

Module-4

Tuple: Creating Tuples; Operations on Tuples; Built-in Functions on Tuple.

Set: Creating Sets; Operations on Sets; Built-in Functions on Sets; Set Methods.

Dictionaries: Creating Dictionaries; Operations on Dictionaries; Built-in Functions on Dictionaries; Dictionary Methods.

Python Tuples:

A **tuple** in Python is an **ordered**, **immutable** collection of elements enclosed in **parentheses** ().

It can store different types of data such as integers, strings, floats, etc.

- Tuples are similar to lists, but unlike lists, they cannot be changed after their creation (i.e., they are immutable).
- Tuples can hold elements of different data types.
- The main characteristics of tuples are being ordered, heterogeneous and immutable.

Creating a Tuple

A tuple is created by placing all the items inside parentheses (), separated by commas. A tuple can have any number of items and they can be of different [data types](#).

```
tup = ()
print(tup)

# Using String
tup = ('Geeks', 'For')
print(tup)

# Using List
li = [1, 2, 4, 5, 6]
print(tuple(li))

# Using Built-in Function
tup = tuple('Geeks')
print(tup)
```

Output:

```
()  
( 'Geeks', 'For' )  
(1, 2, 4, 5, 6)  
( 'G', 'e', 'e', 'k', 's' )
```

Creating a Tuple with Mixed Datatypes.

Tuples can contain elements of various data types, including other tuples, [lists](#), [dictionaries](#) and even [functions](#).

```
tup = (5, 'Welcome', 7, 'Geeks')  
print(tup)
```

```
# Creating a Tuple with nested tuples
```

```
tup1 = (0, 1, 2, 3)  
tup2 = ('python', 'geek')  
tup3 = (tup1, tup2)  
print(tup3)
```

```
# Creating a Tuple with repetition
```

```
tup1 = ('Geeks',) * 3  
print(tup1)
```

```
# Creating a Tuple with the use of loop
```

```
tup = ('Geeks')  
n = 5  
for i in range(int(n)):  
    tup = (tup,)  
    print(tup)
```

Output

```
(5, 'Welcome', 7, 'Geeks')
((0, 1, 2, 3), ('python', 'geek'))
('Geeks', 'Geeks', 'Geeks')
('Geeks',)
(('Geeks',),)
((( 'Geeks',),),)
(((( 'Geeks',),),),)
((((('Geeks',),),),),),)
```

Python Tuple Basic Operations

Below are the Python tuple operations.

- Accessing of Python Tuples
- Concatenation of Tuples
- Slicing of Tuple
- Deleting a Tuple

Accessing of Tuples

We can access the elements of a tuple by using indexing and [slicing](#), similar to how we access elements in a list. Indexing starts at 0 for the first element and goes up to n-1, where n is the number of elements in the tuple. Negative indexing starts from -1 for the last element and goes backward.

```
# Accessing Tuple with Indexing
tup = tuple("Geeks")
print(tup[0])

# Accessing a range of elements using slicing
print(tup[1:4])
print(tup[:3])

# Tuple unpacking
tup = ("Geeks", "For", "Geeks")

# This line unpack values of Tuple1
a, b, c = tup
print(a)
print(b)
print(c)
```

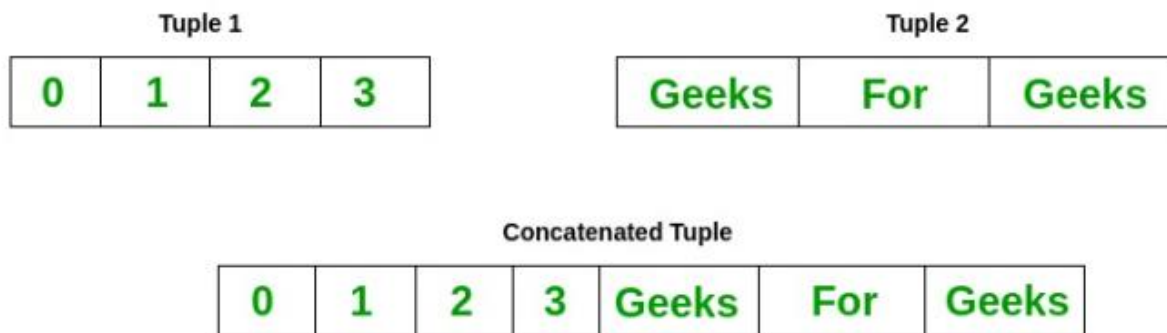
Output

```
G
('e', 'e', 'k')
('G', 'e', 'e')
Geeks
For
Geeks
```

Concatenation of Tuples

Tuples can be concatenated using the + operator. This operation combines two or more tuples to create a new tuple.

***Note:** Only the same datatypes can be combined with concatenation, an error arises if a list and a tuple are combined.*



```
tup1 = (0, 1, 2, 3)
tup2 = ('Geeks', 'For', 'Geeks')

tup3 = tup1 + tup2
print(tup3)
```

Output

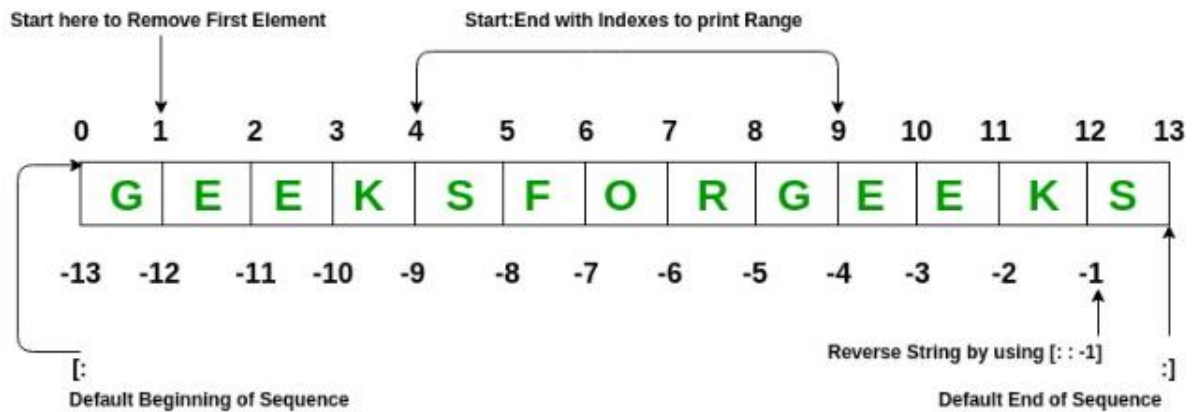
```
(0, 1, 2, 3, 'Geeks', 'For', 'Geeks')
```

Slicing of Tuple

[Slicing a tuple](#) means creating a new tuple from a subset of elements of the original tuple.

The slicing syntax is tuple[start:stop:step].

Note- Negative Increment values can also be used to reverse the sequence of Tuples.



```
tup = tuple('GEEKSFORGEEKS')

# Removing First element
print(tup[1:])

# Reversing the Tuple
print(tup[::-1])

# Printing elements of a Range
print(tup[4:9])
```

Output

```
('E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S')
('S', 'K', 'E', 'E', 'G', 'R', 'O', 'F', 'S', 'K', 'E', 'E', 'G')
('S', 'F', 'O', 'R', 'G')
```

Deleting a Tuple

Since tuples are immutable, we cannot delete individual elements of a tuple. However, we can delete an entire tuple using [del statement](#).

Note: Printing of Tuple after deletion results in an Error.

```
tup = (0, 1, 2, 3, 4)
del tup

print(tup)
```

Output

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 6, in <module>
NameError: name 'tup' is not defined
```

Tuple Unpacking with Asterisk (*)

In Python, the " *" operator can be used in tuple unpacking to grab multiple items into a list. This is useful when you want to extract just a few specific elements and collect the rest together.

```
tup = (1, 2, 3, 4, 5)
```

```
a, *b, c = tup
```

```
print(a)
print(b)
print(c)
```

Output

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

Explanation:

- **a** gets the first item.
- **c** gets the last item.
- ***b** collects everything in between into a list.

Built-in Functions on Tuple.

1. len() — Returns the length of the tuple

Example Program

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

print("Tuple:", t)
print("Length of tuple:", len(t))
```

Output

```
python

Tuple: (10, 20, 30, 40)
Length of tuple: 4
```

2. max() — Returns the maximum element

Example Program

```
t = (5, 10, 25, 2, 18)

print("Tuple:", t)
print("Maximum value:", max(t))
```

Output

```
yaml

Tuple: (5, 10, 25, 2, 18)
Maximum value: 25
```

3. min() — Returns the minimum element

Example Program

```
t = (5, 10, 25, 2, 18)

print("Tuple:", t)
print("Minimum value:", min(t))
```

Output

```
yaml

Tuple: (5, 10, 25, 2, 18)
Minimum value: 2
```

4. sum() — Returns the sum of all elements (works only for numeric tuples)

Example Program

```
t = (10, 20, 30)

print("Tuple:", t)
print("Sum of elements:", sum(t))
```

Output

```
yaml

Tuple: (10, 20, 30)
Sum of elements: 60
```


5. tuple() — Converts another data type into a tuple

Example Program

```
list1 = [1, 2, 3, 4]

t = tuple(list1)

print("Converted tuple:", t)
print("Type:", type(t))
```

Output

```
python

Converted tuple: (1, 2, 3, 4)
<class 'tuple'>
```

Python Sets: Creating Sets; Operations on Sets; Built-in Functions on Sets; Set Methods.

In Python, a Set is one of the four built-in data types used to store numerous items in a single variable. Set is an unindexed and unordered collection of unique elements. For example, a set is suitable option when storing information about employee IDs as these IDs cannot have duplicates.

A Set in Python is used to store a collection of items with the following properties.

- No duplicate elements. If try to insert the same item again, it overwrites previous one.
- An unordered collection. When we access all items, they are accessed without any specific order and we cannot access items using indexes as we do in lists.
- Internally use [hashing](#) that makes set efficient for search, insert and delete operations. It gives a major advantage over a [list](#) for problems with these operations.
- Mutable, meaning we can add or remove elements after their creation, the individual elements within the set cannot be changed directly.

202	205	204	209	207
-----	-----	-----	-----	-----

Set of Employee ID

Example

```
# creating a Set
S = {202, 205, 204, 209, 207}

print(S)
```

Output:

```
{209, 202, 204, 205, 207}
```

Explanation:

In the above example, we have created a simple set consisting of multiple items. Here, we can see that the items in the initialized set are unordered.

A set is a mutable data type, meaning we can remove or add data elements to it. Python Sets are similar to the sets in mathematics, where we can perform operations like intersection, union, symmetric difference, and more.

Characteristics of Python Sets

Set in **Python** is a **data type**, which is:

- **Unordered:** Sets do not maintain the order of how elements are stored in them.
- **Unindexed:** We cannot access the data elements of sets.
- **No Duplicate Elements:** Each data element in a set is unique.
- **Mutable (Changeable):** Sets in Python allow modification of their elements after creation.

Creating a Set

Creating a set in Python is quite simple and easy process. Python offers two ways to create a set:

1. Using curly braces
2. Using set() function

Using Curly Braces

A set can be created by enclosing elements within curly braces '{}', separated by commas.

Let us see a simple example showing the way of creating a set using curly braces.

Example

```
# simple example to create a set using curly braces

int_set = {12, 6, 7, 9, 11, 10} # set of integers
print(int_set)

str_set = {'one', 'two', 'three', 'four', 'five'} # set of strings
print(str_set)

mixed_set = {12, 'tpointtech', 7.2, 6e2} # mixed set
print(mixed_set)
```

Output:

```
{6, 7, 9, 10, 11, 12}
{'one', 'three', 'two', 'four', 'five'}
{600.0, 'tpointtech', 12, 7.2}
```

In this example, we have used the curly braces to create different types of sets. Moreover, we can observe that a set can store any number of items of different types, like [integer](#), float, [tuple](#), [string](#), etc. However, a set cannot store mutable elements like [lists](#), sets, or [dictionaries](#).

Using the set() Function

Python offers an alternative way of creating a set with the help of its built-in function called [set\(\)](#). This function allow us to create a set from a passed iterable.

The following example shows the way of using the set() function:

Example

```
# simple example to create a set using set() function

# given list
int_list = [6, 8, 1, 3, 7, 10, 4]

# creating set using set() function
int_set = set(int_list)

print("Set 1:", int_set)

# creating an empty set
empty_set = set()

print("Set 2:", empty_set)
```

Output:

```
Set 1: {1, 3, 4, 6, 7, 8, 10}
Set 2: set()
```

Explanation:

In the above example, we have used the `set()` function to create set from a given list. We have also created an empty set by using the `set()` function without any arguments.

Note: Creating an empty set is a bit tricky. Empty curly braces '{}' will make an empty dictionary in Python.

Accessing Elements of a Set

Since sets are unordered and unindexed, we cannot access the elements by position. However, we can iterate through a set with the help of loops.

Let us see a simple example showing the way of iterating through a set in Python.

Example

```
# simple example to show how to iterate through a set

# given set
set_one = {11, 17, 12, 5, 7, 8}
print("Given Set:", set_one)

# iterating through the set using for loop
print("Iterating through the Set:")
for num in set_one:
    print(num)
```

Output:

```
Given Set: {17, 5, 7, 8, 11, 12}
Iterating through the Set:
17
5
7
8
11
12
```

Adding Elements to the Set

Python provides methods like `add()` and `update()` to add elements to a set.

- **add():** This method is used to add a single element to the set.
- **update():** This method is used to add multiple elements to the set.

Let us see a simple example showing the way of adding elements to the set in Python.

Example

```
# simple example to show how to add elements to the set

# given set
subjects = {'physics', 'biology', 'chemistry'}
print("Given Set:", subjects)

# adding a single element to the set
subjects.add('maths')    # using add()
print("Updated Set (Added single element):", subjects)

# adding multiple elements to the set
subjects.update(['computer', 'english'])    # using update()
print("Update Set (Added Multiple elements):", subjects)
```

Output:

```
Given Set: {'physics', 'biology', 'chemistry'}
Updated Set (Added single element): {'physics', 'biology', 'chemistry', 'maths'}
Update Set (Added Multiple elements): {'physics', 'chemistry', 'english',
'biology', 'computer', 'maths'}
```

Explanation:

In this example, we have given a set consisting of 3 elements. We have then used the `add()` method to add a new element to the set. We have also used the `update()` method to add multiple elements to the given set.

Removing Elements from the Set

In Python, we can easily remove elements from a given set using methods like `remove()`, `discard()`, `pop()`, and `clear()`.

- **remove():** This method allow us to remove a specific element from the set. It will raise a `KeyError` if the element is not found in the given set.

- **discard():** This method is also used to remove a specified element from the set; however, it does not raise any error if the element is not found.
- **pop():** This method is used to remove and returns a random element from the set.
- **clear():** This method is used to remove all the elements from the given set.

Here is a simple example showing the working of these methods to remove elements from a set in Python.

Example

```
# simple example to show how to remove elements from the set

# given set
subjects = {'physics', 'chemistry', 'english', 'biology', 'computer', 'maths'}
print("Given Set:", subjects)

# removing a specified element from the set
subjects.remove('maths')    # using remove()
print("Updated Set (Removed 'maths'):", subjects)

# removing a specified element from the set
subjects.discard('chemistry') # using discard()
print("Updated Set (Removed 'chemistry'):", subjects)

# removing a random element from the set
subjects.pop()    # using pop()
print("Updated Set (Removed a random element):", subjects)

# removing all elements from the set
subjects.clear()    # using clear()
print("Updated Set (Removed all elements):", subjects)
```

Output:

```
Given Set: {'physics', 'chemistry', 'english', 'computer', 'biology', 'maths'}
Updated Set (Removed 'maths'): {'physics', 'chemistry', 'english', 'computer', 'biology'}
Updated Set (Removed 'chemistry'): {'physics', 'english', 'computer', 'biology'}
Updated Set (Removed a random element): {'english', 'computer', 'biology'}
Updated Set (Removed all elements): set()
```

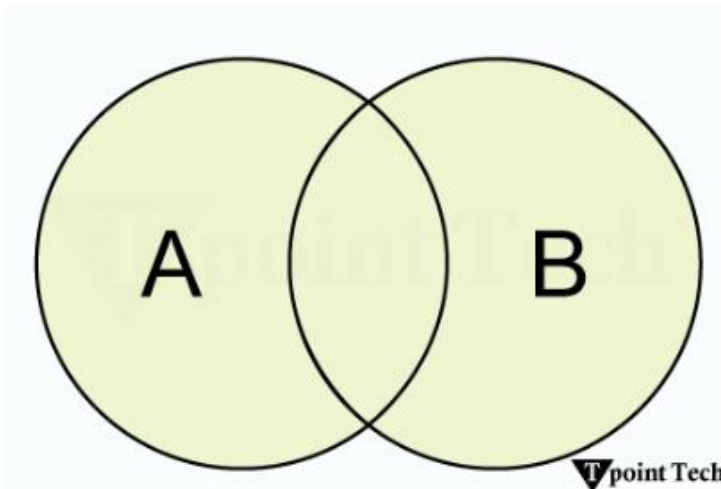
Set Operations in Python

Similar to the [Set Theory](#) in Maths, Python sets also provide support to various [mathematical operations](#) like union, intersection, difference, symmetric difference and more.

Let us discuss some of these operations with the help of examples.

Union of Sets

In mathematical terms, union of sets A and B is defined as the set of all those elements which belongs to A or B or both and is denoted by $A \cup B$.



For instance, let $A = \{1, 2, 3\}$, and $B = \{2, 3, 4, 5\}$. Therefore, $A \cup B = \{1, 2, 3, 4, 5\}$.

Similarly in Python, we can perform union of sets by combining their elements, and eliminating duplicates with the help of the `|` operator or `union()` method.

Let us see a simple example showing the union of sets in Python.

Example

```
# simple example on union of sets

set_A = {1, 2, 3}  # set A
print("Set A:", set_A)

set_B = {2, 3, 4, 5} # set B
print("Set B:", set_B)

print("\nUnion of Sets A and B:")  # union of sets
print("Method 1:", set_A | set_B)  # using |
print("Method 2:", set_A.union(set_B)) # using union()
```


Output:

```
Set A: {1, 2, 3}
Set B: {2, 3, 4, 5}

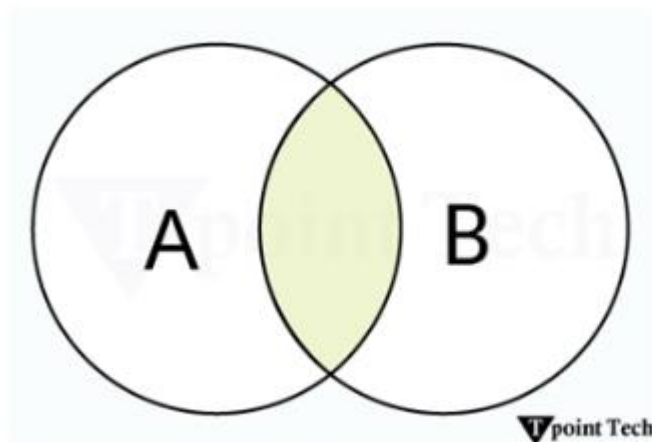
Union of Sets A and B:
Method 1: {1, 2, 3, 4, 5}
Method 2: {1, 2, 3, 4, 5}
```

Explanation:

In the above example, we have defined two sets and performed their union using the `|` operator and `union()` method.

Intersection of Sets

In mathematical terms, intersection of two sets A and B is defined as the set of all those elements which belongs to both A and B and is denoted by $A \cap B$.



For instance, let $A = \{1, 2, 3\}$, and $B = \{2, 3, 4, 5\}$. Therefore, $A \cap B = \{2, 3\}$.

Similarly in Python, we can perform intersection of sets by using the `&` operator or `intersection()` method to return the elements common in both sets.

Let us see a simple example showing the intersection of sets in Python.

Example

```
# simple example on intersection of sets

set_A = {1, 2, 3}    # set A
print("Set A:", set_A)

set_B = {2, 3, 4, 5} # set B
print("Set B:", set_B)

print("\nIntersection of Sets A and B:")    # intersection of sets
print("Method 1:", set_A & set_B)           # using &
print("Method 2:", set_A.intersection(set_B)) # using intersection()
```

Output:

```
Set A: {1, 2, 3}
Set B: {2, 3, 4, 5}

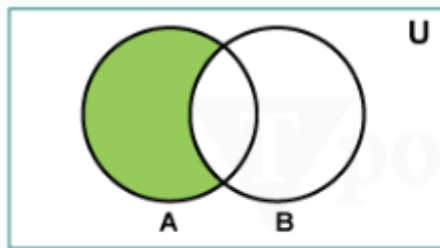
Intersection of Sets A and B:
Method 1: {2, 3}
Method 2: {2, 3}
```

Explanation:

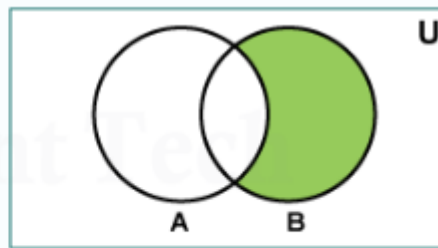
In the above example, we have defined two sets and performed their intersection using the & operator and intersection() method.

Difference of Sets

In mathematical terms, difference of two sets A and B is defined as the set of all those elements which belongs to A, but do not belong to B and is denoted by A-B.



A-B



B-A

point Tech

For instance, let $A = \{1, 2, 3\}$, and $B = \{2, 3, 4, 5\}$. Therefore, $A-B = \{1\}$ and $B-A = \{4, 5\}$.

Similarly in Python, we can perform difference of sets by using the `-` operator or `difference()` method to return the elements present in the first set but not in the second one.

Let us see a simple example showing the difference of sets in Python.

Example

```
# simple example on difference of sets

set_A = {1, 2, 3}    # set A
print("Set A:", set_A)

set_B = {2, 3, 4, 5} # set B
print("Set B:", set_B)

print("\nA - B:")    # difference of sets
print("Method 1:", set_A - set_B)    # using -
print("Method 2:", set_A.difference(set_B)) # using difference()

print("\nB - A:")
print("Method 1:", set_B - set_A)    # using -
print("Method 2:", set_B.difference(set_A)) # using difference()
```

Output:

```
Set A: {1, 2, 3}
Set B: {2, 3, 4, 5}

A - B:
Method 1: {1}
Method 2: {1}

B - A:
Method 1: {4, 5}
Method 2: {4, 5}
```

Explanation:

In the above example, we have defined two sets and performed their difference using the `-` operator and `difference()` method.

Python Set Methods

In Python, **set methods** are used to manipulate the data of a set in an effective and efficient way. These methods allow us to add, remove, and update the elements of sets. [Sets in Python](#), is an unordered and [mutable data type](#) allowing us to store a collection of unique objects in a single variable.

Let us take a look at various Set methods available in Python.

Method	Description
<code>add()</code>	This method is utilized to add a data element to the set.
<code>clear()</code>	This method is utilized to remove all data elements from the set.
<code>copy()</code>	This method is used to return a shallow copy of the set.
<code>discard()</code>	This method is used to remove a data element if it is a member. It will not return any error if the specified element is not found in the set.
<code>remove()</code>	This method is utilized to remove a data element from the given set; however, it raises a <code>KeyError</code> if the specified element is not found.
<code>pop()</code>	This method is used to remove and return an arbitrary element.
<code>update()</code>	This method is utilized to add elements from other sets or iterables.

1) add()

The **add()** method is used to add a new element to a set while ensuring uniqueness. If the passed element already exists, the set remains unchanged.

The syntax of the add() method is shown below:

Syntax:

1. `set_name.add(item)`

We will now look at a simple example showing the use case of set's add() method in Python.

Example

```
# python program to show the use of set add() method

# creating a set
set_of_fruits = {'apple', 'mango', 'banana', 'orange', 'guava'}

# printing the set
print("Set of Fruits:", set_of_fruits)

# using the add() method
set_of_fruits.add('grapes')

# printing the updated set
print("Updated Set of Fruits:", set_of_fruits)
```

Output:

```
Set of Fruits: {'banana', 'orange', 'apple', 'guava', 'mango'}
Updated Set of Fruits: {'banana', 'orange', 'apple', 'guava', 'mango', 'grapes'}
```

Explanation:

In the above example, we have used the add() method to add a new element 'grapes' to the given set.

2) clear()

The **clear()** method is utilized to remove all the elements from the given set.

The following is the syntax of the clear() method:

Syntax:

1. `set_name.clear()`

2. We will now look at an example to understand the working of set's clear() method in Python.

Example

```
# python program to show the use of set clear() method

# creating a set
game_set = {'football', 'cricket', 'volleyball', 'basketball', 'hockey'}
# printing the set
print("Given Set:", game_set)

# using the clear() method
game_set.clear()

# printing the updated set
print("Updated Set:", game_set)
```

Output:

```
Given Set: {'basketball', 'hockey', 'football', 'cricket', 'volleyball'}
Updated Set: set()
```

Explanation:

In the above example, we have used the clear() method to remove all the elements from the given set.

3) copy()

The **copy()** method is used to return a shallow copy of the set in Python.

Here is the syntax of the copy() method:

Syntax:

1. set_name.copy()

We will now see an example showing the use of set's copy() method in Python.

Example

```
# python program to show the use of set copy() method

# creating a set
vegetable_set = {'potato', 'eggplant', 'tomato', 'cabbage', 'broccoli'}

# printing the set
print("Given Set:", vegetable_set)

# using the copy() method
dummy_set = vegetable_set.copy()

# printing the updated set
print("Dummy Set:", dummy_set)
```

Output:

```
Given Set: {'potato', 'tomato', 'broccoli', 'eggplant', 'cabbage'}
Dummy Set: {'potato', 'tomato', 'broccoli', 'eggplant', 'cabbage'}
```

Explanation:

In this example, we have used the `copy()` method to create a shallow copy of the given set.

4) discard()

The **discard()** method is utilized to remove the elements from the set. This method does not return any error in case the particular element is not found in the given set.

The following is the syntax of the `discard()` method:

Syntax:

1. `set_name.discard(item)`

We will now look at a simple example showing the usage of set's `discard()` method in Python.

Example

```
# python program to show the use of set discard() method

# creating a set
beverage_set = {'milk', 'juice', 'soda', 'tea', 'coffee'}

# printing the set
print("Given Set:", beverage_set)

# using the discard() method
beverage_set.discard('soda')

# printing the updated set
print("Updated Set:", beverage_set)
```

Output:

```
Given Set: {'juice', 'milk', 'coffee', 'tea', 'soda'}
Updated Set: {'juice', 'milk', 'coffee', 'tea'}
```

Explanation:

In this example, we have used the `discard()` method to remove the specified element from the given set.

5) `remove()`

The **`remove()`** method is utilized to delete the specified element from the set. It will raise an error if the passed element does not exist in the given set.

Here is the syntax of the `remove()` method:

Syntax:

1. `set_name.remove(item)`

Let us now see the example showing how to use set's `remove()` method in Python.

Example

```
# python program to show the use of set remove() method

# creating a set
country_set = {'India', 'Brazil', 'Japan', 'China', 'USA'}
# printing the set
print("Given Set:", country_set)

# using the remove() method
country_set.remove('China')

# printing the updated set
print("Updated Set:", country_set)
```

Output:

```
Given Set: {'Brazil', 'China', 'India', 'Japan', 'USA'}
Updated Set: {'Brazil', 'India', 'Japan', 'USA'}
```

Explanation:

In the above example, we have used the `remove()` method to remove the specified element from the given set.

6) `pop()`

Python set **`pop()`** method allows us to remove any random element from the set. This method returns the removed element.

The syntax of the `pop()` method is shown below:

Syntax:

1. `set_name.pop()`

We will now look at a simple example showing the implementation of set's `pop()` method in Python.

Example

```
# python program to show the use of set pop() method

# creating a set
state_set = {'New York', 'Delhi', 'Tokyo', 'Penang', 'Ontario'}
# printing the set
print("Given Set:", state_set)

# using the pop() method
popped_item = state_set.pop()

# printing the updated set
print("Updated Set:", state_set)
print("Popped Element:", popped_item)
```

Output:

```
Given Set: {'Penang', 'Ontario', 'Tokyo', 'New York', 'Delhi'}
Updated Set: {'Ontario', 'Tokyo', 'New York', 'Delhi'}
Popped Element: Penang
```

Explanation:

In the above example, we have used the pop() method to remove and return a random element from the given set.

7) update()

Python set **update()** method is used to add elements from another set, list, tuple, or any other iterable to the set. Since sets are collections of unique elements, the update() method will only add the unique elements from the specified iterable to the targeted set.

The following is the syntax of the update() method:

Syntax:

```
set_name.update(*others)
```

We will now look at a simple example showing the use case of set's update() method in Python.

Example

python program to show the use of set update() method

given sets

```
num_set_1 = {4, 7, 8, 11, 19}
```

```
num_set_2 = {2, 5, 7, 8, 10}
```

```
print("Set 1:", num_set_1)
```

```
print("Set 2:", num_set_2)
```

using the update() method

```
num_set_1.update(num_set_2)
```

printing the updated set

```
print("Updated Set:", num_set_1)
```

Output:

```
Set 1: {19, 4, 7, 8, 11}
```

```
Set 2: {2, 5, 7, 8, 10}
```

```
Updated Set: {2, 4, 5, 7, 8, 10, 11, 19}
```

Explanation:

Here, we have used the update() method to add the elements from the second set to the first set. As a result, only the unique elements are added to the set.

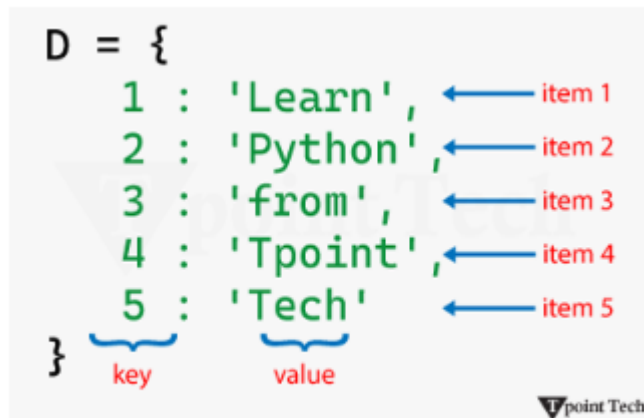
Python Dictionaries

Creating Dictionaries; Operations on Dictionaries; Built-in Functions on Dictionaries;

Dictionary Methods

A **Python Dictionary** is one of the built-in data types used to store data in '**key: value**' pairs.

The dictionary is an *unordered*, *mutable* and *indexed* collection where each key is unique and maps to a value. It is often used to store related data, like information associated with a particular entity or object, where we can easily get value on the basis of its key.



Let us take a look at a simple example of a dictionary.

Example

```
# creating a Dictionary  
D = {1: 'Learn', 2: 'Python', 3: 'from', 4: 'Tpoint', 5: 'Tech'}  
  
print(D)
```

Output:

```
{1: 'Learn', 2: 'Python', 3: 'from', 4: 'Tpoint', 5: 'Tech'}
```

Explanation:

In the above example, we have created a simple dictionary consisting of multiple 'key: value' pairs.

As we can observe, a dictionary in [Python](#) is a [mapping data type](#) where the value of one object maps to another. In order to establish the mapping between a key and a value, we have used the colon ':' symbol between the two.

Characteristics of Python Dictionary

A dictionary in Python is a [data type](#) with the following characteristics:

- **Mutable:** Dictionaries can be modified after initialization allowing us to add, remove or update 'key: value' pairs.

- **Unordered:** Python dictionary does not follow a particular order to store items. However, **starting from Python 3.7**, the feature for the dictionary to maintain the insertion order of the items was added.
 - **Indexed:** Unlike [lists](#) or [tuples](#), which are indexed by position, dictionaries use keys to access values, offering faster and more readable data retrieval.
 - **Unique Keys:** Each key in a dictionary must be unique. If we try to assign a value to an existing key, the old value will be replaced by the new one.
 - **Heterogeneous:** Keys and values in a dictionary can be of any type.
-

Creating a Dictionary

In Python, we can create a dictionary by enclosing the sequence of 'key: value' pairs with curly braces separated by commas. As an alternate option, we can use the Python's built-in [dict\(\) function](#).

Here is a simple example showing both ways of creating a dictionary in Python.

Example

```
# simple example to create python dictionary

# creating dictionaries
dict_zero = {} # empty dictionary
dict_one = {"name": "Lucy", "age": 19, "city": "New Jersey"} # using {}
dict_two = dict(name = "John", age = 21, city = "Havana") # using dict()

# printing the results
print("Empty Dictionary:", dict_zero)
print("Dictionary 1 (created using {}):", dict_one)
print("Dictionary 2 (created using dict()):", dict_two)
```

Output:

```
Empty Dictionary: {}
Dictionary 1 (created using {}): {'name': 'Lucy', 'age': 19, 'city': 'New Jersey'}
Dictionary 2 (created using dict()): {'name': 'John', 'age': 21, 'city': 'Havana'}
```

Explanation:

The above example shows different ways to create dictionaries in Python. We have also seen how to create an empty dictionary.

Note: The dict() function can also be used to transform an existing data type into a dictionary.

Accessing Dictionary Items

In Python, we can access the value of a dictionary item by **enclosing that particular key with square brackets '[]'**. Another way to access dictionary items is by the use of the `get()` method.

The following is a simple example showing the ways to access dictionary items in Python.

Example

```
# simple example to access dictionary items

# given dictionary
dict_x = {
    "name": "Sachin",
    "age": 18,
    "gender": "male",
    "profession": "student"
}

print("Person's Details")
# accessing dictionary items using keys
print("Name:", dict_x["name"])
print("Age:", dict_x["age"])

# accessing dictionary items using get()
print("Gender:", dict_x.get("gender"))
print("Profession:", dict_x.get("profession"))
```

Output:

```
Person's Details
Name: Sachin
Age: 18
Gender: male
Profession: student
```

Explanation:

Here, we have accessed the different values of the dictionary items using the square brackets and `get()` method.

Adding Items to a Dictionary

The dictionary is a [mutable data type](#) that allows us to **add an item** to it. This can be done by assigning a value to a new key.

Let us take a look at a simple example showing how to add items to a Python dictionary.

Example

```
# simple example to add item to dictionary

# given dictionary
dict_x = {
    "name": "Sachin",
    "age": 18,
    "gender": "male",
    "profession": "student"
}

print("Given Dictionary:", dict_x)

# adding an item to the dictionary
dict_x["country"] = "India"

print("Updated Dictionary:", dict_x)
```

Output:

```
Given Dictionary: {'name': 'Sachin', 'age': 18, 'gender': 'male', 'profession': 'student'}
Updated Dictionary: {'name': 'Sachin', 'age': 18, 'gender': 'male', 'profession': 'student', 'country': 'India'}
```

Explanation:

In this example, we have added a new 'key: value' pair to the dictionary using the assignment operator.

Removing Items from a Dictionary

Python offers multiple ways to **remove items from** a given dictionary, such as:

- **del**: This keyword is used to *remove* an item by key.
- **pop()**: This method is used to *remove* an item by key. It also *returns* the value of the removed item.
- **popitem()**: This method *removes* and *returns* the last 'key: value' pair.
- **clear()**: This method is used to *remove all items* from the dictionary.

Here is an example showing the use of different methods to remove items from a Python dictionary.

Example

```
# simple example to remove items from a dictionary

# given dictionary
dict_x = {
    "name": "Sachin",
    "age": 18,
    "gender": "male",
    "profession": "student",
    "country": "India"
}

print("Given Dictionary:", dict_x)

# removing items from the dictionary
del dict_x['age']    # using del
print("Updated Dictionary (Removed 'age'):", dict_x)

popped_value = dict_x.pop('gender') # using pop()
print("Updated Dictionary (Removed 'gender'):", dict_x)
print("Popped Value:", popped_value)

popped_item = dict_x.popitem() # using popitem()
print("Updated Dictionary (Removed last item):", dict_x)
print("Popped Item:", popped_item)

dict_x.clear() # using clear()
print("Update Dictionary (Removed all items):", dict_x)
```


Output:

```
Given Dictionary: {'name': 'Sachin', 'age': 18, 'gender': 'male', 'profession': 'student', 'country': 'India'}
Updated Dictionary (Removed 'age'): {'name': 'Sachin', 'gender': 'male', 'profession': 'student', 'country': 'India'}
Updated Dictionary (Removed 'gender'): {'name': 'Sachin', 'profession': 'student', 'country': 'India'}
Popped Value: male
Updated Dictionary (Removed Last item): {'name': 'Sachin', 'profession': 'student'}
Popped Item: ('country', 'India')
Update Dictionary (Removed all items): {}
```

Explanation:

In this example, we are given a dictionary. We have used several methods like `del` keyword, `pop()`, `popitem()`, and `clear()` methods to remove the items from the dictionary.

Changing Dictionary Items

In Python, we can **change the values of an item** in the dictionary by referring to its key.

Let us take a simple example to understand how to change dictionary items in Python.

Example

```
# simple example to change dictionary items
```

```
# given dictionary
```

```
dict_x = {
    "name": "Sachin",
    "age": 18,
    "gender": "male",
    "profession": "student",
    "country": "India"
}
```

```
print("Given Dictionary:", dict_x)
```

```
# changing dictionary items
```

```
dict_x["age"] = 20
dict_x["profession"] = "developer"
```

```
print("Updated Dictionary:", dict_x)
```

Output:

```
Given Dictionary: {'name': 'Sachin', 'age': 18, 'gender': 'male', 'profession': 'student', 'country': 'India'}
Updated Dictionary: {'name': 'Sachin', 'age': 20, 'gender': 'male', 'profession': 'developer', 'country': 'India'}
```

Explanation:

In this example, we have used the assignment operator to change the value of existing keys in the given dictionary. As a result, the dictionary items are updated.

Dictionary Methods in Python

Python offers several dictionary methods in order to manipulate the data of a dictionary. These methods are commonly used to add, update, delete, and return elements from the dictionaries. Some of these methods are as follows:

Dictionary Method	Description
<code>get()</code>	This method returns the value associated with a specific key.
<code>update()</code>	This method is utilized to add a new item to the dictionary or update the value of an existing key.
<code>copy()</code>	This method is utilized to return a copy of the dictionary.
<code>pop()</code>	This method removes the item with the given key from the dictionary.
<code>popitem()</code>	This method is utilized to return the last inserted key and value as a tuple.
<code>clear()</code>	This method removes all items from the dictionary.
<code>keys()</code>	This method returns all the keys in the dictionary.
<code>values()</code>	This method is utilized to return all the values in the dictionary.