413-513 Notes- Week 6

**[ Dictionaries and Sets ]**

**[ Dictionaries ]**

A Python dictionary is a container of key-value pairs. It is mutable and can contain mixed types. A dictionary is an unordered collection. Python dictionaries are called associative arrays or hash tables in other languages. The keys in a dictionary must be immutable objects like strings or numbers. They must also be unique within a dictionary.

There are many ways to create a dictionary - check it

**[ Creating Dictionaries ]**

**# dictionary for days of the week**

**weekend = { "Sun": "Sunday", "Mon": "Monday" }**

**# dictionary for ordinal numbers**

**vals = dict(one=1, two=2)**

**# dictionary of state capitcals**

**capitals = {}**

**capitals["il"] = "SpringField"**

**capitals["wisc"] = "Madison"**

**capitals["misso"] = "Jefferson City"**

**# dictionary to generate values**

**d = { i: object() for i in range(4) }**

**#\*use dict comprehension ( defined below )**

**# displaying the contents of the dictionaries**

**print (weekend)**

**print (vals)**

**print (capitals)**

**print (d)**

\*A dictionary is created using a dictionary comprehension. The comprehension has two parts. The first part is the i: object() expression, which is executed for each cycle of a loop. The second part is the for i in range(4) loop. The dictionary comprehension creates a dictionary having four pairs, where the keys are numbers 0, 1, 2, and 3 and the values are simple objects.

**[ Some Supportive Dictionary Functions ]**

[len(dict)](http://www.tutorialspoint.com/python/dictionary_len.htm)

Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

[str(dict)](http://www.tutorialspoint.com/python/dictionary_str.htm)

Produces a printable string representation of a dictionary

[type(variable)](http://www.tutorialspoint.com/python/dictionary_type.htm)

Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

[dict.clear()](http://www.tutorialspoint.com/python/dictionary_clear.htm)

Removes all elements of dictionary dict

[dict.copy()](http://www.tutorialspoint.com/python/dictionary_copy.htm)

Returns a shallow copy of dictionary dict

[dict.fromkeys()](http://www.tutorialspoint.com/python/dictionary_fromkeys.htm)

Create a new dictionary with keys from seq and values set to value.

[dict.get(key, default=None)](http://www.tutorialspoint.com/python/dictionary_get.htm)

For key key, returns value or default if key not in dictionary

[dict.has\_key(key)](http://www.tutorialspoint.com/python/dictionary_has_key.htm)

Returns true if key in dictionary dict, false otherwise

[dict.items()](http://www.tutorialspoint.com/python/dictionary_items.htm)

Returns a list of dict's (key, value) tuple pairs

[dict.keys()](http://www.tutorialspoint.com/python/dictionary_keys.htm)

Returns list of dictionary dict's keys

[dict.update(dict2)](http://www.tutorialspoint.com/python/dictionary_update.htm)

Adds dictionary dict2's key-values pairs to dict

[dict.values()](http://www.tutorialspoint.com/python/dictionary_values.htm)

Returns list of dictionary dict's values

**[ Deleting Dictionary Elements ]**

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. The following is a simple example −

**# define a dictionary of names**

**dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};**

**del dict['Name']; # remove entry with key 'Name'**

**dict.clear(); # remove all entries in dict**

**del dict ; # delete entire dictionary**

**print ("dict['Age']: ", dict['Age'])**

**print ("dict['School']: ", dict['School'])**

**This produces the following result. Note that an exception is raised because after del dict dictionary does not exist any more −**

**dict['Age']:**

**Traceback (most recent call last):**

**File "test.py", line 8, in <module>**

**print "dict['Age']: ", dict['Age'];**

**TypeError: 'type' object is unsubscriptable**

**[ Example of a Dictionary ]**

**#dictionary example.**

**dict = {"Cubs": ["Schwarber", "Castro", "Rizzo"], "Mets": ["Ugly", "Dopey", "Sneezy"]}**

**print(dict)**

**print(dict.keys())**

**players = dict.get("Cubs", "Team not found.")**

**print(players)**

**if "Cubs" in dict:**

**print("Found the Cubs!")**

**for keys, values in dict.items():**

**if "Schwarber" in values:**

**print("Found a Cubs player!")**

**for values in dict.values():**

**if "Rizzo" in values:**

**print("Found another Cubs player!")**

**dict.pop("Mets", "Team not found")**

**print(dict)**

**dict["Mets"] = ["Ugly", "Dopey", "Sneezy"]**

**print(dict)**

**team, players = dict.popitem()**

**print("Team [key] randomly removed:", team)**

**print("Players [values] randomly removed:", players)**

**print(dict)**

**#Exercise Morse Code Converter below**

**#key: value pairs**

**morse\_encode = {" ": "(space)", ",": "--..--", ".": ".-.-.-", "?": "..--..", "0": "-----", "1": ".----", "2": "..---", "3": "...--", "4": "....-", "5": ".....",**

**"6": "-....", "7": "--...", "8": "---..", "9": "----.", "A": ".-", "B": "-...",**

**"C": "-.-.", "D": "-..", "E": ".", "F": "..-.", "G": "--.", "H": "....",**

**"I": "..", "J": ".---", "K": "-.-", "L": ".-..", "M": "--", "N": "-.", "O": "---",**

**"P": ".--.", "Q": "--.-", "R": ".-.", "S": "...", "T": "-", "U": "..-",**

**"V": "...-", "W": ".--", "X": "-..-", "Y": "-.-", "Z": "--.."}**

**morse\_encode["&"] = ".-..." #Add to the dictionary**

**print(morse\_encode["I"]) #Access the value of a key**

**if "M" in morse\_encode:**

**print(morse\_encode["M"])**

**for key in morse\_encode:**

**print(key, morse\_encode[key])**

**sentence = input("Please enter a string to convert into Morse Code: ").upper()**

**words = sentence.split(" ")**

**encoded\_words = []**

**for ch in sentence:**

**encoded\_words.append(morse\_encode.get(ch))**

**for w in words:**

**print(w, end= " ")**

**print()**

**for ew in encoded\_words:**

**print(ew, end=" ")**

**[ Sets ]**

Sets are collections of objects, numbers, names or things of the same type.

The individual items within a set are referred to as set elements.

Sets can be comprised of a finite or infinite set of elements.

A set can be considered as an ordered list.

There are ( 2 N − 1 ) proper subsets are there for a set with N members.

A subset can be comprised of zero or more elements from the original set.

**[ Example - Sets ]**

A set designated to hold the prime factors of 54 is defined as [ 2 3 3 3 ] .

**[ Set Operations - On One Set ]**

Operations on a particular set include:

• Membership: is an item in the set?

• Insert: adding an item to the set

• Remove: take an item from the set

• Size: the number of items in the set

**[ Example - Sets ]**

Set [ 3 1 5 8 6 ] ( without any significance in order )

• Membership: 5 is member but 9 is not

• Inserting 7 yields [ 3 1 7 5 8 6 ]

• Removing 6 yields [ 3 1 7 5 8 ]

• Size of set is 5

A Boolean type variable is generally used to check for set membership. Boolean types return TRUE or FALSE.

Set membership checking requires a linear search and is *O* ( *n* ) . *O* ( *n* ) means my order of magnitude is dependent of the size *n* of the set.

**[ Set Operations - Between Two Sets ]**

• Union – items that are in either set

• Intersection – items that are in both sets

• Difference – items in one set but not the other

**[ Example - Set Union ]**

If SetA is defined as [ a b c d e ] and SetB is defined as [ b c f ] , then SetA U SetB is defined as [ a b c d e f ] .

**[ Example - Set Intersection ]**

If SetA is defined as [ a b c d e ] and SetB is defined as [ b c f ] , then SetA ∩ SetB is defined as [ b c ] .

**[ Example - Set Difference ]**

If SetA is defined as [ 1 2 6 7 8 ] and SetB is defined as [ 0 2 3 4 7 9 ] , then

( SetA - SetB ) is defined as [ 1 6 8 ] .

**[ Example - Symmetric Difference ]**

If SetA is defined as [ 1 2 6 7 8 ] and SetB is defined as [ 0 2 3 4 7 9 ] , then

( SetA ~ SetB ) is defined as [ 0 1 3 4 6 8 9 ] .

i.e. SetA − SetB and SetB − SetA combined

**[ Applications of Sets: Bitwise Sets ]**

The bitwise shift operators are as follows:

• Right shift ( >> )

• Left shift ( << )

syntax: variable << number\_of\_bits variable >> number\_of\_bits

1 << 0 == 1

1 << 1 == 2

1 << 2 == 4

1 << 3 == 8

...

1 << 8 == 256

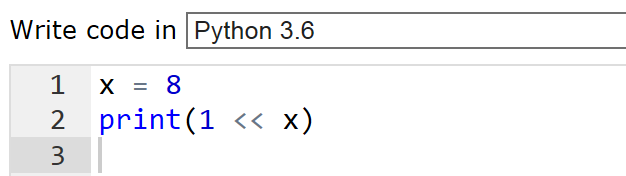
1 << 9 == 512

1 << 10 == 1024

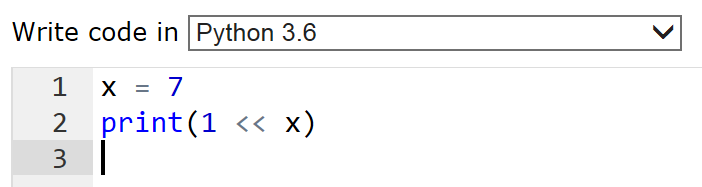
The value of a right - shift expression x >> y is: x / (2 \* y)

The value of a left - shift expression x << y is: x \* (2 \* y)

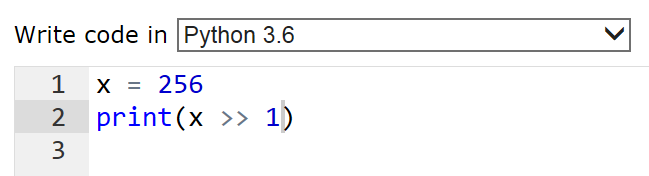
Example:



Returns 256



Returns 128



Returns 128

Compare this to the following C++ version:

#include <iostream>

using namespace std;

int main()

{

int x = 256;

cout << (x >> 1) << endl;

system("pause");

return 0;

}

Set of binary powers: { 1024 , 512, 256, 128, 64, 32, 16, 8, 4, 2, 1 }

8 as decimal is binary 0100

**[ Examples ]**

**(1)** What is the result of the given bit shift operation? cout << (1 << 10);

1024 2



**(2)** What is the result of this bit wise AND operation?

cout << (100 & 101);

1 AND 1 = 1 0 AND 0 = 0 0 AND 1 = 0



**(3)** What is the result of this bit wise OR operation? cout << (110 | 10);

****

1 OR 1 = 1 1 OR 0 = 1 0 OR 0 = 0

**(4)** What is the result of this bit wise operation? 100 << 11 | 20

****

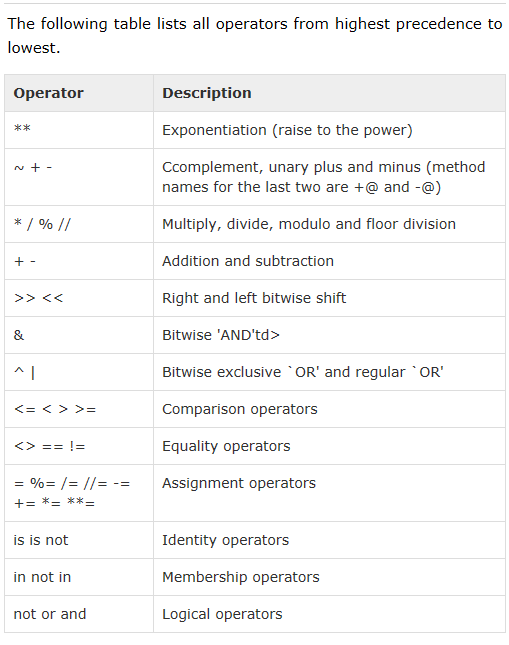
**(5)** What is the result of the given bit wise XOR operation? 100 ^ 111



1 XOR 1 = 0 1 XOR 0 = 1 0 XOR 1 = 1

<http://www.tutorialspoint.com/python/operators_precedence_example.htm>

Python bitwise operator precedence



**Python Bitwise Shift**

print 1 << 10



**Python Bitwise Comparison**

print (100 & 101)



print 110 | 10

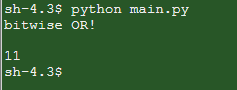


print 100 << 11 | 20



print "bitwise OR!\n"

print 100 ^ 111



**[ Applications ]**

**( Symmetric Difference Between Two Sets )**

If SetA is defined as [ a b c d k m p r ] and SetB is defined as [ a c d m p s ] , then determine the symmetric difference between SetA and SetB .

**[ Solution ]**

The symmetric difference ~ of two sets A and B is the set of all elements of A or B which are not in both A and B .

SetA = [ a b c d k m p r ]

SetB = [ a c d m p s ]

SetA ~ SetB = [ a b c d k m p r ] ~ [ a c d m p s ]

SetA ~ SetB = [ b k r s ]

**( Least Common Multiple )**

How would you use sets to find the least common multiple of positive integers *M* and

*N* , that is, the smallest number of which both *M* and *N* are factors?

**[ Solution ]**

One approach would be to use the set of all counting numbers ( natural numbers )

c = [ 1 2 3 4 5 6 . . . ] and then loop though each of these to check this condition:

**if(c % M == 0 && c % N == 0)**

**return c;**

**//c is the LCM**

For example, the least common multiple ( LCM ) of 48 and 32 is 96 .

**( Set Union and Intersection )**

Write the code of a Python program that simulates set union and set intersection operations. Declare two arrays defined as follows:

**SetA[ 5 ] = { 'b', 'c', 'd', 'e', 'h' };**

**SetB[ 7 ] = { 'a', 'b', 'd', 'f', 'g', 'm', 'n' };**

Then code your program to determine the elements of SetA ∪ SetB and SetA ∩ SetB .

**[ Solution ]**

One approach for a set union is:

# Python "Set Union"

U = set(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',   
'x', 'y', 'z'])

A = set( ['a', 'b', 'c'])

B = set( ['d', 'e', 'f'])

C = A.union(B)

print("Consider some universal set U =",U)

print("If A and B are two sets taken from the universe U such that")

print("A =",A)

print("and")

print("B =",B)

print("then")

print("C = (A | B) =",C)

In C++ one approach for a set intersection is:

**//test for intersection**

**char SetA[ 5 ] = { 'b', 'c', 'd', 'e', 'h' };**

**char SetB[ 7 ] = { 'a', 'b', 'd', 'f', 'g', 'm', 'n' };**

**bool test = false;**

**cout << "The intersection of sets SetA and SetB is\n";**

**for (int j = 0; j < 7; j++)**

**{**

**test = false;**

**for (int i = 0; i < 5; i++)**

**{**

**if (SetB[j] == SetA[i])**

**test = true;**

**}**

**if (test == true)**

**cout << SetB[j] << " ";**

**}**

**cout << endl;**

**Sample Python Program using Sets**

**A = set( ['a', 'a', 'b', 'b', 'c', 'c', 'd', 'e', 'f' ] )**

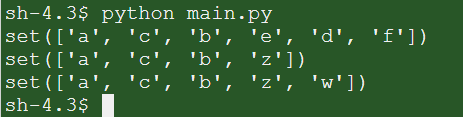
**B = set( ['a', 'b', 'c', 'z' ] )**

**C = set( ['a', 'b', 'c', 'w', 'z' ] )**

**print (A)**

**print (B)**

**print (B)**



**print('The Union A | B is a new set with elements from both A and B =', A | B)**

**print('The Intersection A & B is a new set with elements common to A and B =', A & B)**

**print('The Difference A - B is a new set with elements in A but not in B =', A - B)**

**print('The Symmetric Difference A ^ B is a new set with elements in either A or B but not both =', A ^ B)**

**print('Is B a subset of C? In other words, is every element in B also in C?',B <= C)**

**print('Is C a superset of B? In other words, is every element in B also in C?',C >= B)**

**Output Display**

**>>>**

**The Union A | B is a new set with elements from both A and B = {'z', 'd', 'e', 'f', 'a', 'b', 'c'}**

**The Intersection A & B is a new set with elements common to A and B = {'a', 'b', 'c'}**

**The Difference A - B is a new set with elements in A but not in B = {'d', 'e', 'f'}**

**The Symmetric Difference A ^ B is a new set with elements in either A or B but not both = {'z', 'd', 'e', 'f'}**

**Is B a subset of C? In other words, is every element in B also in C? True**

**Is C a superset of B? In other words, is every element in B also in C? True**

**>>>**

**[ Summary of Set Operations ]**

**The set operations include:**

**"Union" (|) all elements in either set A or set B**

**"Intersection" (&) all elements in set A in common with set B   
"Difference" (-) all elements in set A but not in set B**

**"Symmetric Difference" (^) all elements in set A or set B but not both**

**( also called "Exclusive Or" )**

**[ Applications ]**

**( ISBN Numbers and Sets of Digits )**

Books are assigned identification numbers called ISBNs. Each ISBN is a ten – digit number such as 1 - 55953 - 407 - **9** . The final digit is called the check digit which is determined on the basis of modular arithmetic. The final ( check ) digit is obtained as follows: start at the left and multiply the first nine digits of the ISBN number by 10 ,

9 , 8 , 7 , 6 , 5 , 4 , 3 and 2 , respectively. Then add these products. The check digit is

the smallest number that must be added to the result to arrive at a multiple of 11 .

**[ Example ]**

Show that the check digit ( right - most digit ) is 9 for the following ISBN :

1 - 55953 - 407 - **9**

**[ Solution ]**

|  |  |  |
| --- | --- | --- |
| ***digit*** | ***factor*** | ***product*** |
|  |  |  |
| 1 | 10 | 10 |
| 5 | 9 | 45 |
| 5 | 8 | 40 |
| 9 | 7 | 63 |
| 5 | 6 | 30 |
| 3 | 5 | 15 |
| 4 | 4 | 16 |
| 0 | 3 | 0 |
| 7 | 2 | 14 |
|  | ***sum*** | **233** |

233 compared to nearest factors of 11 :

|  |  |
| --- | --- |
| ***factors of 11*** | ***product*** |
|  |  |
| 198 | 11 × 18 |
| 209 | 11 × 19 |
| 220 | 11 × 20 |
| 231 | 11 × 21 |
| 242 | 11 × 22 |
| 253 | 11 × 23 |

The nearest multiple of 11 that is more than or equal to 233 is 242 .

The difference is : 242 - 233 = 9 , which is the check digit.

Python Lists

<http://www.tutorialspoint.com/python/python_lists.htm>

<http://www.tutorialspoint.com/execute_python_online.php>

<http://www.dotnetperls.com/set-python>

<http://www.programiz.com/python-programming/set>

Review: Creating a Set in Python

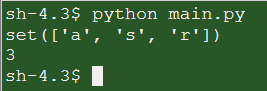
**# Create a set.**

**items = {"a", "s", "a", "a", "r"}**

**# Print set.**

**print(items)**

**print(len(items))**



**# Use in-keyword.**

**items = {"a", "s", "a", "a", "r"}**

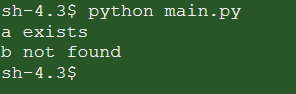
**if "a" in items:**

**print("a exists")**

**# Use not-in keywords.**

**if "" not in items:**

**print("b not found")**



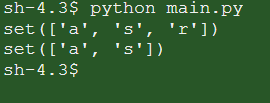
**# Use in-keyword.**

**items = {"a", "s", "a", "a", "r"}**

**print items**

**items.discard("r")**

**print items**



**# Use in-keyword.**

**items = {"a", "s", "a", "a", "r"}**

**print items**

**items.update("r")**

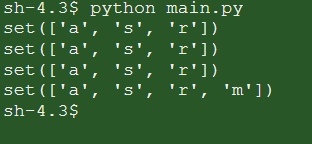
**print items**

**items = {"a", "s", "a", "a", "r"}**

**print items**

**items.update("m")**

**print items**



**numbers1 = {1, 3, 5, 7}**

**numbers2 = {1, 3}**

**# Is subset.**

**if numbers2.issubset(numbers1):**

**print("Is a subset")**

**# Is superset.**

**if numbers1.issuperset(numbers2):**

**print("Is a superset")**

**# Intersection of the two sets.**

**print(numbers1.intersection(numbers2))**

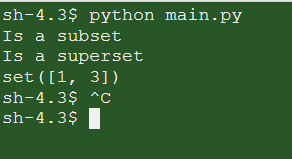
**#** **python multiple a list by a scalar**

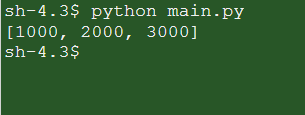
**values = {10, 20, 30}**

**# Multiply all values in the set by 100.**

**result = map(lambda x: x \* 100, values)**

**print result**





**Further example on Sets**

**#Sets**

**letters1 = set("ABCDEFGHIJKLMNOPQRSTUVWXYZ")**

**letters2 = set(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',**

**'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'])**

**print(letters1) #the values in the set are randomly inserted**

**print(letters2) #the same random seed is used here**

**numbers1 = set(range(0,10))**

**print(numbers1)**

**numbers2 = set(range(7,15))**

**print(numbers2)**

**numbers2.add(15)**

**print(numbers2)**

**numbers2.remove(9)**

**print(numbers2)**

**numbers2.discard(20) #No exception is raised**

**#numbers2.remove(20) #A KeyError exception is raised**

**numbers3 = numbers1.union(numbers2)**

**print(numbers3)**

**numbers4 = numbers1.intersection(numbers2)**

**print(numbers4)**

**In Python, when to use a Dictionary, List or Set?**

**-A comparison from Stackoverflow**

[**http://stackoverflow.com/questions/3489071/in-python-when-to-use-a-dictionary-list-or-set**](http://stackoverflow.com/questions/3489071/in-python-when-to-use-a-dictionary-list-or-set)

A list keeps order, dict and set don't: when you care about order, therefore, you must use list (if your choice of containers is limited to these three, of course;-).

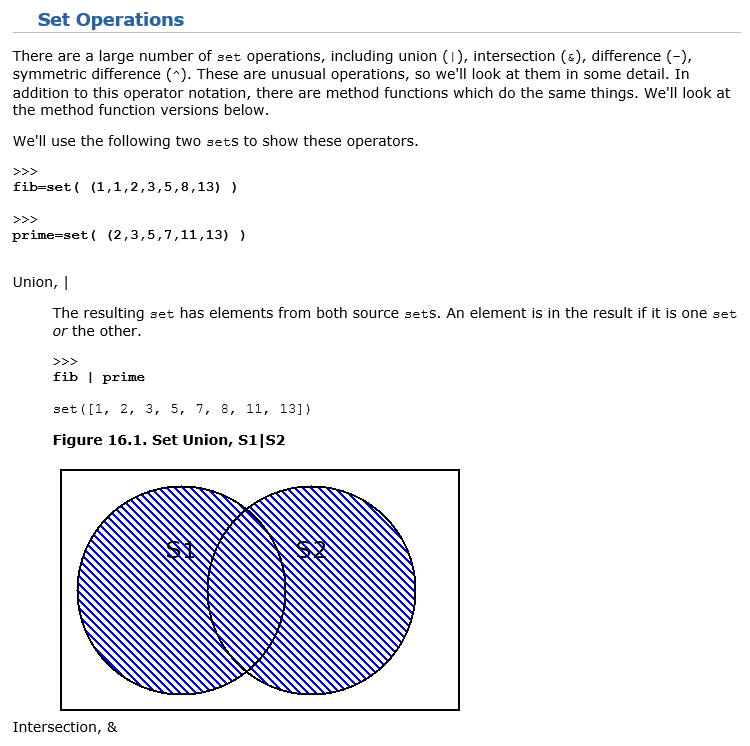
dict associates with each key a value, while list and set just contain values: very different use cases, obviously.

set requires items to be hashable, list doesn't: if you have non-hashable items, therefore, you cannot use set and must instead use list.

set forbids duplicates, list does not: also a crucial distinction. (A "multiset", which maps duplicates into a different count for items present more than once, can be found in collections.Counter -- you could build one as a dict, if for some weird reason you couldn't import collections, or, in pre-2.7 Python as a collections.defaultdict(int), using the items as keys and the associated value as the count).

Checking for membership of a value in a set (or dict, for keys) is blazingly fast (taking about a constant, short time), while in a list it takes time proportional to the list's length in the average and worst cases. So, if you have hashable items, don't care either way about order or duplicates, and want speedy membership checking, set is better than list.

Set Operations…



<http://www.linuxtopia.org/online_books/programming_books/python_programming/python_ch16s03.html>

[ Python Programming Examples ]

<https://www.programiz.com/python-programming/examples>

[ Routines for Sorting Data ]

• Bubble Sort a.k.a. Exchange Sort

• Insertion Sort

• Selection Sort

• Quicksort

• Merge Sort

• Heap Sort

• Shell Sort

• Solitaire Sort

[ Review of Python Set Union Operation ]

A = set( ['a', 'b', 'c', 'd'])

B = set( ['d', 'e', 'f'])

C = A.union(B)

print(A)

print(B)

print(C)

A = set( ['a', 'b', 'c', 'd'])

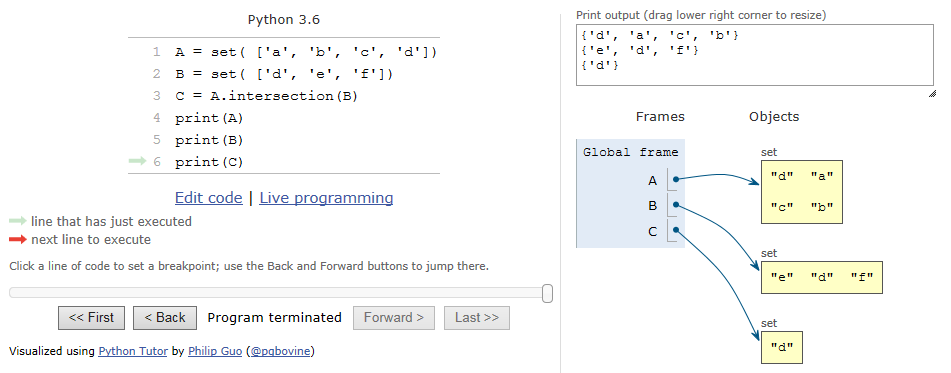
B = set( ['d', 'e', 'f'])

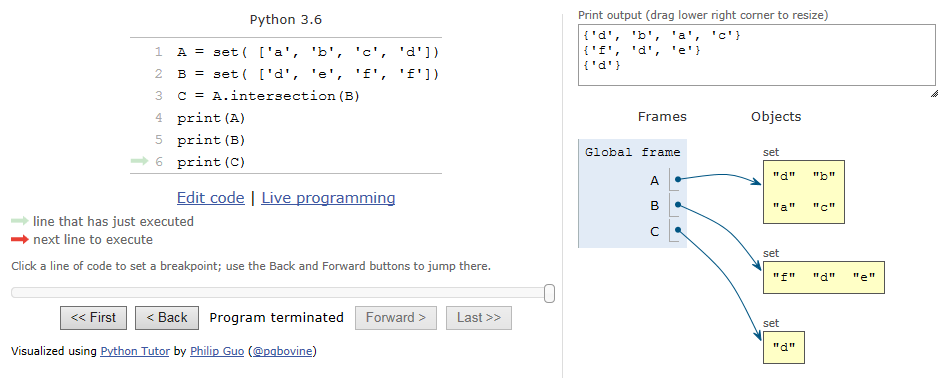
C = A.intersection(B)

print(A)

print(B)

print(C)





Duplicate elements in a set is not allowed.

**Next week:**

**Classes and Objects**