L2. Python

UCLA Masters of Applied Economics

Fall 2018

Melody Y. Huang

Recall from last week:

- White spaces matter!
- Indexing starts from 0
- Lists: `x = [[1,2,3], [4,5,6], [7,8,9]]`
- Dictionaries: `dict = {key: value}`

Attributes

- Python objects have **attributes**

- Example:
  - Strings have attributes like:
    - split: splits a string on a particular character
    - upper: converts everything to uppercase
    - find: looks for a specific element in the string

```
x = “Hello World!”
x.split(” ")
#Will return: “Hello”, “World”
```

Attributes (cont.)

- Other common attributes that are nice:
  - Lists have an attribute called “append”
    - Allows us to add things to the end list
- So how do we remember all the different attributes of an object?
  - www.google.com

Solution

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```
#Parse string into list by every space
s_parsed=s.split(" ")
#Total number of words:
len(s_parsed)
#To find how many times each word appears:
s_parsed.count("sit")
```

Solution (cont.)

```
Total number of words: 100
```

```
['Lorem', 'ipsum', 'dolor', 'sit', 'sit', 'amet,',
'consectetur', 'adipiscing', 'elit,', '', '', '', '', '',
'sed', 'do', 'eiusmod', 'tempor', 'incididunt', 'ut', 'labore',
'et', 'dolore', 'magna', 'aliqua.', '', '', '', '', '', 'Ut',
'enim', 'ad', 'minim', 'veniam,', 'quis', 'nostrud',
'exercitation', 'ullamco', 'laboris', '', '', '', '', '',
'nisi', 'ut', 'aliquip', 'ex', 'ea', 'commodo', 'consequat.',
'Duis', 'aute', 'irure', 'dolor', 'in', '', '', '', '', '',
'reprehenderit', 'in', 'voluptate', 'velit', 'esse', 'cillum',
'dolore', 'eu', 'fugiat', 'nulla', '', '', '', '', '',
'pariatur.', 'Excepteur', 'sint', 'occaecat', 'cupidatat',
'non', 'proident,', 'sunt', 'in', '', '', '', '', '', 'culpa',
'qui', 'officia', 'deserunt', 'mollit', 'anim', 'id', 'est',
'laborum.']
```

Solution (cont.)

```
set_strings = set(s_parsed)
len(set_strings)
dict_words ={x: s_parsed.count(x) for x in set_strings}
```

```
Total number of unique words: 65
{'': 30, 'magna': 1, 'qui': 1, 'pariatur.': 1, 'Ut': 1,
'cupidatat': 1, 'id': 1, 'ad': 1, 'cillum': 1, 'proident,': 1,
'irure': 1, 'elit,': 1, 'dolore': 2, 'deserunt': 1, 'minim': 1,
'labore': 1, 'officia': 1, 'amet,': 1, 'sit': 2, 'esse': 1,
'occaecat': 1, 'do': 1, 'ipsum': 1, 'Excepteur': 1, 'tempor': 1,
'consectetur': 1, 'nostrud': 1, 'consequat.': 1, 'aliqua.': 1,
'enim': 1, 'nulla': 1, 'quis': 1, 'anim': 1, 'dolor': 2,
'laborum.': 1, 'Duis': 1, 'Lorem': 1, 'sed': 1, 'aute': 1,
'exercitation': 1, 'non': 1, 'sint': 1, 'mollit': 1, 'laboris': 1,
'sunt': 1, 'adipiscing': 1, 'culpa': 1, 'ex': 1, 'ullamco': 1,
'ut': 2, 'incididunt': 1, 'commodo': 1, 'est': 1, 'et': 1, 'eu': 1,
'fugiat': 1, 'in': 3, 'velit': 1, 'eiusmod': 1, 'veniam,': 1,
'reprehenderit': 1, 'voluptate': 1, 'nisi': 1, 'aliquip': 1, 'ea':
1}
```

Solution (cont.)

- As it turns out, this isn't the most computationally efficient solution
- Why?
- Every time you call the count function, Python must iterate through the list to find how many times the word appears
- How many words are there in total?

Solution (cont.)

- As it turns out, this isn't the most computationally efficient solution
- Why?
- Every time you call the count function, Python must iterate through the list to find how many times the word appears
- How many words are there in total?
  - 99 words
  - 65 unique words

Solution (cont.)

- That means each time the count function is called, we iterate through 99 words
- How many times is the count function called?
  - 65 times
- This means we have go through 6,435 iterations!

Solution (cont.)

- A better alternative:
  - Construct a dictionary, where each unique word is attached to a count of 0
  - Iterate through all the words in the original string
  - Each time you get to the word, you add one to the counter in the dictionary

Solution (cont.)

```
dict_words2={x:0 for x in set_strings}
#Initialize all to be zero count
for i in s_parsed:
    dict_words2[i] = dict_words2[i]+1
```

- We now only iterate 99 times!
  - This is a lot fewer iterations