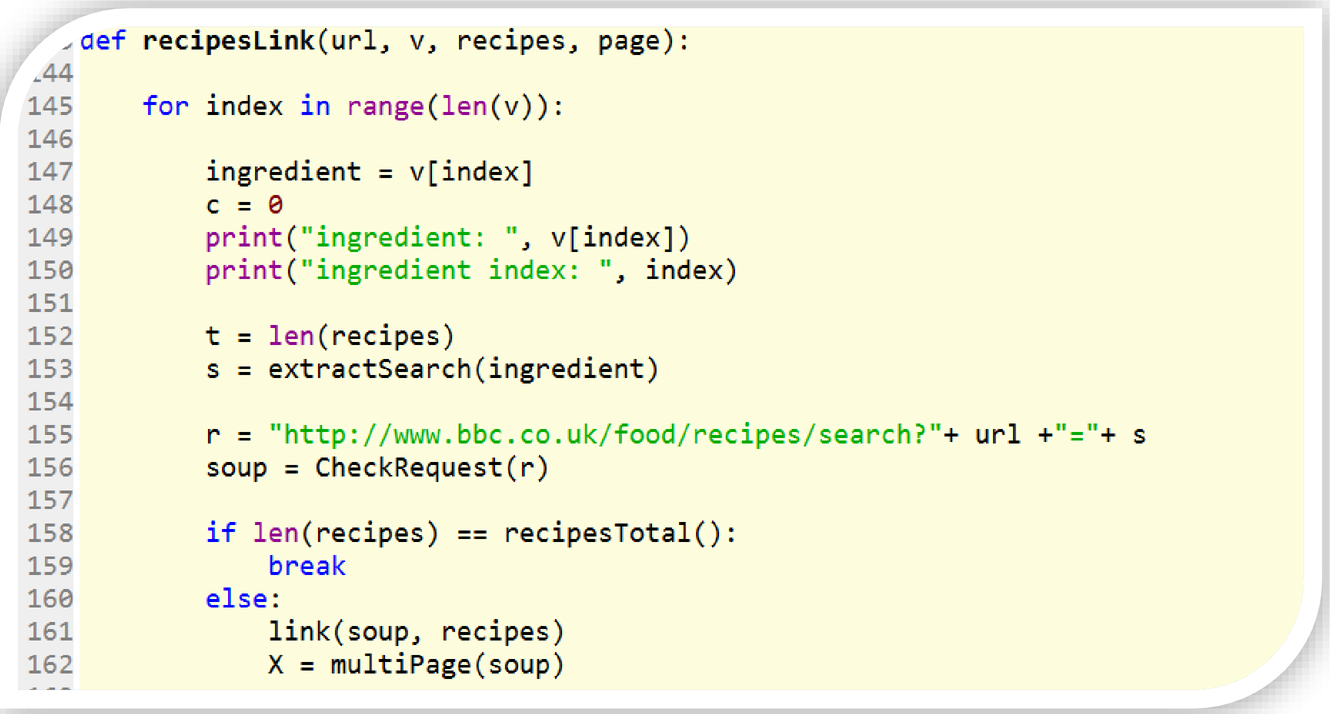


```
def link(soup, recipes):

    for link in soup.find_all("a"):
        if link.get("href") != None:

                    cond = link.get("href") != "/food/recipes/" and "search" not in link.get("href")
                    if link.get("href").startswith("/food/recipes") and cond:

                        rec = link["href"]
                        rec = rec.split("/")
                        try:
                            rec = rec[3]      # possible index problem!!!!

                            recipes.add(rec)

                        except:
                            continue

    return recipes
```

The last function of **lib1** is recipesLink, in fact it takes as input a word to rebuild an url, a vector of ingredients, a set of recipe titles and the number of the page from which we want to take recipes' links. This function works with an html page with the following url

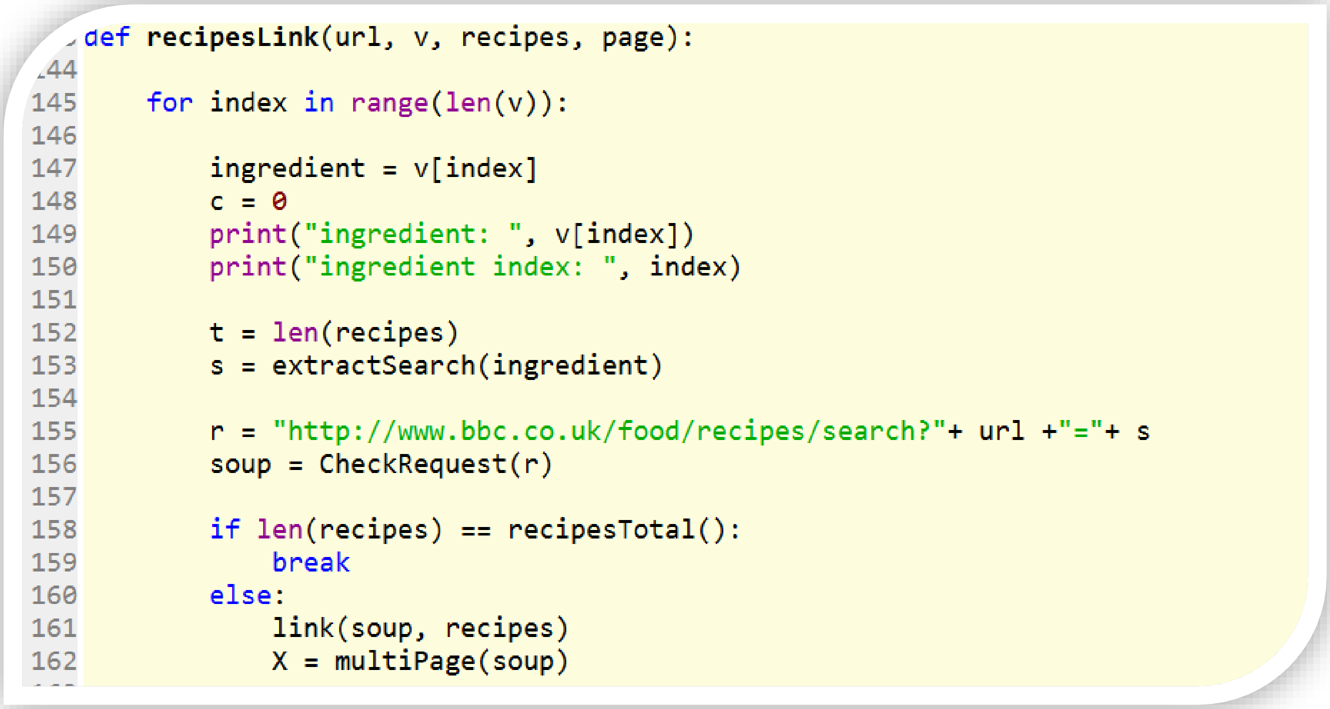


ⓘ www.bbc.co.uk/food/recipes/search?keywords=cashew



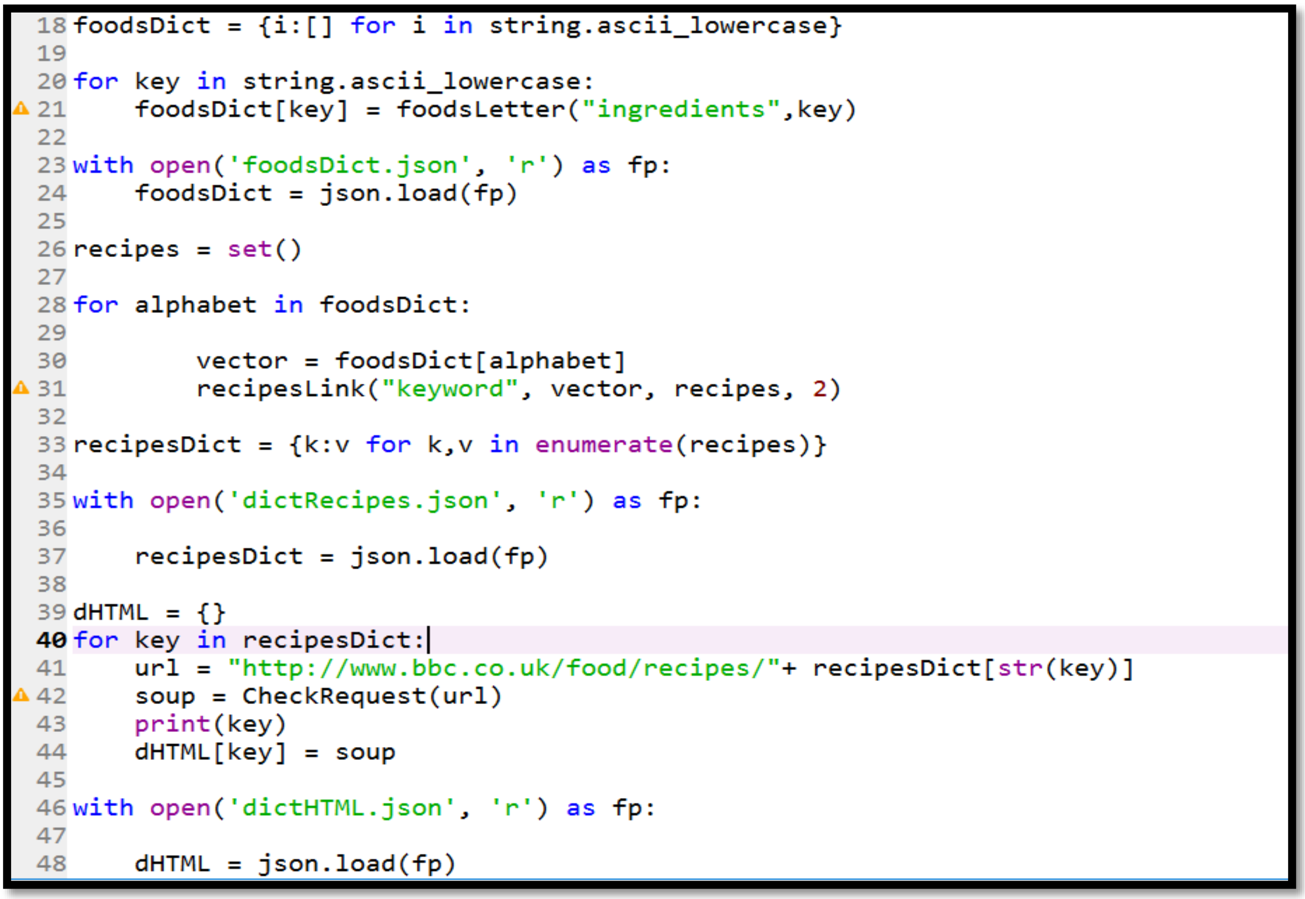
```
def recipesLink(url, v, recipes, page):

    for index in range(len(v)):

        ingredient = v[index]
        c = 0
        print("ingredient: ", v[index])
        print("ingredient index: ", index)

        t = len(recipes)
        s = extractSearch(ingredient)

        r = "http://www.bbc.co.uk/food/recipes/search?"+ url +"="+ s
        soup = CheckRequest(r)

        if len(recipes) == recipesTotal():
            break
        else:
            link(soup, recipes)
            X = multiPage(soup)
```

The second part of the function can handle an important issue, if the function can't finish because of a connection problem, changing the page number we can recall it again and continue filling the set from the point it's been interrupted.



```
def recipesLink(url, v, recipes, page):

    for index in range(len(v)):

        ingredient = v[index]
        c = 0
        print("ingredient: ", v[index])
        print("ingredient index: ", index)

        t = len(recipes)
        s = extractSearch(ingredient)

        r = "http://www.bbc.co.uk/food/recipes/search?"+ url +"="+ s
        soup = CheckRequest(r)

        if len(recipes) == recipesTotal():
            break
        else:
            link(soup, recipes)
            X = multiPage(soup)
```

1.3. The main script (**script_part1.py**) to download all the web-pages and to create some convenient data structure to store the html pages.



```
foodsDict = {i:[] for i in string.ascii_lowercase}

for key in string.ascii_lowercase:
    foodsDict[key] = foodsLetter("ingredients",key)

with open('foodsDict.json', 'r') as fp:
    foodsDict = json.load(fp)

recipes = set()

for alphabet in foodsDict:

        vector = foodsDict[alphabet]
        recipesLink("keyword", vector, recipes, 2)

recipesDict = {k:v for k,v in enumerate(recipes)}

with open('dictRecipes.json', 'r') as fp:

    recipesDict = json.load(fp)

dHTML = {}
for key in recipesDict:
    url = "http://www.bbc.co.uk/food/recipes/"+ recipesDict[str(key)]
    soup = CheckRequest(url)
    print(key)
    dHTML[key] = soup

with open('dictHTML.json', 'r') as fp:

    dHTML = json.load(fp)
```

First of all, we created a dictionary where we stored as key each letter of the alphabet and as values the whole set of ingredients starting with this letter. All the data structures created in this file have been stored in a json format file.

Then we have built a dictionary of recipes titles each of them with a numerical keys. Finally we created another dictionary with the same numerical key of the last one, but this time with the whole html parserd of each recipe.

So we used 4 different data structures, 3 dictionaries and one set(a set of the recipes because we didn’t want repetitions).

At last we have saved the following json formats with the requested information:

- **foodsDict.json**
- **dictRecipes.json**
- **dictHTML.json**

At the beginning we had thought of a more complex structure to store the information of each category, to be more explicit: we wanted to create a dictionary, taking the ingredient as key and a set of recipes as value, all those information would have been stored inside an another dictionary with alphabet letters as key and the list of dictionaries mentioned before.

```
{a: [ {ingredient1:set(recipe1of1, recipe2of1, ....., recipenof1)},
      {ingredient2:set(recipe1of2, recipe2of2,..., recipenof2)} .....],
 b: [ {ingredient1:set(recipe1of1, recipe2of1, ....., recipenof1)},
      {ingredient2:set(recipe1of2, recipe2of2,..., recipenof2)}.....],
 ...
z: [ {ingredient1:set(recipe1of1, recipe2of1, ....., recipenof1)},
     {ingredient2:set(recipe1of2, recipe2of2,...., recipenof2)}.....] }
```

We abandoned this idea because it would have been hard take each single recipe inside a so complicated structure. So we opted for the following:

foodsDict is a file containing the dictionary of ingredients as values:

```
{a:[ acidulated water, ackee,……],b:[bacon,bagel,…],..,z :[zander, zest]}
```

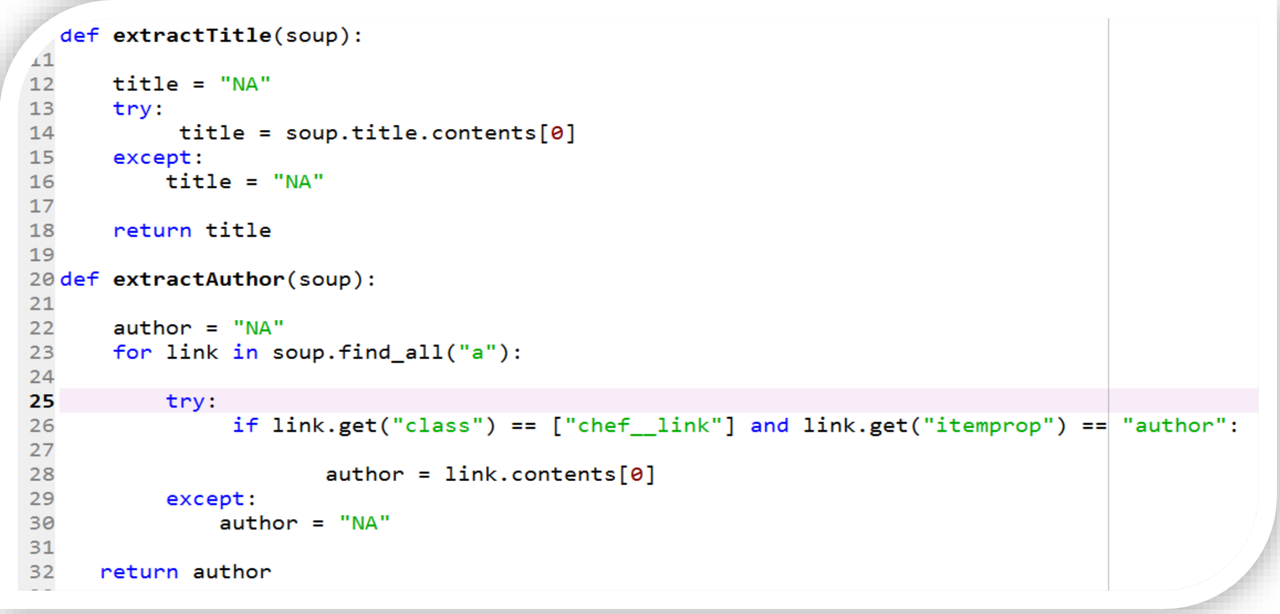
RecipesDict contains a dictionary done in this way:

```
{0:‘tamarind_seared_tuna_97406’, 1:‘wildrabbitandmorelst_92455’ ,....}
```

DictHTML has the same key of the last dictionary and contains the html scripts of each recipes title in **RecipesDict**.

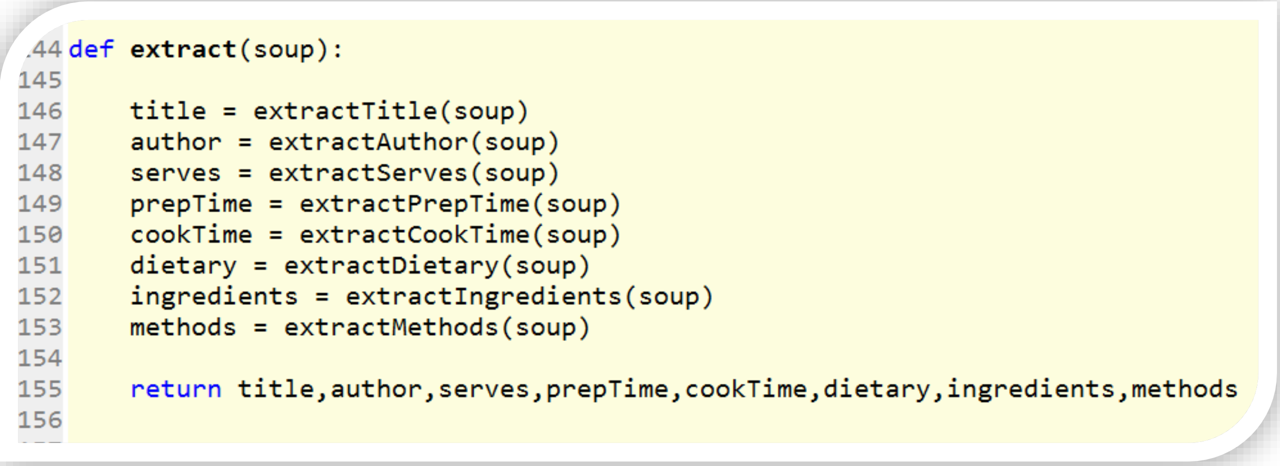
2.1.

Once we stored all the web pages in a Jason file, we write some simple functions (**lib3.py**) based on BeautifulSoup libraries. They are really similar to each other, so we decided to attach just two of them, and the principle one recalls all of the previous functions.



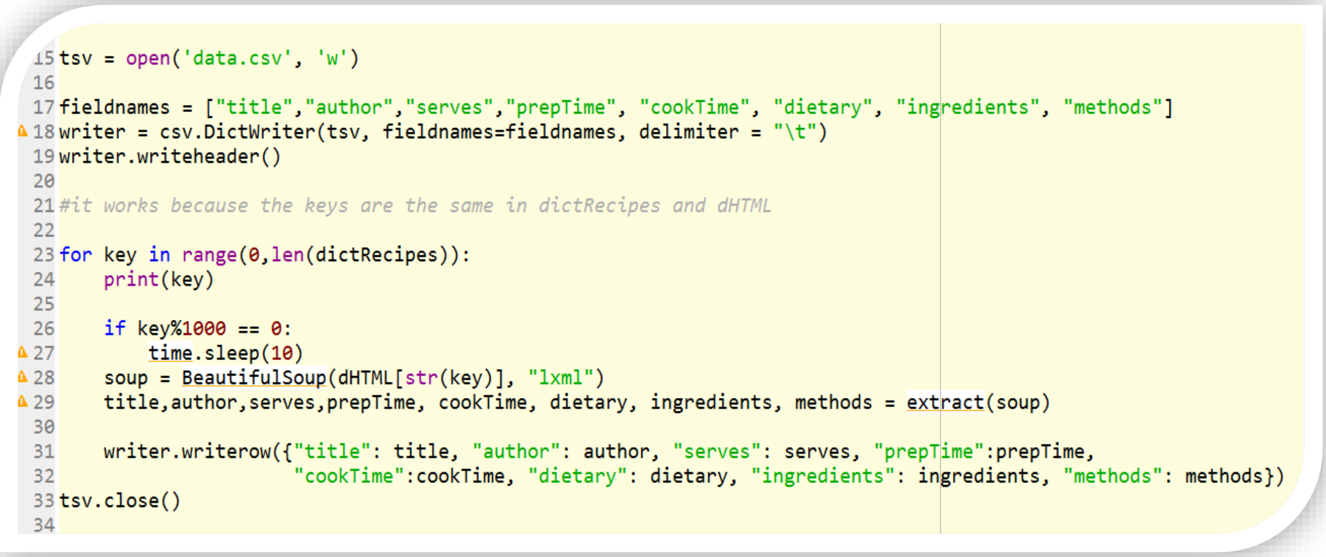
```
def extractTitle(soup):

    title = "NA"
    try:
         title = soup.title.contents[0]
    except:
        title = "NA"

    return title

def extractAuthor(soup):

    author = "NA"
    for link in soup.find_all("a"):

        try:
             if link.get("class") == ["chef__link"] and link.get("itemprop") == "author":

                    author = link.contents[0]
        except:
            author = "NA"

   return author
```

We wrote a function to extract each portion of information inside the web page and as the other functions , it takes as input an html page, then we called them all in the following function named **extract**.



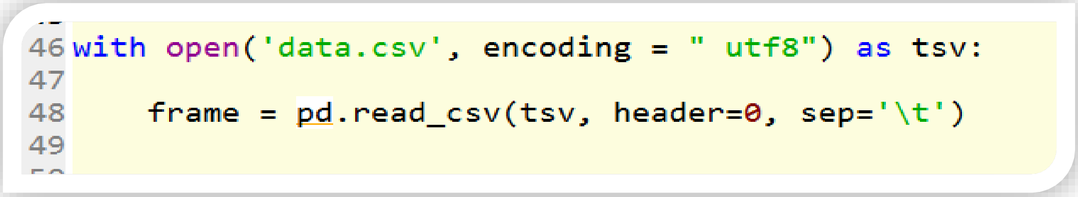
```
def extract(soup):

    title = extractTitle(soup)
    author = extractAuthor(soup)
    serves = extractServes(soup)
    prepTime = extractPrepTime(soup)
    cookTime = extractCookTime(soup)
    dietary = extractDietary(soup)
    ingredients = extractIngredients(soup)
    methods = extractMethods(soup)

    return title,author,serves,prepTime,cookTime,dietary,ingredients,methods
```

After opening ‘dictHTML.json and dictRecipes.json and initializing then as variables called Dhtml and dictRecipes respectively . We used some functions Lib/csv.py documentation online. With the following portion of script(script_part2.py) we structured all the recipes’ information in a sequence of lists.

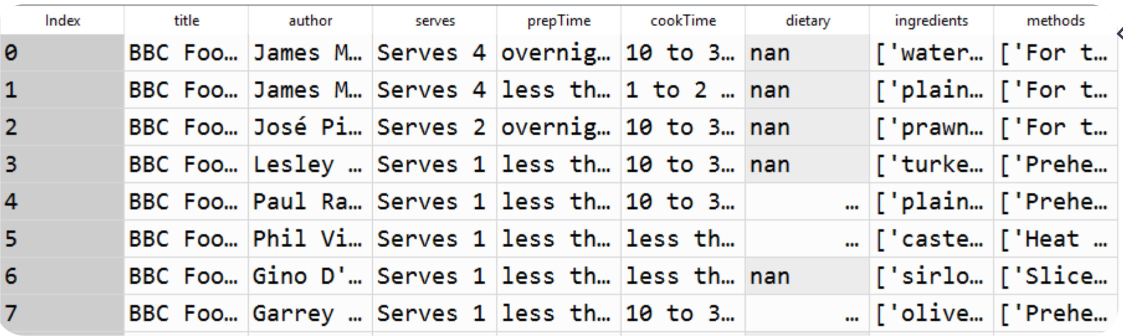


```python
tsv = open('data.csv', 'w')

fieldnames = ["title","author","serves","prepTime", "cookTime", "dietary", "ingredients", "methods"]
writer = csv.DictWriter(tsv, fieldnames=fieldnames, delimiter = "\t")
writer.writeheader()

#it works because the keys are the same in dictRecipes and dHTML

for key in range(0,len(dictRecipes)):
    print(key)

    if key%1000 == 0:
        time.sleep(10)
    soup = BeautifulSoup(dHTML[str(key)], "lxml")
    title,author,serves,prepTime, cookTime, dietary, ingredients, methods = extract(soup)

    writer.writerow({"title": title, "author": author, "serves": serves, "prepTime":prepTime,
                     "cookTime":cookTime, "dietary": dietary, "ingredients": ingredients, "methods": methods})
tsv.close()

```

Then we converted the Comma Separated Values into a panda’s data frame



```python
with open('data.csv', encoding = " utf8") as tsv:

    frame = pd.read_csv(tsv, header=0, sep='\t')

```



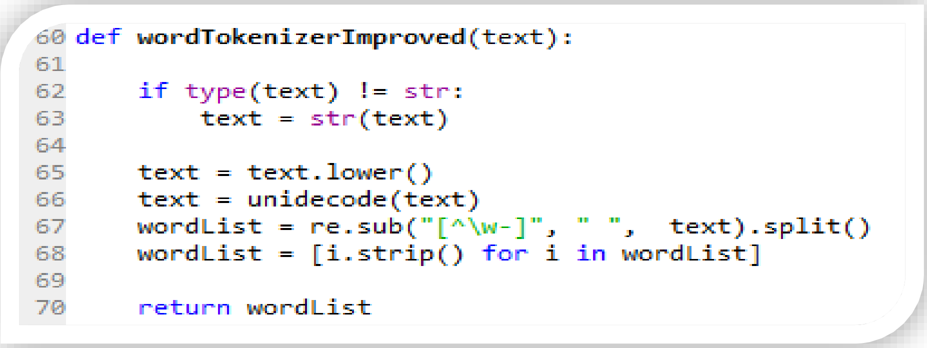
Index	title	author	serves	prepTime	cookTime	dietary	ingredients	methods
0	BBC Foo…	James M…	Serves 4	overnig…	10 to 3…	nan	['water…	['For t…
1	BBC Foo…	James M…	Serves 4	less th…	1 to 2 …	nan	['plain…	['For t…
2	BBC Foo…	José Pi…	Serves 2	overnig…	10 to 3…	nan	['prawn…	['For t…
3	BBC Foo…	Lesley …	Serves 1	less th…	10 to 3…	nan	['turke…	['Prehe…
4	BBC Foo…	Paul Ra…	Serves 1	less th…	10 to 3…	…	['plain…	['Prehe…
5	BBC Foo…	Phil Vi…	Serves 1	less th…	less th…	…	['caste…	['Heat …
6	BBC Foo…	Gino D'…	Serves 1	less th…	less th…	nan	['sirlo…	['Slice…
7	BBC Foo…	Garrey …	Serves 1	less th…	10 to 3…	…	['olive…	['Prehe…

2.2.

The lib3.py functions are used for the preprocessing and are the following:

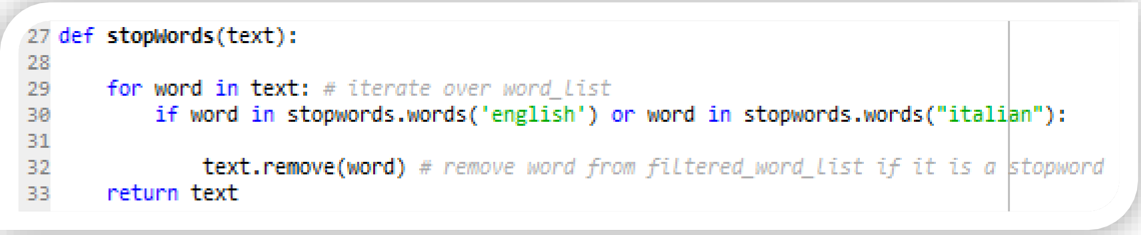
- wordTokenizer(text)
- stopWords(text)
- stemmerLancaster(text)
- Lemmatize(text)
- preprocess(text)

The **wordTokenizerImproved** function, after transforming unicode characters in ASCII, substitutes all characters of *text*, which are not alfanumeric or "-", with a space, splits *text* in its composing words and saves them in a list, **wordList**, which the function returns.



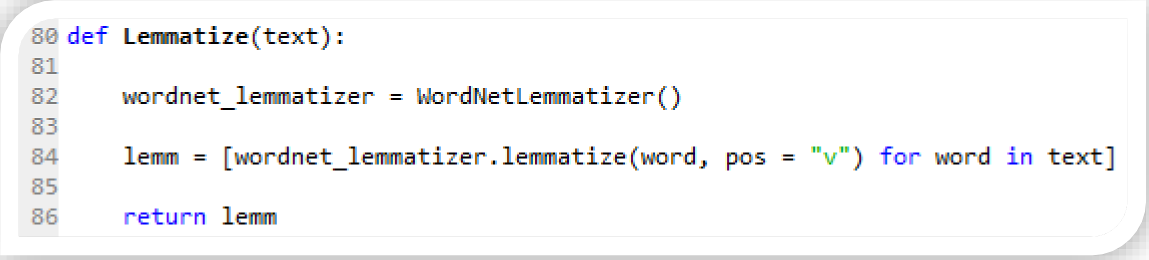
```python
def wordTokenizerImproved(text):

    if type(text) != str:
        text = str(text)

    text = text.lower()
    text = unidecode(text)
    wordList = re.sub("[^\w-]", " ",  text).split()
    wordList = [i.strip() for i in wordList]

    return wordList
```

The *stopWords* function eliminates the english and italian stopWords from the list *text*. This is done by comparing the list *text* with the **stop.words('english')** and **stop.words('italian')** lists from the ntlk library.



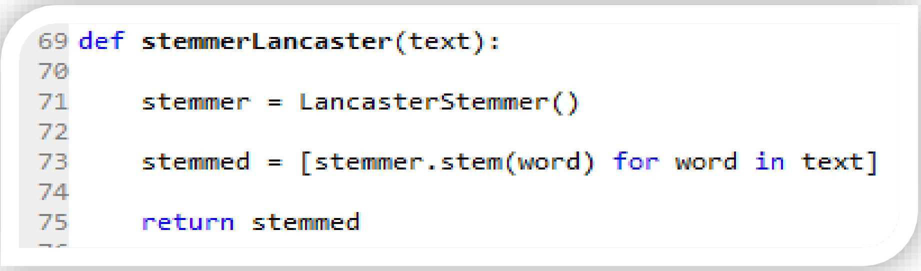
```python
def stopWords(text):

    for word in text: # iterate over word_list
        if word in stopwords.words('english') or word in stopwords.words("italian"):

            text.remove(word) # remove word from filtered_word_list if it is a stopword
    return text
```

The Lemmatize function lemmatizes the words contained in the list *text* using the **WordNetLemmatizer().lemmatize()** function from the nltk library and returns the lemmatized words in a list, *lemm*.



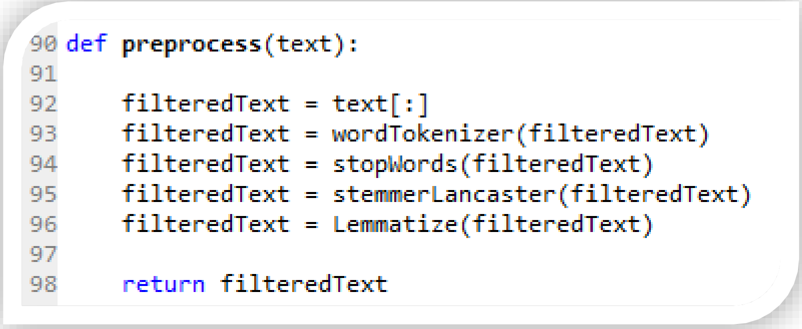
```
def Lemmatize(text):

    wordnet_lemmatizer = WordNetLemmatizer()

    lemm = [wordnet_lemmatizer.lemmatize(word, pos = "v") for word in text]

    return lemm
```

The stemmerLancaster function applies the **LancasterStemmer().stem()** function from the nltk library to the words of the list *text* and returns a list of stemmed words *stemmed*.



```
def stemmerLancaster(text):

    stemmer = LancasterStemmer()

    stemmed = [stemmer.stem(word) for word in text]

    return stemmed
```

The *preprocess* function preprocesses a string using the functions of **lib3.py** that we have just seen. The words composing the string are separated, preprocessed and saved in a list *filteredText* which is then returned by the function.

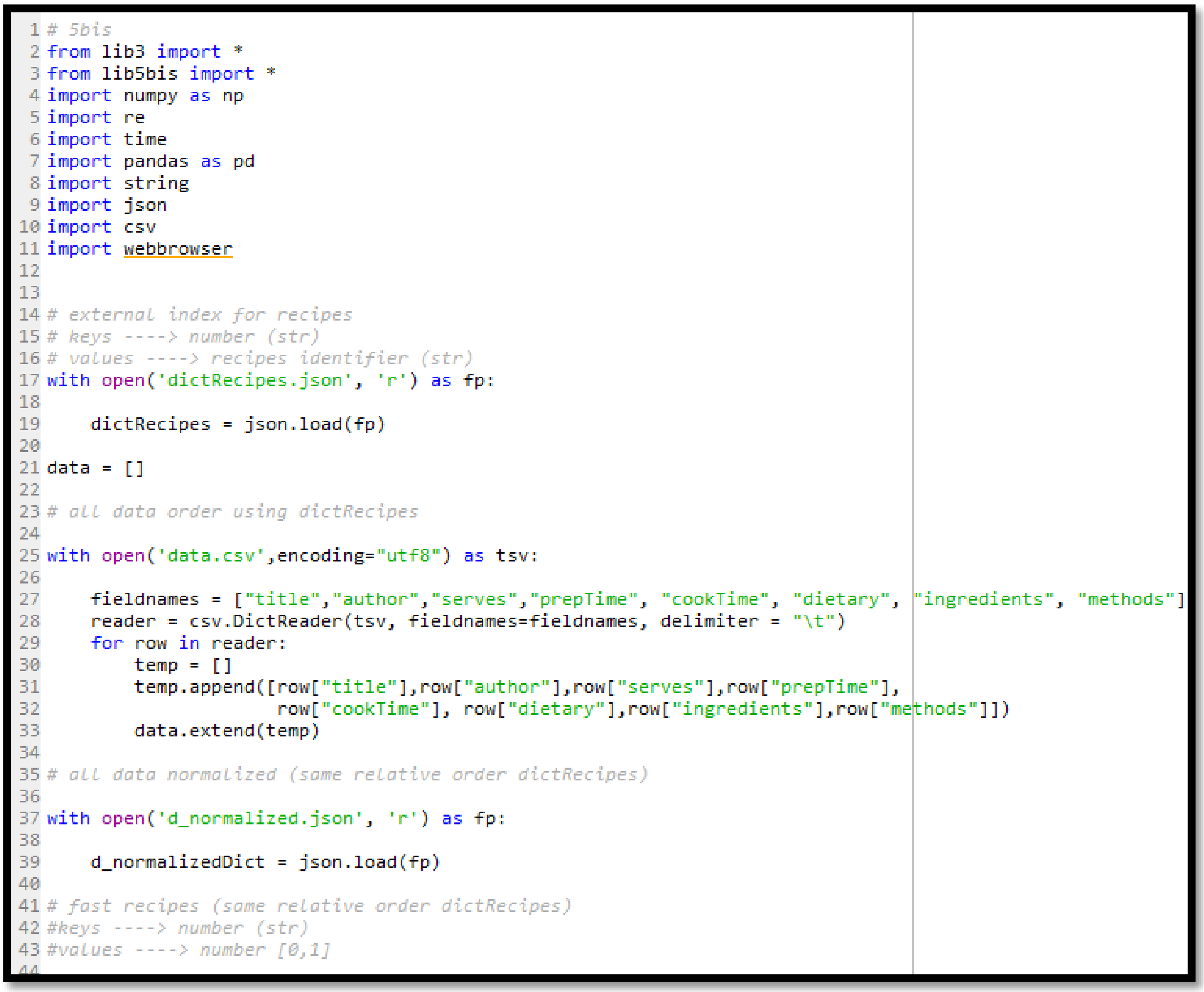


```
def preprocess(text):

    filteredText = text[:]
    filteredText = wordTokenizer(filteredText)
    filteredText = stopWords(filteredText)
    filteredText = stemmerLancaster(filteredText)
    filteredText = Lemmatize(filteredText)

    return filteredText
```

3.

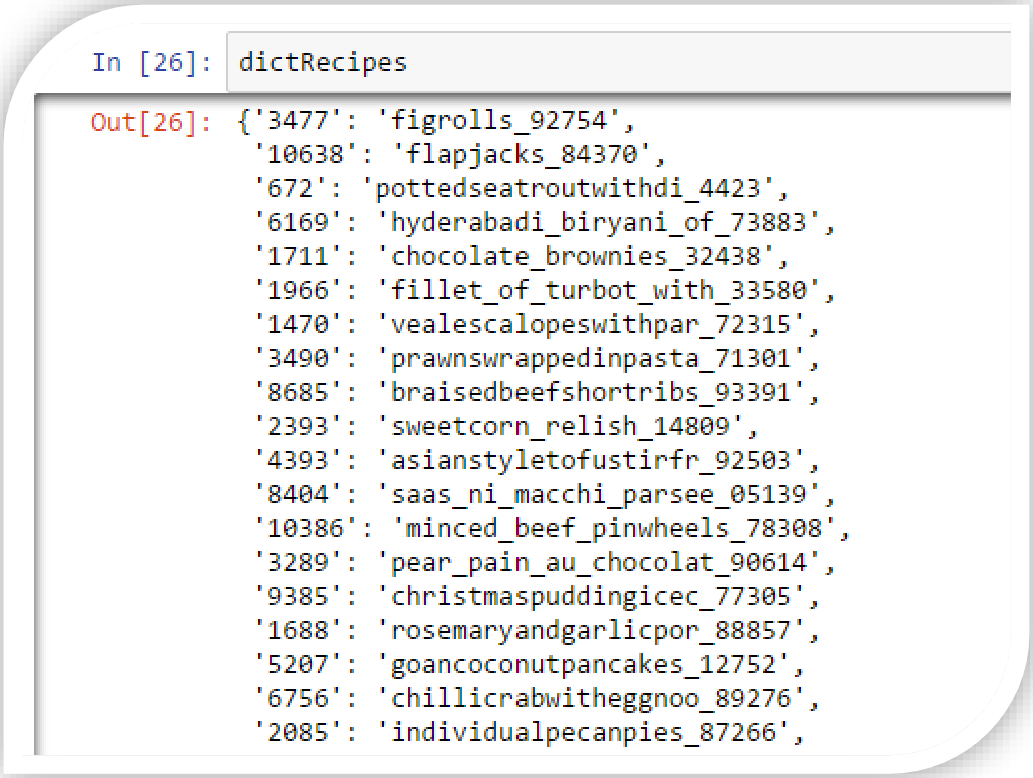
In this part we built a search-engine index. First we built an inverted index and stored it in a json file. Using the cosine similarity measure, the index allows to perform proximity queries.
For the query-processing part we used an algorithm with pointers to bring the most related recipes.

The script is composed of 3 parts, a first part where data is loaded from the csv into dictionaries and lists, a preprocessing part where we create new data structures that will be used in the search engine.



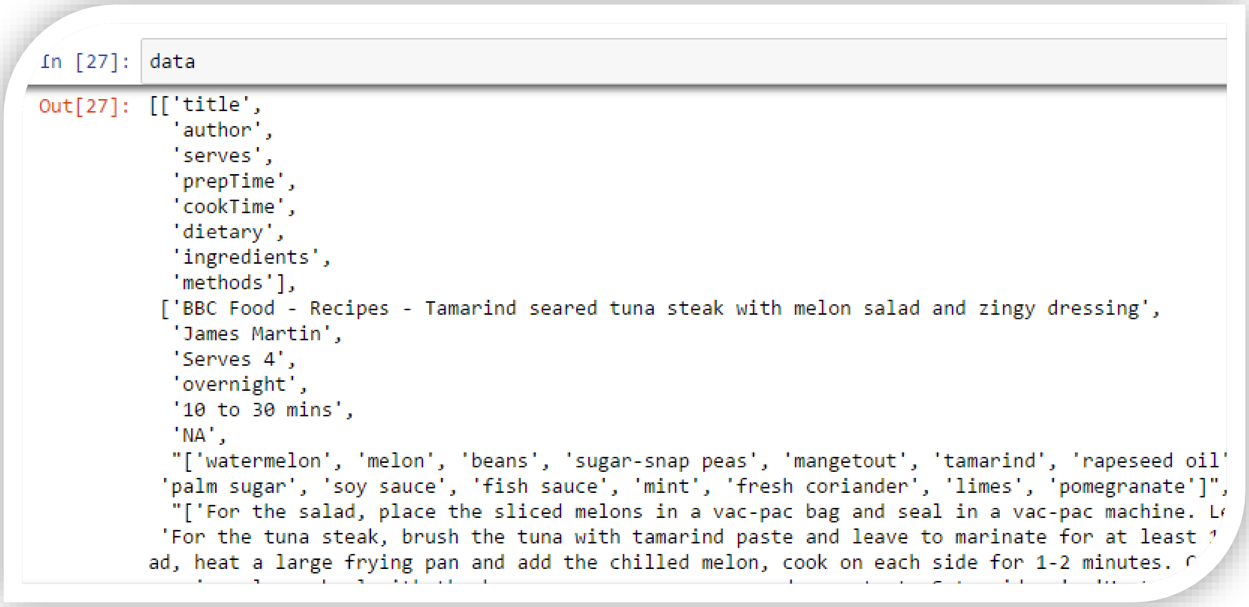
```
# 5bis
from lib3 import *
from lib5bis import *
import numpy as np
import re
import time
import pandas as pd
import string
import json
import csv
import webbrowser


# external index for recipes
# keys ----> number (str)
# values ----> recipes identifier (str)
with open('dictRecipes.json', 'r') as fp:

    dictRecipes = json.load(fp)

data = []

# all data order using dictRecipes

with open('data.csv',encoding="utf8") as tsv:

    fieldnames = ["title","author","serves","prepTime", "cookTime", "dietary", "ingredients", "methods"]
    reader = csv.DictReader(tsv, fieldnames=fieldnames, delimiter = "\t")
    for row in reader:
        temp = []
        temp.append([row["title"],row["author"],row["serves"],row["prepTime"],
                    row["cookTime"], row["dietary"],row["ingredients"],row["methods"]])
        data.extend(temp)

# all data normalized (same relative order dictRecipes)

with open('d_normalized.json', 'r') as fp:

    d_normalizedDict = json.load(fp)

# fast recipes (same relative order dictRecipes)
#keys ----> number (str)
#values ----> number [0,1]
```

The data in **dictRecipes.json** is saved into a dictionary called **dictRecipes**, in this dictionary the keys are the ids of the recipes and the values are the last pieces of the url, this dictionary, as we will see later, is needed to call the internet page of the recipe.



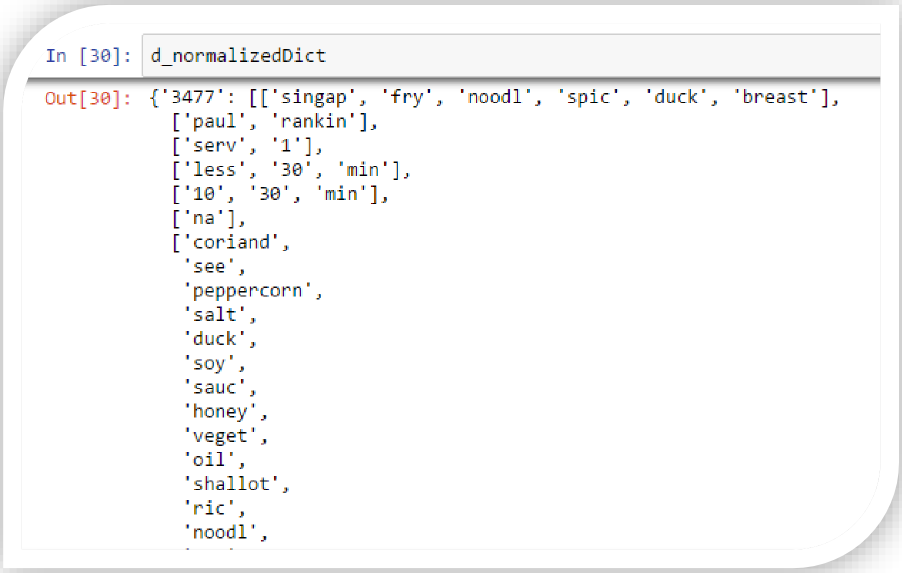
```
In [26]: dictRecipes

Out[26]: {'3477': 'figrolls_92754',
          '10638': 'flapjacks_84370',
          '672': 'pottedseatroutwithdi_4423',
          '6169': 'hyderabadi_biryani_of_73883',
          '1711': 'chocolate_brownies_32438',
          '1966': 'fillet_of_turbot_with_33580',
          '1470': 'vealescalopeswithpar_72315',
          '3490': 'prawnswrappedinpasta_71301',
          '8685': 'braisedbeefshortribs_93391',
          '2393': 'sweetcorn_relish_14809',
          '4393': 'asianstyletofustirfr_92503',
          '8404': 'saas_ni_macchi_parsee_05139',
          '10386': 'minced_beef_pinwheels_78308',
          '3289': 'pear_pain_au_chocolat_90614',
          '9385': 'christmaspuddingicec_77305',
          '1688': 'rosemaryandgarlicpor_88857',
          '5207': 'goancoconutpancakes_12752',
          '6756': 'chillicrabwitheggnoo_89276',
          '2085': 'individualpecanpies_87266',
```

The data contained in **data.csv** is saved in a list *data*. It contains the data regarding the recipes, i.e. title, author, serves, prepTime, cookTime, dietary, ingredients, methods. While creating the list data a first row is added with the field-names.



```
In [27]: data

Out[27]: [['title',
           'author',
           'serves',
           'prepTime',
           'cookTime',
           'dietary',
           'ingredients',
           'methods'],
          ['BBC Food - Recipes - Tamarind seared tuna steak with melon salad and zingy dressing',
           'James Martin',
           'Serves 4',
           'overnight',
           '10 to 30 mins',
           'NA',
           "['watermelon', 'melon', 'beans', 'sugar-snap peas', 'mangetout', 'tamarind', 'rapeseed oil'
          'palm sugar', 'soy sauce', 'fish sauce', 'mint', 'fresh coriander', 'limes', 'pomegranate']",
           "['For the salad, place the sliced melons in a vac-pac bag and seal in a vac-pac machine. L
          'For the tuna steak, brush the tuna with tamarind paste and leave to marinate for at least
         ad, heat a large frying pan and add the chilled melon, cook on each side for 1-2 minutes. C
```

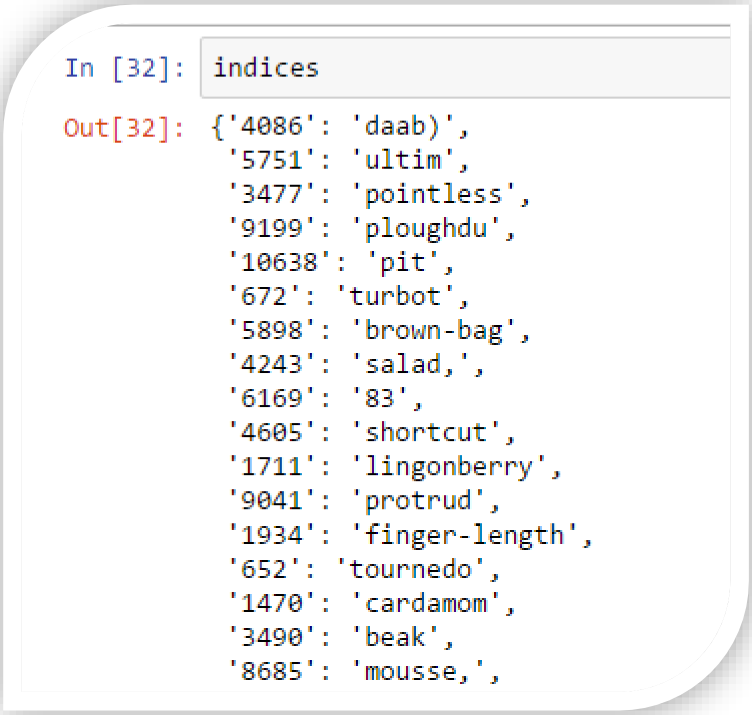
The data contained in **d_normalized.json** is imported into a dictionary called ***d*_normalizedDict**. Each entry of this dictionary corresponds to a recipe, the key is the recipe's id, the value is a list of lists organized according to the fields.



```
In [30]: d_normalizedDict

Out[30]: {'3477': [['singap', 'fry', 'noodl', 'spic', 'duck', 'breast'],
           ['paul', 'rankin'],
           ['serv', '1'],
           ['less', '30', 'min'],
           ['10', '30', 'min'],
           ['na'],
           ['coriand',
            'see',
            'peppercorn',
            'salt',
            'duck',
            'soy',
            'sauc',
            'honey',
            'veget',
            'oil',
            'shallot',
            'ric',
            'noodl',
```

3.1.

The data from **indexImproved.json** is imported into the indices dictionary. Each key is an integer to which a word is associated. The word were taken from all the fields of all the recipes.



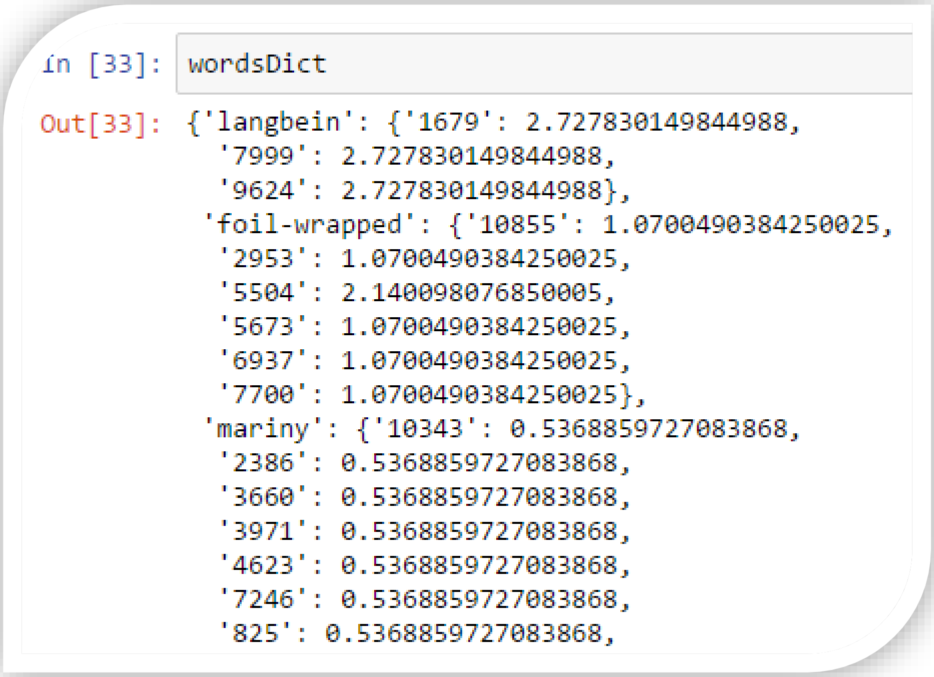
```
In [32]: indices

Out[32]: {'4086': 'daab)',
          '5751': 'ultim',
          '3477': 'pointless',
          '9199': 'ploughdu',
          '10638': 'pit',
          '672': 'turbot',
          '5898': 'brown-bag',
          '4243': 'salad,',
          '6169': '83',
          '4605': 'shortcut',
          '1711': 'lingonberry',
          '9041': 'protrud',
          '1934': 'finger-length',
          '652': 'tournedo',
          '1470': 'cardamom',
          '3490': 'beak',
          '8685': 'mousse,',
```

wordsDict dictionary contains data imported from **wordsDictImproved.json**. The keys of this dictionary are all the words of all the recipes, while the values are dictionaries themselves that have as keys the ids of the all the recipes that contain the aforementioned word associated to the words frequency in the text.

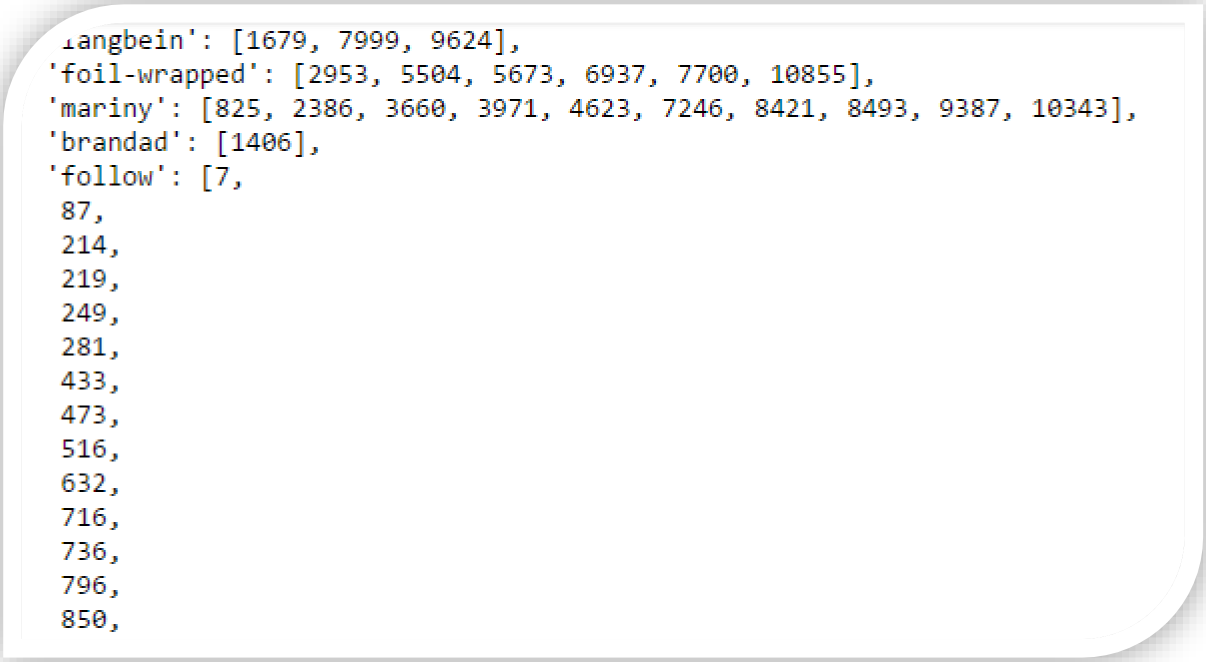
Here and in the following by frequency is meant:

$$t_{fid} = frequency\ of\ terms\ in\ document\ d$$

$$id_{fi} = \log(\frac{N}{Ni})$$

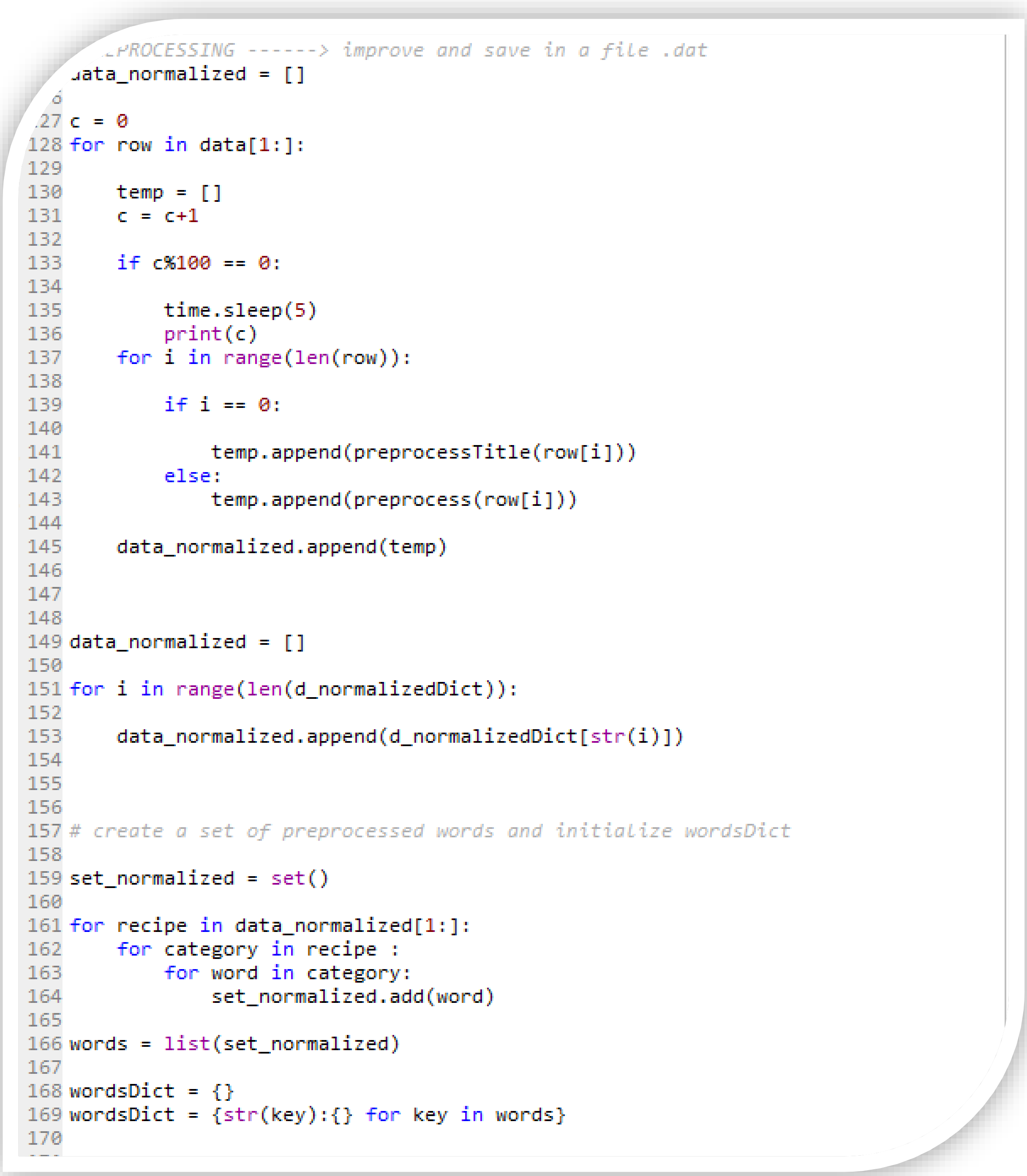


```
In [33]: wordsDict

Out[33]: {'langbein': {'1679': 2.727830149844988,
           '7999': 2.727830149844988,
           '9624': 2.727830149844988},
          'foil-wrapped': {'10855': 1.0700490384250025,
           '2953': 1.0700490384250025,
           '5504': 2.140098076850005,
           '5673': 1.0700490384250025,
           '6937': 1.0700490384250025,
           '7700': 1.0700490384250025},
          'mariny': {'10343': 0.5368859727083868,
           '2386': 0.5368859727083868,
           '3660': 0.5368859727083868,
           '3971': 0.5368859727083868,
           '4623': 0.5368859727083868,
           '7246': 0.5368859727083868,
           '825': 0.5368859727083868,
```

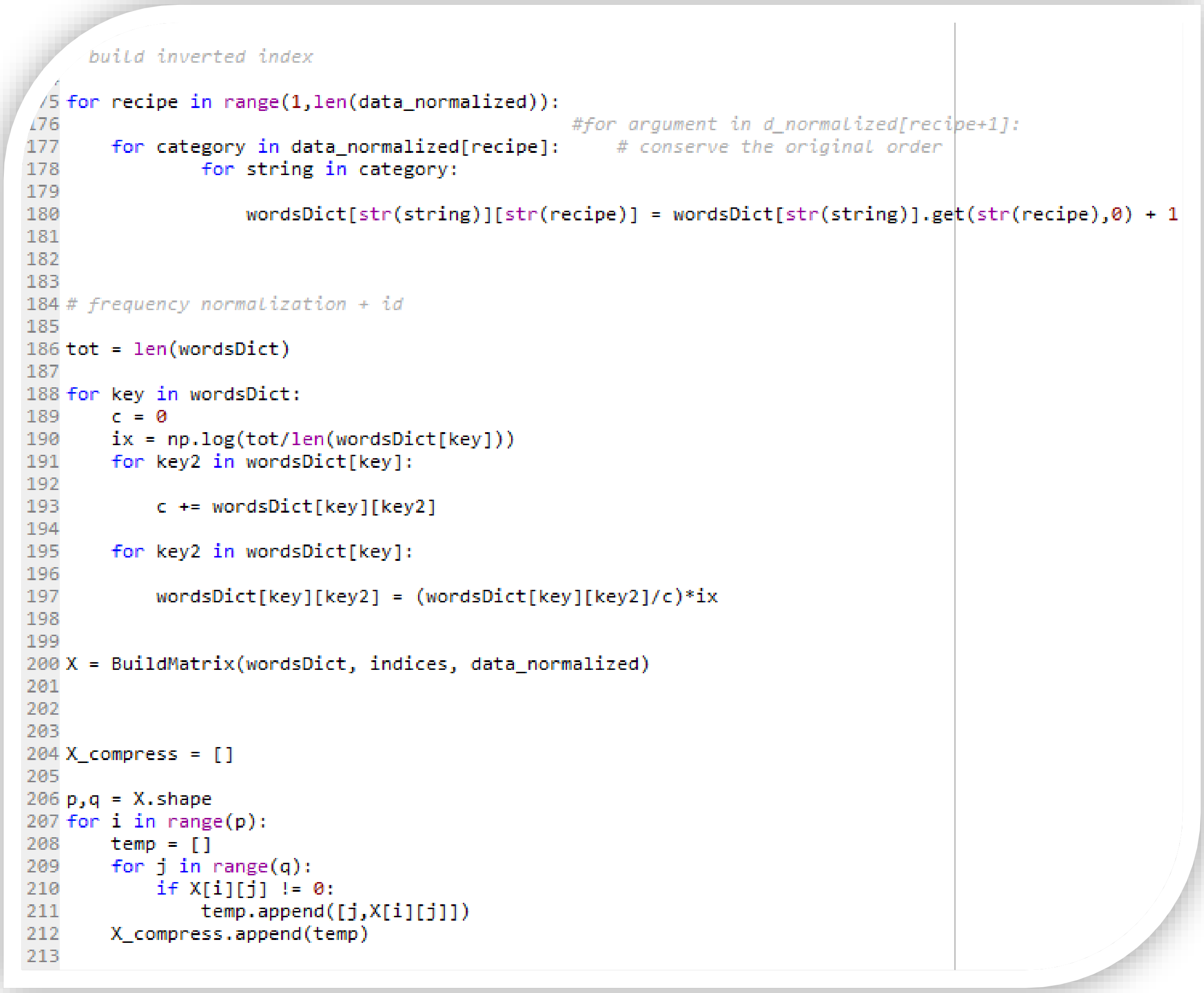


```
 langbein': [1679, 7999, 9624],
'foil-wrapped': [2953, 5504, 5673, 6937, 7700, 10855],
'mariny': [825, 2386, 3660, 3971, 4623, 7246, 8421, 8493, 9387, 10343],
'brandad': [1406],
'follow': [7,
 87,
 214,
 219,
 249,
 281,
 433,
 473,
 516,
 632,
 716,
 736,
 796,
 850,
```

invertedPost is a dictionary with data imported from *invertedPostImproved.json*. It has as keys the words of the recipes and as values an ordered list with all the recipes in which the key-word is contained.



```
PROCESSING ------> improve and save in a file .dat
ata_normalized = []

c = 0
for row in data[1:]:

    temp = []
    c = c+1

    if c%100 == 0:

        time.sleep(5)
        print(c)
    for i in range(len(row)):

        if i == 0:

            temp.append(preprocessTitle(row[i]))
        else:
            temp.append(preprocess(row[i]))

    data_normalized.append(temp)



data_normalized = []

for i in range(len(d_normalizedDict)):

    data_normalized.append(d_normalizedDict[str(i)])



# create a set of preprocessed words and initialize wordsDict

set_normalized = set()

for recipe in data_normalized[1:]:
    for category in recipe :
        for word in category:
            set_normalized.add(word)

words = list(set_normalized)

wordsDict = {}
wordsDict = {str(key):{} for key in words}
```



```
 build inverted index

for recipe in range(1,len(data_normalized)):
                                             #for argument in d_normalized[recipe+1]:
    for category in data_normalized[recipe]:     # conserve the original order
            for string in category:

                wordsDict[str(string)][str(recipe)] = wordsDict[str(string)].get(str(recipe),0) + 1



# frequency normalization + id

tot = len(wordsDict)

for key in wordsDict:
    c = 0
    ix = np.log(tot/len(wordsDict[key]))
    for key2 in wordsDict[key]:

        c += wordsDict[key][key2]

    for key2 in wordsDict[key]:

        wordsDict[key][key2] = (wordsDict[key][key2]/c)*ix


X = BuildMatrix(wordsDict, indices, data_normalized)



X_compress = []

p,q = X.shape
for i in range(p):
    temp = []
    for j in range(q):
        if X[i][j] != 0:
            temp.append([j,X[i][j]])
    X_compress.append(temp)

```

X is a multidimensional numpy array, it's values are imported from the *XcompressImproved.csv*. Each column corresponds to a recipe, while each row correspond to word, the values are the frequency of the words in the recipes.

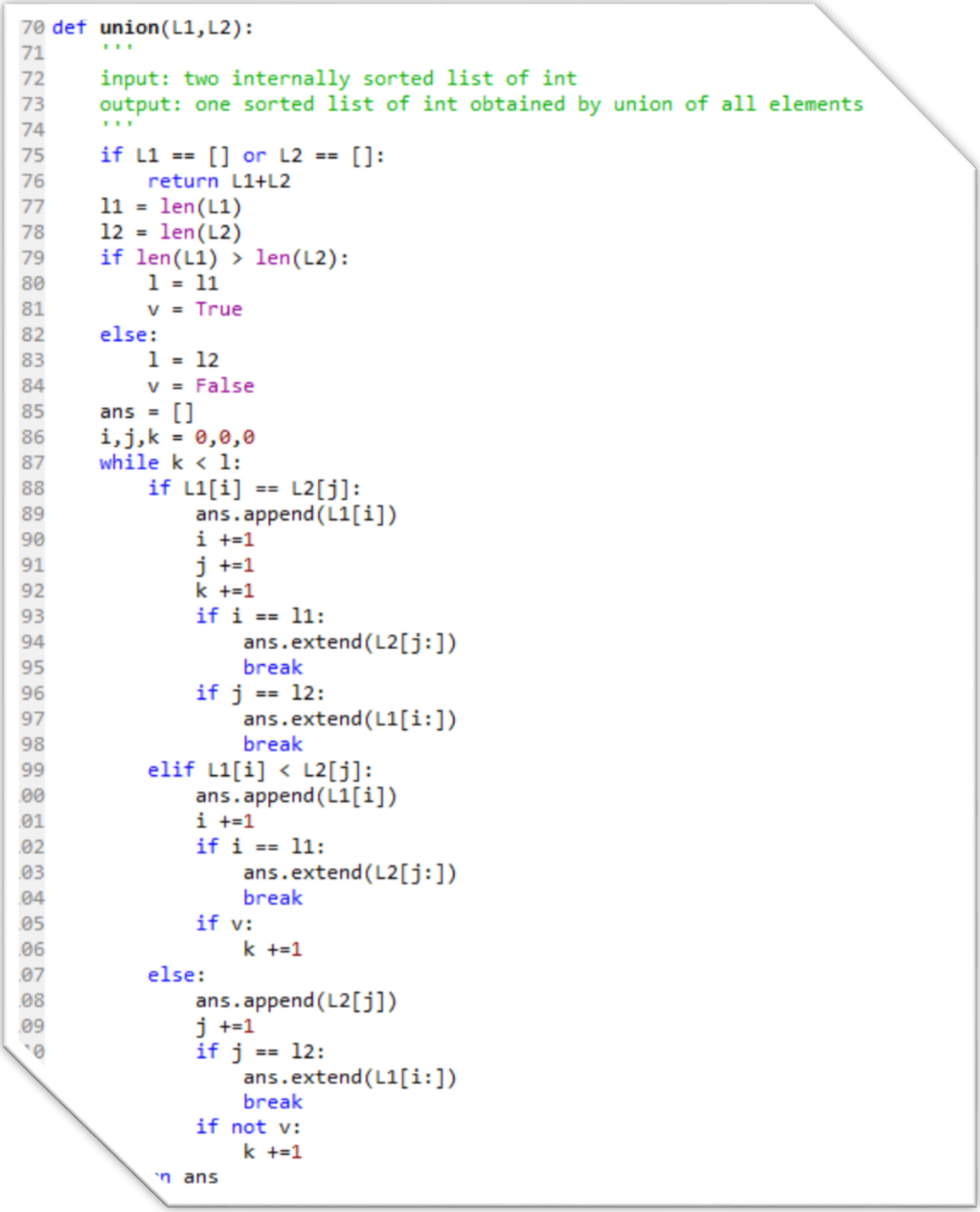
Xweight is a numpy array similar to *X*, its only difference being that instead of the frequency the values are the weighted frequencies.

The *lib5bis.py* functions are the following:

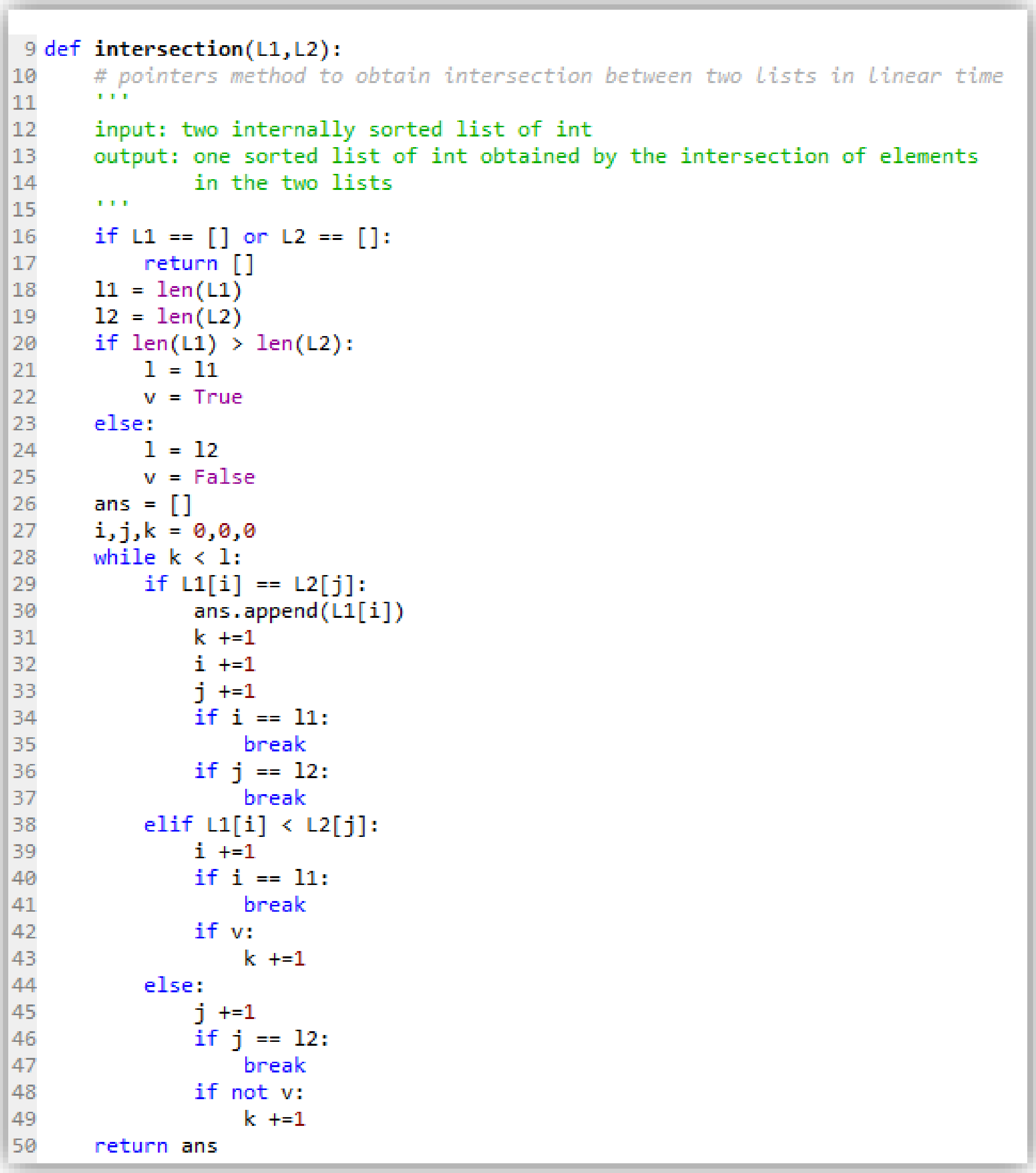
- union(L1,L2)
- intersection(L1,L2)
- skiPointers(L1,L2)
- negation(L1,L2)
- BuildQueryFromMatrix(X,r)
- BuildVector(indices, string)
- cosineSimilarity(X,y,r)
- rank(resDict)

- subroutine(Xtot,r,s,data,indices)
- queryPointers(Xtot, s,n ,invertedPost, indices,f)

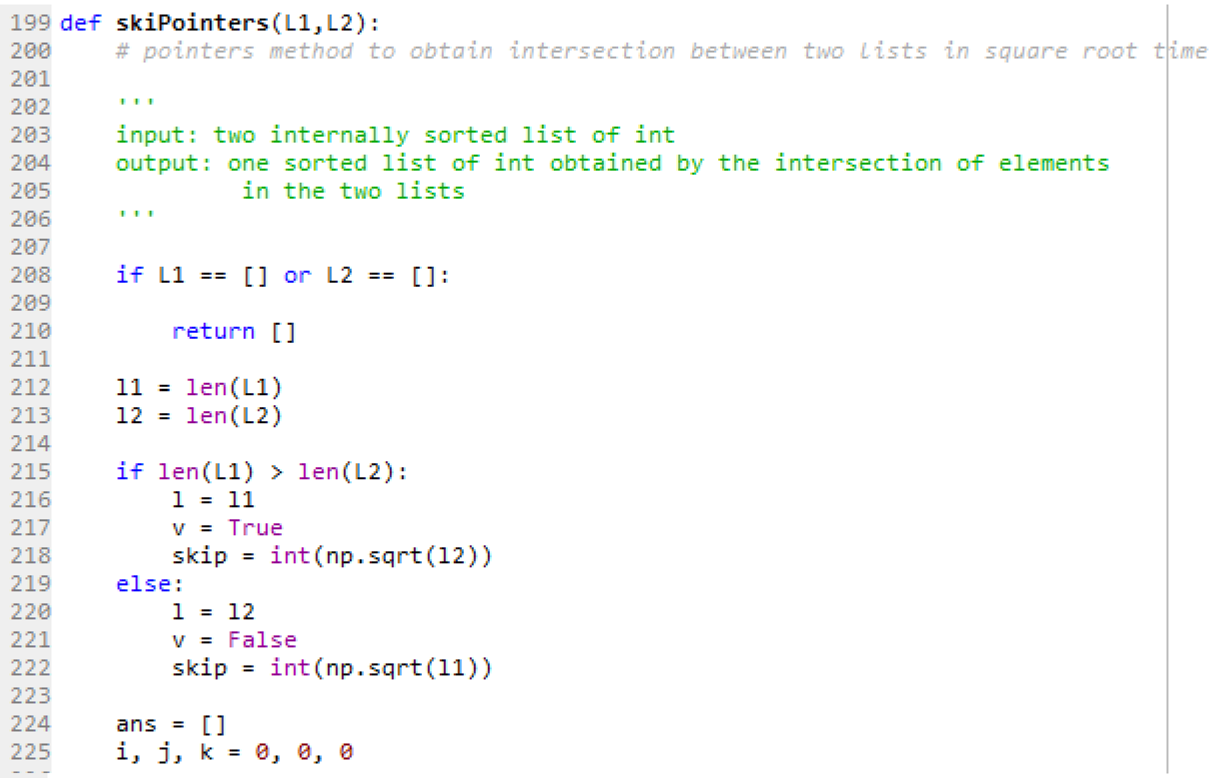
The functions union(L1,L2), intersection(L1,L2), skiPointers(L1,L2), negation(L1,L2) use an algorithm with pointers.



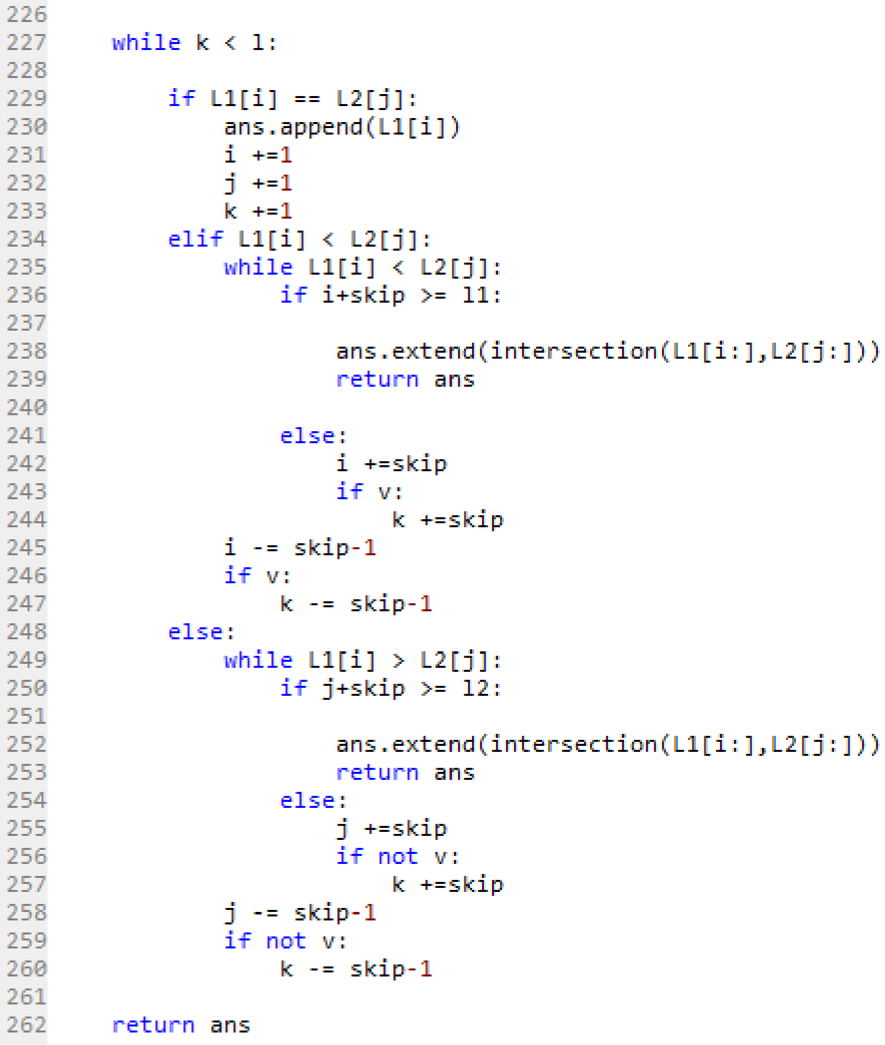
```
def union(L1,L2):
    '''
    input: two internally sorted list of int
    output: one sorted list of int obtained by union of all elements
    '''
    if L1 == [] or L2 == []:
        return L1+L2
    l1 = len(L1)
    l2 = len(L2)
    if len(L1) > len(L2):
        l = l1
        v = True
    else:
        l = l2
        v = False
    ans = []
    i,j,k = 0,0,0
    while k < l:
        if L1[i] == L2[j]:
            ans.append(L1[i])
            i +=1
            j +=1
            k +=1
            if i == l1:
                ans.extend(L2[j:])
                break
            if j == l2:
                ans.extend(L1[i:])
                break
        elif L1[i] < L2[j]:
            ans.append(L1[i])
            i +=1
            if i == l1:
                ans.extend(L2[j:])
                break
            if v:
                k +=1
        else:
            ans.append(L2[j])
            j +=1
            if j == l2:
                ans.extend(L1[i:])
                break
            if not v:
                k +=1
         n ans
```



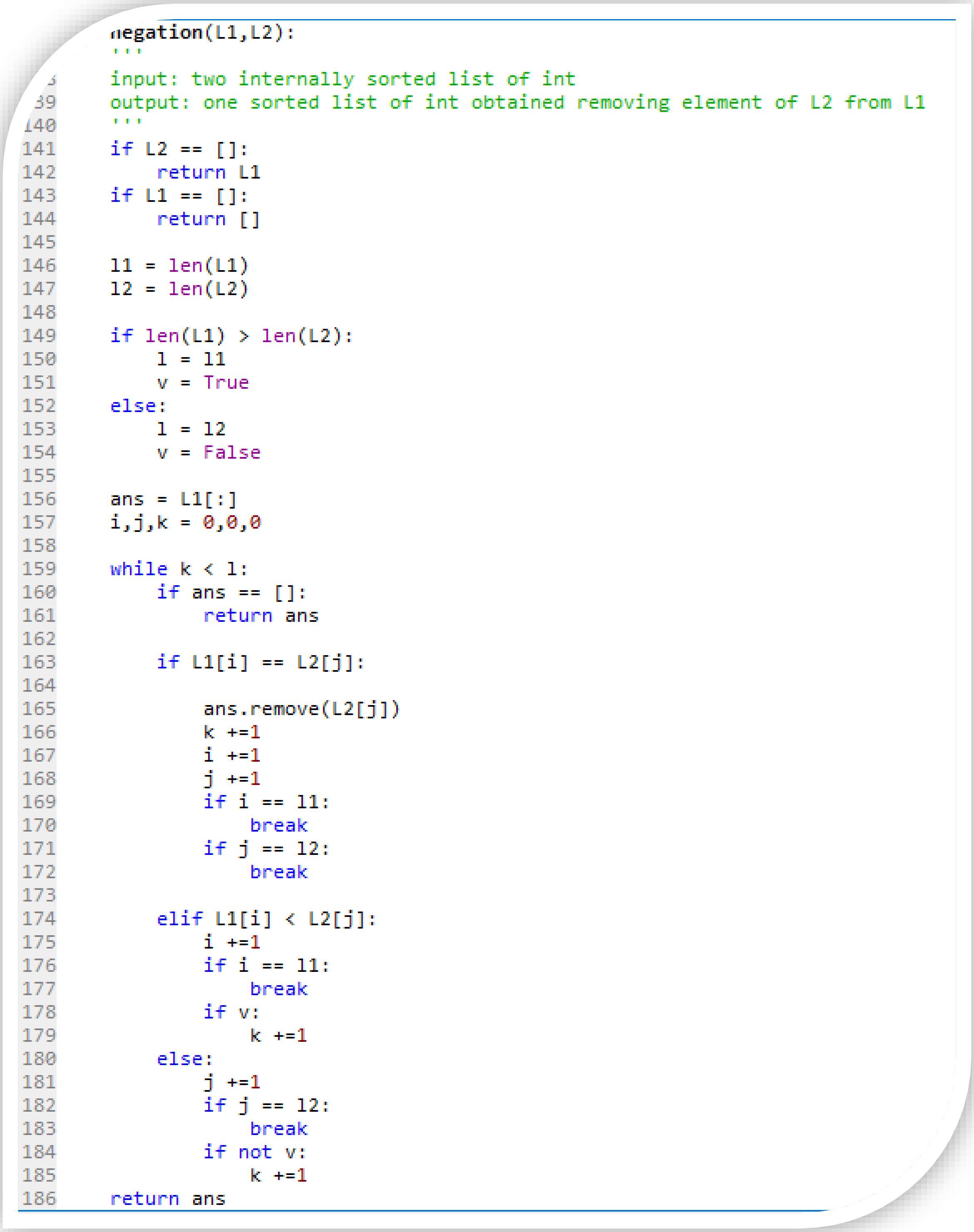
```
def intersection(L1,L2):
    # pointers method to obtain intersection between two lists in linear time
    '''
    input: two internally sorted list of int
    output: one sorted list of int obtained by the intersection of elements
            in the two lists
    '''
    if L1 == [] or L2 == []:
        return []
    l1 = len(L1)
    l2 = len(L2)
    if len(L1) > len(L2):
        l = l1
        v = True
    else:
        l = l2
        v = False
    ans = []
    i,j,k = 0,0,0
    while k < l:
        if L1[i] == L2[j]:
            ans.append(L1[i])
            k +=1
            i +=1
            j +=1
            if i == l1:
                break
            if j == l2:
                break
        elif L1[i] < L2[j]:
            i +=1
            if i == l1:
                break
            if v:
                k +=1
        else:
            j +=1
            if j == l2:
                break
            if not v:
                k +=1
    return ans
```



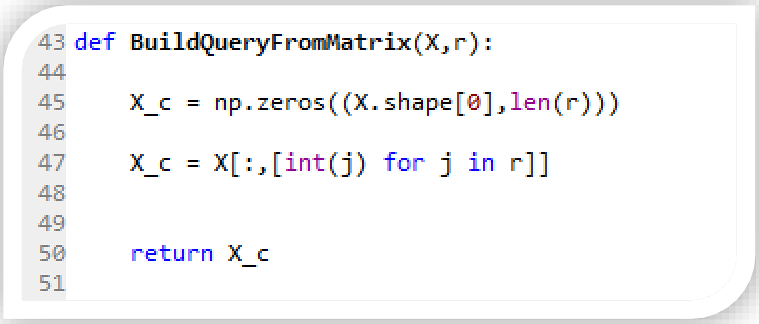
```
def skiPointers(L1,L2):
    # pointers method to obtain intersection between two lists in square root time

    '''
    input: two internally sorted list of int
    output: one sorted list of int obtained by the intersection of elements
             in the two lists
    '''

    if L1 == [] or L2 == []:

        return []

    l1 = len(L1)
    l2 = len(L2)

    if len(L1) > len(L2):
        l = l1
        v = True
        skip = int(np.sqrt(l2))
    else:
        l = l2
        v = False
        skip = int(np.sqrt(l1))

    ans = []
    i, j, k = 0, 0, 0
```



```

    while k < l:

        if L1[i] == L2[j]:
            ans.append(L1[i])
            i +=1
            j +=1
            k +=1
        elif L1[i] < L2[j]:
            while L1[i] < L2[j]:
                if i+skip >= l1:

                    ans.extend(intersection(L1[i:],L2[j:]))
                    return ans

                else:
                    i +=skip
                    if v:
                        k +=skip
            i -= skip-1
            if v:
                k -= skip-1
        else:
            while L1[i] > L2[j]:
                if j+skip >= l2:

                    ans.extend(intersection(L1[i:],L2[j:]))
                    return ans
                else:
                    j +=skip
                    if not v:
                        k +=skip
            j -= skip-1
            if not v:
                k -= skip-1

    return ans
```



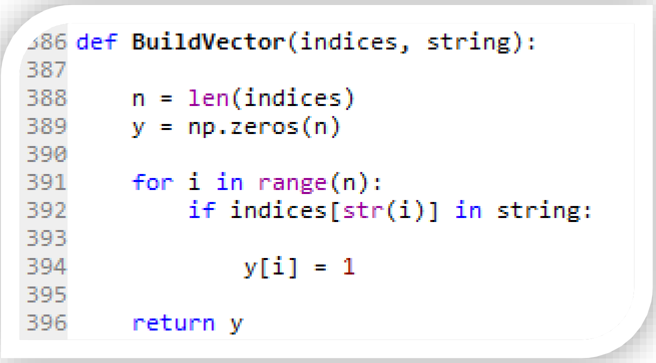
```
negation(L1,L2):
'''
input: two internally sorted list of int
output: one sorted list of int obtained removing element of L2 from L1
'''
if L2 == []:
    return L1
if L1 == []:
    return []

l1 = len(L1)
l2 = len(L2)

if len(L1) > len(L2):
    l = l1
    v = True
else:
    l = l2
    v = False

ans = L1[:]
i,j,k = 0,0,0

while k < l:
    if ans == []:
        return ans

    if L1[i] == L2[j]:

        ans.remove(L2[j])
        k +=1
        i +=1
        j +=1
        if i == l1:
            break
        if j == l2:
            break

    elif L1[i] < L2[j]:
        i +=1
        if i == l1:
            break
        if v:
            k +=1
    else:
        j +=1
        if j == l2:
            break
        if not v:
            k +=1
return ans
```



```python
def BuildQueryFromMatrix(X,r):

    X_c = np.zeros((X.shape[0],len(r)))

    X_c = X[:,[int(j) for j in r]]


    return X_c

```

The *BuildQueryFromMatrix* function creates a matrix *X_c* from the matrix *X* which contains only the recipes (so only the columns) that contain the searched terms (and don't contain the terms the user wants to exclude).

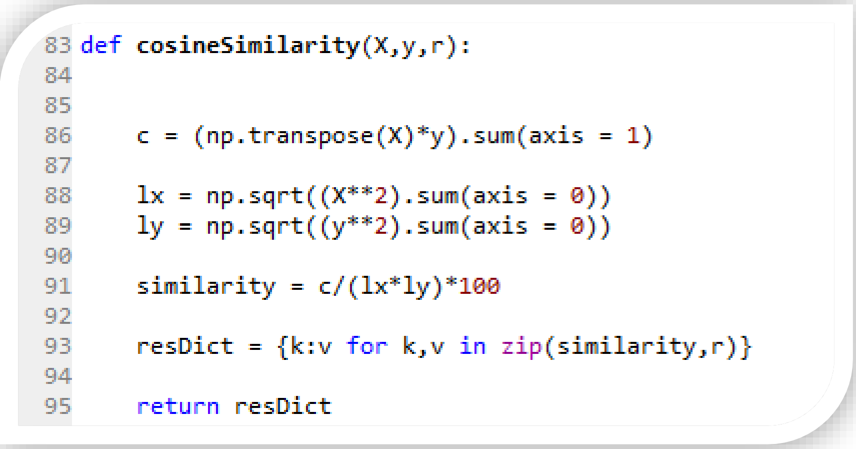
In the script the input matrix *X* will be the matrix with the frequency of the words for each recipe while *r* will be a list with the IDs of the recipes which contain the searched words (and don't contain the words which have to be excluded).
X_c has number of rows equal to the number of rows of the matrix *X* and number of columns equal to the length of *r*.



```python
def BuildVector(indices, string):

    n = len(indices)
    y = np.zeros(n)

    for i in range(n):
        if indices[str(i)] in string:

            y[i] = 1

    return y
```

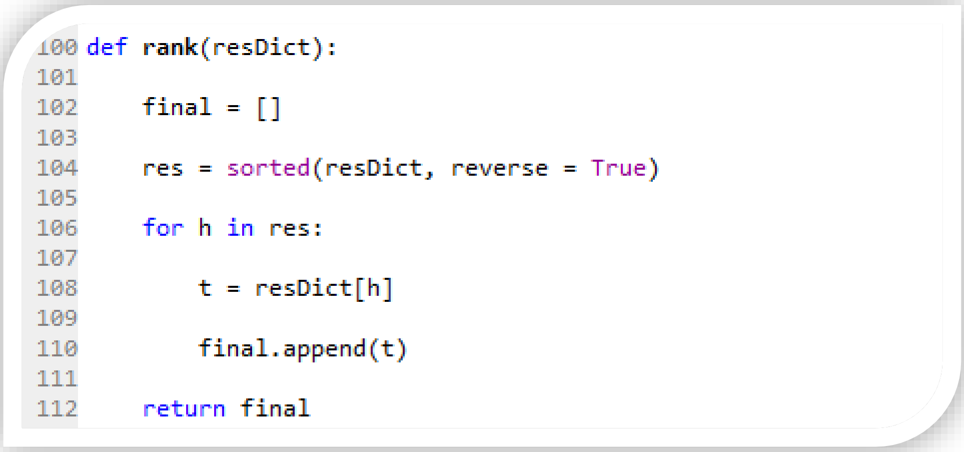
The *BuildVector* function returns a numpy array *y* with length equal to the total number of different words existing in all recipes. *y* is made of zeros and ones, the value is *1* if the position corresponds to the index number of one of the searched terms, zero otherwise.

The *cosineSimilarity* function implements the cosine similarity. The function returns a dictionary *resDict* with as keys the results of the cosine similarity operation and as values the corresponding element of the set *r*.

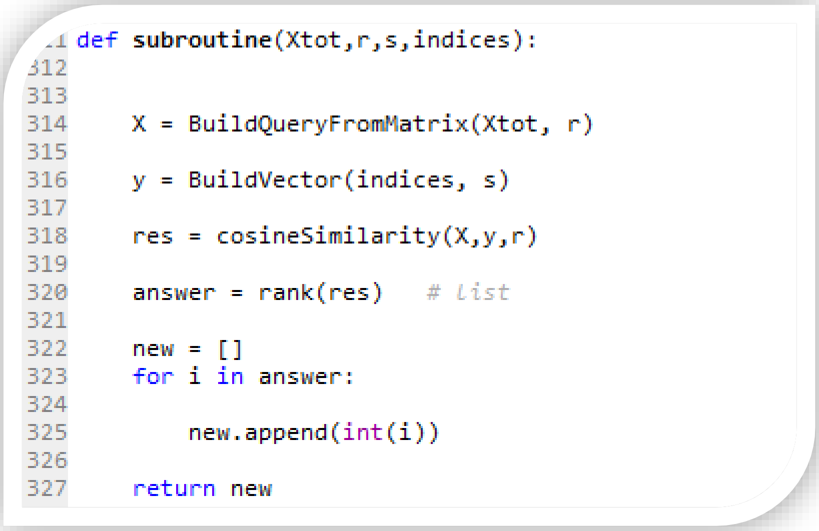


```python
def cosineSimilarity(X,y,r):


    c = (np.transpose(X)*y).sum(axis = 1)

    lx = np.sqrt((X**2).sum(axis = 0))
    ly = np.sqrt((y**2).sum(axis = 0))

    similarity = c/(lx*ly)*100

    resDict = {k:v for k,v in zip(similarity,r)}

    return resDict
```

The *rank* function takes as input a dictionary, the dictionary returned by the *cosineSimilarity* function, as can be seen in the script. This function returns a list of recipes, *final*, ordered according to higher cosine similarity.

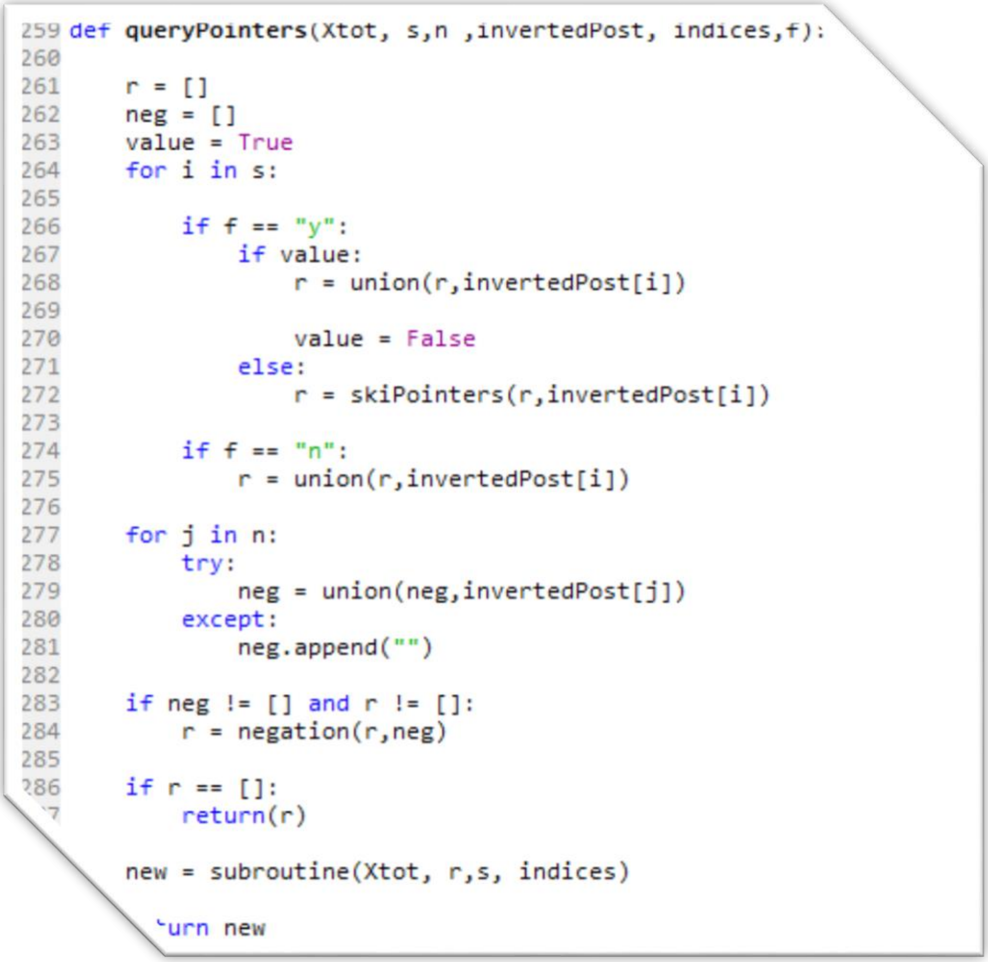


```python
def rank(resDict):

    final = []

    res = sorted(resDict, reverse = True)

    for h in res:

        t = resDict[h]

        final.append(t)

    return final
```



```python
def subroutine(Xtot,r,s,indices):


    X = BuildQueryFromMatrix(Xtot, r)

    y = BuildVector(indices, s)

    res = cosineSimilarity(X,y,r)

    answer = rank(res)    # List

    new = []
    for i in answer:

        new.append(int(i))

    return new
```

The subroutine function takes the indices from the list returned by *rank* and returns the data of the recipes (e.g title, author, ingredients, methods etc.) ordered by cosine similarity as a list.



```
def queryPointers(Xtot, s,n ,invertedPost, indices,f):

    r = []
    neg = []
    value = True
    for i in s:

        if f == "y":
            if value:
                r = union(r,invertedPost[i])

                value = False
            else:
                r = skiPointers(r,invertedPost[i])

        if f == "n":
            r = union(r,invertedPost[i])

    for j in n:
        try:
            neg = union(neg,invertedPost[j])
        except:
            neg.append("")

    if neg != [] and r != []:
        r = negation(r,neg)

    if r == []:
        return(r)

    new = subroutine(Xtot, r,s, indices)

      urn new
```

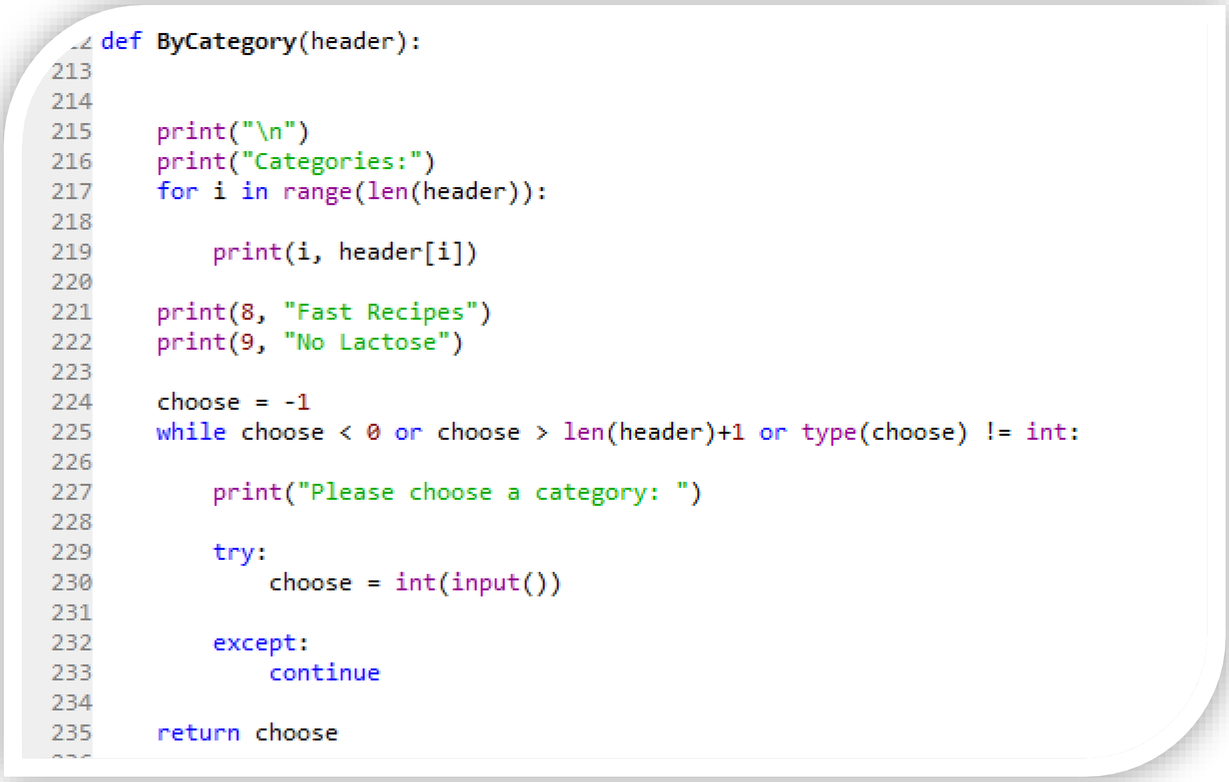
The *queryPointers* function returns a list with the data of the recipes which contain the searched words and don't contain the words the user wants exclude.

In the script, *s* is the list of (preprocessed) words which need to be in the recipes, while *n* is the list of (preprocessed) words which need to be excluded. *f* indicates whether the user chose to see only results containing the searched terms or not.

3.2. Extra features

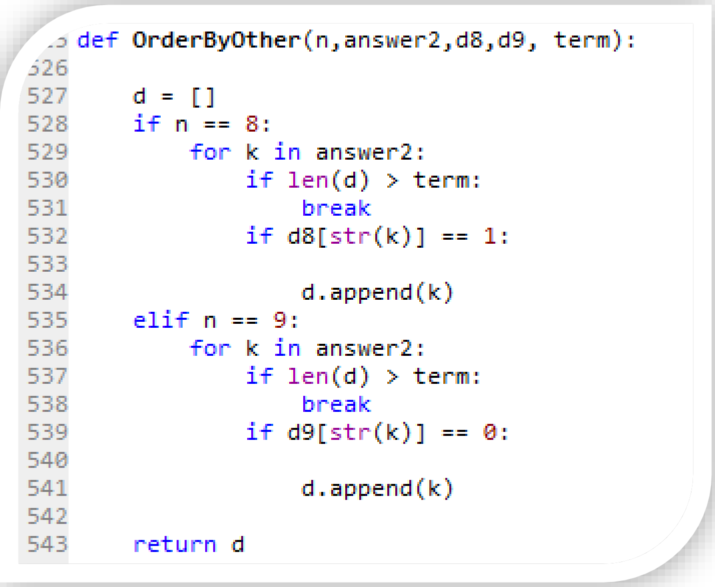
The extra features offered, i.e. weighted search and search by category, are implemented in the following functions (lib5bis.py):

- ByCategory(header)
- OrderByOther(n,answer2,d8,d9, term)
- OrderByCategory(n,ans, s, term)
- chooseRec(d, num, data, dictRecipes)
- formatting(recipe,ss)



```
def ByCategory(header):


    print("\n")
    print("Categories:")
    for i in range(len(header)):

        print(i, header[i])

    print(8, "Fast Recipes")
    print(9, "No Lactose")

    choose = -1
    while choose < 0 or choose > len(header)+1 or type(choose) != int:

        print("Please choose a category: ")

        try:
            choose = int(input())

        except:
            continue

    return choose
```

The *ByCategory* function prints the categories and asks the user which category he/she wants to choose.

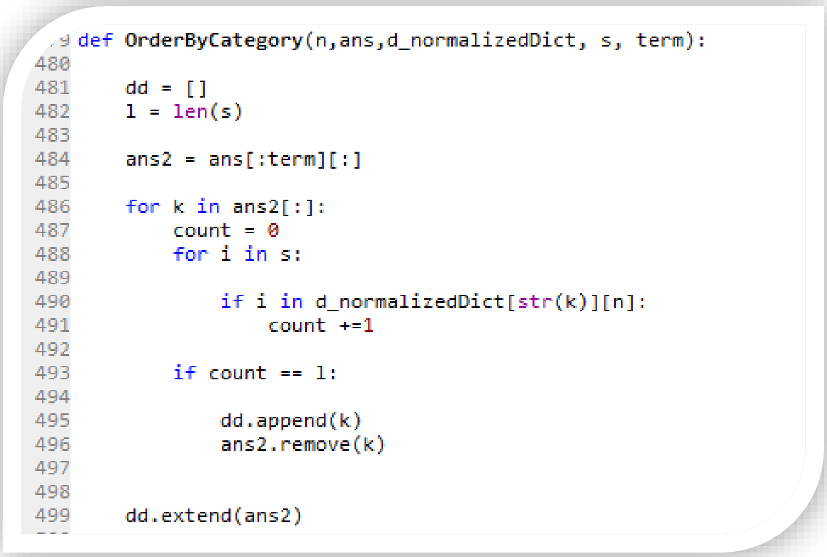


```
def OrderByOther(n,answer2,d8,d9, term):

    d = []
    if n == 8:
        for k in answer2:
            if len(d) > term:
                break
            if d8[str(k)] == 1:

                d.append(k)
    elif n == 9:
        for k in answer2:
            if len(d) > term:
                break
            if d9[str(k)] == 0:

                d.append(k)

    return d
```

Depending on the choice made by the user, the *OrderByOther* function selects out of all the recipes in *answer2* either the fast recipes or the recipes with no lactose, and returns their indices with the list *d*.

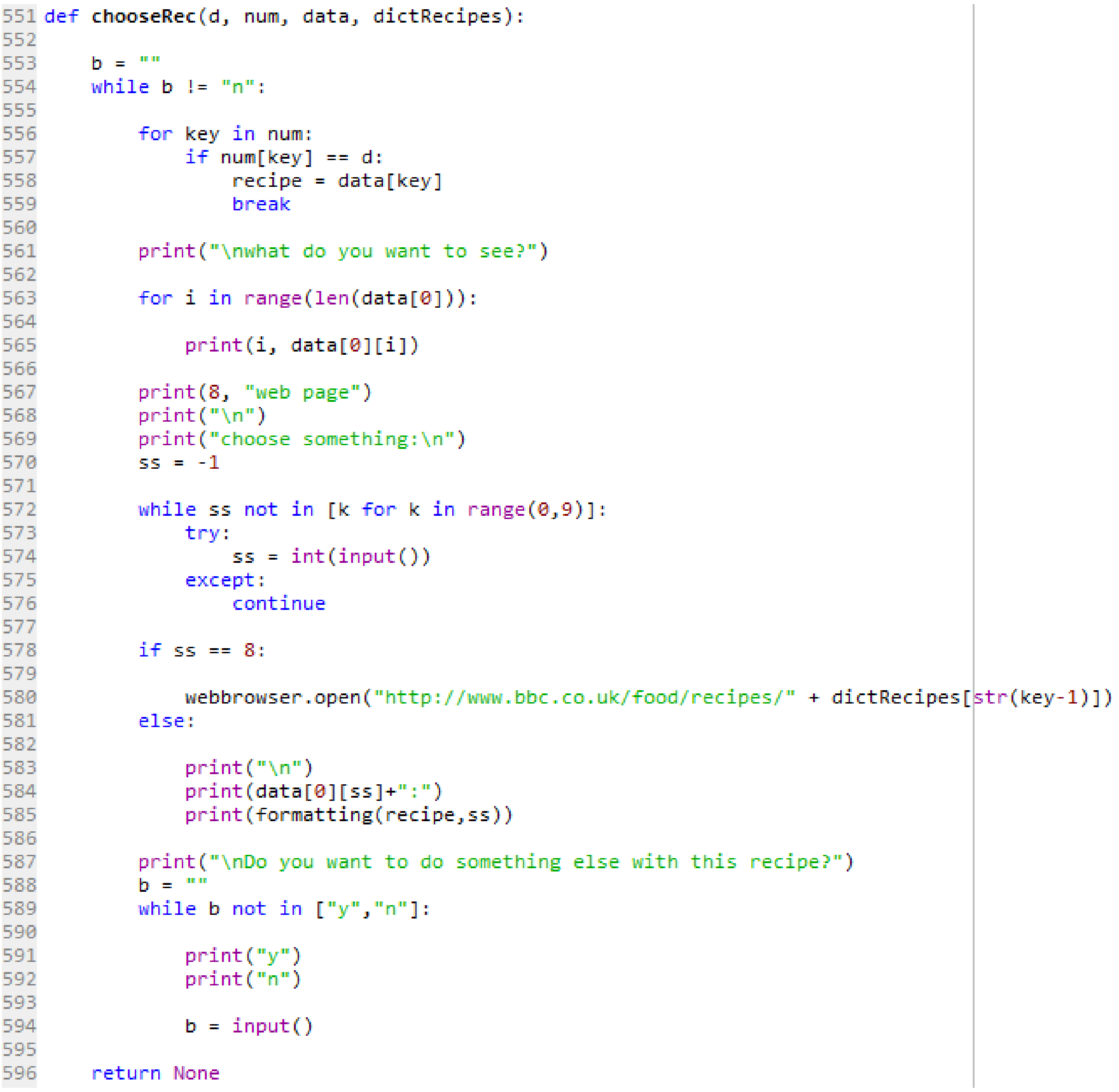
The choice of the user is given to the function through the value *n*.

term specifies how many recipes *d* will contain (in the script the value is 51).



```
def OrderByCategory(n,ans,d_normalizedDict, s, term):

    dd = []
    l = len(s)

    ans2 = ans[:term][:]

    for k in ans2[:]:
        count = 0
        for i in s:

            if i in d_normalizedDict[str(k)][n]:
                count +=1

        if count == 1:

            dd.append(k)
            ans2.remove(k)


    dd.extend(ans2)
```

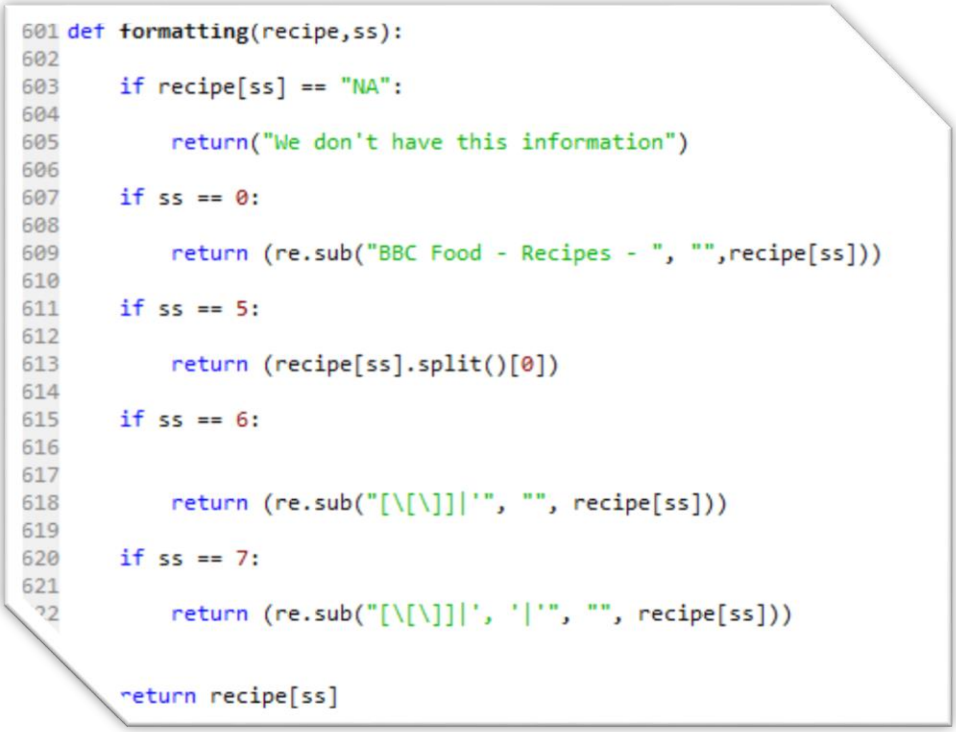
The *OrderByCategory* function selects, out of the recipes which were already ordered using the cosine similarity, the first *term* (= 51 in the script) recipes in such a way that they contain the search-words in the category that the user chose. *n* is the integer which tells which category was chosen and *ans* is the listof recipes (with all the data for each recipe i.e. title, author, etc.) ordered using the cosine similarity.



```
def chooseRec(d, num, data, dictRecipes):

    b = ""
    while b != "n":

        for key in num:
            if num[key] == d:
                recipe = data[key]
                break

        print("\nwhat do you want to see?")

        for i in range(len(data[0])):

            print(i, data[0][i])

        print(8, "web page")
        print("\n")
        print("choose something:\n")
        ss = -1

        while ss not in [k for k in range(0,9)]:
            try:
                ss = int(input())
            except:
                continue

        if ss == 8:

            webbrowser.open("http://www.bbc.co.uk/food/recipes/" + dictRecipes[str(key-1)])
        else:

            print("\n")
            print(data[0][ss]+":")
            print(formatting(recipe,ss))

        print("\nDo you want to do something else with this recipe?")
        b = ""
        while b not in ["y","n"]:

            print("y")
            print("n")

            b = input()

    return None
```

The *chooseRec* function asks the user what specific field of the recipe should be shown and then prints it to the monitor.

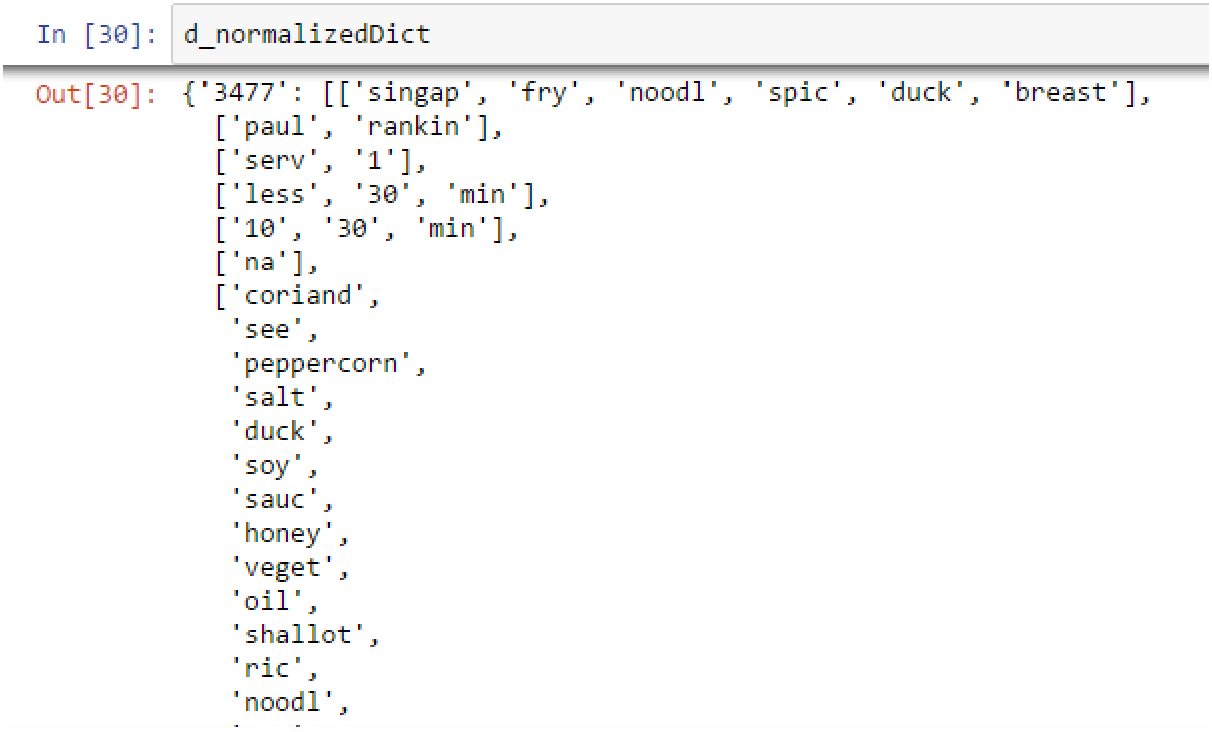
An extra feature implemented in this function is the web page option, which opens the URL of the recipe.



```
def formatting(recipe,ss):

    if recipe[ss] == "NA":

        return("We don't have this information")

    if ss == 0:

        return (re.sub("BBC Food - Recipes - ", "",recipe[ss]))

    if ss == 5:

        return (recipe[ss].split()[0])

    if ss == 6:


        return (re.sub("[\[\]]|'", "", recipe[ss]))

    if ss == 7:

        return (re.sub("[\[\]]|', '|'", "", recipe[ss]))

    return recipe[ss]
```

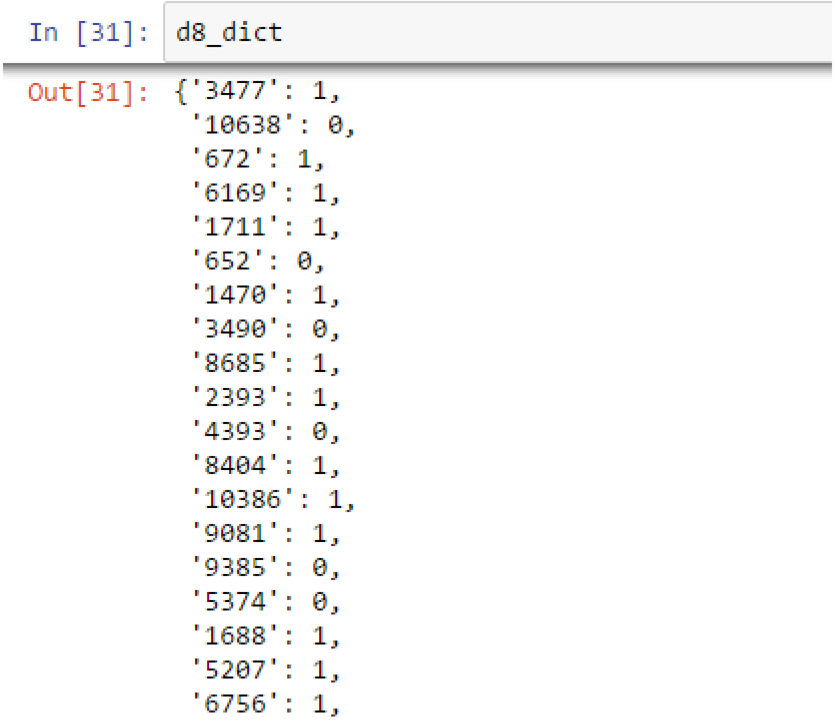
The formatting function does some formatting.

Query that satisfies lactose intolerant and people that search a fast recipe to prepare.



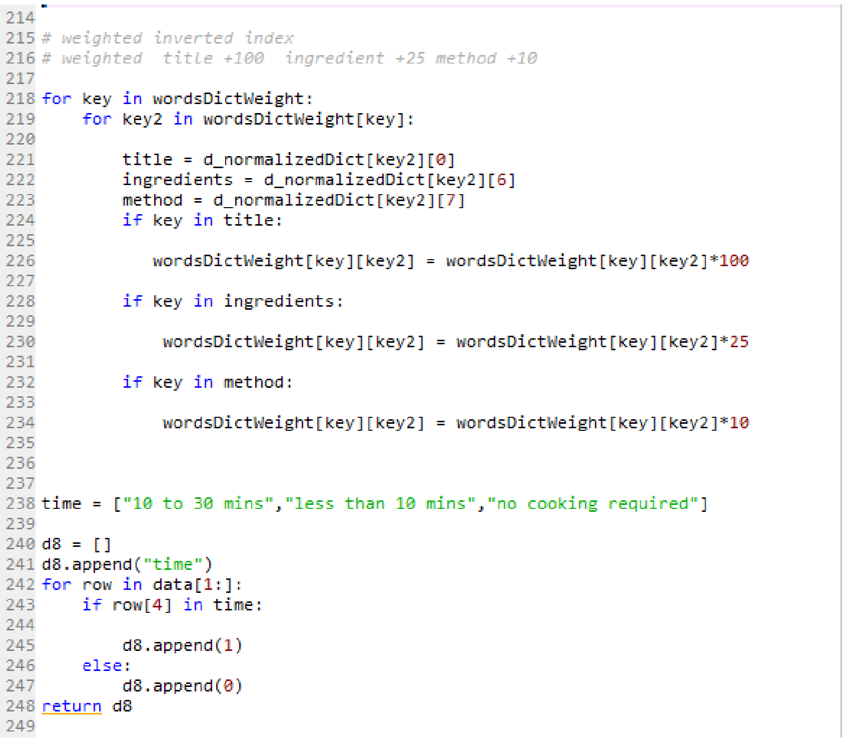
```
In [30]: d_normalizedDict

Out[30]: {'3477': [['singap', 'fry', 'noodl', 'spic', 'duck', 'breast'],
           ['paul', 'rankin'],
           ['serv', '1'],
           ['less', '30', 'min'],
           ['10', '30', 'min'],
           ['na'],
           ['coriand',
            'see',
            'peppercorn',
            'salt',
            'duck',
            'soy',
            'sauc',
            'honey',
            'veget',
            'oil',
            'shallot',
            'ric',
            'noodl',
```

d8_dict.json's data is imported into the *d8_dict* dictionary. In this dictionary the key is the recipe's id while the value is either a *1* or a *0*. A *1* if the recipe is a fast recipe, a *0* otherwise. We considered as fast recipes recipes for which the cooking time (cookTime) is less than 30 minutes or no cooking is required.

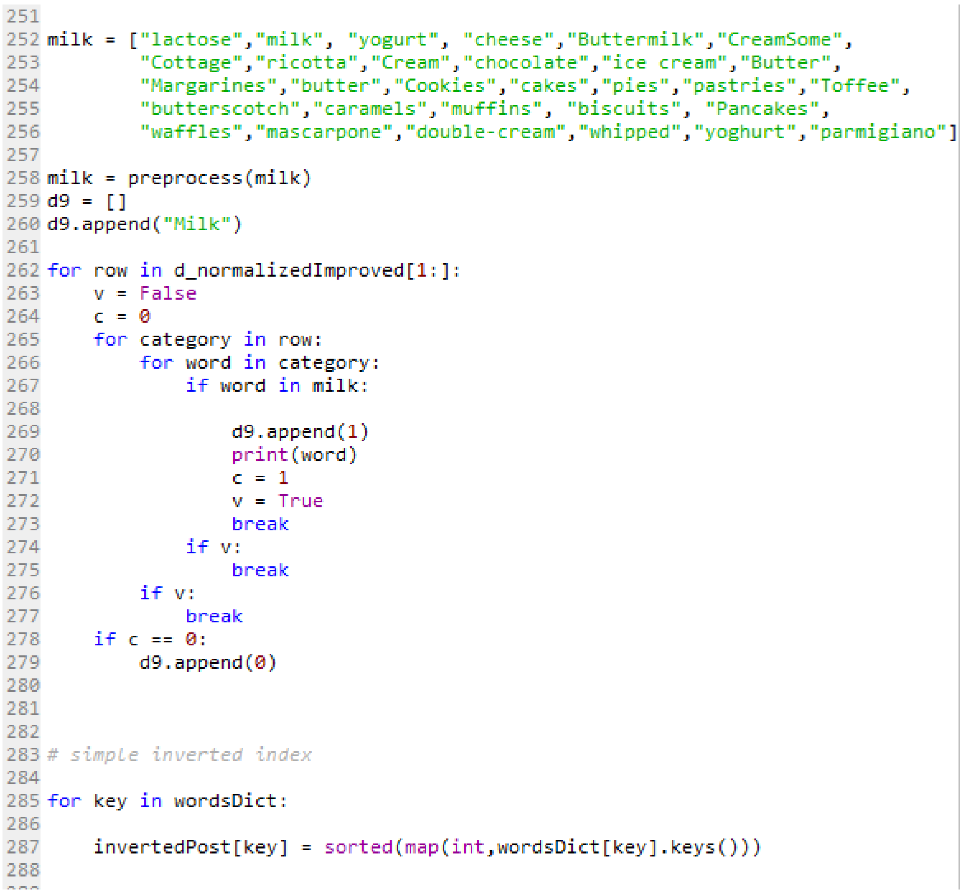


```
In [31]: d8_dict

Out[31]: {'3477': 1,
          '10638': 0,
          '672': 1,
          '6169': 1,
          '1711': 1,
          '652': 0,
          '1470': 1,
          '3490': 0,
          '8685': 1,
          '2393': 1,
          '4393': 0,
          '8404': 1,
          '10386': 1,
          '9081': 1,
          '9385': 0,
          '5374': 0,
          '1688': 1,
          '5207': 1,
          '6756': 1,
```

The *d9_dict* dictionary contains data imported from the *d9_dict.json*. Similarly to the *d8_dict* dictionary, the key is the recipe's id while the value is either a *1* or a *0*. In this case thought, a 1 indicates that the recipe is lactose-free.



```

# weighted inverted index
# weighted  title +100  ingredient +25 method +10

for key in wordsDictWeight:
    for key2 in wordsDictWeight[key]:

        title = d_normalizedDict[key2][0]
        ingredients = d_normalizedDict[key2][6]
        method = d_normalizedDict[key2][7]
        if key in title:

           wordsDictWeight[key][key2] = wordsDictWeight[key][key2]*100

        if key in ingredients:

            wordsDictWeight[key][key2] = wordsDictWeight[key][key2]*25

        if key in method:

            wordsDictWeight[key][key2] = wordsDictWeight[key][key2]*10



time = ["10 to 30 mins","less than 10 mins","no cooking required"]

d8 = []
d8.append("time")
for row in data[1:]:
    if row[4] in time:

        d8.append(1)
    else:
        d8.append(0)
return d8

```



```
milk = ["lactose","milk", "yogurt", "cheese","Buttermilk","CreamSome",
        "Cottage","ricotta","Cream","chocolate","ice cream","Butter",
        "Margarines","butter","Cookies","cakes","pies","pastries","Toffee",
        "butterscotch","caramels","muffins", "biscuits", "Pancakes",
        "waffles","mascarpone","double-cream","whipped","yoghurt","parmigiano"]

milk = preprocess(milk)
d9 = []
d9.append("Milk")

for row in d_normalizedImproved[1:]:
    v = False
    c = 0
    for category in row:
        for word in category:
            if word in milk:

                d9.append(1)
                print(word)
                c = 1
                v = True
                break
            if v:
                break
        if v:
            break
    if c == 0:
        d9.append(0)



# simple inverted index

for key in wordsDict:

    invertedPost[key] = sorted(map(int,wordsDict[key].keys()))

```

4.

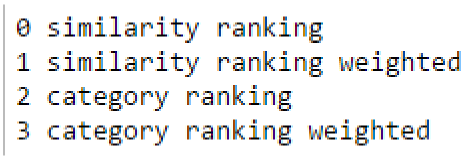
4.1.What the program does:

Given one or more terms the program finds the recipes which are most related. It is also possible to exclude terms which should not be present in the recipe.

In the case of two or more search-terms, it can be chosen whether the shown results should only be the ones which contain both terms, or if the user also wants to see the recipes which contain only one of the two terms. In both cases the terms can be found in any of the fields of the recipe (e.g. title, ingredients, method etc.).

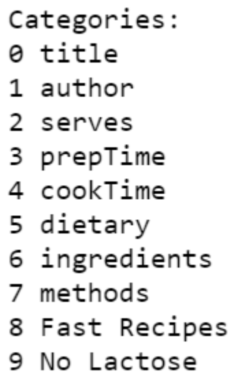
The results of the search can be ordered in two possible ways: by similarity ranking or by category. Furthermore a weighted similarity option is also present, different importance is given to the words depending on where they can be found, for example words contained in the title have more weight. In particular the frequency of the words is multiplied by 100 if the word belongs to the title, by 25 if it belongs to the ingredients and 10 if it's in the methods.

This are the possible choices for how the search-result should be given:



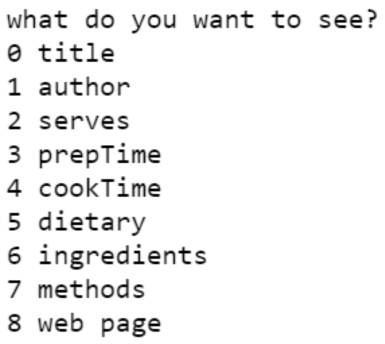
```
0 similarity ranking
1 similarity ranking weighted
2 category ranking
3 category ranking weighted
```

In the case of ranking by category, the options are:



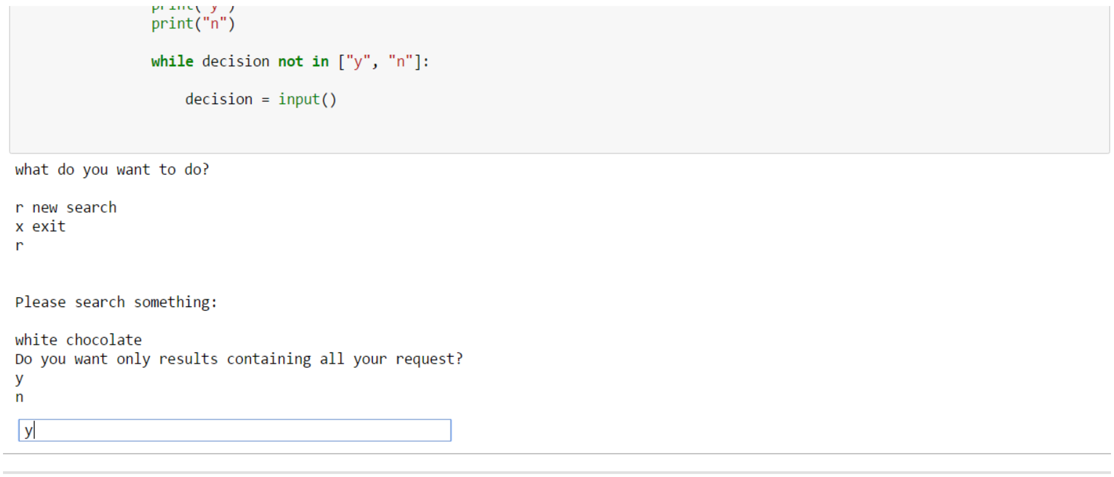
```
Categories:
0 title
1 author
2 serves
3 prepTime
4 cookTime
5 dietary
6 ingredients
7 methods
8 Fast Recipes
9 No Lactose
```

The program requests the user to choose a specific recipe which can then be further inspected. It is possible to ask for the following information:



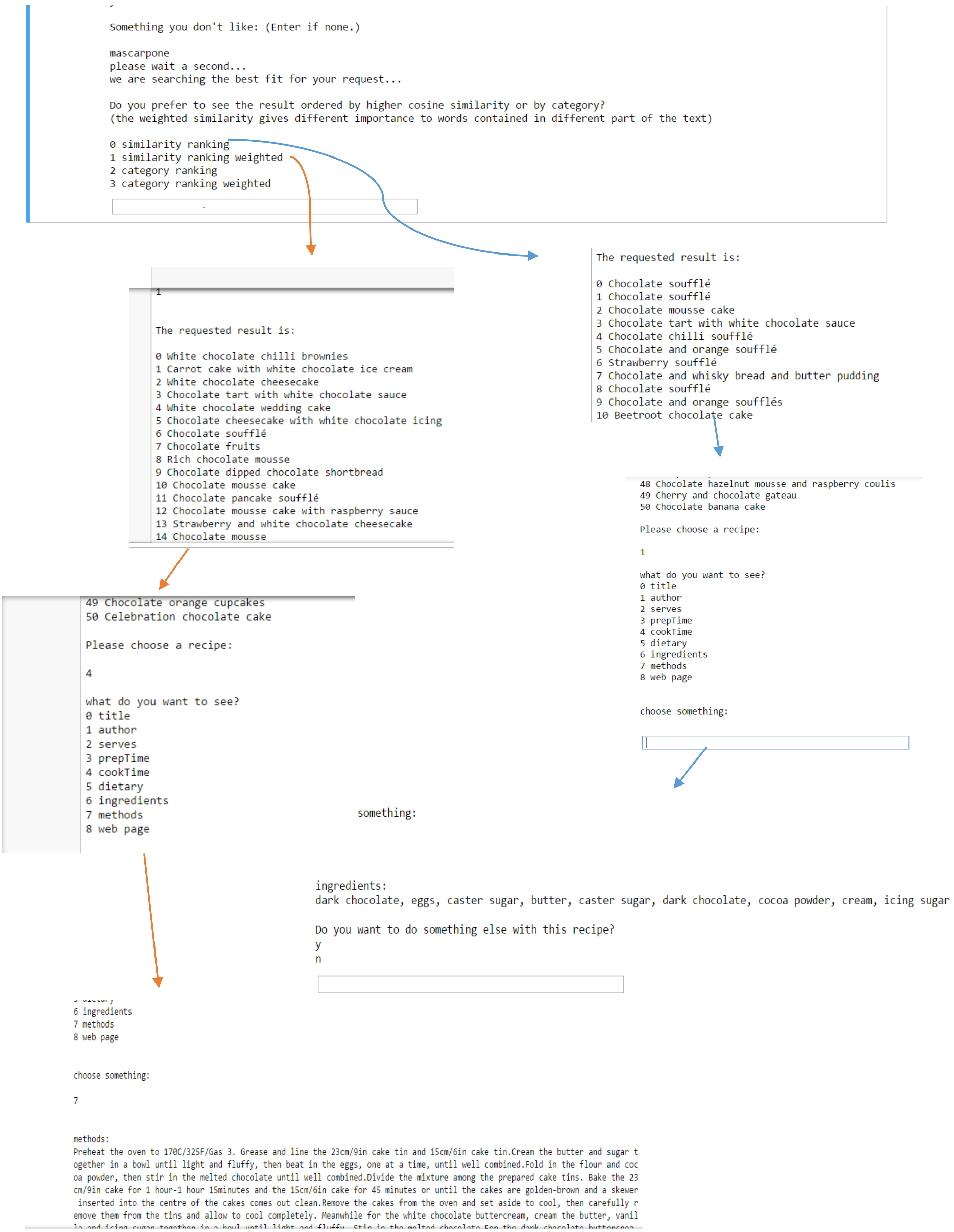
```
what do you want to see?
0 title
1 author
2 serves
3 prepTime
4 cookTime
5 dietary
6 ingredients
7 methods
8 web page
```

4.2. Some query and results (screenshots):



```
                print("y")
                print("n")

                while decision not in ["y", "n"]:

                    decision = input()

what do you want to do?

r new search
x exit
r


Please search something:

white chocolate
Do you want only results containing all your request?
y
n

y
```

QUERY :WHITE CHOCOLATE



```
Something you don't like: (Enter if none.)

mascarpone
please wait a second...
we are searching the best fit for your request...

Do you prefer to see the result ordered by higher cosine similarity or by category?
(the weighted similarity gives different importance to words contained in different part of the text)

0 similarity ranking
1 similarity ranking weighted
2 category ranking
3 category ranking weighted
```

```
1

The requested result is:

0 White chocolate chilli brownies
1 Carrot cake with white chocolate ice cream
2 White chocolate cheesecake
3 Chocolate tart with white chocolate sauce
4 White chocolate wedding cake
5 Chocolate cheesecake with white chocolate icing
6 Chocolate soufflé
7 Chocolate fruits
8 Rich chocolate mousse
9 Chocolate dipped chocolate shortbread
10 Chocolate mousse cake
11 Chocolate pancake soufflé
12 Chocolate mousse cake with raspberry sauce
13 Strawberry and white chocolate cheesecake
14 Chocolate mousse
```

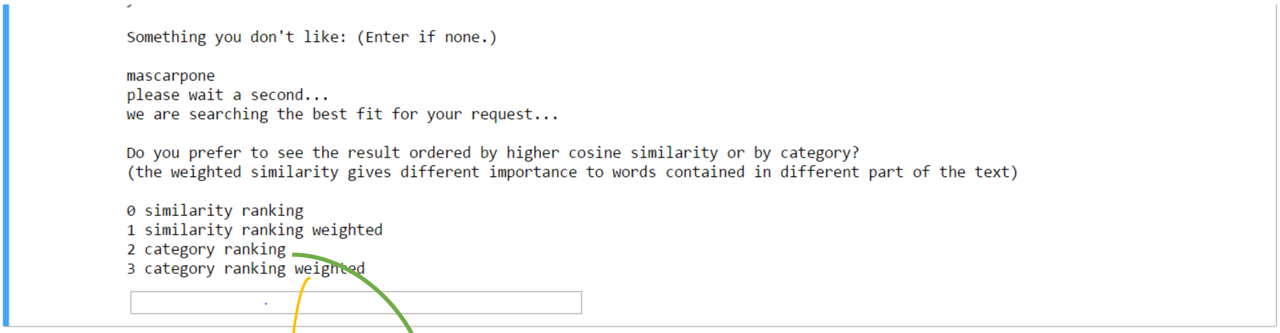
```
49 Chocolate orange cupcakes
50 Celebration chocolate cake

Please choose a recipe:

4

what do you want to see?
0 title
1 author
2 serves
3 prepTime
4 cookTime
5 dietary
6 ingredients
7 methods
8 web page
```

```
6 ingredients
7 methods
8 web page


choose something:

7


methods:
Preheat the oven to 170C/325F/Gas 3. Grease and line the 23cm/9in cake tin and 15cm/6in cake tin.Cream the butter and sugar t
ogether in a bowl until light and fluffy, then beat in the eggs, one at a time, until well combined.Fold in the flour and coc
oa powder, then stir in the melted chocolate until well combined.Divide the mixture among the prepared cake tins. Bake the 23
cm/9in cake for 1 hour-1 hour 15minutes and the 15cm/6in cake for 45 minutes or until the cakes are golden-brown and a skewer
 inserted into the centre of the cakes comes out clean.Remove the cakes from the oven and set aside to cool, then carefully r
emove them from the tins and allow to cool completely. Meanwhile for the white chocolate buttercream, cream the butter, vanil
la and icing sugar together in a bowl until light and fluffy. Stir in the melted chocolate.For the dark chocolate buttercrea
```

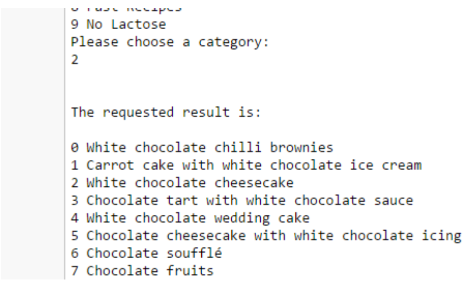
```
The requested result is:

0 Chocolate soufflé
1 Chocolate soufflé
2 Chocolate mousse cake
3 Chocolate tart with white chocolate sauce
4 Chocolate chilli soufflé
5 Chocolate and orange soufflé
6 Strawberry soufflé
7 Chocolate and whisky bread and butter pudding
8 Chocolate soufflé
9 Chocolate and orange soufflés
10 Beetroot chocolate cake
```

```
48 Chocolate hazelnut mousse and raspberry coulis
49 Cherry and chocolate gateau
50 Chocolate banana cake

Please choose a recipe:

1

what do you want to see?
0 title
1 author
2 serves
3 prepTime
4 cookTime
5 dietary
6 ingredients
7 methods
8 web page


choose something:
```

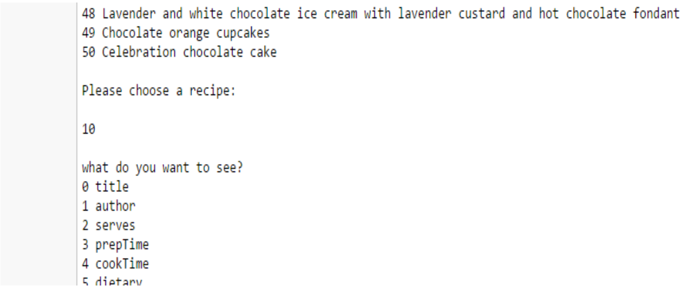
```
something:


ingredients:
dark chocolate, eggs, caster sugar, butter, caster sugar, dark chocolate, cocoa powder, cream, icing sugar

Do you want to do something else with this recipe?
y
n
```



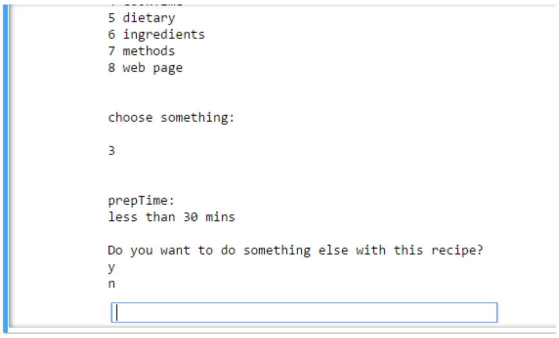
```
Something you don't like: (Enter if none.)

mascarpone
please wait a second...
we are searching the best fit for your request...

Do you prefer to see the result ordered by higher cosine similarity or by category?
(the weighted similarity gives different importance to words contained in different part of the text)

0 similarity ranking
1 similarity ranking weighted
2 category ranking
3 category ranking weighted
```



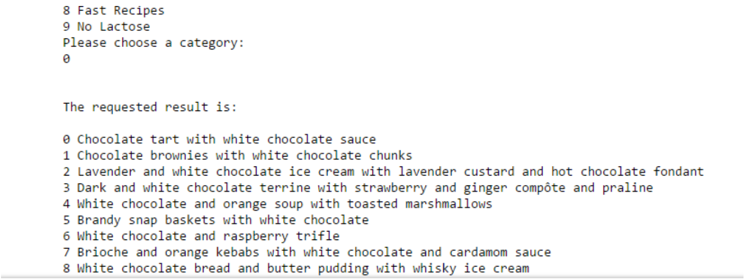
```
9 No Lactose
Please choose a category:
2


The requested result is:

0 White chocolate chilli brownies
1 Carrot cake with white chocolate ice cream
2 White chocolate cheesecake
3 Chocolate tart with white chocolate sauce
4 White chocolate wedding cake
5 Chocolate cheesecake with white chocolate icing
6 Chocolate soufflé
7 Chocolate fruits
```



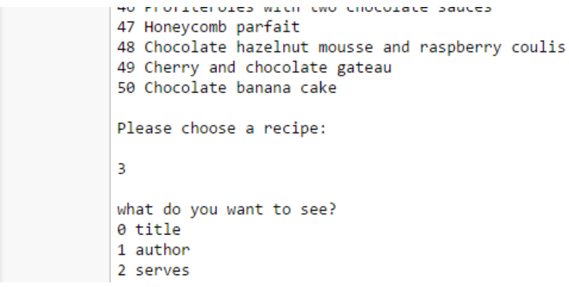
```
48 Lavender and white chocolate ice cream with lavender custard and hot chocolate fondant
49 Chocolate orange cupcakes
50 Celebration chocolate cake

Please choose a recipe:

10

what do you want to see?
0 title
1 author
2 serves
3 prepTime
4 cookTime
```



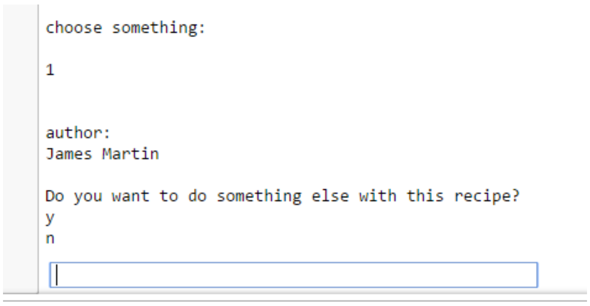
```
5 dietary
6 ingredients
7 methods
8 web page


choose something:

3


prepTime:
less than 30 mins

Do you want to do something else with this recipe?
y
n
```



```
8 Fast Recipes
9 No Lactose
Please choose a category:
0


The requested result is:

0 Chocolate tart with white chocolate sauce
1 Chocolate brownies with white chocolate chunks
2 Lavender and white chocolate ice cream with lavender custard and hot chocolate fondant
3 Dark and white chocolate terrine with strawberry and ginger compôte and praline
4 White chocolate and orange soup with toasted marshmallows
5 Brandy snap baskets with white chocolate
6 White chocolate and raspberry trifle
7 Brioche and orange kebabs with white chocolate and cardamom sauce
8 White chocolate bread and butter pudding with whisky ice cream
```

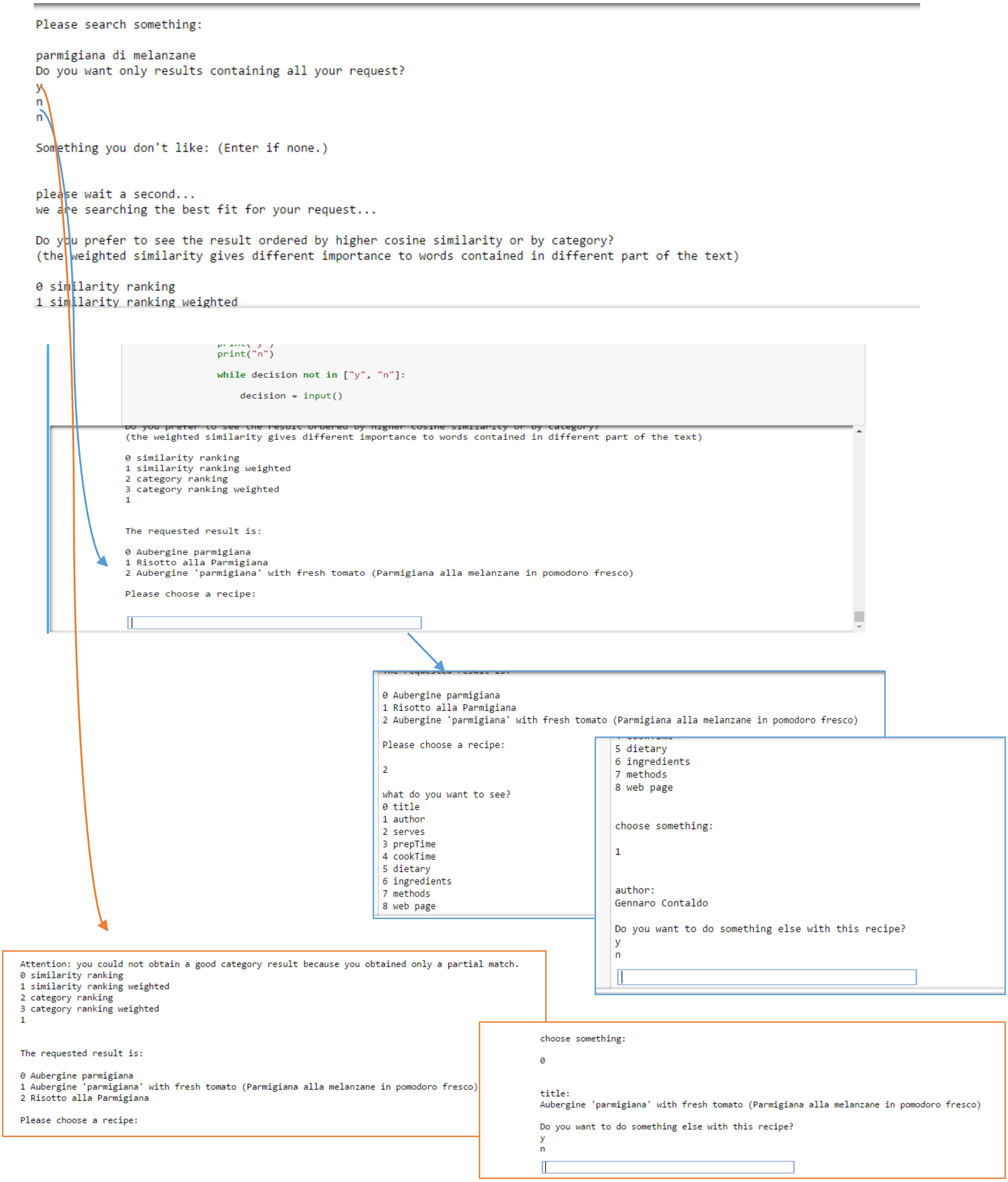


```
47 Honeycomb parfait
48 Chocolate hazelnut mousse and raspberry coulis
49 Cherry and chocolate gateau
50 Chocolate banana cake

Please choose a recipe:

3

what do you want to see?
0 title
1 author
2 serves
```



```
choose something:

1


author:
James Martin

Do you want to do something else with this recipe?
y
n
```

QUERY: PARMIGIANA DI MELANZANE



```
Please search something:

parmigiana di melanzane
Do you want only results containing all your request?
y
n
n

Something you don't like: (Enter if none.)


please wait a second...
we are searching the best fit for your request...

Do you prefer to see the result ordered by higher cosine similarity or by category?
(the weighted similarity gives different importance to words contained in different part of the text)

0 similarity ranking
1 similarity ranking weighted
```

```
                print("y")
                print("n")

                while decision not in ["y", "n"]:

                    decision = input()
```

```
Do you prefer to see the result ordered by higher cosine similarity or by category?
(the weighted similarity gives different importance to words contained in different part of the text)

0 similarity ranking
1 similarity ranking weighted
2 category ranking
3 category ranking weighted
1


The requested result is:

0 Aubergine parmigiana
1 Risotto alla Parmigiana
2 Aubergine 'parmigiana' with fresh tomato (Parmigiana alla melanzane in pomodoro fresco)

Please choose a recipe:
```

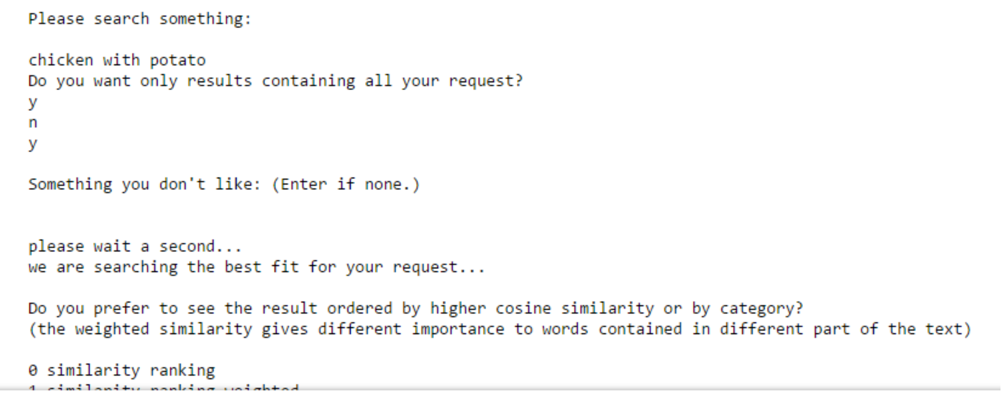
```
The requested result is:

0 Aubergine parmigiana
1 Risotto alla Parmigiana
2 Aubergine 'parmigiana' with fresh tomato (Parmigiana alla melanzane in pomodoro fresco)

Please choose a recipe:

2

what do you want to see?
0 title
1 author
2 serves
3 prepTime
4 cookTime
5 dietary
6 ingredients
7 methods
8 web page
```

```
4 cookTime
5 dietary
6 ingredients
7 methods
8 web page


choose something:

1


author:
Gennaro Contaldo

Do you want to do something else with this recipe?
y
n
```

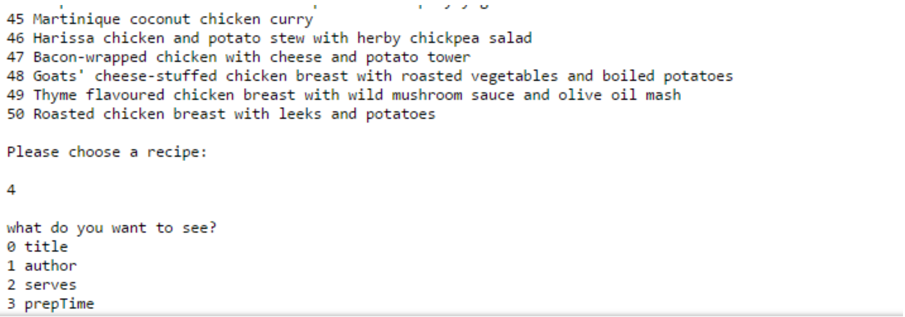
```
Attention: you could not obtain a good category result because you obtained only a partial match.
0 similarity ranking
1 similarity ranking weighted
2 category ranking
3 category ranking weighted
1


The requested result is:

0 Aubergine parmigiana
1 Aubergine 'parmigiana' with fresh tomato (Parmigiana alla melanzane in pomodoro fresco)
2 Risotto alla Parmigiana

Please choose a recipe:
```

```
choose something:

0


title:
Aubergine 'parmigiana' with fresh tomato (Parmigiana alla melanzane in pomodoro fresco)

Do you want to do something else with this recipe?
y
n
```

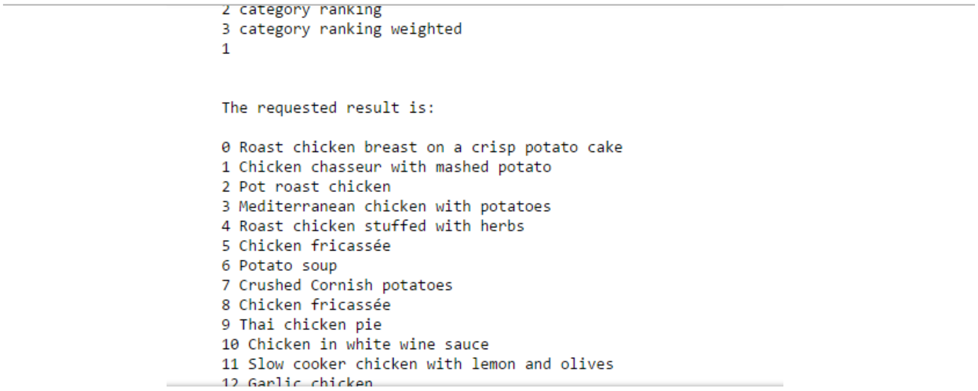
QUERY : CHICKEN AND POTATO



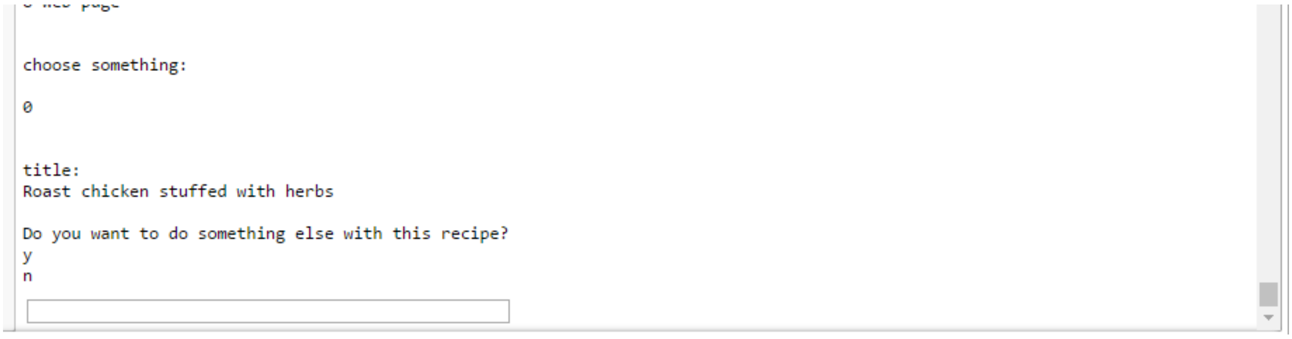
```
Please search something:

chicken with potato
Do you want only results containing all your request?
y
n
y

Something you don't like: (Enter if none.)


please wait a second...
we are searching the best fit for your request...

Do you prefer to see the result ordered by higher cosine similarity or by category?
(the weighted similarity gives different importance to words contained in different part of the text)

0 similarity ranking
1 similarity ranking weighted
```



```
45 Martinique coconut chicken curry
46 Harissa chicken and potato stew with herby chickpea salad
47 Bacon-wrapped chicken with cheese and potato tower
48 Goats' cheese-stuffed chicken breast with roasted vegetables and boiled potatoes
49 Thyme flavoured chicken breast with wild mushroom sauce and olive oil mash
50 Roasted chicken breast with leeks and potatoes

Please choose a recipe:

4

what do you want to see?
0 title
1 author
2 serves
3 prepTime
```



```
2 category ranking
3 category ranking weighted
1


The requested result is:

0 Roast chicken breast on a crisp potato cake
1 Chicken chasseur with mashed potato
2 Pot roast chicken
3 Mediterranean chicken with potatoes
4 Roast chicken stuffed with herbs
5 Chicken fricassée
6 Potato soup
7 Crushed Cornish potatoes
8 Chicken fricassée
9 Thai chicken pie
10 Chicken in white wine sauce
11 Slow cooker chicken with lemon and olives
12 Garlic chicken
```



```
6 web page


choose something:

0


title:
Roast chicken stuffed with herbs

Do you want to do something else with this recipe?
y
n
```