Chapter-5 String,List, Tuple and Dictionary

Prepared By:
Prof. Vishal A. Polara
Asst. Prof.
IT Department
BVM Engg. College

Outline

- String
- Tuples
- Lists
- Dictionaries
- Set
- Frozenset

Strings:

- Strings are *immutable.*
- It Can use single or double quotes, and three double quotes for a multi-line string.
- It is recommended to use single quotes.
- String literal can be span multiple line using backslash(\) at the end of each line.
- E.g. ‘abc’ or “abc”
- There is no char type like in C++ or Java.
- + is overloaded to do concatenation.

Example:

- >>> greeting = 'Hello, world!'
- >>> greeting[0] = 'J'
- TypeError: 'str' object does not support item assignment
- >>> greeting = 'Hello, world!'
- >>> new_greeting = 'J' + greeting[1:]
- >>> print(new_greeting)
- Jello, world!

String and Operators

- >>> x = 'hello'
- >>> x = x + ' there'
- >>> x
- 'hello there'
- >>>’3’+’4’
- 34
- >>>len(x)
- 10
- >>> s="abc"
- >>> p=s*3
- >>> p
- 'abcabcabc'

String Indexing and Slicing

- **Indexing** can be used to extract individual character from a string.
- **Slicing** is used to extract substrings of arbitrary length. If s is string the expression s[start:end] denotes the substring of s that starts at index start and ends at index end -1.
- E.g. ‘abc’[1:3]=‘bc’

String Indexing

B	I	R	L	A
0	1	2	3	4
-5	-4	-3	-2	-1

Examples:

```
>>> s = '012345'
>>> s[3]
'3'
>>> s[1:4]
'123'
>>> s[2:]
'2345'
>>> s[:4]
'0123'
>>> s[-2]
'4'
```

- **len**(String) – returns the number of characters in the String
- **str**(Object) – returns a String representation of the Object

```
>>> len(x)
6
>>> str(10.3)
'10.3'
```

Slicing of String

```
>>>str1="this is python“
>>> print("slice of string",str1[1:4:1])
slice of string his
>>> print("slice of string",str1[0:-1:2])
slice of string ti spto
>>> print("slice of string",str1[1:4:2])
slice of string hs
>>> print("slice of string",str1[1:8:1])
slice of string his is
```

Use of In Operators

```
>>> fruit = 'banana'
>>> len(fruit)
6
>>> last = fruit[length-1]
>>> print(last)
a
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

String Parsing

- It is the way of finding substring from a given string.
- >>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
- >>> atpos = data.find('@')
- >>> print(atpos)
- 21
- >>> sppos = data.find(' ',atpos)
- >>> print(sppos)
- 31
- >>> host = data[atpos+1:sppos]
- >>> print(host)
- uct.ac.za
- >>>

String Methods

Function Name	Description
S.lower()	Convert string in lower case
S.upper()	Convert stting in Upper case
S.strip() - (s.lstrip(), rstrip())	Use to remove space before an after string (by default left side)
S.isalpha(), S.isdigit(), S.isspace()	To check sting is alphabetic, dig or space
S.startswith('Word'), S.endswith('Word')	To find string is start or end with specific character or word
S.find('keyword'),S.find('word',3)	Use to find word in a given strin
S.replace('old','new')	To replace old word with new word
S.split('delimiter')	To split space using space or ,

- >>> a="Hello world"
- >>> type(a)
- <class 'str'>
- >>> dir(a)
- ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

- >>> help(p.capitalize)

Help on built-in function capitalize:

capitalize(...) method of built in s.str instance

 S.capitalize() -> str

Return a capitalized version of S, i.e. make the first character

have upper case and the rest lower case.

Examples:

- >>> s="hello world"
- >>> s.upper()
- 'HELLO WORLD'
- >>> s.lower()
- 'hello world'
- >>> s.strip()
- 'hello world'
- >>> s.isalpha()
- False
- >>> s.isdigit()
- False
- >>> s.isspace()
- False

```
>>> s.startswith("hello")
True
>>> s.endswith("world")
True
>>> s.find("world")
6
>>> s.find("ll",7)
-1
>>> s.replace("world","how are you")
'hello how are you‘
>>> s.split(' ')
['hello', 'world']
>>> s="hello“
>>> s.isalpha()
True
```

Example:

```
>>> s="hello world“
>>> p="how"
>>> s.join(p)
'hhello worldohello worldw'
```

Processing String using loop

- word = 'banana'
- count = 0
- **for letter in word:**
  - **if letter == 'a':**
    - count = count + 1
- print(count)

Lists

- It is an Ordered collection of data
- It contains data of different types unlike string.
- Lists are *mutable.*
- Few operations of strings are also available for List.
- There is a difference of (parenthesis) while defining values between list and tuples.
- It is also possible to create two dimension list.
- A list within another list is *nested.*

Examples:

- >>> x=[]   //empty list
- >>> x
- []
- >>> x.insert(0,10)
- >>> x.insert(1,20)
- >>> x
- [10, 20]
- >>> x = [1,'hello', (3 + 2j)]
- >>> x[2]
- (3+2j)

Operator operation on List

- >>> a = [1, 2, 3]
- >>> b = [4, 5, 6]
- >>> c = a + b
- >>> print(c)
- [1, 2, 3, 4, 5, 6]
- >>> [0] * 4
- [0, 0, 0, 0]
- >>> [1, 2, 3] * 3
- [1, 2, 3, 1, 2, 3, 1, 2, 3]

Convert string into list

- >>> s='vishal'
- >>> t=list(s)
- >>> print(t)
- ['v', 'i', 's', 'h', 'a', 'l']
- >>> s="my name is vishal"
- >>> t=s.split()
- >>> print(t)
- ['my', 'name', 'is', 'vishal']

In Operator

- >>> x = [1,'hello', (3 + 2j)]
- >>> 1 in x
- True
- >>> 'hello' in x
- True

List Slicing

- >>> t = ['a', 'b', 'c', 'd', 'e', 'f']
- >>> t[1:3]
- ['b', 'c']
- >>> t[:4]
- ['a', 'b', 'c', 'd']
- >>> t[3:]
- ['d', 'e', 'f']
- >>> t[:]
- ['a', 'b', 'c', 'd', 'e', 'f']
- >>> t = ['a', 'b', 'c', 'd', 'e', 'f']
- >>> t[1:3] = ['x', 'y']   //Updating Multiple elements
- >>> print(t)
- ['a', 'x', 'y', 'd', 'e', 'f']

Traversing a list

- >>> for i in x: //Traversing a list
-         print(i)
- 1
- hello
- (3+2j)

Identical and Equality

- If two objects are identical, they are also equivalent, but if they are equivalent, they are not necessarily identical.
- X=[1,3,4]
- Y=[1,3,4]
- Two lists are equivalent because they have the same elements but not identical because they are not same objects.

Examples:

```
>>> x=[1,2,3]
>>> y=x
>>> id(x)
33141624
>>> id(y)
33141624
>>> y
[1, 2, 3]
>>> y[0]=5
>>> x
[5, 2, 3]
>>> y
[5, 2, 3]
```

List Methods

Function Name	Description
L.Insert(I, e)	Inserts the object e into L at index I
L.Append(e)	Adds the object e to the end of L
L.Extend([L1])	Adds the items in list L1 to the end of L
L.Remove(e)	Deletes the first occurrence of e from L
L.Index(e)	Returns the index of the first occurrence of e in L. It raises an exception if e is not in L
L.Pop(i)	Removes and returns the item at index i in L. if is omitted, it defaults to -11 to remove and return the last element of L
L.Sort()	Sorts the elements of L in ascending order.
L.Reverse()	Reverses the order of the elements in L.
L.Count(e)	Returns the number of times that e occurs in L

Append and Extend

- The method append is used to modifies the list
- Extend takes a Entire list as an argument.
- Append takes a singleton or Single element as an argument.

- >>> t = ['a', 'b', 'c']
- >>> t.append([‘d‘,’e’])
- >>> print(t)
- ['a', 'b', 'c', ['d‘,’e’]]
- >>> t1 = ['a', 'b', 'c']
- >>> t2 = ['d', 'e']
- >>> t1.extend(t2)
- >>> print(t1)
- ['a', 'b', 'c', 'd', 'e']

Example:

- >>> L.append(6)
- >>> L
- [1, 2, 3, 4, 5, 6]
- >>> L.count(1)
- 1
- >>> L.insert(7,4)
- >>> L
- [1, 2, 3, 4, 5, 6, 4]
- >>> L.extend([1,2,4])
- >>> L
- [1, 2, 3, 4, 5, 6, 4, 1, 2, 4]
- >>> L.extend([2])
- >>> L
- [1, 2, 3, 4, 5, 6, 4, 1, 2, 4, 2]

- >>> L.remove(5)
- >>> L
- [1, 2, 3, 4, 6, 4, 1, 2, 4, 2]
- >>> L.index(6)
- 4
- >>> L.count(4)
- 3
- >>> L.pop(8)
- 4

- >>> L.sort()
- >>> L
- [1, 1, 2, 2, 2, 3, 4, 4, 6]
- >>> L.reverse()
- >>> L
- [6, 4, 4, 3, 2, 2, 2, 1, 1]
- >>> a=[1,3,3,6,5]
- >>> sorted(a,reverse=True)
- [6, 5, 3, 3, 1]

List Methods

- >>>li = [‘a’, ‘b’, ‘c’, ‘b’]
- >>> li.index(‘b’) # index of 1st occurrence
- 1
- >>> li.count(‘b’) # number of occurrences
- 2
- >>> li.remove(‘b’) # remove 1st occurrence
- >>> li
-   [‘a’, ‘c’, ‘b’]
- >>> t = ['d', 'c', 'e', 'b', 'a']
- >>> t.sort()
- >>> print(t)
- ['a', 'b', 'c', 'd', 'e']
- >>> li.sort(some_function)
-     # sort in place using user-defined comparison

- >>> nums=[3,5,6,7,10,15]
- >>> print(len(nums))
- 6
- >>> print(max(nums))
- 15
- >>> print(min(nums))
- 3
- >>> print(sum(nums))
- 46
- >>> print(sum(nums)/len(nums))
- 7.666666666666667
- >>>t=[‘my’,’name’,’is’,’vishal’]
- >>> x=t.pop(1)

Use of Del and Remove

- >>> t=[‘my’,’name’,’vishal’]
- name
- >>> del t[1]
- >>> print(t)
- ['my', 'vishal']
- >>> t.remove('my')
- >>> print(t)
- ['vishal']
- >>> li = [5, 2, 6, 8]
- >>> li.reverse()   # reverse the list *in place*
- >>> li
-   [8, 6, 2, 5]

Two dimension list

- >>> list=[1,2,3,[1,2,3]]
- >>> list
- [1, 2, 3, [1, 2, 3]]
- >>> list[0]
- 1
- >>> list[1]
- 2
- >>> list[2]
- 3
- >>> list[3][0]
- 1
- >>> list[3][1]
- 2
- >>> list[3][2]
- 3

- >>> list=[[1,2,3],[4,5,6],[7,8.9]]
- >>> list
- [[1, 2, 3], [4, 5, 6], [7, 8.9]]
- >>> list[0][0]
- 1
- >>> list[1][0]
- 4
- >>> list[2][0]
- 7

Making string out of a list

- >>> h=['hello','world']
- >>> h
- ['hello', 'world']
- >>> a=' '.join(h)
- >>> a
- 'hello world'

List Comprehension

- It is easy to write an expression which expands on whole list
- Syntax: [expr for var in list]

- >>> i=[1,2,3]
- >>> result=[x**2 for x in i]
- >>> result
- [1, 4, 9]
- >>> min=[n for n in i if n<=2]
- >>> min
- [1, 2]

- >>> fruits=['mango','dates','orange']
- >>> fruitss=[s.upper() for s in fruits if 'a' in s]
- >>> fruitss
- ['MANGO', 'DATES', 'ORANGE']

Parsing Lines using list

- From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

- fhand = open('mbox-short.txt')
- **for line in fhand:**
  - line = line.rstrip()
  - **if not line.startswith('From '): continue**
  - words = line.split()
  - print(words[2])

Tuples

- Tuples are ordered sequences of elements.
- The elements of a tuples need not be characters.
- The individual elements can be of any type, and need not be of the same type as each other.
- Tuples are immutable.
- Literals of type tuples are written by enclosing a comma-separated list of elements within parentheses.
- Trailing comma only required for singletons.
- The immutability of tuples means they’re faster than lists.
- It is lightweight then list.

Example:

- >>> t = tuple()
- >>> print(t)
- ()
- >>> x = (1,2,3)
- >>> x[1:]
- (2, 3)
- >>> y = (‘a’,)
- >>> t2=('a‘)
- >>> type(y)
- <class 'tuple'>
- >>> type(t2)
- <class 'str'>

- >>> t = tuple('lupins')
- >>> print(t)
- ('l', 'u', 'p', 'i', 'n', 's')
- >>> t = ('a', 'b', 'c', 'd', 'e')
- >>> print(t[0])
- *'a‘*
- >>> print(t[1:3])
- ('b', 'c')
- >>> t[0]='A'
- Traceback (most recent call last):
-   File "<pyshell#15>", line 1, in <module>
-     t[0]='A'
- TypeError: 'tuple' object does not support item assignment

- >>> t = ('A',) + t[1:]
- >>> print(t)
- ('A', 'b', 'c', 'd', 'e')
- >>> (0, 1, 2) < (0, 3, 4)
- True
- >>> (0, 1, 2000000) < (0, 3, 4)
- True

- >>> (0,1,2)<(0,3,4)
- True
- >>> m=['have','fun']
- >>> x,y=m
- >>> x
- 'have'
- >>> y
- 'fun'
- >>> m=['hello','world']
- >>> (x,y)=m
- >>> x
- 'hello'
- >>> y
- 'world'

Function for string, tuples and list

Function Name	Description
Seq[i]	Returns the ith element in the sequence
Len(seq)	Returns the length of the sequence
Seq1 + seq2	Returns the concatenation of the two sequences
N * seq	Returns a sequence that repeats seq n times
Seq [start:end]	Returns a slice of the sequence
E in seq	Returns true if e is contained in the sequence and false otherwise
E not in seq	Returns true if e is not in the sequence and false otherwise
For e in seq	Iterates over the elements of the sequence

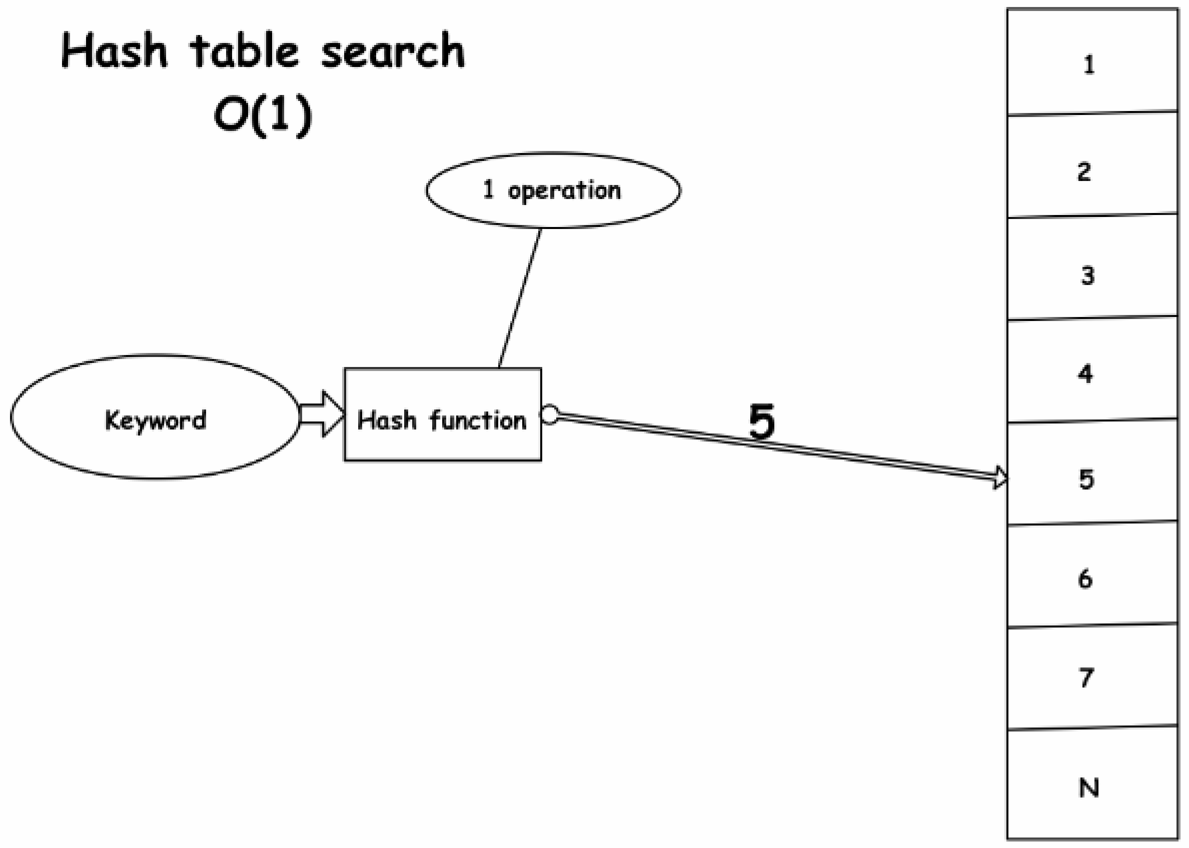
Type	Type of elements	Examples of literals	Mutable
Str	Characters	‘ ‘, ’a’ , ’abc’	No
Tuple	Any type	( ) ,(3,) ,(‘abc’,4)	No
List	Any type	[ ], [3] ,[‘abc’,4]	Yes

Dictionaries

- A set of key-value pairs.
- Dictionaries are *mutable,* but key must be immutable.
- It uses a hash table concept for storing value.
- They are not ordered, we call them keys rather than indices.
- Dictionary keys are case sensitive.
- Incase of duplicate keys only last key will be visible.
- Literals of type dict are enclosed in curly braces and each element is written as a key followed by a colon followed by a value.

Hash Table working

- I(x)= x modulo k
- Here x is word and k is table_size



Example:

- >>> fruits=dict()
- >>> fruits
- {}
- >>> fruits['Apple']=3
- >>> fruits['mango']=2
- >>> fruits['Orange']=6
- >>> fruits
- {'Apple': 3, 'Orange': 6, 'mango': 2}
- >>> fruits['mango']=5
- >>> fruits
- {'Apple': 3, 'Orange': 6, 'mango': 5}

Continue…

- >>> del(fruits['mango'])
- >>> fruits
- {'Apple': 3, 'Orange': 6}
- >>> fruits.items()
- dict_items([('Apple', 3), ('Orange', 6)])
- >>> fruits.clear()
- >>>fruits
- {}

Tuple as a key

- >>> number={}
- >>> number[('vishal','patel')]=9838483833
- >>> number[('rajesh','patel')]=9343934934
- >>> number
- {('rajesh', 'patel'): 9343934934, ('vishal', 'patel'): 9838483833}

How to Copy List and Dictionary ?

- The built-in **list** function will copy a list
- The dictionary has a method called **copy**

```
>>> l1 = [1]
>>> l2 = list(l1)
>>> l1[0] = 22
>>> l1
[22]
>>> l2
[1]
```

```
>>> d = {1 : 10}
>>> d2 = d.copy()
>>> d[1] = 22
>>> d
{1: 22}
>>> d2
{1: 10}
```

Dictionary Functions

Function Name	Description
Len(d)	Returns the number of items in d
d.Keys()	Returns a list containing the keys in d
d.Values()	Returns a list containing the values in d
K in d	Returns true If key k is in d
D[k]	Returns the item in d with key k
d.Get(k,v)	Returns d[k] if k is in d, and v can be message otherwise
D[k]=v	Associates the value v with the key k in d
Del d[k]	Removes the key k from d
For k in d	iterates over the keys in d

Example:

- >>> Dict={'fruit':'apple','elect':'blub','auto':'Wheels}
- >>> Dict
- {'elect': 'blub', 'auto': 'Wheels', 'fruit': 'apple'}
- >>> len(Dict)
- 3
- >>> Dict.keys()
- dict_keys(['elect', 'auto', 'fruit'])
- >>> Dict.values()
- dict_values(['blub', 'Wheels', 'apple'])
- >>> 'fruit'in Dict
- True

Continue…

- >>> Dict['fruit']
- 'apple’
- >>> Dict.get('fruit','apple')
- 'apple'
- >>> Dict['fruit']='orange'
- >>> Dict
- {'elect': 'blub', 'auto': 'Wheels', 'fruit': 'orange'}
- >>> del Dict['auto']
- >>> Dict
- {'elect': 'blub', 'fruit': 'orange'}

```
>>> d={1:'apple',2:'mango',3:'lemon'}
>>> l=list()
>>> for key,val in d.items():
        l.append((val,key))
>>> l
[('apple', 1), ('mango', 2), ('lemon', 3)]
```

Convert Dictionaries to List

```
>>> d={'a':10,'b':20,'c':30}
>>> l=list()
>>> for key,val in d.items():
        l.append((val,key))
>>> d
{'c': 30, 'a': 10, 'b': 20}
>>> l
[(30, 'c'), (10, 'a'), (20, 'b')]
>>> l.sort()
>>> l
[(10, 'a'), (20, 'b'), (30, 'c')]
```

Histogram

- word = 'brontosaurus'
- d = dict()
- **for c in word:**
  - **if c not in d:**
    - d[c] = 1
  - **else:**
    - d[c] = d[c] + 1
- print(d)

Count word in File

- fname = input('Enter the file name: ')
- **try:**
  - fhand = open(fname)
- **except:**
  - print('File cannot be opened:', fname)
  - exit()
- counts = dict()
- **for line in fhand:**
  - words = line.split()
  - **for word in words:**
    - **if word not in counts:**
      - counts[word] = 1
    - **else:**
      - counts[word] += 1
- print(counts)

Set

- Set is mutable. It is possible to add or remove item from it.
- It is an unordered collection of items.
- Every element is unique and immutable.
- It is normally used to remove duplicates from a sequence and for mathematical operations such as union, intersection, difference.
- It is created by using built in function set() or writing values inside {} brackets.
- It is unordered so we can perform slicing or indexing.
- Add() method is used to add single element.
- Update() method adds multiple element.

Examples:

- a=set()
- >>> a={1,2,3}
- >>> type(a)
- <class 'set'>
- >>> b=set()
- >>> type(b)
- <class 'set'>
- >>> a.add(4)
- >>> a.add(5)
- >>> a
- {1, 2, 3, 4, 5}

- >>> l=[10,20,30]
- >>> a.update(l)
- >>> a
- {1, 2, 3, 4, 5, 10, 20, 30}

Set methods and Function

Method Name	Description
Discard()	If item doesnot found it remians unchanged
Remove()	If item doesnot found it gives an error
Pop()	Remove element from leftside
Clear()	Clear all item from set
Len()	Find length of set
Max(), min()	Find max and min elements
Sorted()	Sort elements
Sum()	Do addition of all elements

```
>>> a
{1, 2, 3, 4, 5, 10, 20, 30}
>>> a.pop()
1
>>> a.pop()
2
>>> len(a)
8
>>> max(a)
30
>>> min(a)
1
>>> sorted(a)
[1, 2, 3, 4, 5, 10, 20, 30]
>>> sum(a)
75
```

```
>>> a.discard(40)
>>> a
{3, 4, 5, 10, 20, 30}
>>> a.discard(10)
>>> a
{3, 4, 5, 20, 30}
>>> a.remove(40)
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    a.remove(40)
KeyError: 40
```

Set Operations

- Union (| operator, A.union(B))
- Intersection (& operator, A.intersection(B))
- Set difference (- operator, A.difference(B))
- Symmetric difference (^ operator, A.symmetric_difference(B))
- Sub set (issubset())
- Super set (issuperset())

Example:

- >>> a={1,2,3,4,5}
- >>> b={3,4,5,6,8}
- >>> a|b
- {1, 2, 3, 4, 5, 6, 8}
- >>> a&b
- {3, 4, 5}
- >>> a-b
- {1, 2}
- >>> b-a
- {8, 6}
- >>> a^b
- {1, 2, 6, 8}

- >>> a.issubset(b)
- False
- >>> b.issubset(a)
- True
- >>> a.issuperset(b)
- True

Frozenset

- Frozendset is same as set but the value of frozenset can not be modifed so update and remove will not work for frozenset.
- It can be created using frozenset() function

Example:

- >>> a={1,2,3}
- >>> type(a)
- <class 'set'>
- >>> b=frozenset(a)
- >>> b
- frozenset({1, 2, 3})
- >>> a.add(6)
- >>> a
- {1, 2, 3, 6}

- >>> b.add(5)
- Traceback (most recent call last):
-   File "<pyshell#14>", line 1, in <module>
-     b.add(5)
- AttributeError: 'frozenset' object has no attribute 'add'

Time of execution of data types

- >>> from timeit import timeit
- >>> timeit("[1,2,3,4,5]")
- 0.3780898293932045
- >>> timeit("(1,2,3,4,5)")
- 0.033304236751957816
- >>> timeit("{1,2,3,4,5)")
- >>> timeit("{1,2,3,4,5}")
- 0.46791443209826156
- >>> timeit("{1:'o',2:'p',3:'a',4:'q'}")
- 0.7418759116919205

Thank You