LẬP TRÌNH PYTHON CƠ BẢN

(Basic Python Programming)

Th.S Nguyễn Hoàng Thành

Email: thanhnh@ptithcm.edu.vn

Tel: 0909 682 711

12. Python - Tuples

- The tuple object (pronounced “toople” or “tuhple”) is roughly like a list that cannot be changed - tuples are ***sequences***, like lists, but they are ***immutable***, like strings.
- Functionally, they’re used to represent fixed collections of items:
- the components of a specific calendar date, for instance.
- Syntactically, they are normally coded in **parentheses**
- They support arbitrary types, arbitrary nesting, and the usual sequence operations.

Creating a Tuple

- A tuple is created by placing all the items (elements) inside **parentheses** (), separated by **commas**. The parentheses are **optional**, however, it is a good practice to use them.
- A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

Exam 1:

```
>>> my_tuple = ()                                    #Empty tuple
>>> print(my_tuple)
()
>>> my_tuple = (1, 2, 3)                             #Tuple having integers
>>> print(my_tuple)
(1, 2, 3)
>>> my_tuple = (1, "Hello", 3.4)                     #Tuple with mixed datatypes
>>> print(my_tuple)
(1, 'Hello', 3.4)
>>> my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))       #Nested tuple
>>> print(my_tuple)
('mouse', [8, 4, 6], (1, 2, 3))
```

Creating a Tuple

A tuple can also be created without using parentheses. This is known as tuple packing.

```
>>> my_tuple = 3, 4.6, "dog"
>>> print(my_tuple)
(3, 4.6, 'dog')
>>> a, b, c = my_tuple
>>> print(a)
3
>>> print(b)
4.6
>>> print(c)
dog
```

Creating a tuple with one element is a bit tricky.

Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is, in fact, a tuple.

```
>>> my_tuple = ("hello")
>>> print(type(my_tuple))
<class 'str'>
>>> my_tuple = ("hello",)                          # Creating a tuple having one element
>>> print(type(my_tuple))
<class 'tuple'>
>>> my_tuple = "hello",                            # Parentheses is optional
>>> print(type(my_tuple))
<class 'tuple'>
```

Access Tuple Elements

- There are various ways in which we can access the elements of a tuple.
  - **Indexing**
  - **Negative Indexing**
  - **Slicing**

Access Tuple Elements - Indexing

- We can use the index operator [] to access an item in a tuple, where the index starts from 0.
- So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an index outside of the tuple index range (6,7,... in this example) will raise an IndexError.
- The index must be an integer, so we cannot use float or other types. This will result in TypeError.

Access Tuple Elements - Indexing

```
>>> my_tuple = ('p','t','i','t','h','c', 'm')        # Accessing tuple elements using indexing
>>> print(my_tuple[0])
p
>>> print(my_tuple[5])
c
>>> n_tuple = ("ptithcm", [2, 7, 1], (1, 8, 3))       # nested tuple
>>> print(n_tuple[0][3])                              # nested index
t
>>> print(n_tuple[1][1])                              # nested index
7
```

Access Tuple Elements - Negative Indexing

- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.

```
>>> my_tuple = ('p','t','i','t','h','c', 'm')        # Accessing tuple elements using indexing
>>> print(my_tuple[-1])
m
>>> print(my_tuple[-6])
t
```

Access Tuple Elements - Slicing

- We can access a range of items in a tuple by using the slicing operator colon ”:”

```
>>> my_tuple = ('p','t','i','t','-','h','c','m')        # Accessing tuple elements using slicing
>>> print(my_tuple[1:4])
('t', 'i', 't')
>>> print(my_tuple[:-6])
('p', 't')
>>> print(my_tuple[6:])
('c', 'm')
>>> print(my_tuple[:])                                  # elements beginning to end
('p', 't', 'i', 't', '-', 'h', 'c', 'm')
```

Changing a Tuple

- Unlike lists, tuples are **immutable**.
- This means that elements of a tuple cannot be changed once they have been assigned. But, if the element is itself a mutable data type like a list, its nested items can be changed.
- We can also assign a tuple to different values (reassignment).

Changing a Tuple

```
>>> my_tuple = (4, 2, 3, [6, 5])                     # Changing tuple values

>>> my_tuple[3][0] = 9

>>> print(my_tuple)

(4, 2, 3, [9, 5])

>>> my_tuple = ('p', 't', 'i', 't', '-', 'h', 'c', 'm')     # Tuples can be reassigned

>>> print(my_tuple)

('p', 't', 'i', 't', '-', 'h', 'c', 'm')
```

Changing a Tuple

- We can use + operator to combine two tuples. This is called **concatenation**.
- We can also **repeat** the elements in a tuple for a given number of times using the * operator.
- Both + and * operations result in a new tuple.

```
>>> print((1, 2, 3) + (4, 5, 6))                     # Concatenation
(1, 2, 3, 4, 5, 6)
>>> print(("Repeat",) * 3)                           # Repeat
('Repeat', 'Repeat', 'Repeat')
```

Deleting a Tuple

- We cannot change the elements in a tuple. It means that we **cannot delete** or **remove** items from a tuple.
- Deleting a tuple entirely, however, is possible using the keyword **del**.

Tuple Methods

- The **count()** method of Tuple returns the number of times the given element appears in the tuple.

  tuple.count(element)

- The **Index()** method returns the first occurrence of the given element from the tuple.

  Syntax:

  tuple.index(element, start, end)

  Parameters:

  - **element:** The element to be searched.
  - **start (Optional):** The starting index from where the searching is started
  - **end (Optional):** The ending index till where the searching is done

Why Tuples?

- Why have a type that is like a list, but supports fewer operations?
- Frankly, tuples are not generally used as often as lists in practice, but their immutability is the whole point. If you pass a collection of objects around your program as a list, it can be changed anywhere; if you use a tuple, it cannot.
- That is, tuples provide a sort of integrity constraint that is convenient in programs larger than those we’ll write here.

Exercise 1: Reverse the tuple

Input

tuple1 = (10, 20, 30, 40, 50)

Output

(50, 40, 30, 20, 10)

Solution

```
>>> tuple1 = (10, 20, 30, 40, 50)

>>> tuple1 = [::-1]

>>> print(tuple1)
```

Exercise 2: Create a tuple with single item 50

Solution

```
>>> tuple1 = (50,)
```

Exercise 4: Unpack the tuple into 4 variables

Input

```
tuple1 = (10, 20, 30, 40)
```

Output

```
tuple1 = (10, 20, 30, 40)
# Your code
print(a) # should print 10
print(b) # should print 20
print(c) # should
print 30 print(d) # should print 40
```

Solution

```
>>> tuple1 = (10, 20, 30, 40)

>>> a, b, c, d =tuple1

>>> print(a)

>>> print(b)

>>> print(c)

>>> print(d)
```

13. Python - Set

13. Python - Set

- A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).
- We can add or remove items from it.
- Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

Creating Python Sets

- A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function.
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.).
- A set cannot have mutable elements like lists, sets or dictionaries as its elements.

```
>>> my_set = {1, 2, 3}                              # set of integers
>>> print(my_set)
{1, 2, 3}
>>> my_set = {1.0, "Hello", (1, 2, 3)}              # set of mixed datatypes
>>> print(my_set)
{1.0, 'Hello', (1, 2, 3)}
>>> my_set = {1, 2, 3, 4, 3, 2}
>>> print(my_set)
{1, 2, 3, 4}
>>> my_set = {[1, 2, 3, 4]}                         # we can make set from a list
>>> print(my_set)
{1, 2, 3, 4}
```

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.

```
>>> my_set = {}                                   # initialize a with {}

>>> print(type(a))

<class 'dict'>

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

>>> print(type(a))

<class ‘set'>
```

Modifying a set in Python

- Sets are mutable. However, since they are unordered, indexing has no meaning.
- We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.
- We can add a 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.

Modifying a set in Python

```
>>> my_set = {1, 2, 5}                          # initialize a with {}
>>> print(my_set)
{1, 2, 5}
>>> my_set.add(3)                               # add an element
>>> print(my_set)
{1, 2, 3, 5}
>>> my_set.update([2, 3, 4])                    # add multiple elements
>>> print(my_set)
{1, 2, 3, 4, 5}
>>> my_set.update([4, 5], [1, 6, 8])            # add multiple elements
>>> print(my_set)
{1, 2, 3, 4, 5, 6, 8}
```

Removing elements from a set

- A particular item can be removed from a set using the methods **discard()** and **remove()**.
- The only difference between the two is that the discard() function leaves a set unchanged if the element is not present in the set.
- The remove() function will raise an error in such a condition (if element is not present in the set).

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

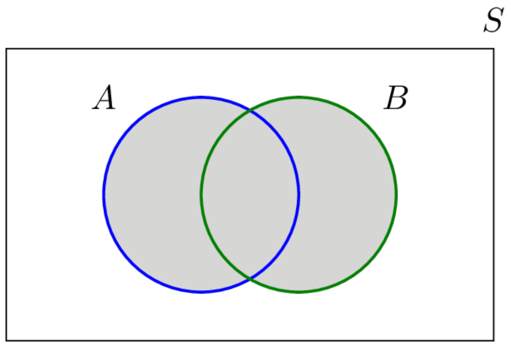
```
>>> print(my_set)

{1, 3, 5}

>>> my_set.discard(2)

>>> print(my_set)

{1, 3, 5}

>>> my_set.remove(2)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 2
```

- We can remove and return an item using the **pop()** method
- Remove all the items from a set using the **clear()** method.

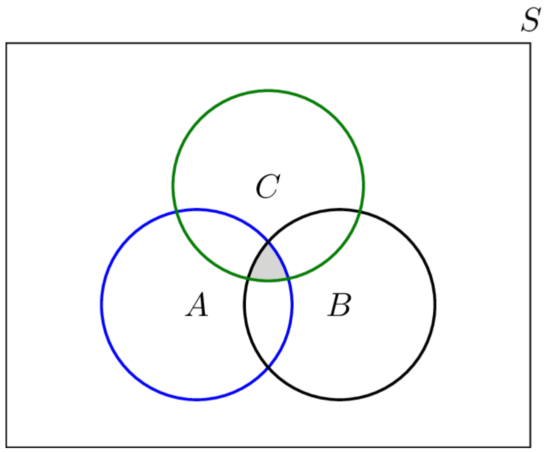
```
>>> my_set = set("PTITHCM")
>>> print(my_set)
{'M', 'P', 'H', 'T', 'C', 'I'}
>>> print(my_set.pop())                    #pop an element
M
>>> my_set.pop()                           # pop another element
'P'
>>> print(my_set)
{'H', 'T', 'C', 'I'}
```

Set Operations

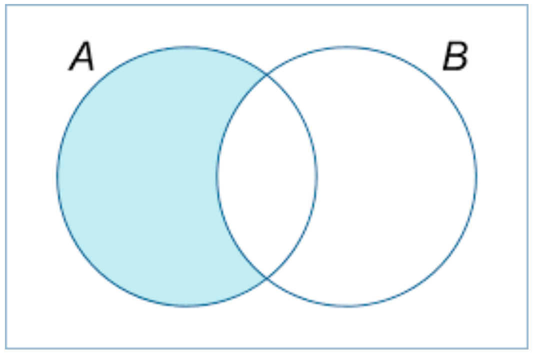
- Sets can be used to carry out mathematical set operations like **union, intersection, difference and symmetric difference**. We can do this with operators or methods.



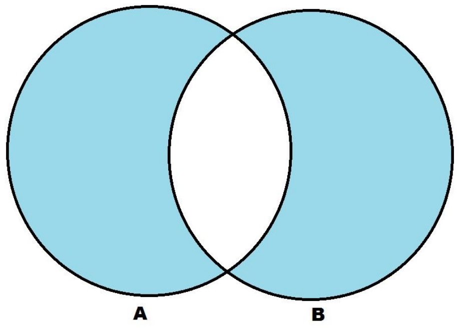
union



difference



intersection

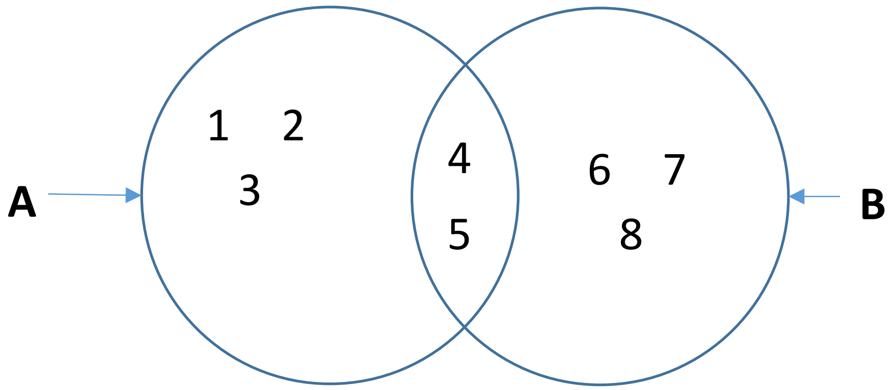


symmetric difference

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 **union()** method.

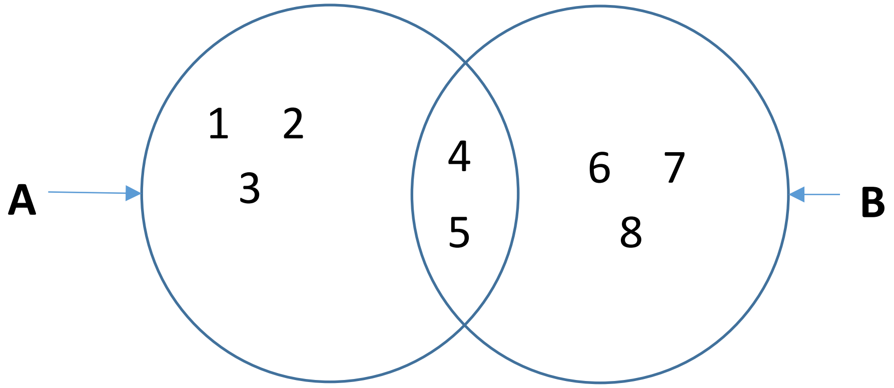
```
>>> A = {1, 2, 3, 4, 5}                    # initialize A and B
>>> B = {4, 5, 6, 7, 8}
>>> print(A | B)
{1, 3, 4, 5, 6, 7, 8}

>>> A.union(B)
{1, 3, 4, 5, 6, 7, 8}
>>> B.union(A)
{1, 3, 4, 5, 6, 7, 8}
```



Set Union

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

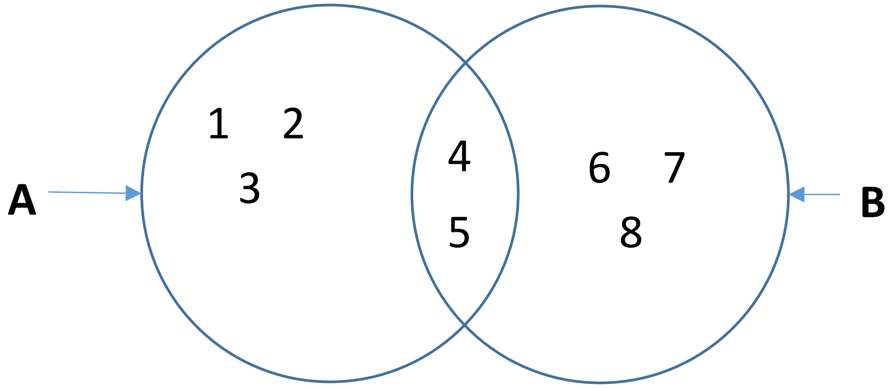
```
>>> A = {1, 2, 3, 4, 5}                    # initialize A and B
>>> B = {4, 5, 6, 7, 8}
>>> print(A | B)
{4, 5,}

>>> A.intersection(B)
{4, 5,}
>>> B. intersection(A)
{4, 5,}
```



Set Difference

- Difference of the set B from set A(A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of elements in B but not in A.
- Difference is performed using **- operator**. Same can be accomplished using the **difference()** method.

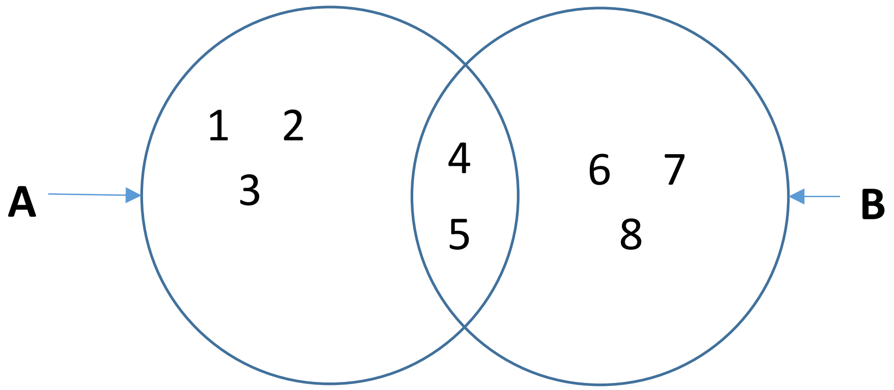
```
>>> A = {1, 2, 3, 4, 5}                         # initialize A and B
>>> B = {4, 5, 6, 7, 8}
>>> print(A - B)
{1, 2, 3}

>>> A.difference(B)
{1, 2, 3}
>>> B. difference(A)
{8, 6, 7}
```



Set Symmetric Difference

- Symmetric Difference of A and B is a set of elements in A and B but not in both (excluding the intersection).
- Symmetric difference is performed using ^ **operator**. Same can be accomplished using the method **symmetric_difference().**

```
>>> A = {1, 2, 3, 4, 5}                    # initialize A and B
>>> B = {4, 5, 6, 7, 8}
>>> print(A ^ B)
{1, 2, 3, 6, 7, 8}

>>> A.symmetric_difference(B)
{1, 2, 3, 6, 7, 8}
>>> B. symmetric_difference(A)
{1, 2, 3, 6, 7, 8}
```



Other Python Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection

Other Python Set Methods

Method	Description
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
pop()	Removes and returns an arbitrary set element. Raises KeyError if the set is empty
remove()	Removes an element from the set. If the element is not a member, raises a KeyError
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

Built-in Functions with Set

- Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with sets to perform different tasks.

Function	Description
all()	Returns True if all elements of the set are true (or if the set is empty).
any()	Returns True if any element of the set is true. If the set is empty, returns False.
enumerate()	Returns an enumerate object. It contains the index and value for all the items of the set as a pair.
len()	Returns the length (the number of items) in the set.
max()	Returns the largest item in the set.
min()	Returns the smallest item in the set.
sorted()	Returns a new sorted list from elements in the set(does not sort the set itself).
sum()	Returns the sum of all elements in the set.

EX1: Python | Find missing and additional values in two lists

- Given two lists, find the missing and additional values in both the lists.
- Input :
  - list1 = [1, 2, 3, 4, 5, 6]
  - list2 = [4, 5, 6, 7, 8]
- Output :
  - Missing values in list1 = [8, 7]
  - Additional values in list1 = [1, 2, 3]
  - Missing values in list2 = [1, 2, 3]
  - Additional values in list2 = [7, 8]

EX2: Concatenated string with uncommon characters in Python

- Two strings are given and you have to modify 1st string such that all the **common** characters of the **2nd string** have to be **removed** and the uncommon characters of the **2nd string** have to be concatenated with uncommon characters of the 1st string.
- Input : S1 = "aacdb”;   S2 = "gafd"
- Output : "cbgf"
- Input : S1 = "abcs"; S2 = "cxzca";
- Output : "bsxz"

Differences and Applications of List, Tuple, Set and Dictionary in Python

- **Lists:** are just like dynamic sized arrays, declared in other languages (vector in C++ and ArrayList in Java). Lists need not be homogeneous always which makes it the most powerful tool in Python.
- **Tuple:** A Tuple is a collection of Python objects separated by commas. In some ways, a tuple is similar to a list in terms of indexing, nested objects, and repetition but a tuple is immutable, unlike lists that are mutable.
- **Set:** A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements. Python's set class represents the mathematical notion of a set.
- **Dictionary:** in Python is an ordered (since Py 3.7) [unordered (Py 3.6 & prior)] collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, Dictionary holds **key:value** pair. Key-value is provided in the dictionary to make it more optimized.
- List, Tuple, Set, and Dictionary are the data structures in python that are used to store and organize the data in an efficient manner.

Differences and Applications of List, Tuple, Set and Dictionary in Python

List	Tuple	Set	Dictionary
List is a non-homogeneous data structure that stores the elements in single row and multiple rows and columns	Tuple is also a non-homogeneous data structure that stores single row and multiple rows and columns	Set data structure is also non-homogeneous data structure but stores in single row	Dictionary is also a non-homogeneous data structure which stores key value pairs
List can be represented by **[ ]**	Tuple can be represented by **( )**	Set can be represented by **{ }**	Dictionary can be represented by **{ }**
List **allows** duplicate elements	Tuple **allows** duplicate elements	Set will **not allow** duplicate elements	Set will **not allow** duplicate elements and dictionary doesn't allow duplicate keys.
List **can** use nested among all	Tuple **can** use nested among all	Set **can** use nested among all	Dictionary **can** use nested among all

Differences and Applications of List, Tuple, Set and Dictionary in Python

List	Tuple	Set	Dictionary
List can be created using **list()** function	Tuple can be created using **tuple()** function.	Set can be created using **set()** function	Dictionary can be created using **dict()** function.
List is **mutable** i.e we can make any changes in list.	Tuple is **immutable** i.e we can not make any changes in tuple	Set is **mutable** i.e we can make any changes in set. But elements are not duplicated.	Dictionary is **mutable**. But **Keys are not duplicated**.
List is **ordered**	Tuple is **ordered**	Set is **unordered**	Dictionary is **ordered** (Python 3.7 and above)

Creating an empty list
l=[]
	Creating an empty Tuple
	t=()
		Creating a set
		a=set()
		b=set(a)
			Creating an empty dictionary
			d={}

Applications of List, Set, Tuple, and Dictionary

- **List:**
  - Used in JSON format
  - Useful for Array operations
  - Used in Databases
- **Tuple:**
  - Used to insert records in the database through SQL query at a time.Ex: (1.'sravan', 34).(2.'geek', 35)
  - Used in parentheses checker

Applications of List, Set, Tuple, and Dictionary

- **Set:**
  - Finding unique elements
  - Join operations
- **Dictionary:**
  - Used to create a data frame with lists
  - Used in JSON

Python – Date and Time

Python datetime module

- In Python, date and time are not a data type of their own, but a module named **datetime** can be imported to work with the date as well as time. **Python Datetime module** comes built into Python, so there is no need to install it externally.
- Python Datetime module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times and time intervals. Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.

The DateTime module is categorized into 6 main classes

- **date** – An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Its attributes are year, month and day.
- **time** – An idealized time, independent of any particular day, assuming that every day has exactly 24*60*60 seconds. Its attributes are hour, minute, second, microsecond, and tzinfo.
- **datetime** – Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.
- **timedelta** – A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.
- **tzinfo** – It provides time zone information objects.
- **timezone** – A class that implements the tzinfo abstract base class as a fixed offset from the UTC (New in version 3.2).

Date class

- The **date class** is used to instantiate date objects in Python. When an object of this class is instantiated, it represents a date in the format YYYY-MM-DD. Constructor of this class needs three mandatory arguments year, month and date.
- **Constructor syntax:**
    - class datetime.date(year, month, day)
- The arguments must be in the following range:
    - MINYEAR <= **year** <= MAXYEAR
    - 1 <= **month** <= 12
    - 1 <= **day** <= number of days in the given month and year

Example 1: Date object representing date in Python

```
>>> from datetime import date                                #import date class
>>> my_date = date(2022, 12, 11)
>>> print("Date passed as argument is", my_date)
Date passed as argument is 2022-12-11
```

Example 2: Get current date in Python

To return the current local date today() function of date class is used. today() function comes with several attributes (year, month and day). These can be printed individually.

```
>>> from datetime import date                                #import date class
>>> today = date.today()
>>> print("Today's date is ", my_date)
Today's date is 2022-02-11
```

Example 2: Get current date in Python

To return the current local date today() function of date class is used.

today() function comes with several attributes (year, month and day).

These can be printed individually.

```
>>> from datetime import date                                    #import date class
>>> today = date.today()
>>> print("Today's date is ", my_date)
Today's date is 2022-02-11
```

Example 3: Get Today’s Year, Month, and Date

- We can get the year, month, and date attributes from the date object using the **year, month and date attribute** of the date class.

```
>>> from datetime import date                              #import date class
>>> today = date.today()
>>>print("Current year:", today.year)
>>>print("Current month:", today.month)
>>>print("Current day:", today.day)
```

Example 4: Get date from Timestamp

- We can create date objects from timestamps y=using the fromtimestamp() method.
- The timestamp is the number of seconds from 1st January 1970 at UTC to a particular date.

```
>>> from datetime import datetime
>>> date_time = datetime.fromtimestamp(1887630000)
>>> print("Datetime from timestamp:", date_time)
Datetime from timestamp: 2029-10-25 20:40:00
```

Example 5: Convert Date to String

- We can convert date object to a string representation using two functions isoformat() and strftime().

```
>>> from datetime import date
>>> today = date.today()
>>> Str = date.isoformat(today)                    # Converting the date to the string
>>> print("String Representation", Str)
>>> print(type(Str))
```

List of Date class Methods

Function Name	Description
ctime()	Return a string representing the date
fromisocalendar()	Returns a date corresponding to the ISO calendar