Midterm Cheatsheet

March 8, 2023

1 Python(lecture4-12)

1.1 Data Structure

1.1.1 creating

- creating a list: L = list() / L = []
- creating a tuple: T = tuple()/ t= ()
- creating a set: S = set() / S = {a,b,c}
- creating a dict: D = dict()/ D = {a:a, b:b, c:c}

1.1.2 Finding the index of an element

```
[ ]: L = [1,2,3,4,5,5,5]
     L.index(2)
```

1.1.3 Finding Frequency in a list

```
[53]: L.count(5)
```

```
[53]: 3
```

1.1.4 Operating on lists

```
[54]: L = ["a","b","c"]
      trying = [1,2,3,4,5]
      L + trying
```

```
[54]: ['a', 'b', 'c', 1, 2, 3, 4, 5]
```

```
[55]: L*2
```

```
[55]: ['a', 'b', 'c', 'a', 'b', 'c']
```

1.1.5 Sorting lists

- by default this is in an ascending order

- *When using sorted(), to apply the sorting changes to the list sorted(list) needs to be assigned back to the list==:*
- the key of sorting could be followed by functions defined by lambda, self-defined functions, or built-in method

[56]:
```
L = [[1,5,6],[6,4],[5,5],[5,1]]
L = sorted(L, key = lambda x:x[0])
L
```

[56]:
```
[[1, 5, 6], [5, 5], [5, 1], [6, 4]]
```

[57]:
```
L = sorted(L, key = len)
L
```

[57]:
```
[[5, 5], [5, 1], [6, 4], [1, 5, 6]]
```

or we can do

[58]:
```
L.sort(key = lambda x:x[1], reverse = True)
L
```

[58]:
```
[[5, 5], [1, 5, 6], [6, 4], [5, 1]]
```

1.1.6 Set comparisons(?)

[59]:
```
s1 = {'candice', 'jason', 'june'}
s2 = {'candice','june','victor', 'lily', 'suzy'}
```

Union

[60]:
```
s1|s2
#or
s1.union(s2)
```

[60]:
```
{'candice', 'jason', 'june', 'lily', 'suzy', 'victor'}
```

Intersection

[61]:
```
s1.intersection(s2)
#or
s1&s2
```

[61]:
```
{'candice', 'june'}
```

Assymetric Difference

[62]:
```
print(s2.difference(s1)) # what is in s2 that is not in s1
#or
print(s2-s1) # what is in s2 that is not in s1
print(s1.difference(s2)) # what is in s1 that is not in s2
#or
print(s1-s2)# what is in s1 that is not in s2
```

```
{'victor', 'suzy', 'lily'}
{'victor', 'suzy', 'lily'}
{'jason'}
{'jason'}
```

Symmetric Difference

```
[63]: s1 ^ s2
print(s1.symmetric_difference(s2))
print(s2.symmetric_difference(s1))
```

```
{'jason', 'victor', 'suzy', 'lily'}
{'jason', 'victor', 'suzy', 'lily'}
```

1.1.7 Dictionary

```
[64]: d={'james':'two thousands', 'sedra':2500, 'alma':[200, 500, 700], 0:123,␣
 ↪('hasan','aljabbouli'):{'age':43,'salary':3000,'kids':3}}
```

*Retriving elements in the dictionary:

```
[65]: list(d.items())
list(d.keys())
list(d.values())
```

```
[65]: ['two thousands',
 2500,
 [200, 500, 700],
 123,
 {'age': 43, 'salary': 3000, 'kids': 3}]
```

- Getting the number of items

```
[66]: len(list(d.items()))
```

```
[66]: 5
```

```
[67]: min([2,3,4])
```

```
[67]: 2
```

1.1.8 Standard practice for finding frequency

[68]:
```
data=['james','hasan','james','hasan','hasan','alma','sedra',␣
↪'sedra','hasan','james','james']
```

[69]:
```
# find the frequencies
f={}
for name in data:
    if name not in f:
        f[name]=1
    else:
        f[name]+=1
f
```

[69]:
```
{'james': 4, 'hasan': 4, 'alma': 1, 'sedra': 2}
```

1.1.9 Iterators

[70]:
```
names=['james','alma','sedra','hala','hana','hasan']
i=iter(names)
next(i) # iterate the 1st ele
```

[70]:
```
'james'
```

[71]:
```
next(i) # iterate the 2nd ele
```

[71]:
```
'alma'
```

[72]:
```
list(i) # the rest to be iterated; if it reaches the last element it will be an␣
↪empty list
```

[72]:
```
['sedra', 'hala', 'hana', 'hasan']
```

1.1.10 Deep and Shallow copy

1. Deep vs. virtually deep copy

[73]:
```
import copy

original_list = [1, 2, 3, [4, 5]]
new_list = list(original_list)
new_list[3].append(6)

print(original_list) # Output: [1, 2, 3, [4, 5, 6]]
print(new_list) # Output: [1, 2, 3, [4, 5, 6]]

original_object = [1, 2, [3, 4]]
new_object = copy.deepcopy(original_object)
```

```
original_object[2].append(5)

print(original_object) # Output: ~~[1, 2, [3, 4]]~~
print(new_object) # Output: ~~[1, 2, [3, 4, 5]]~~
```

mistaken

```
[1, 2, 3, [4, 5, 6]]
[1, 2, 3, [4, 5, 6]]
[1, 2, [3, 4, 5]]
[1, 2, [3, 4]]
```

2. Virtually deep copy vs Shallow copy

- Virtual deep copy is different from shallow copy in that it creates a different object(using seperate memory space) which is the same as the original list. It is not *pointing* toward the same list as shallow copy would do.
  - Therefore, altering the virtual deep copy will not affect the original list while shallow copy does

[74]:
```
original_list = [1, 2, 3]
new_list_1 = original_list
new_list_2 = list(original_list)

#new_list_1.append(4)
new_list_2.append(5)

print(original_list) # Output: [1, 2, 3, 4]
print(new_list_1) # Output: [1, 2, 3, 4]
print(new_list_2) # Output: [1, 2, 3, 5]
```

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 3, 5]
```

1.1.11 Map

Basic Syntax:

- map(function, input data to be mapped) -function could be lambda, built-in functions, or self-defined functions
- *note that the result has to be made into a list in order for the results of mapping to be displayed.*

[75]:
```
data=['james','hasan','james','hasan','hasan','alma','sedra',␣
 ↪'sedra','hasan','james','james']
```

[76]:
```
map(len, data)
```

[76]:
```
<map at 0x7fd8f5dd2020>
```

[77]:
```
list(map(len, data))
```

[77]:
```
[5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5]
```

Using self-defined functions:

[78]:
```
salaries=[2000,3000,4000,1500,2500]
def increase_by_100(s):
    return s+100
list(map(increase_by_100,salaries))
# increase each salary by 100:
list(map(lambda s:s+100,salaries))
# or
list(map(increase_by_100,salaries))
```

[78]:
```
[2100, 3100, 4100, 1600, 2600]
```

1.1.12 Mapping practices

ex1: write one line code that gives the minimum salary of all employees

[79]:
```
Employees={'james':[20, 2000, 3], 'alma':[25, 2500, 1], 'hana':[22, 1900, 0],␣
↪'hasan':[27, 2400, 3]}
```

[80]:
```
min(list(map(lambda x:x[1], Employees.values())))
```

[80]:
```
1900
```

[81]:
```
min(list(map(lambda x:x[1][1], Employees.items())))
```

[81]:
```
1900
```

write one line code to increase the salary of each employee by 100 for each child

ex2: write one line code to increase the salary of each employee by 100 for each child

[82]:
```
Employees={'james':{'age':20,'salary':2000,'kids':3},
           'alma':{'age':25,'salary':2500,'kids':1},
           'hana':{'age':22,'salary':1900,'kids':0},
           'hasan':{'age':27,'salary':2400,'kids':3}}
```

[83]:
```
list(map(lambda x:x['salary'] + x['kids']*100, Employees.values()))
```

[83]:
```
[2300, 2600, 1900, 2700]
```

[84]:
```
dict(list((map(lambda x:(x[0],{'age':x[1]['age'],'salary':x[1]['salary']␣
↪+x[1]['kids']*100, 'kids':x[1]['kids']}), Employees.items()))))
```

[84]:
```
{'james': {'age': 20, 'salary': 2300, 'kids': 3},
 'alma': {'age': 25, 'salary': 2600, 'kids': 1},
 'hana': {'age': 22, 'salary': 1900, 'kids': 0},
 'hasan': {'age': 27, 'salary': 2700, 'kids': 3}}
```

IMPORTANT: to write mapping results into a dictionary, we can first write lambda function to a tuple, and than convert it to a dictionary

Tuple to dict syntax

- dict((key1, value1),(key2, value2))
- write nested dictionaries in dict form!!

[85]:
```
two_babes = dict((('a',{'age':1}),('b',{'age':2})))
```

[86]:
```
#increasing each babe's age by 10
```

[87]:
```
two_old_babes = dict(map(lambda x:(x[0],{'age':x[1]['age']+10}), two_babes.
 ↪items()))
two_old_babes
```

[87]:
```
{'a': {'age': 11}, 'b': {'age': 12}}
```

Yay! Now they are older.

1.1.13 Filter

The Difference Between Filter and Map

- *map returns booleans because it operates the comparison assigned by lambda to x, while filter filters out the False ones*

[88]:
```
Employees=[['james',20,2000],['mike',23, 2500], ['alma', 25, 3000], ['sarah',␣
 ↪19, 1900]]
```

[89]:
```
list(map(lambda x:len(x[0]) > 4, Employees))
```

[89]:
```
[True, False, False, True]
```

[90]:
```
list(filter(lambda x:len(x[0]) > 4, Employees))
```

[90]:
```
[['james', 20, 2000], ['sarah', 19, 1900]]
```

Exercises

1. write one line code to convert the List Employees into a dict

```
[91]: emp_dict = dict(map(lambda x:(x[0],{'age':x[1],'salary':x[2]}), Employees))
```

2. filter items in the dictionary and get only employees with salary >=2500

```
[92]: list(filter(lambda x:emp_dict[x]['salary'] >=2500, emp_dict.keys()))
      #but this only gives the name
```

```
[92]: ['mike', 'alma']
```

```
[93]: #or
      list(filter(lambda e:e[1]['salary']>=2500, emp_dict.items()))
```

```
[93]: [('mike', {'age': 23, 'salary': 2500}), ('alma', {'age': 25, 'salary': 3000})]
```

3. write one line code to get the average salary of all employees

```
[94]: sum(list(map(lambda x:x['salary'], emp_dict.values())))/len(emp_dict.items())
```

```
[94]: 2350.0
```

```
[95]: #or
      sum(list(map(lambda e:e[1]['salary'],emp_dict.items())))/len(emp_dict)
```

```
[95]: 2350.0
```

1.1.14 Zip

- think of it as a way to merge a column name and a value in the column into *a tuple*
  - for instance ('james',20) is the outcome of zipping ['james', 'mike'] and [20,25]

```
[96]: L1=['james','mike','alma', 'sarah']
      L2=[20,30,19]
      L3=[2000,3000,1900]
```

```
[97]: result=list(zip(L1, L2, L3))
      result
```

```
[97]: [('james', 20, 2000), ('mike', 30, 3000), ('alma', 19, 1900)]
```

```
[98]: print(type(result))
      result
```

```
<class 'list'>
```

```
[98]: [('james', 20, 2000), ('mike', 30, 3000), ('alma', 19, 1900)]
```

unzipping

```
[102]: names, ages, salaries= list(zip(*result))
```

```
[103]: names
```

```
[103]: ('james', 'mike', 'alma')
```

```
[104]: ages
```

```
[104]: (20, 30, 19)
```

```
[105]: salaries
```

```
[105]: (2000, 3000, 1900)
```

1.2 Classes

```
[116]: class person:
           counter=0
           def __init__(self, n='', a=0):
               self.name=n
               self.age=a
               person.counter+=1;

           def intro(self):
               return "I am a preson"

           def add(self, n1, n2):
               return n1+n2

           def reset_counter():
               person.counter=0

           def __str__(self):
               return "I am a person, my name is: " + self.name + " and my age is:␣
        ↪"+str(self.age)
```

```
[117]: p1=person('james', 20)
```

```
[118]: p2=person()
       p3=person()
```

```
[119]: person.reset_counter()
```

```
[120]: person.counter
```

```
[120]: 0
```

```
[121]: print(p1)
```

```
I am a person, my name is: james and my age is: 20
```

```
[122]: print(p2)
```

```
I am a person, my name is:  and my age is: 0
```

```
[123]: str(p1)
```

```
[123]: 'I am a person, my name is: james and my age is: 20'
```

```
[124]: p1.__str__()
```

```
[124]: 'I am a person, my name is: james and my age is: 20'
```

Example

```
[125]: Employees={'james': {'age': 20, 'salary': 2000},
        'mike': {'age': 23, 'salary': 2500},
        'alma': {'age': 25, 'salary': 3000},
        'sarah': {'age': 19, 'salary': 1900}}
```

```
[126]: Employees
```

```
[126]: {'james': {'age': 20, 'salary': 2000},
        'mike': {'age': 23, 'salary': 2500},
        'alma': {'age': 25, 'salary': 3000},
        'sarah': {'age': 19, 'salary': 1900}}
```

```
[127]: class employee:
            def __init__(self, name, age, salary):
                self.name=name
                self.age=age
                self.salary=salary
            def get_tax(self):
                return self.salary /self.age
```

```
[128]: e1=employee('james',20,2000)
       e2=employee('mike',23,2500)
       e3=employee('alma',25,3000)
```

```
[129]: Employees=[e1,e2,e3]
```

[130]:
```
list(map(lambda e:e.get_tax(),Employees))
```

[130]:
```
[100.0, 108.69565217391305, 120.0]
```

1.2.1 Inheritance

[130]:
```
class Student():
    s_n = 1
    def __init__(self,name='', age =20):
        Student.s_n +=1
        self.s_n = Student.s_n
```

[131]:
```
s1 = Student('M',21)
s1.s_n = 10
s2 = Student('A',21)
s3 = Student('P',21)
print(Student.s_n)
```

```
4
```

[132]:
```
e1=employee('james',20,2000)
e2=employee('mike',23,2500)
e3=employee('alma',25,3000)
```

example

[133]:
```
class person:
    '''This class can be used to .....'''
    # constructor
    def __init__(self, name="", age=0):
        self.name=name
        self.age=age
    def say_hi(self):
        print("hi...")
    def __str__(self):
        return "I am a person, my name is: "+self.name
```

[134]:
```
person?
```

[135]:
```
p1=person("james",20)
```

[136]:
```
p1.say_hi()
```

```
hi…
```

[137]:
```
print(p1)
```

```
I am a person, my name is: james
```

[138]:
```
str(p1)
```

[138]:
```
'I am a person, my name is: james'
```

[139]:
```
class employee(person):
    def __init__(self, name="", age=0, salary=0):
        super().__init__(name, age)
        self.salary=salary
    def __str__(self):
        return 'I am an employee..my salary is:'+str(self.salary)
```

[140]:
```
e1=employee("james",20, 2000)
```

[141]:
```
e1.salary
```

[141]:
```
2000
```

[142]:
```
print(e1)
```

```
I am an employee..my salary is:2000
```

1.2.2 Example-2

[143]:
```
class mylist(list):
    def first_last(self):
        return self[0]==self[-1]
```

[144]:
```
L1=[1,2,3,4,5]
L1=mylist([1,2,3,4,5])
L1.first_last()
```

[144]:
```
False
```

1.2.3 List Comprehensions

basic syntax

[145]:
```
salaries=[1000,2000,3000,5000,500,2500]
print([s+100 for s in salaries])
[s+100 if s>2000 else s for s in salaries]
```

```
[1100, 2100, 3100, 5100, 600, 2600]
```

[145]:
```
[1000, 2000, 3100, 5100, 500, 2600]
```

1.3 Exceptions

[146]:
```python
raise Exception("you did something wrong")
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Input In [146], in <cell line: 1>()
----> 1 raise Exception("you did something wrong")

Exception: you did something wrong
```

[147]:
```python
raise ValueError("you used  wrong value")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [147], in <cell line: 1>()
----> 1 raise ValueError("you used  wrong value")

ValueError: you used  wrong value
```

[148]:
```python
names=['james','alma','sedra','hasan','william','tara']
for n in names:
    if n=='hasan':
        raise Exception("I do not like hasan")
    print('hi', n)
```

```
hi james
hi alma
hi sedra
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Input In [148], in <cell line: 2>()
      2 for n in names:
      3     if n=='hasan':
----> 4         raise Exception("I do not like hasan")
      5     print('hi', n)

Exception: I do not like hasan
```

1.3.1 Handle errors

[149]:
```
# try:
#     # do something
# Except:
#     # do something else
```

Example-1

[150]:
```
def divide(n,m):
    result=0
    try:
        result= n/m
    except TypeError:
        try:
            result= divide(int(n),int(m))
        except:
            result= 'unable to divide'
    except ZeroDivisionError:
        result= 'Nan'
        print("zero division")
    except:
        result= 'unable to divide'
    else:
        print("everything was OK..")
    finally:
        print("Thank you..")
    return result
```

[151]:
```
divide(3,0)
```

```
zero division
Thank you..
```

[151]:
```
'Nan'
```

[152]:
```
divide(10,2)
```

```
everything was OK..
Thank you..
```

[152]:
```
5.0
```

[154]:
```
#try:
    # read data from main file
#except FileNotFound:
    # read data from the backup file
#except:
    # mail(admin)
#else:
    #add to the logfile(someone read the data successfully)
#finally:
    #add to the logfile(someone tried to acces the data at this time..)
```

1.3.2 Dictionary Comprehension

[155]:
```
employees=[('james',2000),('alma',2500),('mike',3000)]
{i[0]:i[1] for i in employees}
```

[155]:
```
{'james': 2000, 'alma': 2500, 'mike': 3000}
```

[156]:
```
names=['james','mike','alma']
{n:len(n) for n in names}
```

[156]:
```
{'james': 5, 'mike': 4, 'alma': 4}
```

1.3.3 Text Handling

- special characters
  - /n, /t, //
  - for /t, in string it is going to be recognized as a tab
  - so if we want to print out the “” when it happens to be followed by a”t”, we do “\”; or we use r-string:
    - Python raw string treats the backslash character () as a literal character.

[3]:
```
s='the first item/the second item'
print(s)
d = 'the first item//the second item'
print(d)
r=r'the first item/the second item' #r-string
print(r)
```

```
the first item/the second item
the first item//the second item
the first item/the second item
```

1.3.4 String Methods

[169]:
```
s="welcome everyone. today is        Tuesday"
```

upper and lower cases

[170]:
```
s.capitalize() # capitalize the first letter
```

[170]:
```
'Welcome everyone. today is        tuesday'
```

[171]:
```
s.title() # capitalize the first letter of every word
```

[171]:
```
'Welcome Everyone. Today Is        Tuesday'
```

[172]:
```
s.swapcase() # exchange lower and upper from original
```

[172]:
```
'WELCOME EVERYONE. TODAY IS        tUESDAY'
```

[195]:
```
#s.upper()
#s.lower()
```

Counting and Finding

[174]:
```
s.count('e')
```

[174]:
```
6
```

- string.find('searchitem', startposition, endposition(optional))
- the results returns the first occurence of the search item starting at the specified position

[194]:
```
s.find('e',10)  #find the first presence of 'e' starting from position 10
```

[194]:
```
10
```

Stripping

[212]:
```
s = ' some text       \n \t  '
```

[200]:
```
print(s.strip())
print(s.rstrip())
print(s.lstrip())
```

```
some text
 some text
some text
```

Replacing

[211]:
```
s.replace(' ', '')
```

[211]:
```
'sometext\n\t'
```

Using Replace to clean the text

[221]:
```
for punc in ' \n\t':
    if punc in s:
        s = s.replace(punc, '')
s
```

```
[221]: 'sometext'
```

1.3.5 Regular Expressions

```
[68]: import re
```

```
[224]: s="Hi, how are you today"
```

```
[235]: result = re.search('Hi',s)
       print(result)
```

```
<re.Match object; span=(0, 2), match='Hi'>
```

```
[236]: print(result.start()) # the starting index of the search item
       print(result.end()) # the ending index of the search item
       print(result.span()) # the span
```

```
0
2
(0, 2)
```

Using RE to see if a substring is in a string

```
[237]: re.search('Hi',s)!= None
```

```
[237]: True
```

```
[238]: re.search('Hello',s)!= None
```

```
[238]: False
```

Identifiers

```
[39]: # . = any character except new line
      print(re.search('a','aa')!=None)
      print(re.search('a.','aa')!=None)
      print(re.search('a','ab')!=None)
      print(re.search('a.','ab')!=None)
      # if a is followed by a new line it returns false
      print(re.search('a.','a\n')!=None)
```

```
True
True
True
True
False
```

```
[46]: # {} = match a range of number of times
print(re.search('ba{1,3}b','bab')!=None)
print(re.search('ba{1,3}b','baab')!=None)
print(re.search('ba{1,3}b','baaab')!=None)
print(re.search('ba{1,3}b','bb')!=None)
print(re.search('ba{1,3}b','baaaab')!=None)
print(re.search('b{0,1}ab','baaaabbbb')!=None)
# it seems like this identifier could only be used once in a search, if we put␣
  ↪more,
# only the first one will be evaluated, see the following example
print(re.search('b{0,1}ab{1,2}','baaaabbbb')!=None)
```

```
True
True
True
False
False
True
True
```

```
[10]: # ? = matches 0 or 1 repetitions
print(re.search('ba?b','bb')!=None)
print(re.search('ba?b','bab')!=None)
print(re.search('ba?b','baab')!=None)
```

```
True
True
False
```

```
[11]: # * = matches 0 or more repetitions
print(re.search('ba*b','bb')!=None)
print(re.search('ba*b','bab')!=None)
print(re.search('ba*b','baab')!=None)
print(re.search('ba*b','baaaaaaaaab')!=None)
print(re.search('ba*b','baaaaaagb')!=None)
```

```
True
True
True
True
False
```

```
[90]: # + = matches 1 or more
print(re.search('ba+b','bb')!=None)
print(re.search('ba+b','bab')!=None)
print(re.search('ba+b','baaaaab')!=None)
print(re.search('ba+b','baaaaagb')!=None)
```

```
False
True
True
False
```

[55]:
```
# ^ = matches start of a string
# it is placed to the left of the evaluated string!
print(re.search('^abc','abcd')!=None)
print(re.search('^a','abcabc')!=None)
print(re.search('^a',' abc')!=None)
```

```
True
True
False
```

[56]:
```
# $ = matches end of string
# it is placed to the right of the evaluated string!
print(re.search('a$','aba')!=None)
print(re.search('a$','abaaba')!=None)
print(re.search('a$','aba ')!=None)
```

```
True
True
False
```

COMBINED USECASES

[63]:
```
#start and end with 'a' with one of any character other than new line in-betwn
print(re.search('^a.a$','aba')!=None)
print(re.search('^a.a$','aaa')!=None)
print(re.search('^a.a$','acba')!=None) # more than 1 char in-between
```

```
True
True
False
```

[69]:
```
# start and end with 'a' with 0 or 1 any char in between
print(re.search('^a.?a$','aba')!=None)
print(re.search('^a.?a$','aa')!=None)
print(re.search('^a.?a$','aaaaaaa')!=None) # false because center chars exceed 1
```

```
True
True
False
```

Checking a string with specified beginning and end

[71]:
```
#anything that starts and ends with 'a'
# * is 0 or more so it maximizes the flexibility of middle char(s)
```

```
print(re.search('^a.*a$','awelcomea')!=None)
```

```
True
```

Important Differentiation between 2 RE

```
[49]: txt = "The Rain in Spain"
re.search("^The.*Spain$", txt)!=None
#the . here is used to specify any character other than \n in the middle
# so it matches
```

```
[49]: True
```

```
[50]: txt = "The Rain in Spain"
re.search("^The*Spain$", txt)!=None
#without . the * is used to find 0/1 occurence of "e", although 0 "e" is␣
  ↪present
#after "The" so this matches, but the evaluation cannot be passed if there are␣
  ↪other
#characters before the beginning and end!
```

```
[50]: False
```

Checking a string with specified middle-part length

```
[73]: # anything that starts and ends with 'a'
# any middle char(.) could has the length of 2-5 characters
print(re.search('^a.{2,5}a$','aba')!=None)
print(re.search('^a.{2,5}a$','ahia')!=None)
print(re.search('^a.{2,5}a$','ahelloa')!=None)
```

```
False
True
True
```

brackets

```
[81]: print(re.search('[123abc]','abcde')!=None)
print(re.search('[123abc]','defg')!=None)
print(re.search('[1-9]','a7bcde')!=None)
print(re.search('[1-3]','a7bcde')!=None)
print(re.search('[a-z]','a7bcde')!=None)
print(re.search('[A-Z]','a7bcde')!=None)
print(re.search('[1-8A-C]','a7bcde')!=None) # [1-3] or [a-c]
```

```
True
False
True
False
```

```
True
False
True
```

Parentheses

- Capture and group

[100]:
```python
print(re.search('(abc){1,3}','abc')!=None)
print(re.search('(abc){1,3}','abcabcabc')!=None)
print(re.search('^(abc){1,3}$','abcabcabcabc')!=None)
print(re.search('(abc){1}','abcabc')!=None)
print(re.search('^(abc){1,2}$','abcaabc')!=None)
print(re.search('^(abc){1,2}$','abcabc')!=None)
```

```
True
True
False
True
False
True
```

Vertical bar

- either or

[121]:
```python
print(re.search('abc|123','a')!=None)
print(re.search('abc|123','ab')!=None)
print(re.search('abc|123','abc')!=None)
print(re.search('abc|123','123')!=None)
```

```
False
False
True
True
```

Backslash & Special Characters

[83]:
```python
print(re.search('\?','how are you today')!=None)  # looking for ?
print(re.search('\+','how are you today')!=None)  # looking for +
print(re.search('\+?','how are you today')!=None)  # looking for 0 or 1 +
# check if a string contains white spaces:
print(re.search('\s','how are you today')!=None)
# check if a string contains no white space:
print(re.search('\S','howareyoutoday')!=None)
# check if a string has number
print(re.search('\d','how are you today')!=None)
# check if a string contains no number
print(re.search('\D','how are you today')!=None)
print(re.search('\W','how are you today')!=None)
```

```
False
False
True
True
True
False
True
True
```

Re.split() split whenever there's space:

```
[122]: txt="weclome..... today   is Tuesday,,,,,,,,, it    is cold     outside."
       print(txt.split(' '))
       re.split(' +',txt) # strip using 1 or more spaces
```

```
['weclome…', 'today', '', '', 'is', 'Tuesday,,,,,,,,', 'it', '', '', '',
'is', 'cold', '', '', '', '', 'outside.']
```

```
[122]: ['weclome…',
        'today',
        'is',
        'Tuesday,,,,,,,,',
        'it',
        'is',
        'cold',
        'outside.']
```

```
[121]: txt.split()
```

```
[121]: ['weclome…',
        'today',
        'is',
        'Tuesday,,,,,,,,',
        'it',
        'is',
        'cold',
        'outside.']
```

```
[141]: re.split('[\n ,\.]+',txt) #\. is period
```

```
[141]: ['weclome', 'today', 'is', 'Tuesday', 'it', 'is', 'cold', 'outside', '']
```

Re.findall

- different from search, this one returns what we find

```
[86]: txt="my name is hasan, I am  43 years old, and my salary is 2000!!"
      print(re.findall('[0-9]', txt))
      print(re.findall('[0-9]+', txt)) # the + allows us to extract the numbers as␣
        ↪they are
      print(re.findall('[a-z]+', txt)) # find all the words - this can also be used␣
        ↪to split
```

```
['4', '3', '2', '0', '0', '0']
['43', '2000']
['my', 'name', 'is', 'hasan', 'am', 'years', 'old', 'and', 'my', 'salary', 'is']
```

[96]:
```
txt = '1.5 2.5 10:05'
print(re.findall('\d\d?', txt))
```

```
['1', '5', '2', '5', '10', '05']
```

[98]:
```
ist = ['b','a','c']
''.join(ist[::-1])
```

[98]:
```
'cab'
```

[99]:
```
txt="""
- Bob has been working for IT department for 1 year.
- Sam has been working for Trade department for 3 years.
- Peter has been working for HR department for 8 years.
- Mike has been working for IT department for 2 years.
- Sarah has been working for Trade department for 2 years.
- Sali has been working for IT department for 7 years.
- Tim has been working for IT department for 12 years.
- Maya has been working for HR department for 10 years.
- Susan has been working for IT department for 7 years.
- Kati has been working for IT department for 11 years.
- Robert has been working for Trade department for 9 years.
- Shahd has been working for Trade department for 2 years.
- Ema has been working for IT department for 7 years.
- Hala has been working for HR department for 12 years.
- Joe has been working for HR department for 18 years.
- Ali has been working for HR department for 4 years.
- Nader has been working for IT department for 15 years.
- Kim has been working for Trade department for 5 years.
- Bilal has been working for Trade department for 2 years.
- Albert has been working for IT department for 7 years.
- Hind has been working for Trade department for 12 years.
- Alberto has been working for HR department for 10 years.
"""
```

[106]:
```
sorted([int(i) for i in re.findall('\d+', txt)], reverse = True)[0]
```

[106]:
```
18
```

```
[107]: l0 = [1,2]
       l1 = [1,2,3,l0]
       l2 = l1
       l2[-1].append(5)
       print(l0+l1)

       [1, 2, 5, 1, 2, 3, [1, 2, 5]]

[108]: l1 = [1,2,3,5]
       l2 = l1
       l2[-1] = 6
       l1[-2] = 10
       l1= l2
       print(l1)

       [1, 2, 10, 6]

[115]: l0 = [1,2]
       l2 =[8,9]
       l0.append(l2)
       l2[-1]=10
       l0

[115]: [1, 2, [8, 10]]

[159]: print(re.findall('\S+',txt)) #return everything that is not whitespace

       ['my', 'name', 'is', 'hasan,', 'I', 'am', '43', 'years', 'old,', 'and', 'my',
       'salary', 'is', '2000']

[161]: re.findall('\d+',txt) # return every number

[161]: ['43', '2000']

[163]: re.findall('\D+',txt) #return everything that is not number

[163]: ['my name is hasan, I am  ', ' years old, and my salary is ']

[164]: re.findall('\s',txt) #return all the whitespace

[164]: [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
```

1.3.6 Handling Text Files

```
[183]: # read from files
       f=open("./shared/datafiles/names.txt",'r')
       # content=f.read()     # reads the entire content at once
       # lines=f.readlines()  # reads the lines at once
       # for i in f:            # loop through the content
       #     print(i)
       # f.close()
```

[182]:
```
# try:
#      while True:
#          l=next(f)
#          print(l)
# except:
#      print("done..")
```

[127]:
```
# write to file
f=open('mydata.txt','w')
f.write('hi')
f.write('welcome')
f.close()
f=open('mydata.txt','r')
print(len(f.readlines()))
```

```
1
```

[122]:
```
cat mydata.txt
```

```
hiwelcome
```

[187]:
```
# append to a file
f=open('mydata.txt','a')
f.write('\nthank you')
f.close()
```

[188]:
```
cat mydata.txt
```

```
hi
welcome
thank you
```

[192]:
```
# open the file for read and append
f=open('mydata.txt','r+') # read + append
content=f.read()
f.write('\ngood bye..')
f.close()
```

[190]:
```
abcabc
content = abcabc
abcabcabcabc
```

```
[190]: 'hi\nwelcome\nthank you'
```

```
[47]: cat mydata.txt
```

```
hi
welcome
thank you
good bye..
```

1.3.7 lec10 Ex1(my version)

```
[ ]: names_file = open("./shared/datafiles/names.txt","r")
new_file = open("./firstnames.txt","w")
names_content = names_file.read().split("\n")
names_file.close()
#firstnames = []
# for line in names_content:
#     firstnames.append(line.split()[0].lower())
firstnames = [i.split()[0].lower() for i in names_content] # get new list␣
  ↪without for loop
# for i in firstnames:
#     new_file.write(i+"\n")
new_file.write("\n".join(firstnames)) # write new file without for loop
new_file.close()
```

```
[ ]: #cat firstnames.txt
```

1.4 ex2

```
[ ]: cat ./shared/datafiles/chat.txt
```

```
[ ]: f = open("./shared/datafiles/chat.txt",'r')
#chat = f.read().split("\n")
chat = f.readlines() # this is better than read() in this case since it already␣
  ↪makes it into a list
f.close()
names = [i.split()[0] for i in chat[::2]] # using step to substitue modulo␣
  ↪operator
people = set(names)
#names = [i.split() for index, i in enumerate(chat) if index %2 ==0][:-1]
# names = [i.split() for i in chat[:-1:2]]
# names = sorted([i[0] for i in names])
# people = set(names)
# print(names)
# print(people)
```

```
[ ]: ## counting frequency using dictionary:
     freq = {}
     for name in names:
         if name not in freq:
             freq[name] = 1
         else:
             freq[name] += 1
     print(freq)

     #find the most participated person
     sorted(freq.items(), key = lambda x:x[1])[-1][0]
```

```
[ ]: #pip install nltk
     #nltk.download('stopwords')
     import string
     from nltk.corpus import stopwords
```

```
[ ]: stop = list(stopwords.words('english'))
```

```
[ ]: ## find the top 10 keywords
     chat_content = ' '.join([i.strip() for i in chat[1::2]]).lower()
     for punc in string.punctuation:
         chat_content = chat_content.replace(punc, '')
     word_list =[i for i in chat_content.split() if i not in stop]
     #print(word_list)
     # print(chat_content)
     # uniq_words = set(word_list)
     freq = {}
     for word in word_list:
         if word not in freq:
             freq[word] = 1
         else:
             freq[word] += 1
     #print(freq)
     sorted(freq.items(), key = lambda x:x[1])[-1:-11:-1]
     #or we can use reverse and start from above
     def divide(n,m):
         result=0
         try:
             result= n/m
         except TypeError:
             try:
                 result= divide(int(n),int(m))
             except:
                 result= 'unable to divide'
         except ZeroDivisionError:
             result= 'Nan'
```

```
        print("zero division")
    except:
        result= 'unable to divide'
    else:
        print("everything was OK..")
    finally:
        print("Thank you..")
    return result
```

1.4.1 lec10 exercise1(answerkey)

[62]:
```
# write code to read the names from ./shared/datafiles/names.txt
# extract first names, convert them into lower case, save these first names in␣
 ↪order in a new file (firstnames.txt)
f=open('./shared/datafiles/names.txt')
content=f.read()
first_names=[l.split()[0] for l in content.lower().split('\n')]
first_names.sort()
f.close()
```

[65]:
```
f=open('firstnames.txt','w')
for n in first_names:
    f.write(n)
    f.write('\n')
f.close()
```

[74]:
```
# Alternative method
f=open('./shared/datafiles/names.txt')
first_names=[l.split()[0].lower() for l in f]
f.close()
```

[78]:
```
f=open('firstnames.txt','w')
f.write('\n'.join(sorted(first_names)))
f.close()
```

Exercise-2(answerkey)

[82]:
```
# write code to extract names of particpant in this chat
# find the name of person who particpated most
f=open('./shared/datafiles/chat.txt')
lines=f.readlines()
f.close()
```

[86]:
```
# write code to extract names of particpant in this chat
names=set([l.split()[0] for l in lines[::2]])
```

```
[86]: {'Daniel',
       'Elizabeth',
       'Emily',
       'Emma',
       'Jack',
       'Jayden',
       'John',
       'Mary',
       'Michael',
       'William'}
```

```
[95]: # find the name of person who particpated most
      f={}
      for n in [l.split()[0] for l in lines[::2]]:
          if n not in f:
              f[n]=1
          else:
              f[n]+=1
      sorted(f.items(), key=lambda i:i[1])[-1][0]
```

```
[95]: 'Emily'
```

```
[126]: # find the most important keywords used in this chat (top 10)
       import string
       txt=' '.join(lines[1::2]).lower()
       for c in string.punctuation:
           txt=txt.replace(c,'')
```

```
[132]: # pip install nltk
       import nltk
       from nltk.corpus import stopwords
       # nltk.download('stopwords')
```

```
[ ]: d={}
     for i in stopwords.words('english'):
         d[i]=
         for w in txt.split():
             if w in
```

```
[140]: clean_words=[w for w in txt.split() if w not in stopwords.words('english')]
```

```
[141]: f={}
       for w in clean_words:
           if w not in f:
               f[w]=1
           else:
               f[w]+=1
```

```
[148]: sorted(f.items(), key= lambda i:i[1])[-10:]
```

```
[148]: [('outside', 16),
        ('without', 17),
        ('across', 17),
        ('whatever', 18),
        ('top', 20),
        ('next', 20),
        ('every', 20),
        ('behind', 20),
        ('near', 21),
        ('past', 25)]
```

1.4.2 Placeholders

```
[196]: name = "me"
       age = 19
       # %s - string
       # %d - decimal
       print("I am person, my name is: %s, and my age is: %d"%(name, age))
```

```
I am person, my name is: me, and my age is: 19
```

1.4.3 Binary Files

```
[197]: class person:
           def __init__(self, name="", age=0):
               self.name=name
               self.age=age
           def add(self, x, y):
               return x+y
           def __str__(self):
               return "I am person, my name is: %s, and my age is: %d"%(self.name,␣
        ↪self.age)
```

```
[201]: p1=person("alma",20)
       p2=person("james",25)
```

```
[202]: # save object into a binary file)
       import pickle
       f=open('people_v3.bin','wb') # 'bin' stands for binary file; 'wb' stands for␣
        ↪"write binary file"
       pickle.dump(p1,f) #dump the first person into this binary file
       pickle.dump(p2,f) #dump the second person into this binary file
       f.close()
```

[5]:
```
#cat people_v3.bin
```

[204]:
```
# read from binary file that contains objects
import pickle
f=open('people_v3.bin','rb')# 'wb' stands for "read binary file"
x1=pickle.load(f)
x2=pickle.load(f)
f.close()
```

[205]:
```
print(x1) # print as object
print(x2)
```

```
I am person, my name is: alma, and my age is: 20
I am person, my name is: james, and my age is: 25
```

[209]:
```
# Option 4 (save data in binary file as list of objects)
p1=person("alma",20)
p2=person("james",25)
p3=person("sedra",19)
L=[p1,p2,p3]

f=open('people_v4.bin','wb')
pickle.dump(L,f)
f.close()
```

[4]:
```
#cat people_v4.bin
```

[211]:
```
# read data from binary file that contains the list
f=open('people_v4.bin','rb')
x=pickle.load(f) # note that now a list of object is extracted from the binary␣
 ↪file
f.close()
```

[212]:
```
print(x[0]) # to access single obj in the list we need to specify index
```

```
I am person, my name is: alma, and my age is: 20
```

[60]:
```
for p in x:
    print(p)
```

```
I am person, my name is: alma, and my age is: 20
I am person, my name is: james, and my age is: 25
I am person, my name is: sedra, and my age is: 19
```

[62]:
```
[str(p) for p in x]
```

```
[62]: ['I am person, my name is: alma, and my age is: 20',
       'I am person, my name is: james, and my age is: 25',
       'I am person, my name is: sedra, and my age is: 19']
```

1.5 Fetch Data from URL

```
[66]: # Access Data from different computer
import requests
response= requests.get('https://raw.githubusercontent.com/ha2285/files/main/
 ↪courses.txt')
t=response.text
```

Exercise-1

```
[ ]: # link 1 https://raw.githubusercontent.com/ha2285/files/main/invdata1.txt
# link 2 https://raw.githubusercontent.com/ha2285/files/main/invdata2.txt
# 1.Verify if there is any duplicates in each file
# 2.find data that was scanned more than one time in each file
# 3.find data that's shared between these two files
```

```
[213]: import requests
link1 = requests.get("https://raw.githubusercontent.com/ha2285/files/main/
 ↪invdata1.txt")
file1 = link1.text.strip().split(',')
link2 = requests.get("https://raw.githubusercontent.com/ha2285/files/main/
 ↪invdata2.txt")
file2 = link2.text.strip().split(',')
def get_duplicates(file):
    freq = {}
    for num in file:
        if num not in freq.keys():
            freq[num] =1
        else:
            freq[num] += 1
    items = sorted(freq.items(), key = lambda x:x[1],reverse = True)
    replicated_items = list(filter(lambda x:x[1] >1, items))
    return replicated_items
replicated_1 = get_duplicates(file1)
print(replicated_1)
#second link
freq2 = {}
for num in file2:
    if num not in freq2.keys():
        freq2[num] =1
    else:
        freq2[num] += 1
items2 = sorted(freq2.items(), key = lambda x:x[1],reverse = True)
```

```
rep2 = list(filter(lambda x:x[1] >1, items2))
#print(rep2)

#q3
print(len(file1))
shared = [i for i in file1 if i in file2]
print(len(shared))
```

```
[('596368923888', 4), ('630935116647', 3), ('325584633445', 2), ('409252788948',
2), ('401569104891', 2), ('548338751881', 2), ('157591660778', 2),
('488352816888', 2), ('323568241564', 2), ('633874664823', 2), ('350250775701',
2)]
1000
986
```

[220]:
```
#q3 alternative
#find data that's shared between these two files
#set(file1) & set(file2)
```

[112]:
```
# 4.find items in file 1 not in file 2
```

[221]:
```
#set(file1) - set(file2)
```

[114]:
```
# 4. find non shared data
```

[222]:
```
#set(file1) ^ set(file2)
```

1.5.1 Handling HTML & Websraping

[4]:
```
import requests
reu = requests.get('https://www.reuters.com/').text
#not gonna run it since it is too long for printing
```

[6]:
```
from bs4 import BeautifulSoup
#reu_soup = BeautifulSoup(reu)
```

[34]:
```
txt = [i.getText() for i in reu_soup.find_all('a',{'data-testid':'Heading'})]
txt
```

[34]:
```
['Analysis: Investors revive inflation trades as 6% Fed rate risk grips Wall
Street',
 "Germany set to ban China's Huawei, ZTE from parts of 5G networks -source",
 'Salesforce to add ChatGPT to Slack as part of OpenAI partnership',
 'UK salad shortage weighs on supermarket sales -NIQ',
 "China's Xi has mixed feelings about CATL's battery market dominance",
 'Heineken blames Russia exit delay on local paperwork',
 'Hong Kong loses lustre as retail units go vacant and big brands look to
```

```
China',
 'China says U.S. should change attitude or risk conflict, article with video',
 'Two Israels face off as justice overhaul deepens the divide',
 "Morning Bid: Powell's plan, China challenge, article with image",
 'Biden plans tax on high-earners in bid to save Medicare, article with image',
 "Inside BP's plan to reset renewables as oil and gas boom, article with
gallery",
 "Fed's Powell, in Hill appearance, to update views on status of
'disinflation'",
 'Big Oil and green energy, TV dating in Korea and the battle for Bakhmut:
podcast, article with image',
 "Women's stories stand out in Oscars race, but Hollywood lagging in gender
parity, article with gallery",
 'Diverse talent in Hollywood takes reins to speed up change, article with
video',
 'Tuxedos and jumpsuits: menswear dazzles on awards season red carpets, article
with video',
 'Kyiv vows to send more troops into Bakhmut, seeing chance to break Russian
force, article with video',
 'Strikes spread as French unions intensify pension reform fight, article with
gallery',
 'UK tells Channel migrants: we will detain and remove you, article with
gallery',
 'New EU-US data pact may come too late for Facebook -regulator, article with
image',
 'Yellen warns climate change could trigger asset value losses, harming US
economy, article with image',
 'Twelve U.S. senators back giving Commerce Secretary new powers to ban TikTok,
article with gallery',
 "U.S. Supreme Court won't decide scope of wage-and-hour class actions, article
with image",
 'White House may restart detention of migrant families - sources, article with
image',
 'Israeli settlers attack Palestinians in flashpoint West Bank town, article
with gallery',
 'Pentagon chief, in unannounced visit to Iraq, pledges continued U.S. troop
presence, article with gallery',
 'Lebanon says it regains UN voting rights after paying dues, article with
image',
 'Turks look to history and foresee rebirth of ancient Antakya from earthquake
ruins, article with gallery',
 "Factbox: Hold or hike? Latin American central banks' wield rates to battle
rising prices, article with image",
 'Venezuelans struggling to afford food - even if they have access to dollars,
article with gallery',
 "Bolsonaro's son revives speculation over former president's return to Brazil
with deleted tweet, article with image",
 'Trudeau orders new probes into alleged election interference by China, article
with video',
 'CERAWEEK-US oil output to grow by 500,000 bpd in 2023 - Occidental Petroleum
exec, article with image',
 'Oil edges lower on stronger dollar and weak Chinese data, article with image',
 'Kremlin says it does not recognise Western price cap on its oil, article with
image',
 "Germany's Schwedt refinery losing out in race from Russian oil, article with
image",
 'Sri Lanka expects approval of $2.9 billion IMF deal after China support,
article with image',
 'Forced labour victims protest in wheelchairs, reject S.Korea deal on Japan,
article with gallery',
 'North Korea warns U.S. against intercepting its test missiles, article with
gallery',
 'Australian officials warn of bushfire threat as heatwave grips Sydney, article
with image',
 'The colors of Holi, article with image',
 'Top Photos of the Day, article with image',
 'Fearing Russian bombs, Ukrainian families seek safety underground, article
with image',
 'Eurovision Song Contest final tickets sell out in 36 minutes, article with
image',
 'With pails and mugs, Philippine residents clean up oil spill, article with
video',
 'Chile announces biological corridor to protect endangered deer, article with
gallery',
 "Louis Vuitton shows playful, French styles at Musee d'Orsay in Paris, article
with gallery"]
```

BeautifulSoup

```
[1]: f=open('mypage.html') #this is the prof wrote on his pc so I cant run it here
     txt=f.read()
     f.close()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [1], in <cell line: 1>()
----> 1 f=open('mypage.html') #this is the prof wrote on his pc so I cant run it
 ↪here
      2 txt=f.read()
      3 f.close()

FileNotFoundError: [Errno 2] No such file or directory: 'mypage.html'
```

[40]:
```
pip install nltk
```

```
Collecting nltk
  Using cached nltk-3.8.1-py3-none-any.whl (1.5 MB)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-packages
(from nltk) (4.64.1)
Collecting joblib
  Using cached joblib-1.2.0-py3-none-any.whl (297 kB)
Collecting regex>=2021.8.3
  Using cached
regex-2022.10.31-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (770
kB)
Requirement already satisfied: click in /opt/conda/lib/python3.10/site-packages
(from nltk) (8.1.3)
Installing collected packages: regex, joblib, nltk
Successfully installed joblib-1.2.0 nltk-3.8.1 regex-2022.10.31
Note: you may need to restart the kernel to use updated packages.
```

[64]:
```
entire_string = ' '.join(txt)
import nltk
import string
from nltk.corpus import stopwords
stopwords = list(stopwords.words('english'))
for i in string.punctuation:
    entire_string.replace(i, '')
word_list = [i.strip() for i in entire_string.split() if i not in stopwords]
f = {}
for word in word_list:
    if word not in f:
        f[word] = 1
    else:
        f[word] += 1
f = dict(sorted(f.items(), key = lambda x:x[1], reverse = True))
f
```

[64]:
```
{'article': 39,
 'image': 20,
 'gallery': 13,
 'video': 6,
 'U.S.': 5,
 'China': 4,
 'says': 3,
 'oil': 3,
 'Russian': 3,
 '-': 3,
 'risk': 2,
 'grips': 2,
 'ban': 2,
 "China's": 2,
 'UK': 2,
 'look': 2,
 'change': 2,
 'Oil': 2,
 'Korea': 2,
 'battle': 2,
 'Hollywood': 2,
 'French': 2,
 'may': 2,
 'warns': 2,
 'new': 2,
 'families': 2,
 'oil,': 2,
 'deal': 2,
 'Analysis:': 1,
 'Investors': 1,
 'revive': 1,
 'inflation': 1,
 'trades': 1,
 '6%': 1,
 'Fed': 1,
 'rate': 1,
 'Wall': 1,
 'Street': 1,
 'Germany': 1,
 'set': 1,
 'Huawei,': 1,
 'ZTE': 1,
 'parts': 1,
 '5G': 1,
 'networks': 1,
 '-source': 1,
 'Salesforce': 1,
 'add': 1,
 'ChatGPT': 1,
 'Slack': 1,
 'part': 1,
 'OpenAI': 1,
 'partnership': 1,
 'salad': 1,
 'shortage': 1,
 'weighs': 1,
 'supermarket': 1,
 'sales': 1,
 '-NIQ': 1,
 'Xi': 1,
 'mixed': 1,
 'feelings': 1,
 "CATL's": 1,
 'battery': 1,
 'market': 1,
 'dominance': 1,
 'Heineken': 1,
 'blames': 1,
 'Russia': 1,
 'exit': 1,
 'delay': 1,
 'local': 1,
 'paperwork': 1,
 'Hong': 1,
 'Kong': 1,
 'loses': 1,
 'lustre': 1,
 'retail': 1,
 'units': 1,
 'go': 1,
 'vacant': 1,
 'big': 1,
 'brands': 1,
 'attitude': 1,
 'conflict,': 1,
 'Two': 1,
 'Israels': 1,
 'face': 1,
 'justice': 1,
 'overhaul': 1,
 'deepens': 1,
 'divide': 1,
 'Morning': 1,
 'Bid:': 1,
 "Powell's": 1,
 'plan,': 1,
 'challenge,': 1,
 'Biden': 1,
 'plans': 1,
 'tax': 1,
 'high-earners': 1,
 'bid': 1,
 'save': 1,
 'Medicare,': 1,
 'Inside': 1,
 "BP's": 1,
 'plan': 1,
 'reset': 1,
 'renewables': 1,
 'gas': 1,
 'boom,': 1,
 "Fed's": 1,
 'Powell,': 1,
 'Hill': 1,
 'appearance,': 1,
 'update': 1,
 'views': 1,
 'status': 1,
 "'disinflation'": 1,
 'Big': 1,
 'green': 1,
 'energy,': 1,
 'TV': 1,
 'dating': 1,
 'Bakhmut:': 1,
 'podcast,': 1,
 "Women's": 1,
 'stories': 1,
 'stand': 1,
 'Oscars': 1,
 'race,': 1,
 'lagging': 1,
 'gender': 1,
 'parity,': 1,
 'Diverse': 1,
 'talent': 1,
 'takes': 1,
 'reins': 1,
 'speed': 1,
 'change,': 1,
 'Tuxedos': 1,
 'jumpsuits:': 1,
 'menswear': 1,
 'dazzles': 1,
 'awards': 1,
 'season': 1,
 'red': 1,
 'carpets,': 1,
 'Kyiv': 1,
 'vows': 1,
 'send': 1,
 'troops': 1,
 'Bakhmut,': 1,
 'seeing': 1,
 'chance': 1,
 'break': 1,
 'force,': 1,
 'Strikes': 1,
 'spread': 1,
 'unions': 1,
 'intensify': 1,
 'pension': 1,
 'reform': 1,
 'fight,': 1,
 'tells': 1,
 'Channel': 1,
 'migrants:': 1,
 'detain': 1,
 'remove': 1,
 'you,': 1,
 'New': 1,
 'EU-US': 1,
 'data': 1,
 'pact': 1,
 'come': 1,
 'late': 1,
 'Facebook': 1,
 '-regulator,': 1,
 'Yellen': 1,
 'climate': 1,
 'could': 1,
 'trigger': 1,
 'asset': 1,
 'value': 1,
 'losses,': 1,
 'harming': 1,
 'US': 1,
 'economy,': 1,
 'Twelve': 1,
 'senators': 1,
 'back': 1,
 'giving': 1,
 'Commerce': 1,
 'Secretary': 1,
 'powers': 1,
 'TikTok,': 1,
 'Supreme': 1,
 'Court': 1,
 'decide': 1,
 'scope': 1,
 'wage-and-hour': 1,
 'class': 1,
 'actions,': 1,
 'White': 1,
 'House': 1,
 'restart': 1,
 'detention': 1,
 'migrant': 1,
 'sources,': 1,
 'Israeli': 1,
 'settlers': 1,
 'attack': 1,
 'Palestinians': 1,
 'flashpoint': 1,
 'West': 1,
 'Bank': 1,
 'town,': 1,
 'Pentagon': 1,
 'chief,': 1,
 'unannounced': 1,
 'visit': 1,
 'Iraq,': 1,
 'pledges': 1,
 'continued': 1,
 'troop': 1,
 'presence,': 1,
 'Lebanon': 1,
 'regains': 1,
 'UN': 1,
 'voting': 1,
 'rights': 1,
 'paying': 1,
 'dues,': 1,
 'Turks': 1,
 'history': 1,
 'foresee': 1,
 'rebirth': 1,
 'ancient': 1,
 'Antakya': 1,
 'earthquake': 1,
 'ruins,': 1,
 'Factbox:': 1,
 'Hold': 1,
 'hike?': 1,
 'Latin': 1,
 'American': 1,
 'central': 1,
 "banks'": 1,
 'wield': 1,
 'rates': 1,
 'rising': 1,
 'prices,': 1,
 'Venezuelans': 1,
 'struggling': 1,
 'afford': 1,
 'food': 1,
 'even': 1,
 'access': 1,
 'dollars,': 1,
 "Bolsonaro's": 1,
 'son': 1,
 'revives': 1,
 'speculation': 1,
 'former': 1,
 "president's": 1,
 'return': 1,
 'Brazil': 1,
 'deleted': 1,
 'tweet,': 1,
 'Trudeau': 1,
 'orders': 1,
 'probes': 1,
 'alleged': 1,
 'election': 1,
 'interference': 1,
 'China,': 1,
 'CERAWEEK-US': 1,
 'output': 1,
 'grow': 1,
 '500,000': 1,
 'bpd': 1,
 '2023': 1,
 'Occidental': 1,
 'Petroleum': 1,
 'exec,': 1,
 'edges': 1,
 'lower': 1,
 'stronger': 1,
 'dollar': 1,
 'weak': 1,
 'Chinese': 1,
 'data,': 1,
 'Kremlin': 1,
 'recognise': 1,
 'Western': 1,
 'price': 1,
 'cap': 1,
 "Germany's": 1,
 'Schwedt': 1,
 'refinery': 1,
 'losing': 1,
 'race': 1,
 'Sri': 1,
 'Lanka': 1,
 'expects': 1,
 'approval': 1,
 '$2.9': 1,
 'billion': 1,
 'IMF': 1,
 'support,': 1,
 'Forced': 1,
 'labour': 1,
 'victims': 1,
 'protest': 1,
 'wheelchairs,': 1,
 'reject': 1,
 'S.Korea': 1,
 'Japan,': 1,
 'North': 1,
 'intercepting': 1,
 'test': 1,
 'missiles,': 1,
 'Australian': 1,
 'officials': 1,
 'warn': 1,
 'bushfire': 1,
 'threat': 1,
 'heatwave': 1,
 'Sydney,': 1,
 'The': 1,
 'colors': 1,
 'Holi,': 1,
 'Top': 1,
 'Photos': 1,
 'Day,': 1,
 'Fearing': 1,
 'bombs,': 1,
 'Ukrainian': 1,
 'seek': 1,
 'safety': 1,
 'underground,': 1,
 'Eurovision': 1,
 'Song': 1,
 'Contest': 1,
 'final': 1,
 'tickets': 1,
 'sell': 1,
 '36': 1,
 'minutes,': 1,
 'With': 1,
 'pails': 1,
 'mugs,': 1,
 'Philippine': 1,
 'residents': 1,
 'clean': 1,
 'spill,': 1,
 'Chile': 1,
 'announces': 1,
 'biological': 1,
 'corridor': 1,
 'protect': 1,
 'endangered': 1,
 'deer,': 1,
 'Louis': 1,
 'Vuitton': 1,
 'shows': 1,
 'playful,': 1,
 'styles': 1,
 'Musee': 1,
 "d'Orsay": 1,
 'Paris,': 1}
```

[ ]:

```

```

[47]:

```
txt
```

[47]:

```
'<HTML>\n\t<Head>\n\t\t<title>My first web
page</title>\n\t</HEAD>\n<body>\n\t<font color="red" size=20><b>some text
here</b></font>\n\t<h1>welcome</h1>\n\n\t<p>My name is hasan, I am a prof. at
NYU</p>\n\t<p>I do teach computer systems , Data science and OOP</p>\n\t<p>if
you like to get more info about me, you can visit my website.\n\t<a
href="http://cs.nyu.edu/~ha2285">click here to visit my website.</a>\n\t<img
src="NYU-Logo.png" width="400" height="400">\n\t</p>\n\t<p>if you want to visit
NYU main website <a href="http://www.nyu.edu">click
here</a></p>\n</body>\n</HTML>'
```

[48]:

```
from bs4 import BeautifulSoup
soup=BeautifulSoup(txt)
```

```
[55]: [t['href'] for t in soup.find_all('a')]
```

```
[55]: ['http://cs.nyu.edu/~ha2285', 'http://www.nyu.edu']
```

1.5.2 Web Scraping

```
[20]: response=requests.get("https://www.bbc.com/")
      #response.text
      # too long not gonna displayed
```

```
[19]: from bs4 import BeautifulSoup
      soup=BeautifulSoup(response.text)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [19], in <cell line: 2>()
      1 from bs4 import BeautifulSoup
----> 2 soup=BeautifulSoup(response.text)

NameError: name 'response' is not defined
```

```
[232]: soup.find_all('h1')[0].getText()
```

```
[232]: 'BBC Homepage'
```

```
[60]: soup.find_all('a')[2]['href']
```

```
[60]: 'https://www.bbc.co.uk/accessibility/'
```

```
[261]: [t['href'] for t in soup.find_all('a')][1:10]
```

```
[261]: ['#orb-modules',
        'https://www.bbc.co.uk/accessibility/',
        'https://account.bbc.com/account',
        'https://www.bbc.co.uk/notifications',
        'https://www.bbc.co.uk',
        'https://www.bbc.co.uk/news',
        'https://www.bbc.co.uk/sport',
        'https://www.bbc.co.uk/weather',
        'https://www.bbc.co.uk/iplayer']
```

```
[17]: headlines=[t.getText().strip().lower() for t in soup.find_all('a',␣
       ↪{'class','media__link'})]
```

```
[18]: # find the headlines that contains 'us'; != -1 means true
      [h for h in headlines if h.find('us')!=-1]
```

```
[18]: []
```

```
[251]: txt=' '.join(headlines) # join all the sentences into a large string so we can␣
        ↪find keywords latter
       import string
       for c in string.punctuation:
           txt=txt.replace(c,"")
```

```
[240]: # pip install nltk
       import nltk
```

```
[250]: txt
```

```
[250]: 'ukraine doubles down in bakhmut defence  zelensky canada to probe alleged
       chinese election meddling exposed for abuse but still revered – japan’s pop
       predator fernandes should not captain man utd again  sutton a surprise hit thats
       an oscars favourite why did the uns quake aid take so long to arrive search
       launched for americans seized in mexico the crane which refuses to leave its
       human saviour danger in a vanishing world  the icecold nerve of will gadd neymar
       set to miss rest of season with ankle injury how do england look seven months
       before world cup europes littleknown progressive city the irish films breaking
       oscars records just how loud are rocket launches why so many lgbtq workers are
       quitting jimmy kimmel returns to host oscars 2023 couple jailed for stealing 17m
       worth of wine smiley face minisphinx unearthed in egypt we cut our websites
       carbon emissions by 520kg in a year the mushrooms you can wear and build with
       cctv of couzens moments before flashing drivethrough workers cctv of couzens
       moments before flashing watch brawl kicks off in georgian parliament ceiling
       collapses nearly hits commuter in us stephen bear watch the day georgia harrison
       fire tears through bangladesh refugee camp australias biggest drug bust nets
       700m of oneminute world news thanks but no big speech in ken bruces tear gas
       fired at greece train crash ros atkins on the creeping tiktok bans the book that
       records all disasters at sea china discusses military budget and warns of
       threats comebacks stunts and selfies 8 awards season highlights worlds most
       premature twins certified by guinness warming could increase uk flood damage
       bill by 20 twitter insiders we cant protect users from trolling under musk half
       of world could be overweight by 2035 putins frustration and resentment against
       the west robotaxi tech improves but can they make money pet crocs and red
       carpets africas top shots the devastated towns near ukraines front line pictures
       show devastation after greece train disaster spectacular northern lights seen
       across uk sag awards the red carpet in pictures'
```

```
[254]: from bs4 import BeautifulSoup
       import datetime
       response=requests.get("https://www.bbc.com/")
       soup=BeautifulSoup(response.text)
       headlines=[t.getText().strip().lower() for t in soup.find_all('a',␣
        ↪{'class','media__link'})]
```

```
d=str(datetime.datetime.today().day)
m=str(datetime.datetime.today().month)
f=open('bbc-'+d+'-'+m+'.txt','w')
f.write('\n'.join(headlines))
f.close()
```

[255]:
```
print(datetime.datetime.today())
```

```
2023-03-07 00:14:05.194525
```

1.6 first quiz MISTAKES

[ ]:
```
l0 = 'me'
l1 = "she"
l2 = [1,2,3,l0,l1]
l2[-1] += ("me")
l2[-2] = "he"
l0
```

[ ]:
```
l = ['james', ['mike'],['sarah','me']]
print(l[-1][-2:][-1])
```

[120]:
```
cat ./shared/datafiles/employees.csv|cut -d, -f1|sort -d| uniq -c|sort -n
```

```
      1
      1 0
      1 1
      1 2
      1 3
      1 4
      1 5
      1 6
      1 7
      1 8
```

[ ]:
```
payments =␣
 ↪[['james',200],['james',150],['james',100],['sarah',300],['sarah',100],['mike',500],['mike'
```

[ ]:
```
print([list(set([p[0] for p in payments])), sum([p[1] for p in payments])])
```

[ ]:
```
name = "candice"
last = "yao"
print("my name is %s, %s"%(name,last))
```

[82]:
```
### practice questions
import re
re.search('ab+$', 'a')!=None
```

```
[82]: False
```

```
[1]: pwd
```

```
[1]: '/home/jovyan'
```

```
[ ]:
```