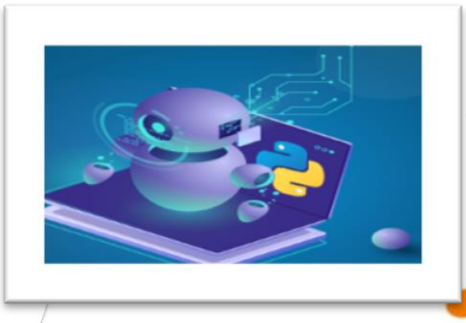
# Data science and Machine learning with Python



Designed by Abdur Rahman Joy - MCSD, MCPD, MCSE, MCTS, OCJP, Sr. Technical Trainer for VFX at IDB BISW (Scholarship program), and C#.net, R, Scala, Kotlin, JAVA, Android/IOS/Windows Mobile Apps, SQL server, Azure, Oracle, SharePoint Development, AWS , CEH, KALI Linux, Python, Data Science, Machine Learning ,Software Testing, Graphics, Multimedia and Game Developer at Joy Infosys and other premises like BITM, SkillsJob, PNTL, Leads Training and New Horizon inc , Cell #: +880-1712587348, email: jspaonline@gmail.com. Web URL: http://www.joyinfosys.com/me.

# Python collections:

## Introduction

Collections in Python are containers that are used to store collections of data, for example, **list, dict, set, tuple** etc. These are built-in collections. Several modules have been developed that provide additional data structures to store collections of data.

## Python Sets: Introduction

A Python set is an unordered collection of comma separated elements inside curly braces '{ }'. Python set itself is mutable allowing addition and removal of items but the elements are immutable which forbids any change in existing items.

Unlike tuples or lists, Python sets don't allow multiple occurrences of the same element. Hence, each element in Python set is unique.

There are two types of built-in set types:

- **set:** Python set type is mutable
- **frozen set:** Python frozen set type is immutable

## How to create Python sets?

Creating set in Python is as easy as placing unique comma separated items in curly braces '{ }' or it can also be created by using built-in function set( ).

We can't create empty sets by using only curly braces in Python sets though, we need to use built-in function set( ) to create an empty set.

**Ex:**

```
# set of integers
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

**Re:**

```
{1, 2, 3}
{1.0, 'Hello', (1, 2, 3)}
```

**Ex:**
```
# set do not have duplicates
# Output: {1, 2, 3, 4}
my_set = {1,2,3,4,3,2}
print(my_set)

# set cannot have mutable items
# here [3, 4] is a mutable list
# If you uncomment line #12,
# this will cause an error.
# TypeError: unhashable type: 'list'

#my_set = {1, 2, [3, 4]}

# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1,2,3,2])
print(my_set)
```

## Creating an empty set is a bit tricky.

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set() function without any argument.

**Ex:**
```
# initialize a with {}
a = {}

# check data type of a
# Output: <class 'dict'>
print(type(a))

# initialize a with set()
a = set()

# check data type of a
# Output: <class 'set'>
print(type(a))
```

## How to change a set in Python?

Sets are mutable. But since they are unordered, indexing have no meaning.

We cannot access or change an element of set using indexing or slicing. Set does not support it.

We can add single element using the add() method and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
# initialize my_set
my_set = {1,3}
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing

#my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2,3,4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4,5], {1,6,8})
print(my_set)
```

A particular item can be removed from set using methods, discard() and remove().
The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

**The following example will illustrate this.**

```
# initialize my_set
my_set = {1, 3, 4, 5, 6}
print(my_set)

# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)
```

```
# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
print(my_set)

# discard an element
# not present in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)

# remove an element
# not present in my_set
# If you uncomment line 27,
# you will get an error.
# Output: KeyError: 2

#my_set.remove(2)
```

Similarly, we can remove and return an item using the pop() method.

Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.

We can also remove all items from a set using clear().

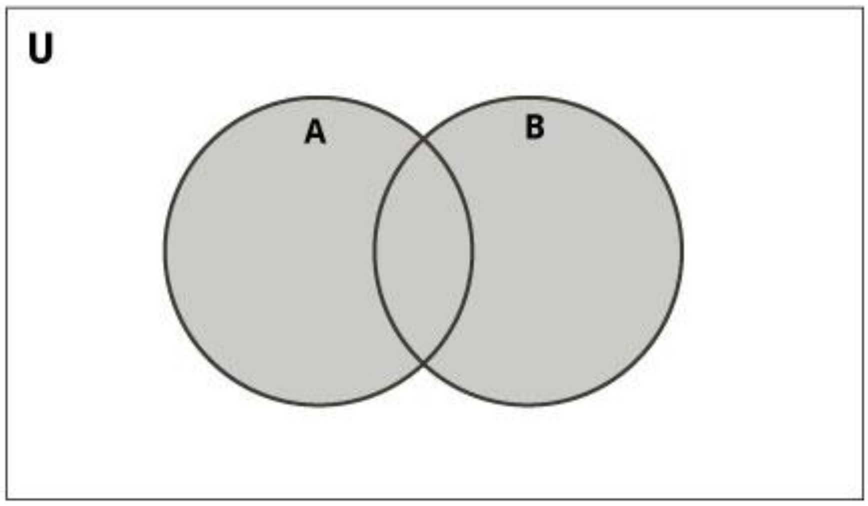
```
# initialize my_set
# Output: set of unique elements
my_set = set("HelloWorld")
print(my_set)

# pop an element
# Output: random element
print(my_set.pop())

# pop another element
# Output: random element
my_set.pop()
print(my_set)

# clear my_set
#Output: set()
my_set.clear()
print(my_set)
```

## Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

Let us consider the following two sets for the following operations.

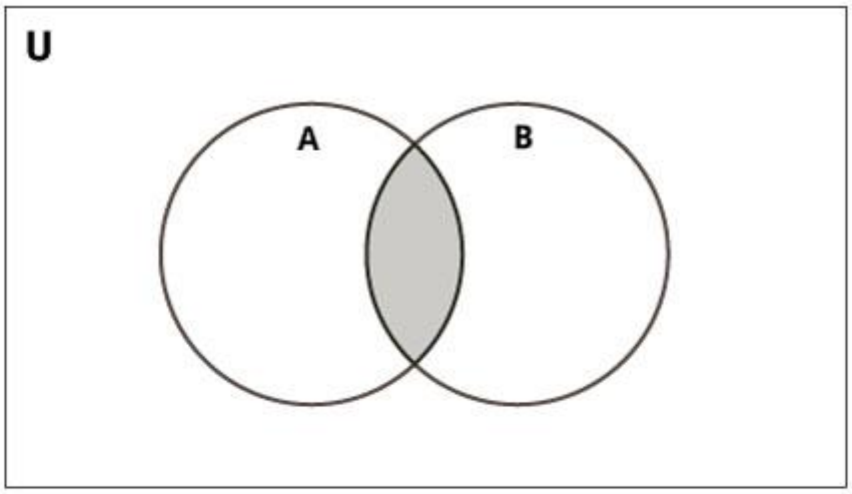
### Set Union



Union of A and B is a set of all elements from both sets.
Union is performed using | operator. Same can be accomplished using the method union().

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

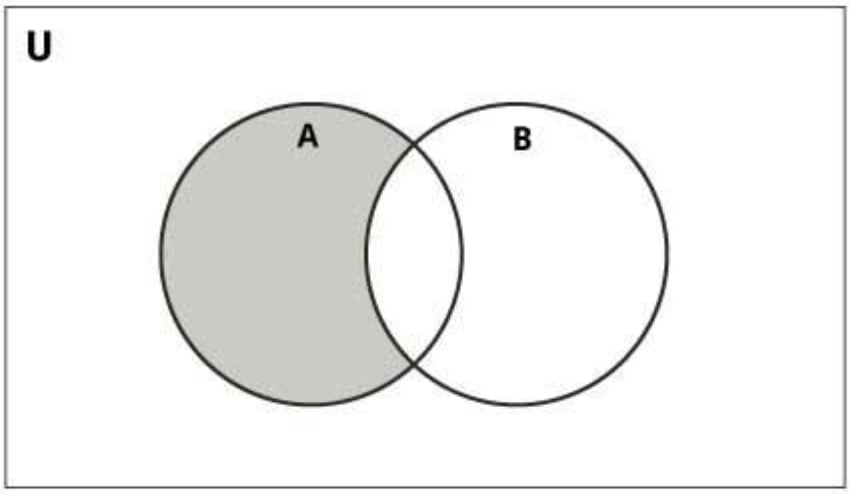
**Set Intersection**



Intersection of A and B is a set of elements that are common in both sets.
Intersection is performed using & operator. Same can be accomplished using the method intersection().

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use & operator
# Output: {4, 5}
print(A & B)
```

**Set Difference**



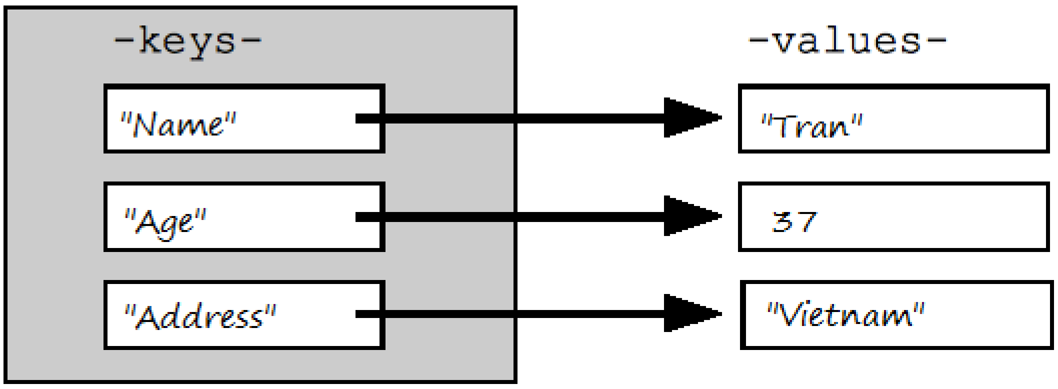
Difference of A and B (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of element in B but not in A.

Difference is performed using - operator. Same can be accomplished using the method difference().

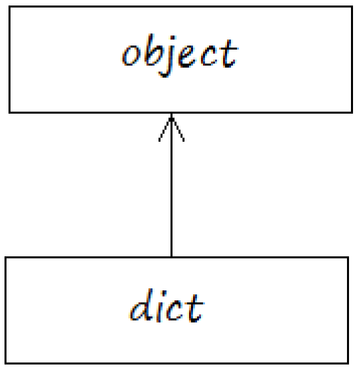
```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```

## Dictionary

In **Python**, dictionary is a data type and it is a list of elements, each of which is a pair of key & value. Dictinary quite similar to the **Map** concept in **Java**.



Dictionaries all are objects of the **dict** class.



In order to write a dictionary, you need to use { }, and write the elements into it, the elements are separated by comma. Each element is a pair of key and value separated by a colon ':'.

## # Dictionary

```
myinfo =  {"Name": "Tran", "Age": 37, "Address" : "Vietnam"  }


# Create a dictionary via constructor of dict class.
mydict = dict()

mydict["E01"] =  "John"
mydict["E02"] =  "King"

print ("mydict: ", mydict)
```

**Features of the value in the dictionary:**

- Each element of the dictionary is a pair of key and value, the value can be a certain type (string, number, user-defined types,...), and can be the same.

**Features of the key in the dictionary.**

- The key in dictionary is a immutable type. So, it can be string, number, Tuple,...
- Some types are not allowed (for example: **List**) because **List** is a mutable data type.
- Key in dictionary is not allowed to duplicate.

## Example:

## # Dictionary

```
myinfo =  {"Name": "Tran", "Age": 37, "Address" : "Vietnam"  }

print ("myinfo['Name'] = ", myinfo["Name"])

print ("myinfo['Age'] = ", myinfo["Age"])

print ("myinfo['Address'] = ", myinfo["Address"])
```

## Update Dictionary

Dictionary allows you to update the value of a certain key, it adds a new element if the key does not exist on the dictionary.

```
# Dictionary
myinfo =  {"Name": "Tran", "Age": 37, "Address" : "Vietnam"  }

# update value for key 'Address'
myinfo["Address"] = "HCM Vietnam"
```

```
# Add new element with key 'Phone'
myinfo["Phone"] = "12345"


print ("Element count: ", len(myinfo) )

print ("myinfo['Name'] = ", myinfo["Name"])

print ("myinfo['Age'] = ", myinfo["Age"])

print ("myinfo['Address'] = ", myinfo["Address"])

print ("myinfo['Phone'] = ", myinfo["Phone"])
```

## Delete dictionary

There are 2 ways to remove an element from a dictionary.

1. Use the **del** operator
2. Using the **"__delitem __ (key)"** method

```
# (Key,Value) = (Name, Phone)
contacts = {"John": "01217000111", \
          "Tom": "01234000111", \
          "Addison":"01217000222", "Jack":"01227000123"}

print ("Contacts: ", contacts)

print ("\n")
print ("Delete key = 'John' ")

# Delete element with key 'John'
del contacts["John"]

 print ("Contacts (After delete): ", contacts)

print ("\n")
print ("Delete key = 'Tom' ")

# Delete element with key 'Tom'
contacts.__delitem__( "Tom")

 print ("Contacts (After delete): ", contacts)
 print ("Clear all element")
```

```
# Clear all element
contacts.clear()

print ("Contacts (After clear): ", contacts)

# Delete dictionary 'contact' from memory
del contacts

# An error occured while accessing a variable that does not exist in memory
print ("Contacts (After delete): ", contacts)
```

## Functions for Dictionary

| Function | Description |
|---|---|
| len(dict) | Return elements count of dict. |
| str(dict) | Produces a printable string representation of a dictionary |
| type(variable) | Returns the type of the passed variable. If passed variable is dictionary, then it will return the object representing the 'dict' class. |
| dir(clazzOrObject) | Returns the members of the passed class (or object). If passed class is **dict** class, then it will return the members of **dict** class. |

```
contacts = {"John": "01217000111" ,"Addison": "01217000222","Jack": "01227000123"}

print ("contacts: ", contacts)

 print ("Element count: ", len(contacts) )

contactsAsString = str(contacts)

print ("str(contacts): ", contactsAsString )

# An object representing the 'dict' class.
aType = type(contacts)

print ("type(contacts): ", aType )

# dir(dict) function returns members of 'dict' class.
print ("dir(dict): ", dir(dict) )
```