04-WordCount

August 10, 2020

1 Wordcount

- Wikipedia
- Word count example reads text files and counts how often words occur.
- Word count is commonly used by translators to determine the price for the translation job.
- This is the "Hello World" program of Big Data.

Some recommendations: - *Don't google too much, ask me or use the python documentation through* ***help*** *function. - Do not try to find a clever or optimized solution, do something that works before. - Please don't get the solution from your colleagues - Notebooks will be updated next week with solutions*

1.1 Create sample text file

[1]:
```
from lorem import text

with open("sample.txt", "w") as f:
    for i in range(10000):
        f.write(text())
```

1.1.1 Exercise 4.1

Write a python program that counts the number of lines, words and characters in that file.

[2]:
```
%%bash
wc sample.txt
du -h sample.txt
```

```
  69936  2010172 14203468 sample.txt
14M     sample.txt
```

- Compute number of lines

[3]:
```
with open("sample.txt") as f:
    lines = list(f)
```

```
nlines = len(lines)
nlines
```

[3]: 69937

- Compute number of words

[4]:
```
nwords = sum([len(line.split()) for line in lines])
nwords
```

[4]: 2010172

[5]:
```
nchars = 0
for line in lines:
    words = line.split()
    nchars += sum([len(word) for word in line.split()])

nchars
```

[5]: 12158329

- `set` gives the list of unique elements from words list.

[6]:
```
s = set(words)
s
```

[6]:
```
{'Dolore',
 'Est',
 'Neque',
 'Sed',
 'adipisci',
 'aliquam',
 'amet',
 'dolor',
 'dolore',
 'dolorem',
 'eius',
 'etincidunt',
 'ipsum',
 'magnam',
 'modi',
 'non',
 'numquam',
 'numquam.',
 'quaerat.',
 'quiquia',
 'quisquam',
```

```
 'sed.',
 'ut.',
 'voluptatem'}
```

1.1.2 Exercise 4.2

Create a function called `map_words` that take a file name as argument and return a lists containing all words as items.

```
map_words("sample.txt")[:5] # first five words
['adipisci', 'adipisci', 'adipisci', 'adipisci', 'adipisci']
```

[7]:
```
def map_words(filename):
    """ take a file name as argument and return a
    lists containing all words as item
    """
    with open(filename) as f:
        data = f.read().lower().replace('.',' ')

    return sorted(data.split())

map_words("sample.txt")[:5]
```

[7]:
```
['adipisci', 'adipisci', 'adipisci', 'adipisci', 'adipisci']
```

1.2 Sorting a dictionary by value

By default, if you use `sorted` function on a `dict`, it will use keys to sort it. To sort by values, you can use operator.itemgetter(1) Return a callable object that fetches item from its operand using the operand's `__getitem__(` method. It could be used to sort results.

[8]:
```
import operator
fruits = [('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]
getcount = operator.itemgetter(1)
dict(sorted(fruits, key=getcount))
```

[8]:
```
{'orange': 1, 'banana': 2, 'apple': 3, 'pear': 5}
```

`sorted` function has also a `reverse` optional argument.

[9]:
```
dict(sorted(fruits, key=getcount, reverse=True))
```

[9]:
```
{'pear': 5, 'apple': 3, 'banana': 2, 'orange': 1}
```

1.2.1 Exercise 4.3

Create a function `reduce` to reduce the list of words returned by `map_words` and return a dictionary containing all words as keys and number of occurrences as values.

```
wordcount('sample.txt')
{'tempora': 2, 'non': 1, 'quisquam': 1, 'amet': 1, 'sit': 1}
```

[10]:
```python
def reduce(sorted_words):
    " Compute word occurences from sorted list of words"

    res = {}
    current_word = None
    for word in sorted_words:
        if word == current_word:
            res[word] += 1
        else:
            res[word] = 1
            current_word = word
    return dict(sorted(res.items(), key=lambda v:v[1], reverse=True))

reduce(map_words("sample.txt"))
```

[10]:
```
{'dolorem': 75332,
 'non': 75321,
 'dolore': 75287,
 'porro': 75224,
 'consectetur': 75188,
 'numquam': 75157,
 'neque': 75049,
 'tempora': 75028,
 'labore': 74940,
 'adipisci': 74935,
 'sed': 74913,
 'voluptatem': 74874,
 'ipsum': 74858,
 'ut': 74839,
 'modi': 74806,
 'sit': 74771,
 'est': 74711,
 'amet': 74640,
 'dolor': 74640,
 'quisquam': 74632,
 'quiquia': 74598,
 'magnam': 74573,
 'quaerat': 74541,
 'eius': 74519,
 'etincidunt': 74346,
```

```
 'aliquam': 74317,
 'velit': 74132}
```

- `reduce` function using python exception KeyError

[11]:
```
def reduce(sorted_words):
    " Compute word occurences from sorted list of words"

    res = {}
    for word in sorted_words:
        try:
            res[word] += 1
        except KeyError:
            res[word] = 1

    return dict(sorted(res.items(), key=lambda v:v[1], reverse=True))

reduce(map_words("sample.txt"))
```

[11]:
```
{'dolorem': 75332,
 'non': 75321,
 'dolore': 75287,
 'porro': 75224,
 'consectetur': 75188,
 'numquam': 75157,
 'neque': 75049,
 'tempora': 75028,
 'labore': 74940,
 'adipisci': 74935,
 'sed': 74913,
 'voluptatem': 74874,
 'ipsum': 74858,
 'ut': 74839,
 'modi': 74806,
 'sit': 74771,
 'est': 74711,
 'amet': 74640,
 'dolor': 74640,
 'quisquam': 74632,
 'quiquia': 74598,
 'magnam': 74573,
 'quaerat': 74541,
 'eius': 74519,
 'etincidunt': 74346,
 'aliquam': 74317,
 'velit': 74132}
```

You probably notice that this simple function is not easy to implement. Python standard library

provides some features that can help.

1.3 Container datatypes

`collection` module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, `dict`, `list`, `set`, and `tuple`.

- `defaultdict` : dict subclass that calls a factory function to supply missing values
- `Counter` : dict subclass for counting hashable objects

1.3.1 defaultdict

When you implement the `wordcount` function you probably had some problem to append key-value pair to your `dict`. If you try to change the value of a key that is not present in the dict, the key is not automatically created.

You can use a `try-except` flow but the `defaultdict` could be a solution. This container is a `dict` subclass that calls a factory function to supply missing values. For example, using list as the default_factory, it is easy to group a sequence of key-value pairs into a dictionary of lists:

[12]:
```
from collections import defaultdict
s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
d = defaultdict(list)
for k, v in s:
    d[k].append(v)

dict(d)
```

[12]:
```
{'yellow': [1, 3], 'blue': [2, 4], 'red': [1]}
```

1.3.2 Exercise 4.4

- Modify the `reduce` function you wrote above by using a defaultdict with the most suitable factory.

[13]:
```
from collections import defaultdict

def reduce(sorted_words):
    " Reduce version using defaultdict, we use factory `int`"
    res = defaultdict(int)
    for word in sorted_words:
        res[word] += 1

    return dict(sorted(res.items(), key=lambda v:v[1], reverse=True))

reduce(map_words("sample.txt"))
```

[13]:
```
{'dolorem': 75332,
 'non': 75321,
 'dolore': 75287,
 'porro': 75224,
 'consectetur': 75188,
 'numquam': 75157,
 'neque': 75049,
 'tempora': 75028,
 'labore': 74940,
 'adipisci': 74935,
 'sed': 74913,
 'voluptatem': 74874,
 'ipsum': 74858,
 'ut': 74839,
 'modi': 74806,
 'sit': 74771,
 'est': 74711,
 'amet': 74640,
 'dolor': 74640,
 'quisquam': 74632,
 'quiquia': 74598,
 'magnam': 74573,
 'quaerat': 74541,
 'eius': 74519,
 'etincidunt': 74346,
 'aliquam': 74317,
 'velit': 74132}
```

1.3.3 Counter

A Counter is a dict subclass for counting hashable objects. It is an unordered collection where elements are stored as dictionary keys and their counts are stored as dictionary values. Counts are allowed to be any integer value including zero or negative counts.

Elements are counted from an iterable or initialized from another mapping (or counter):

[14]:
```
from collections import Counter

violet = dict(r=23,g=13,b=23)
print(violet)
cnt = Counter(violet)  # or Counter(r=238, g=130, b=238)
print(cnt['c'])
print(cnt['r'])
```

```
{'r': 23, 'g': 13, 'b': 23}
0
23
```

[15]:
```
print(*cnt.elements())
```

```
r r r r r r r r r r r r r r r r r r r r r r r r g g g g g g g g g g g g g b b b b
b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b b
```

[16]:
```
cnt.most_common(2)
```

[16]:
```
[('r', 23), ('b', 23)]
```

[17]:
```
cnt.values()
```

[17]:
```
dict_values([23, 13, 23])
```

1.3.4 Exercise 4.5

Use a `Counter` object to count words occurences in the sample text file.

[18]:
```
from collections import Counter

def wordcounter(filename):

    " Wordcount function using the Counter type from collections"

    with open(filename) as f:
        data = f.read()

    c = Counter(data.lower().replace("."," ").split())
    return dict(c.most_common())

wordcounter("sample.txt")
```

[18]:
```
{'dolorem': 75332,
 'non': 75321,
 'dolore': 75287,
 'porro': 75224,
 'consectetur': 75188,
 'numquam': 75157,
 'neque': 75049,
 'tempora': 75028,
 'labore': 74940,
 'adipisci': 74935,
 'sed': 74913,
 'voluptatem': 74874,
 'ipsum': 74858,
 'ut': 74839,
 'modi': 74806,
```

```
 'sit': 74771,
 'est': 74711,
 'amet': 74640,
 'dolor': 74640,
 'quisquam': 74632,
 'quiquia': 74598,
 'magnam': 74573,
 'quaerat': 74541,
 'eius': 74519,
 'etincidunt': 74346,
 'aliquam': 74317,
 'velit': 74132}
```

The Counter class is similar to bags or multisets in some Python libraries or other languages. We will see later how to use Counter-like objects in a parallel context.

1.4 Process multiple files

- Create several files containing `lorem` text named 'sample01.txt', 'sample02.txt'...
- If you process these files you return multiple dictionaries.
- You have to loop over them to sum occurences and return the resulted dict. To iterate on specific mappings, Python standard library provides some useful features in `itertools` module.
- itertools.chain(*mapped_values) could be used for treating consecutive sequences as a single sequence.

[19]:
```
import itertools, operator
fruits = [('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]
vegetables = [('endive', 2), ('spinach', 1), ('celery', 5), ('carrot', 4)]
getcount = operator.itemgetter(1)
dict(sorted(itertools.chain(fruits,vegetables), key=getcount))
```

[19]:
```
{'orange': 1,
 'spinach': 1,
 'banana': 2,
 'endive': 2,
 'apple': 3,
 'carrot': 4,
 'pear': 5,
 'celery': 5}
```

1.4.1 Exercise 4.6

- Write the program that creates files, processes and use `itertools.chain` to get the merged word count dictionary.

[20]:
```
import lorem

for i in range(4): # write 4 sample text files
    with open(f"sample{i:02d}.txt", "w") as f:
        f.write(lorem.text())
```

[21]:
```
from glob import glob

samples = glob("*.txt")
```

[22]:
```
from itertools import chain

words1 = map_words("sample01.txt")
words2 = map_words("sample02.txt")

reduce(chain(words1,words2)) # word count on two files
```

[22]:
```
{'dolore': 18,
 'magnam': 18,
 'adipisci': 15,
 'est': 14,
 'labore': 14,
 'sed': 13,
 'aliquam': 10,
 'eius': 10,
 'ipsum': 10,
 'modi': 10,
 'quisquam': 10,
 'sit': 10,
 'tempora': 10,
 'amet': 9,
 'etincidunt': 9,
 'neque': 9,
 'non': 9,
 'voluptatem': 9,
 'consectetur': 7,
 'dolor': 7,
 'porro': 7,
 'ut': 7,
 'dolorem': 6,
 'numquam': 6,
 'velit': 6,
 'quaerat': 5,
 'quiquia': 5}
```

- wordcount on a list of files

[23]:
```
from itertools import chain
from glob import glob

reduce(chain(*[map_words(file) for file in glob("sample0*.txt")]))
```

[23]:
```
{'dolore': 90,
 'voluptatem': 89,
 'quisquam': 85,
 'magnam': 84,
 'sit': 83,
 'consectetur': 81,
 'dolor': 81,
 'adipisci': 79,
 'labore': 79,
 'sed': 79,
 'ipsum': 78,
 'non': 78,
 'tempora': 78,
 'etincidunt': 77,
 'eius': 76,
 'est': 76,
 'velit': 76,
 'ut': 74,
 'quiquia': 73,
 'amet': 72,
 'numquam': 72,
 'quaerat': 69,
 'dolorem': 68,
 'neque': 68,
 'modi': 67,
 'aliquam': 66,
 'porro': 57}
```

1.4.2 Exercise 4.7

- Create the `wordcount` function in order to accept several files as arguments and return the result dict.

`wordcount(file1, file2, file3, ...)`

Hint: arbitrary argument lists

- Example of use of arbitrary argument list and arbitrary named arguments.

[24]:
```
def func( *args, **kwargs):
    for arg in args:
        print(arg)
```

```
    print(kwargs)

func( "3", [1,2], "bonjour", x = 4, y = "y")
```

```
3
[1, 2]
bonjour
{'x': 4, 'y': 'y'}
```

[25]:
```
from itertools import chain
from glob import glob

def wordcount(*args): # arbitrary argument list

    # MAP
    mapped_values = []
    for filename in args:
        with open(filename) as f:
            data = f.read()
        words = data.lower().replace('.','').strip().split()
        mapped_values.append(sorted(words))

    # REDUCE
    return reduce(chain(*mapped_values))

wordcount(*glob("sample0*.txt"))
```

[25]:
```
{'dolore': 90,
 'voluptatem': 89,
 'quisquam': 85,
 'magnam': 84,
 'sit': 83,
 'consectetur': 81,
 'dolor': 81,
 'adipisci': 79,
 'labore': 79,
 'sed': 79,
 'ipsum': 78,
 'non': 78,
 'tempora': 78,
 'etincidunt': 77,
 'eius': 76,
 'est': 76,
 'velit': 76,
 'ut': 74,
 'quiquia': 73,
 'amet': 72,
```

```
'numquam': 72,
'quaerat': 69,
'dolorem': 68,
'neque': 68,
'modi': 67,
'aliquam': 66,
'porro': 57}
```