Data Structure : list

A data structure has zero or more elements with some relationship. We have already studied the concept of a list.

- A list has zero or more elements
- The element of the list can be of any type. There is no restriction on the type of the element.
- A list has elements which could be of the same type(honogeneous) or of different types(heterogeneous)
- Each element of a list can be referred to by a index or a subscript
- An index is an integer
- Access to an element based on index or position takes the same time no matter where the element is in the list – random access
- A list can grow or shrink. Size of the list can be found using the function len
- There are functions to play with a list
  - append
  - insert
  - extend
  - pop
  - remove
  - sort
  - index
  - …
- Elements in the list can repeat
- List is a sequence
- List is also iterable - is eager and not lazy.
- Lists can be nested. We can have list of lists.
- Assignment of one list to another causes both to refer to the same list.
- Lists can be sliced. This creates a new list.

Data Structure: Set:

A set is a data structure with zero or more elements with the following attributes.

- Elements are unique – does not support repeated elements

- Elements should be hashable - Hashing is a mechanism to convert the given element to an integer. In some storage area, at that location indicated by hashing, the element will be stored in some way. We do not have to worry about it. We should know that hashing is a requirement to put into a set
- set is not ordered – we cannot assume the order of elements in a set.
- set is an iterable – eager and not lazy
- we cannot index on a set. The set is iterable, but is not a sequence.
- We can check for membership using the **in** operator. This would be faster in case of a set compared to a list, a tuple or a string.
- Sets support many mathematical operations on sets.
    - Membership : in
    - union : |
    - intersection : &
    - set difference : -
    - symmetric difference : ^
    - equality and inequality : = !=
    - subset and superset : < <= > >=
    - set constructor { ... }
- To create an empty set, we must use the set constructor set() and not { }. The latter would become a dict.

We use sets for
a) deduplication : removal of repeated elements
b) finding unique elements
c) comparing two iterables for common elements or difference.

Let us look at a few programs illustrating the usefulness and the power of the set data structure.

```
# name : 0_intro_set.py
# set :
#       is a data structure
#       has # of elements
```

```
#       elements are unique
#       make a set : { }
a = { 10, 30, 10, 40, 20, 50, 30 }
print(a) # elements are unique; there is no particular order

# list : l
#       ith element : l[i]
#       next element : l[i + 1]
#       previous element : l[i - 1]
#       is a sequence
#       concept of position for each element

# set :
#       not sequence
#       no concept of an element in a particular position
#       represents a finite set of math

# set in an iterable
for e in a :
        print(e, end =  " ")
print()



# check for membership
print(100 in a) # False
print(20 in a) # True

# set operations
s1 = {1, 2, 3, 4, 5}
s2 = {1, 3, 5, 7, 9}
# union
print(s1 | s2) # {1, 2, 3, 4, 5, 7, 9}
# intersection
print(s1 & s2) # {1, 3, 5}
# set difference
```

```
print(s1 - s2) # {2, 4}
# symmetric difference
print(s1 ^ s2) # {2, 4, 7, 9}



#print("what : ", s1[0])# TypeError: 'set' object does not support indexing

# creates a set ; initialized by called set - which is called the constructor
#       of set
s3 = set()
s4 = set([11, 33, 22, 11, 33, 11, 11, 44, 22])
print(s3)
print(s4)

s5 = set("mississippi")
print(s5) # {'s', 'i', 'm', 'p'}

# string : double or single quote : string should be on a single line
#                       3 double quotes or 3 single quotes : string can appear or span
#               multiple lines
str1 = """do not trouble trouble
till trouble
troubles you"""
#print(str1.split())
print(set(str1.split()))
"""
# output in order
# version 1
l = list(set(str1.split()))
l.sort()
print(l)
"""
# version 2:
print(sorted(set(str1.split())))
```

```
str2 = """betsy botsome bought some butter but the butter was bitter
betsy botsome bought some better butter to make the bitter butter better"""
print(sorted(set(str2.split())))
```

Let us look at some pieces of code from this program.

- This creates a set – all the elements are enumerated.
  The elements shall be unique.
  The order of output is not defined as the set is an unordered collection.
  ```
  a = { 10, 30, 10, 40, 20, 50, 30 }
  print(a) # elements are unique; there is no particular order
  ```
- set is iterable, but not indexable. a[2] will be an error.
  The order of output is still not defined.
  ```
  # set in an iterable
  for e in a :
          print(e, end =  " ")
  print()
  ```
- The code is self evident.
  ```
  # check for membership
  print(100 in a) # False
  print(20 in a) # True
  ```
- set operations
  These are self evident. The order of the output is not defined in each of these cases.
  ```
  # set operations
  s1 = {1, 2, 3, 4, 5}
  s2 = {1, 3, 5, 7, 9}
  # union
  print(s1 | s2) # {1, 2, 3, 4, 5, 7, 9}
  # intersection
  print(s1 & s2) # {1, 3, 5}
  # set difference
  print(s1 - s2) # {2, 4}
  ```

```
# symmetric difference
print(s1 ^ s2) # {2, 4, 7, 9}
```

- empty set creation
  ```
  s3 = set()
  ```

- Finding unique elements in a string
  ```
  s5 = set("mississippi")
  print(s5) # {'s', 'i', 'm', 'p'}
  ```

- Finding unique words in a string str1; words separated by white space
  We split the string based on white space – pass this list of strings as argument to the set constructor. If necessary pass this as argument to the list constructor to make a list.
  ```
  l = list(set(str1.split()))
  ```

- This is same as the earlier case – but the elements of the set are sorted into a list by using the function sorted.
  ```
  print(sorted(set(str2.split())))
  ```

Another example of removing repeated elements.

```
# name : 1_remove_repeated.py
# deduplication : removed repeated elements
a = [11, 33, 11, 33, 11, 44, 22, 55, 55, 11]
a = list(set(a))
print(a)
```

Let us try creating sets with the required elements.

```
# name : 2_operations_set.py
# create sets with the required elements

# make a set of  numbers from 2 to n
#
n = 10
```

```
#s = {}  #not an empty set

"""
# not good; it works
# version 1
s = set()
for e in range(2, n + 1):
        s.add(e)
print(s, type(s))
"""

# version 2
s = set(range(2, n + 1))
print(s)

#-------------------------------------

# is set empty or not
s1 = set()
s2 = set("fool")
print("empty : ", len(s1) == 0)
print("empty : ", len(s2) == 0)
"""
if len(s1) == 0 :
        print("empty")
else:
        print("non empty")

if len(s2) == 0 :
        print("empty")
else:
        print("non empty")
"""
if s1 :
        print("empty")
```

```
else:
        print("non empty")

if s2 :
        print("empty")
else:
        print("non empty")
# empty data structure => False
# non-empty data structure => True

# remove elements from a set
s3 = set(range(10)) # 0 .. 9
# remove 2 4 6 8  # remove multiples of 2
for i in range(2, 10, 2) :
        s3.remove(i)
print(s3)

s3 = set(range(10)) # 0 .. 9
s3 = s3 - set(range(2, 10, 2))
print(s3)
```

Let us discuss the fragments of the code from this example.

- Create a set of numbers from 2 to n.
  - Method 1: use a for loop; iterate on the range object range(2, n + 1); add each element to the set
  - Method 2: pass the range object as an argument to the set constructor This method is clean and elegant.
    s = set(range(2, n + 1))
- check whether a set is empty
  - len(s) == 0 # not preferred
  - s
    - empty set is False; non-empty set is True.
- Remove elements from a set
  . Given a set of elements s3, remove elements 2, 4, 6, 8.

- method 1:
  iterate through a range or a list object containing 2, 4, 6, 8. call remove of each element on the set.
- Method 2:
  create a set of the elements to be removed – use set difference to remove the elements. This is preferred.
  s3 = s3 - set(range(2, 10, 2))

We shall now discuss a very interesting method to generate prime numbers which does not involve any division operator at all. This algorithm is called Sieve of Eratosthenes.

```
# name: 3_sieve.py
# generate prime numbers (no division; most efficient algorithm)
# sieve of Eratosthenes
# get a number(say n)
# make a set of numbers from 2 to n - say sieve
# while sieve is not empty
#       find the smallest (small)
#       print it (that is a prime)
#       remove small and its multiples from the sieve

n = int(input("Enter an integer : "))
# make a set of numbers from 2 to n - say sieve
sieve = set(range(2, n + 1))
#print(sieve)
while sieve :
        small = min(sieve)
        print(small, end = " ")
        sieve -= set(range(small, n + 1,small))
```

Make a set called sieve of elements from 2 to a given number n.
While the sieve is not empty, remove the smallest element – which will be a prime. Then remove that element and its multiples.
Thats all. Are the sets powerful?

Data Structure: dict : dictionary :

dict is a data structure which has key value pairs.

- Keys are unique
- keys are immutable – cannot be changed
- Keys are hashable
- Key value pairs are stored at the hashed location in some way
- dict like the set are unordered collection
- dict is indexable based on the key – key could be a string, an integer or a tuple or any immutable type
- An empty dict is created by using {} or by the constructor dict()
- we can extract keys using the method dict.keys() and the values using the method dict.values(). Both these return iterable objects. There will be one-to-one mapping between the keys in the dict.keys() and the values in the dict.values()
- we can iterate through a dict – we actually iterate through the keys.

Let us solve some problems and choose the right data structures.

The input to all these problems is a string having # of lines. Each line has a language name, a writer's name and his work.

We can split the string based on new line and split each of these lines based on white space and capture whatever is required.

a) question 1 : find the number of books

Solution: count the number of lines in the string.

```
print("# of books : ", len(all.split('\n')))
```

b) question 2: find the number of languages

Solution:

split the string into lines

split each line and extract the first entry only

put them into a data structure where the entries shall be unique – that is set.

count them

```
langset = set()
for line in all.split('\n'):
        langset.add(line.split()[0])
```

```
print("# of lang : ", len(langset))
```

c) question 3: count the number of books in each language

This is similar to the last example. But set will not be sufficient. For each language we require a count. We require a language:count pair where the languages are unique. The data structure for this is dict.

In these sort of problems where a data structure has to be created, we will initialize before a loop. We build the data structure element by element within the loop. If the lang is not present, we create the lang as the key and make the corresponding value 0 in the dict. We then count that book by adding one to the value stored in the value field for that language.

```
lang_book_count = {}
for line in all.split('\n'):
        lang = line.split()[0]
        if lang not in lang_book_count :
                lang_book_count[lang] = 0
        lang_book_count[lang] += 1

for lang in lang_book_count :
        print(lang, " => ",  lang_book_count[lang])
```

 # name: 5_dict.py

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
```

```
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""

# find the # of books
print("# of books : ", len(all.split('\n')))
#for l in enumerate(all.split('\n')) :
#       print(l)
#print(l)

# find the number of languages
langset = set()
for line in all.split('\n'):
        #print(line.split()[0])
        langset.add(line.split()[0])
#print(langset)
print("# of lang : ", len(langset))



# count the number of books in each language
lang_book_count = {}
for line in all.split('\n'):
        lang = line.split()[0]
        #print(lang)
        if lang not in lang_book_count :
                lang_book_count[lang] = 0
        lang_book_count[lang] += 1

for lang in lang_book_count :
        print(lang, " => ",  lang_book_count[lang])
```

d) question 4: find list of authors for each language

Names of the authors would repeat as each author may have more than one book. So the data structure shall be a dict where key is the language and the value is a set of authors.

Check the comments at the end of each line.

```
lang_author = {}  # create an empty dict
for line in all.split('\n'):
        (lang, author) = line.split()[:2] # slice and pick up the first two elements
        if lang not in lang_author: # if key lang does not exist, add that key
                lang_author[lang] =  set() # make the value an empty set
        lang_author[lang].add(author) # add to the set  uniquely
```

 # name : 6_dict.py

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""

# find list of authors for each language
# dict of sets
lang_author = {}
for line in all.split('\n'):
        (lang, author) = line.split()[:2]
#       print(lang, author)
        if lang not in lang_author:
                lang_author[lang] = set()
        lang_author[lang].add(author)

for lang in lang_author:
        print(lang)
```

```
    for author in lang_author[lang]:
        print("\t", author)
```

e) question 5: find number of books of each author.
The data structure shall be a dict of dict of int.
The key for the outer dict shall be the lang.
The key for the inner dict will be the author.
The value shall be int. Check the comments at the end of each line.

```
lang_author = {} # empty dict
for line in all.split('\n'):
    (lang, author) = line.split()[:2] # slice and pick up what is required
    if lang not in lang_author : # check and create an empty dict as the value
        lang_author[lang] = {}
    if author not in lang_author[lang] : # if the key does not exist, put the key
        lang_author[lang][author] = 0 # with the value as 0
    lang_author[lang][author] += 1 # increment the count
```

 # name: 7_dict.py

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""

# find # of books of each author in each language
# soln: dict of dict of int
```

```
lang_author = {}
for line in all.split('\n'):
        (lang, author) = line.split()[:2]
        if lang not in lang_author :
                lang_author[lang] = {}
        if author not in lang_author[lang] :
                lang_author[lang][author] = 0
        lang_author[lang][author] += 1

for lang in lang_author:
        print(lang)
        for author in lang_author[lang]:
                print("\t", author, "=>",
                                lang_author[lang][author])
```

f) question 6: create a language to author to book mapping.

We will create a dict as the solution. The key will be the language. The value will be a dict. In that dict, key will be the author and the value will be a list. Check the comments added at the end of each line.

```
Info = {} # create an empty dict
for line in all.split('\n'):
        (lang, author, title) = line.split() # get the required info by splitting
        if lang not in info : # if lang does not exist, add that as the key in info
                info[lang] = {}
        if author not in info[lang] : # if author does not exist, add that as the key
                                                in info[lang]
                info[lang][author] = [] # make the value an empty list
        info[lang][author].append(title) # append to the list.
```

 # name: 8_dict.py

```
all = """sanskrit kalidasa shakuntala
english r_k_narayan malgudi_days
kannada kuvempu ramayanadarshanam
sanskrit bhasa swapnavasavadatta
kannada kuvempu malegalalli_madumagalu
english r_k_narayan dateless_diary
kannada karanta chomanadudi
sanskrit baana harshacharita
kannada karanta sarasatammana_Samadhi
sanskrit kalidasa malavikagnimitra
sanskrit kalidasa raghuvamsha
sanskrit baana kadambari
sanskrit bhasa pratijnayogandhararayana"""

# find list of titles of each author of each lang
# soln: dict of dict of list

info = {}
for line in all.split('\n'):
        (lang, author, title) = line.split()
        if lang not in info :
                info[lang] = {}
        if author not in info[lang] :
                info[lang][author] = []
        info[lang][author].append(title)

for lang in info:
        print(lang)
        for author in info[lang]:
                print("\t", author)
                for title in info[lang][author]:
                        print("\t\t", title)
```