Data Structure: Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example

```
fruitset = { “apple”, “Orange”, “Kiwi”, “banana”, “cherry”}
```

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 unordered – we cannot assume the order of elements in a set.**
- **Set is an iterable – eager and not lazy**
- **We cannot index on a set. You cannot access items in a set by referring to an index, since sets are unordered the items has no index.**
- **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

- **Deduplication : removal of repeated elements**
- **Finding unique elements**
- **Comparing two iterables for common elements or differences.**

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
{40, 10, 50, 20, 30}

# 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 is an iterable

a = { 10, 30, 10, 40, 20, 50, 30 }
for i in a :
      print(i, end = " ")
print()
40 10 50 20 30

# 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 diference
print(s1 - s2) # {2, 4}

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

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

# Creates a set; initialized by calling the constructor of set
s3 = set()
s4 = set([11, 33, 22, 11, 33, 11, 11, 44, 22])
print(s3)      # set()
print(s4)      # { 33, 11, 44, 22 }

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 i in a :
      print(i, 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 diference**
    ```
    print(s1 - s2) # {2, 4}
    print(s2 - s1) # {7, 9}
    ```
  - **# symmetric diference**
    ```
    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,   it is <class dict>

# not good; it works
# version 1
s = set()
for e in range(2, n + 1):
      s.add(e)
print(s, type(s))    #{2, 3, 4, 5, 6, 7, 8, 9, 10} <class 'set'>

# version 2
s = set(range(2, n + 1))
print(s, type(s))    # {2, 3, 4, 5, 6, 7, 8, 9, 10} <class 'set'>
#-------------------------------------

# is set empty or not
s1 = set()
s2 = set("fool")      # len(s2) is 3
print("empty : ", len(s1) == 0)     # empty : True
print("empty : ", len(s2) == 0)     # empty : False

"""
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 ele in range(2, 10, 2) :
      s3.remove(ele)
print(s3)

s3 = set(range(10)) # s3 is {1, 2, 3, 4, 5, 6, 7, 8, 9}
s3 = s3 - set(range(2, 10, 2))   # set(range(2, 10, 2)) is {8, 2, 4, 6}
print(s3)   # {0, 1, 3, 5, 7, 9}
```

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     # preferred, but empty set is False; non-empty set is True`

- **Remove elements from a set**

  Given a set 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))
```

```
Enter an integer: 30
2 3 5 7 11 13 17 19 23 29
```

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?**

Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
`add()`	Adds an element to the set
`clear()`	Removes all the elements from the set
`copy()`	Returns a copy of the set
`difference()`	Returns a set containing the difference between two or more sets
`difference_update()`	Removes the items in this set that are also included in another, specified set
`discard()`	Remove the specified item
`intersection()`	Returns a set, that is the intersection of two other sets
`intersection_update()`	Removes the items in this set that are not present in other, specified set(s)
`isdisjoint()`	Returns whether two sets have a intersection or not
`issubset()`	Returns whether another set contains this set or not
`issuperset()`	Returns whether this set contains another set or not
`pop()`	Removes an element from the set
`remove()`	Removes the specified element
`symmetric_difference()`	Returns a set with the symmetric differences of two sets
`symmetric_difference_update()`	inserts the symmetric differences from this set and another
`union()`	Return a set containing the union of sets
`update()`	Update the set with the union of this set and others

Data Structure: dict() : dictionary :

A dictionary is a collection which is unordered, mutable and indexed through keys. In Python dictionaries are written with curly brackets, and they have key-value pairs.

```
cardict = { "brand": "Ford",
            "model": "Mustang",
            "year": 1964,
            "colors":["red", "blue", "yellow", "grey"]
          }
```

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

- **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 sorts 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)
```

Python Dictionaries

- A dictionary is a collection which is unordered, mutable and indexed through keys.
- In Python dictionaries are written with curly brackets, and they have key-value pairs.
- Each key-value pair in a Dictionary is separated by a colon(**:)**, whereas many **key:value** pairs are separated by a comma.
- A Dictionary in Python works similar to the Dictionary in a real world. **Keys of a Dictionary must be unique and of *immutable* data type such as Strings, Integers and tuples**, but the values associated with keys can be repeated and be of any type.

Example

Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
x = thisdict["model"]
```

There is also a method called `get()` that will give you the same result:

Example

Get the value of the "model" key:

```
x = thisdict.get("model")
```

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

```
thisdict = { "brand": "Ford",  "model": "Mustang",  "year": 1964 }
thisdict["year"] = 2018
```

Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

Example

Print all key names in the dictionary, one by one:

```
for x in thisdict:
  print(x)
```

Example

Print all *values* in the dictionary, one by one:

```
for x in thisdict:
  print(thisdict[x])
```

Example

You can also use the values() function to return values of a dictionary:

```
for x in thisdict.values():
  print(x)
```

Example

Loop through both *keys* and *values*, by using the items() function:

```
print(thisdict.items())
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

for x, y in thisdict.items():
  print(x, y)
brand Ford
model Mustang
year 1964
```

Check if Key Exists

To determine if a specified key is present in a dictionary use the `in` keyword:

Example

Check if "model" is present in the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Dictionary Length

To determine how many items (key-value pairs) a dictionary has, use the `len()` method.

Example

Print the number of items in the dictionary:

```
print(len(thisdict))
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Removing Items

There are several methods to remove items from a dictionary:

Example

The `pop()` method removes the item with the specified key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Example

The `popitem()` method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

Example

The `del` keyword removes the item with the specified key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

Example

The `del` keyword can also delete the dictionary completely:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

Example

The `clear()` keyword empties the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

Copy a Dictionary

You cannot copy a dictionary simply by typing `dict2 = dict1`, because: `dict2` will only be a *reference* to `dict1`, and changes made in `dict1` will automatically also be made in `dict2`.

There are ways to make a copy, one way is to use the built-in Dictionary method `copy()`.

Example

Make a copy of a dictionary with the `copy()` method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

Another way to make a copy is to use the built-in method `dict()`.

Example

Make a copy of a dictionary with the `dict()` method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.

Example

Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

Or, if you want to nest three dictionaries that already exists as dictionaries:

Example

Create three dictionaries, than create one dictionary that will contain the other three dictionaries:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

The dict() Constructor

It is also possible to use the `dict()` constructor to make a new dictionary:

Example

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
print(thisdict)
```

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
values()	Returns a list of all the values in the dictionary
copy()	They copy() method returns a shallow copy of the dictionary.
clear()	The clear() method removes all items from the dictionary.
pop()	Removes and returns an element from a dictionary having the given key.
popitem()	Removes the arbitrary key-value pair from the dictionary and returns it as tuple.
get()	It is a conventional method to access a value for a key.
str()	Produces a printable string representation of a dictionary.
update()	Adds dictionary dict2's key-values pairs to dict
setdefault()	Set dict[key]=default if key is not already in dict
keys()	Returns list of dictionary dict's keys
items()	Returns a list of dict's (key, value) tuple pairs
has_key()	Returns true if key in dictionary dict, false otherwise
fromkeys()	Create a new dictionary with keys from seq and values set to value.
type()	Returns the type of the passed variable.
cmp()	Compares elements of both dict.