Python Objects

Container or Collection Objects

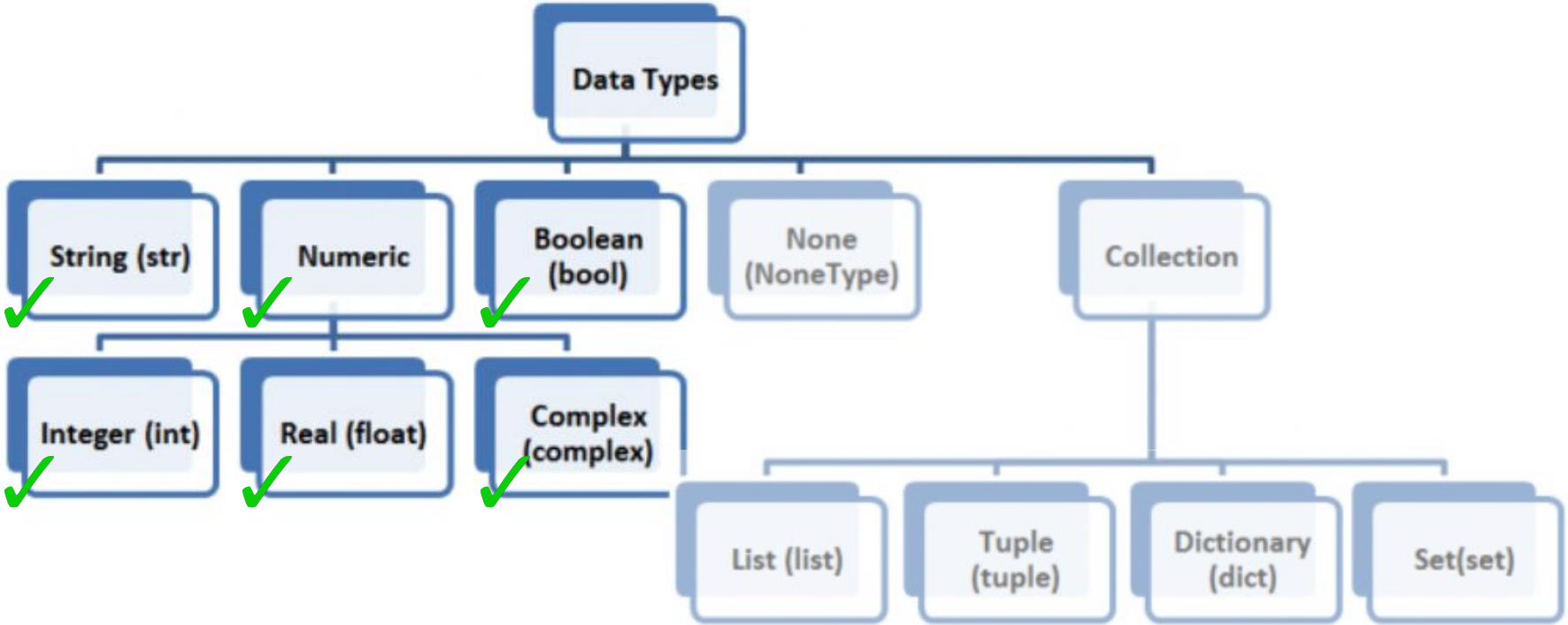
Container or collection objects

- Container or collection objects are objects that can hold any number of arbitrary objects

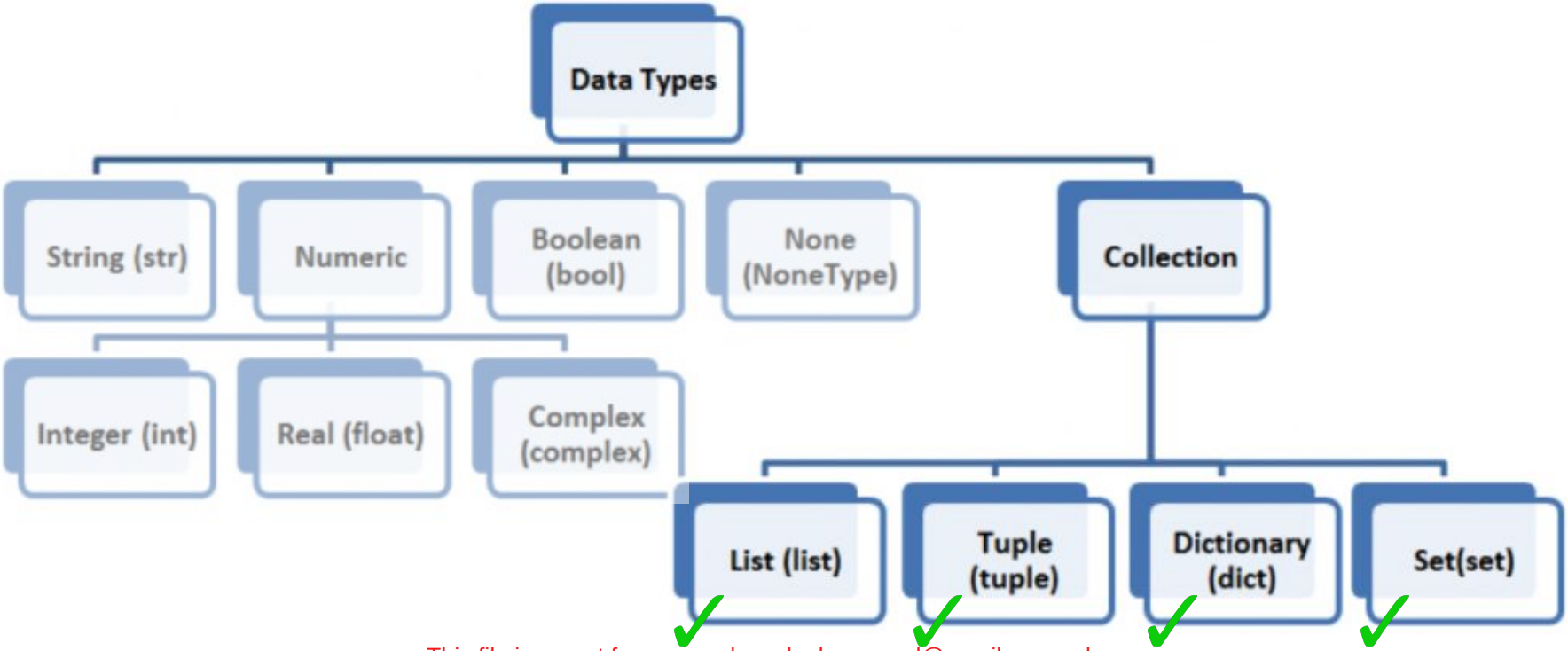
- Container provides a way to access the child objects and iterate over them

- Container or collection objects:
  - Dictionary
  - Tuple
  - List
  - Set

In our last session we covered...



We learn the following in our session today



List

List

- A list is a container object
- Can have duplicates unlike a Set
- Homogeneous
- Supports the following:
  - Append new elements
  - Concatenate
  - Access using indexes
  - min(), max()
  - In, not in

Creating lists

```
# Creating Lists
numbers = range(1,20)
print(numbers[0])
print(numbers[2])

characters = ["Python", "Scala", "Spark"]
print(characters[0])
print(characters[1])
```

```
1
3
Python
Scala
```

```
# Concatenating Lists
states_in_india = ["Tamil Nadu", "Karnataka", "Haryana"]
states_in_us = ["California", "Florida", "Texas", "Alabama"]
print(states_in_india + states_in_us)
```

```
['Tamil Nadu', 'Karnataka', 'Haryana', 'California', 'Florida', 'Texas', 'Alabama']
```

```
# Append
states_in_india.append("Kerala")
print(states_in_india)
```

```
['Tamil Nadu', 'Karnataka', 'Haryana', 'Kerala']
```

Sorting a list using the built-in sort() function

```
# Sorting a list
a = [10, 12, 11, 16, 71, 12, 9, 56]
b = [11, 16, 2, 34, 21, 11, 8, 18]
c = ["Zebra", "Apple", "Anchor", "Assets", "Baseball", "Basket"]
```

```
# Using built-in sort function
print(a)
a.sort()
print(a)
```

```
[10, 12, 11, 16, 71, 12, 9, 56]
[9, 10, 11, 12, 12, 16, 56, 71]
```

```
# Sorting text in a list
print(c)
c.sort()
print(c)
```

```
['Zebra', 'Apple', 'Anchor', 'Assets', 'Baseball', 'Basket']
['Anchor', 'Apple', 'Assets', 'Baseball', 'Basket', 'Zebra']
```

Sort the list in reverse order by passing the reverse=True parameter to the sort() function

```
# Sorting a list
a = [10, 12, 11, 16, 71, 12, 9, 56]
b = [11, 16, 2, 34, 21, 11, 8, 18]
c = ["Zebra", "Apple", "Anchor", "Assets", "Baseball", "Basket"]
```

```
# Using built-in sort function
print(a)
a.sort()
print(a)
```

```
[10, 12, 11, 16, 71, 12, 9, 56]
[9, 10, 11, 12, 12, 16, 56, 71]
```

Sorting a list using the sorted() function

```
# using sorted() function
print(b)
bs = sorted(b)
print(bs)
```

```
[11, 16, 2, 34, 21, 11, 8, 18]
[2, 8, 11, 11, 16, 18, 21, 34]
```

Sort the list in reverse order by passing the reverse=True parameter to the sorted() function

Hint: sorted(b, reverse=True)

```
# using sorted() function
print(b)
bs = sorted(b)
print(bs)
```

```
[11, 16, 2, 34, 21, 11, 8, 18]
[2, 8, 11, 11, 16, 18, 21, 34]
```

More list operations

```
# given 2 lists which have only numbers
List1 = [1,2,3,4]
List2 = [5,6,7,8]
Concatenation = List1 + List2
Concatenation
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
# given 2 list which have numbers and string
List1 = [1,2,3,4]
List2 = ["a","b"]
Concatenation = List1 + List2
Concatenation
```

```
[1, 2, 3, 4, 'a', 'b']
```

```
# list with mixed data type and add 2 list
List1 = ["a",2,"b",4]
List2 = ["a","b",1,2]
Concatenation = List1 + List2
Concatenation
```

```
['a', 2, 'b', 4, 'a', 'b', 1, 2]
```

More list operations

```
# Check existence of element in a list
List = [1,2,3,4,5]
print(2 in List)
```

```
True
```

```
# Find the number of elements in a list
List = [1,2,3,4,5]
len(List)
```

```
5
```

```
# Iterating through a list
List = [1,2,3,4,5]
for i in List:
    print(List)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

More list operations

```
# Max value in a list
List = [1,2,3,4,4,5,67,8,565,4,65,76,7654,654,6543,7654]
max(List)
```

```
7654
```

```
# Min value in a list
List = [1,2,3,4,4,5,67,8,565,4,65,76,7654,654,6543,7654]
min(List)
```

```
1
```

```
# Type cast a string to a list
string = 'qwerty'
list(string)
```

```
['q', 'w', 'e', 'r', 't', 'y']
```

More list operations

```
# Append element to a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
List.append('pig')
List
```

```
# Clear a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
List.clear()
List
```

```
# Copy a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
new_list = List.copy()
List.append('pig')
print('Old List: ', List)
print('New List: ', new_list)
```

```
Old List:  ['Penguin', 'cat', 'Hippopotam', 'dog', 'rabbit', 'pig']
New List:  ['Penguin', 'cat', 'Hippopotam', 'dog', 'rabbit']
```

More list operations

```
# Count number of specific elements
List = ['a', 'e', 'i', 'o', 'i', 'u']
List.count('i')
```

```
2
```

```
# Extend a list
List1 = [1,2,3,4]
List2 = [5,6,7,8]
List1.extend(List2)
print('Language List: ', List1)
```

```
Language List:  [1, 2, 3, 4, 5, 6, 7, 8]
```

```
# Find the index position of an element
List = ['a', 'e', 'i', 'o', 'i', 'u']
List.index('i')
```

```
2
```

```
# Insert element at specific index position
List = ['a', 'e', 'i', 'u']
List.insert(3, 'o')

print('Updated List: ', List)
```

```
Updated List:  ['a', 'e', 'i', 'o', 'u']
```

More list operations

```
# Remove an element from a list by index position
List = ['a', 'e', 'i', 'u']
return_value = List.pop(3)
print('Return Value: ', return_value)
print('Updated List: ', List)
```

```
Return Value:  u
Updated List:  ['a', 'e', 'i']
```

```
# Remove an element from a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
List.remove('rabbit')
print('Updated list: ', List)
```

```
Updated list:  ['Penguin', 'cat', 'Hippopotam', 'dog']
```

```
# Reverse a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
List.reverse()
List
```

```
['rabbit', 'dog', 'Hippopotam', 'cat', 'Penguin']
```

We learnt the list object...

Create using [ ]

Editable

Tuple

Tuples

- Type of container object
- Known as sequence types
- Can have heterogeneous sequence of elements
- Immutable
- Elements are separated by comma, enclosed in ( ) parenthesis
- Supported operations:
  - In, Not in
  - Min(), Max()
  - Compare
  - Concatenate, Slice, Index

Creating tuples

```
# Create a tuple
my_tuple = (100, 250, "Robert")

# Access the elements in a tuple
print(my_tuple[0])
print(my_tuple[-1])
```

```
100
Robert
```

```
# YOU CANNOT CHANGE AN ELEMENT
my_tuple[0]=450
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-3a8a9921f822> in <module>()
      1 # YOU CANNOT CHANGE AN ELEMENT
----> 2 my_tuple[0]=450

TypeError: 'tuple' object does not support item assignment
```

Concatenate & slice tuple

```
# Concatenate two tuples
my_tuple = (100, 250, 650)
your_tuple = ("Daniel", 450, 200, 750, "Siva")

print(my_tuple+your_tuple)
```

```
(100, 250, 650, 'Daniel', 450, 200, 750, 'Siva')
```

```
#Slice a tuple, min(), max()
print(your_tuple[0])
print(your_tuple[2:3])
print(your_tuple[1:5])
print(min(my_tuple) + max(my_tuple))
```

```
Daniel
(200,)
(450, 200, 750, 'Siva')
750
```

Tuple operations

```
# Tuple Repetition
Tuple = (1,2,3,4)
Repetition = Tuple *2
Repetition
```

```
(1, 2, 3, 4, 1, 2, 3, 4)
```

```
# Concatenate a tuple
Tuple1 = (1,2,3,4)
Tuple2 = (5,6,7,8)
Concatenation = Tuple1 + Tuple2
Concatenation
```

```
(1, 2, 3, 4, 5, 6, 7, 8)
```

```
# Checking existence of an element in a tuple
Tuple = (1,2,3,4,5)
print(2 in Tuple)
```

```
True
```

Tuple operations

```
# Length of a tuple
Tuple = (1,2,3,4,5)
len(Tuple)
```

```
5
```

```
# Iterating a tuple
Tuple = (1,2,3,4,5)
for i in Tuple:
    print(i)
```

```
1
2
3
4
5
```

Tuple operations

```
# Finding maximum
Tuple = [1,2,3,4,4,5,67,8,565,4,65,76,7654,654,6543,7654]
max(Tuple)
```

```
7654
```

```
# Finding minimum
Tuple = [1,2,3,4,4,5,67,8,565,4,65,76,7654,654,6543,7654]
min(Tuple)
```

```
1
```

Tuple operations

```
# type casting - convert a string into a tuple
Tuple = 'qwerty'
tuple(Tuple)
```

```
('q', 'w', 'e', 'r', 't', 'y')
```

```
## Sorted - output will be sorted but the original data will not be changed
age = (40,8,35,5)
print(sorted(age))

# default is ascending order
print(sorted(age , reverse = True))
print(age)
```

```
[5, 8, 35, 40]
[40, 35, 8, 5]
(40, 8, 35, 5)
```

Tuple operations

```
## Sorted - output will be sorted but the original data will not be changed
age = (40,8,35,5)
print(sorted(age))

# default is ascending order
print(sorted(age , reverse = True))
print(age)
```

```
[5, 8, 35, 40]
[40, 35, 8, 5]
(40, 8, 35, 5)
```

```
# Slicing a tuple
age = (40,8,35,5)
print(age[1])
print(age[0:])
print(age[:len(age)])
print(age[:-1])
print(age[-0:-1])
```

```
8
(40, 8, 35, 5)
(40, 8, 35, 5)
(40, 8, 35)
(40, 8, 35)
```

Tuple operations

```
# Reversing a tuple
fruits = ("orange","apple","cherry","apple","grapes")

print("reverse string :",fruits[::-1])
print("list from 2nd element til last :",fruits[2:])
print("print last 3 element :",fruits[-3:])
```

```
reverse string : ('grapes', 'apple', 'cherry', 'apple', 'orange')
list from 2nd element til last : ('cherry', 'apple', 'grapes')
print last 3 element : ('cherry', 'apple', 'grapes')
```

```
# Add tuple
age = (40,8,35,5)
name = ('jack','jerry')

ad_tup = age + name
ad_tup
```

```
(40, 8, 35, 5, 'jack', 'jerry')
```

Tuple operations

```
## sub tuple tuples together, sub will not work
ad_tup = age - name
ad_tup
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-582e9fba70da> in <module>
      1 ## sub tuple tuples together, sub will not work
----> 2 ad_tup = age - name
      3 ad_tup

TypeError: unsupported operand type(s) for -: 'tuple' and 'tuple'
```

```
del age[1] # tuple cannot delete an object in it
del age # tuple can be deleted completly
age # its not available after deleting
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-05932a2264c9> in <module>
----> 1 del age[1] # tuple cannot delete an object in it
      2 del age # tuple can be deleted completly
      3 age # its not available after deleting

TypeError: 'tuple' object doesn't support item deletion
```

did you know?

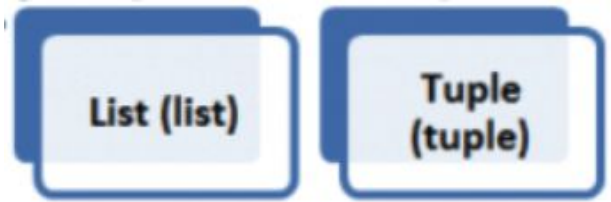
Due to mutability, you need more memory for lists and less memory for tuples.

WANT TO KNOW MORE?

To reduce memory fragmentation and speed up allocations, Python reuses old tuples. If a tuple no longer needed and has less than 20 items, instead of deleting it permanently Python moves it to a free list.

A free list is divided into 20 groups, where each group represents a list of tuples of length n between 0 and 20. Each group can store up to 2 000 tuples. The first (zero) group contains only 1 element and represents an empty tuple

We learnt list & tuple...



Create using	[ ]	( )
Editable	✎	⊘

Dictionary

Dictionary object

- A Python dictionary is a mapping of unique keys to values
- Use {} curly brackets to construct the dictionary
- [] square brackets to index it
- Dictionaries are mutable
- We will learn how to:
  - Access values in a dictionary
  - Update dictionary
  - Delete dictionary elements

A sample dictionary object

Key 1 Value 1

```
dictionary_list = { 'name'    :  'Ariel' ,
                    'hobbies':  ['painting' , 'singing' , 'cooking'] }
```

Key 2 Value 2

Dictionary object

```
# Create a NEW Dictionary object
books = {
    "R": 480,
    "Python": 650,
    "PySpark": 450,
    "Scala": 780,
    "Basic Stats": 650
}
```

```
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Scala': 780, 'Basic Stats': 650}
```

```
# Add elements to the books dictionary
books['Hadoop']=850
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Scala': 780, 'Basic Stats': 650, 'Hadoop': 850}
```

```
# Remove an element from the dictionary
del books['Scala']
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Basic Stats': 650, 'Hadoop': 850}
```

```
# Check the length of the dictionary object
len(books)
```

```
5
```

Dictionary object

```
# Using Dictionary Objects

# Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()
```

```
['A', 'Quick', 'Brown', 'Fox', 'Jumps', 'over', 'the', 'Lazy', 'Dog']
```

```
# Initializa a dictionary object
# Use {} curly brackets to construct a dictionary object
my_dictionary = {}
```

```
# We will perform a word count using the dictionary object
for word in my_text.split() :
    if word not in my_dictionary :
        my_dictionary[word]=1
    else:
        my_dictionary[word]+=1

# The above code buils a frequency table for every word
```

```
# Print the output
# Output ia  key-value pair
print(my_dictionary)
```

```
{'A': 1, 'Quick': 1, 'Brown': 1, 'Fox': 1, 'Jumps': 1, 'over': 1, 'the': 1, 'Lazy': 1, 'Dog': 1}
```

Dictionary object: Default dictionary

```
# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = {}

# We will perform a word count using the dictionary object
for word in my_text.split() :
#    if word not in my_dictionary :
#        my_dictionary[word]=1
#    else:
        my_dictionary[word]+=1
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-19-47206380e9a7> in <module>()
     12 #        my_dictionary[word]=1
     13 #    else:
---> 14         my_dictionary[word]+=1
     15

KeyError: 'A'
```

Dictionary object: Default dictionary

- Import the defaultdict from collections module
- Initialize the dictionary object with defaultdict()
- Note: We passed int() to the defaultdict()
- When the dictionary object encounters a key that was not seen before, it initializes the key with a value returned by int(), in this case 0 (zero)

Dictionary object: Default dictionary

- Import the defaultdict from collections module
- Initialize the dictionary object with defaultdict()
- Note: We passed int() to the defaultdict()
- When the dictionary object encounters a key that was not seen before, it initializes the key with a value returned by int(), in this case 0

```
# Import for the defaultdict from collections module
from collections import defaultdict

# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = defaultdict(int)

# We will perform a word count using the dictionary object
for word in my_text.split() :
#    if word not in my_dictionary :
#        my_dictionary[word]=1
#    else:
        my_dictionary[word]+=1

print(my_dictionary)
```

```
defaultdict(<class 'int'>, {'A': 1, 'Quick': 1, 'Brown': 1, 'Fox': 1, 'Jumps': 1, 'over': 1, 'the': 1, 'Lazy': 1, 'Do
g': 1})
```

Loop through the dictionary object

- Use keys() to loop through the keys in the dictionary object
- Use values() to loop through the values in the dictionary object

```
# Import for the defaultdict from collections module
from collections import defaultdict

# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = defaultdict(int)

# We will perform a word count using the dictionary object
for word in my_text.split() :
    my_dictionary[word]+=1

# Using key(), value() functions of dictionary obejct
for key, value in my_dictionary.items():
    print(key, value)
```

Built-in dictionary functions

```
# Checking length of dictionary object
dict = {'Name': 'vema', 'Age': 27};
len(dict)
```

```
2
```

```
# Converts the dictionary into the printable string representation
dict = {'Name': 'vema', 'Age': 27};
str(dict)
```

```
"{'Name': 'vema', 'Age': 27}"
```

```
# Checking the type of object
dict = {'Name': 'vema', 'Age': 27};
type(dict)
```

```
dict
```

Built-in dictionary functions

```
employee = {"Name": "Johny", "Age": 32}
employee.clear()
employee
```

```
{}
```

```
employee = {"Name": "Johny", "Age": 32}
new_employee = employee.copy()
new_employee
```

```
{'Name': 'Johny', 'Age': 32}
```

```
# seq is a tuple
seq = ('name', 'age', 'sex')
# tuple is type casted into dict and all the elements are converted into keys. So the key will not have any values.
dict = dict.fromkeys(seq)

print ("New Dictionary : ",  str(dict))

# assign a default value as 10 for all the keys
dict = dict.fromkeys(seq, 10)
print ("New Dictionary : ",  str(dict))
```

```
New Dictionary :  {'name': None, 'age': None, 'sex': None}
New Dictionary :  {'name': 10, 'age': 10, 'sex': 10}
```

Built-in dictionary functions

```
employee = {'Name': 'vema', 'Age': 27};
print('Name: ', employee.get('Name'))
print('Age: ', employee.get('Age'))

# salary key is not vailable and its value by default will be None
print('Salary: ', employee.get('salary'))
print('Salary: ', employee.get('salary',"Not Found ! "))
```

```
Name:  vema
Age:  27
Salary:  None
Salary:  Not Found !
```

```
# get all the keys and its respective values
sales = { 'MacBook Pro': 10, 'iPhone': 132, 'iPad': 78 }
print(sales.items())
```

```
dict_items([('MacBook Pro', 10), ('iPhone', 132), ('iPad', 78)])
```

```
# get only the keys
employee = {'Name': 'vema', 'Age': 27};
print(employee.keys())

empty_dict = {}
print(empty_dict.keys())
```

Built-in dictionary functions

```
employee = {'name': 'Phill'}
print(employee.keys())
print(employee.values())
print(employee.items())
```

```
dict_keys(['name'])
dict_values(['Phill'])
dict_items([('name', 'Phill')])
```

```
# key is not in the dictionary
salary = employee.setdefault('salary' )
print('person = ',employee)
print('salary = ',salary)
```

```
person =  {'name': 'Phill', 'salary': None}
salary =  None
```

```
employee = {'name': 'Phill'}
# key is not in the dictionary, add salarya nd give default value as 5000
salary = employee.setdefault('salary' , 5000)
print('person = ', employee)
print('salary = ', salary)
```

```
person =  {'name': 'Phill', 'salary': 5000}
salary =  5000
```

Built-in dictionary functions

```
# dict1, value of key 2 is three which has to be updated as two from dict2
Dict1 = {1: "one", 2: "three",4:"five"}
print("before update :",Dict1)
Dict2 = {2: "two",4: "four"}


#value of key 2 and 4 from Dict2 is updated with dict1

Dict1.update(Dict2)
print("After update :",Dict1)
```

```
before update : {1: 'one', 2: 'three', 4: 'five'}
After update : {1: 'one', 2: 'two', 4: 'four'}
```

```
# type cast a dict into list
Dict1
list(Dict1)
```

```
[1, 2, 4]
```

Built-in dictionary functions

```
Dict = { 'apple': 2, 'orange': 3, 'grapes': 4 }
print(Dict.values())
```

```
dict_values([2, 3, 4])
```

```
# pass the key and get the value
Dict["apple"]
```

```
2
```

```
#update a dict
Dict["apple"] = 8
Dict
```

```
{'apple': 8, 'orange': 3, 'grapes': 4}
```

```
## delete an object
del Dict["grapes"]
Dict
```

```
{'apple': 8, 'orange': 3}
```

Built-in dictionary functions

```
## each key should be unique
std  = {"Eddy":26,"Eddy":23,"Eddy":28}
print(std) # holds only the last value when the keys are same
```

```
{'Eddy': 28}
```

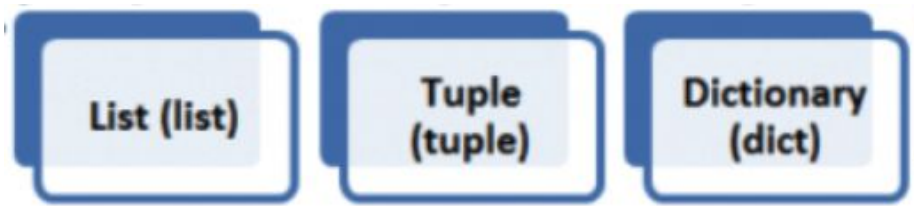
did you know?

Tuples are hashable and lists are not. It means that you can use a tuple as a key in a dictionary. The list can't be used as a key in a dictionary, whereas a tuple can be used

did you know?

If the tuple contains a list or a dictionary inside it, those can be changed even if the tuple itself is immutable.

We learnt list, tuple and dictionary...



Create using	[ ]	( )	{ }
Editable	✎	⊘	✎

Set

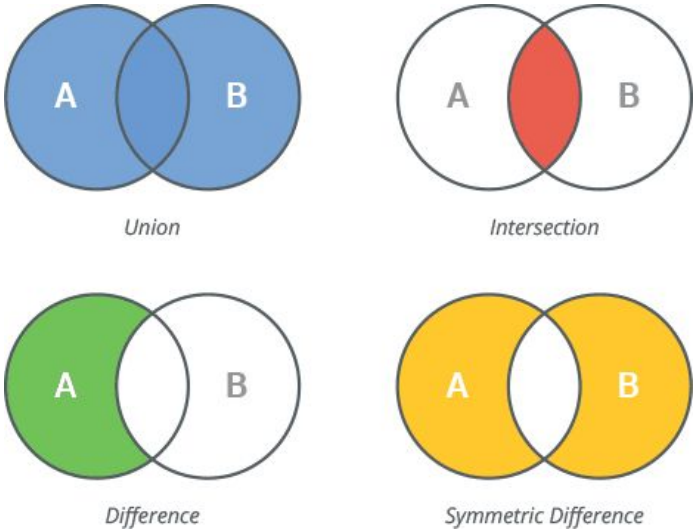
Sets

- Sets are very similar to list data structures
- Do not allow duplicates
- Unordered collections of homogeneous elements
- Sets are generally used to remove duplicate elements from a list

Sets

Sets supports the following operations:

- Intersection
- Union
- Difference
- Symmetric Difference



Sets

```
# Working with sets
# Initialize two sentences.
sentence_1 = "There is nothing new in the world except history you do not know"
sentence_2 = "With the new day comes new strength and new thoughts"
```

```
# Create set of words from strings
sentence_1_words = set(sentence_1.split())
sentence_2_words = set(sentence_2.split())
```

```
# Find out the number of unique words in each set, vocabulary size.
no_words_in_sentence_1 = len(sentence_1_words)
no_words_in_sentence_2 = len(sentence_2_words)
```

```
# Find out the list of common words between the two sets & their count
common_words = sentence_1_words.intersection(sentence_2_words)
number_of_common_words = len(sentence_1_words.intersection(sentence_2_words))
```

```
# Find a list of unique words between the two sets and their count
unique_words = sentence_1_words.union(sentence_2_words)
number_of_unqiue_words = len(sentence_1_words.union(sentence_2_words))
```

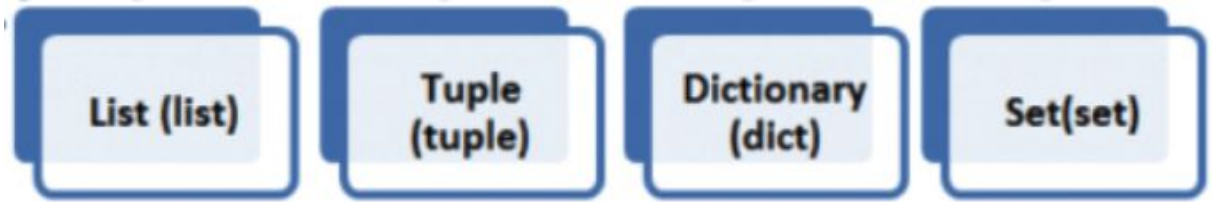
Sets

```
# Find a list of unique words between the two sets and their count
unique_words = sentence_1_words.union(sentence_2_words)
number_of_unqiue_words = len(sentence_1_words.union(sentence_2_words))
```

```
print("Words in sentence 1 = ", sentence_1_words)
print("No of words in sentence 1 = %d"% no_words_in_sentence_1)
print("Words in sentence 2 = ", sentence_2_words)
print("No of words in sentence 2 = %d"% no_words_in_sentence_2)
print("No of words in common = %d"% number_of_common_words)
print("Common words are ", common_words)
print("number of unique words are = %d"%  number_of_unqiue_words)
print("Unique words are ", unique_words)
```

```
Words in sentence 1 =  {'is', 'do', 'not', 'in', 'know', 'nothing', 'the', 'There', 'world', 'you', 'history', 'excep
t', 'new'}
No of words in sentence 1 = 13
Words in sentence 2 =  {'thoughts', 'comes', 'and', 'the', 'With', 'day', 'new', 'strength'}
No of words in sentence 2 = 8
No of words in common = 2
Common words are  {'the', 'new'}
number of unique words are = 19
Unique words are  {'is', 'do', 'not', 'in', 'know', 'comes', 'nothing', 'the', 'There', 'world', 'With', 'history',
'day', 'except', 'thoughts', 'and', 'you', 'new', 'strength'}
```

We learnt the collection objects...



Create using	[ ]	( )	{ }	{ }
Editable	✎	⊘	✎	✎

Conditional Statements

Let's explore conditional statements

- Conditional statements are handled by 'if' statements in python
- Conditional statement perform computations or actions depending on the boolean constraint is evaluated as true or false
- If the constraint is 'True', the statements in the body are executed, and if it is 'False', the statements in body are skipped
- Conditional statements:
  - If statement
  - If-else statement
  - If elif else statement
  - Nested if-else

If Statement

If statement

- The syntax of the if-statement is

```
if condition:
   statement 1
   statement 2
      ....
```

- Python logical conditions supported are
  - Equivalence `==`
  - Inequivalence `!=`
  - Less than `>`
  - Greater than `<`
  - Less than or equal to `<=`
  - Greater than or equal to `>=`

- If statement can be nested

If statement

```
# A demo of if statement
if (4 ** 2 >= 16):
    print("It's true!")
```

```
It's true!
```

```
if (4 * 2 < 100):
    print("This won't run")
```

```
This won't run
```

```
# Check if the given value is between 0 to 100
value = 56
if((value>0) and (value<100)):
     print("Given number is between 0 and 100")
```

```
Given number is between 0 and 100
```

The if...else Statement

If-else statement

- The syntax of the if-statement is

```
if condition:
   statement(s)
    else:
   statement(s)
```

- The statement(s) under ‘else:’ are executed when the condition in ‘if ’ is not satisfies, ie., the boolean output is ‘False’

If-else statement

```
days = int(input("How many days are in June?: "))
if days == 30:
    print("You passed the test.")
else:
    print("You failed the test.")
print("Thank You!")
```

```
How many days are in June?: 31
You failed the test.
Thank You!
```

If-else statement

```
# calculate the circumference of the circle using if else
radius = int(input("Enter radius: "))

if radius >= 0:
    print("Circumference = ", 2 * 3.14 * radius)
else:
    print("Please enter a positive number")
```

```
Enter radius: 4
Circumference =  25.12
```

If elif else Statement

If elif else statement

- Elif statement is used to control multiple conditions only if the given if condition false
- It's similar to an if-else statement and the only exception is that in else we are not checking the condition but in elif, we check the condition
- The syntax of the if elif else is

```
if (condition):
elif (condition):
else:
```

If elif else statement

```
# check the ticket price according to your age
my_age = input('Enter your age:')
my_age = int(my_age)
if 0<my_age<3:
    print('Free ticket')
elif 3<my_age<10:
    print('Ticket price is 200')
elif 10<my_age<20:
    print('Ticket price is 250')
else:
    print('Ticket price is 300')
```

```
Enter your age:17
Ticket price is 250
```

If elif else statement

```
day = input(str('Enter the day: '))
if (day == 'Monday'):
    print('Today is Monday')
elif (day == 'Tuesday'):
    print('Today is Tuesday')
elif (day == 'Wednesday'):
    print('Today is Wednesday')
else:
    print('Today is Holiday')
```

```
Enter the day: Monday
Today is Monday
```

Nested if and if...else Statement

Nested If-else statement

- The syntax of the if-statement is

```
if condition:
  statement(s)
  if condition:
    statement(s)
  else:
    statement(s)
else:
  statement(s)
```

- Note that a ‘if’ statements are within the body of another if statement
- The statements in this manner are called ‘nested if- statements’
- If the first ‘if’ condition is satisfied then the program executes the commands within that ‘if’

Nested If-else statement

```
# find the largest number among three numbers using nested if else
x = 234
y = 453
z = 223
if x > y:
    print('x is greater')
else:
    if z > y:
        print('z is greater')
    else:
        print('y is greater')
```

```
y is greater
```

Nested If-else statement

```
winning_number = 27
Num = input('Enter your score:')
Num = int(Num)
if Num == winning_number:
    print('YOU WIN !!')
else:
    if Num < winning_number:
        print('Your score is too low')
    else:
        print('Your score is too high')
```

```
Enter your score:45
Your score is too high
```

Nested If statement

```
userid = input(str('Enter your userid: '))
password = input('Enter your password: ')
password = int(password)
if userid == 'simplilearn@gmail.com':
              if password == 1234:
                            print('You are successfully logged in')
```

```
Enter your userid: simplilearn@gmail.com
Enter your password: 1234
You are successfully logged in
```

Thank You