Python Collections (Arrays)

There are four collection data types in the Python programming language:

* [**List**](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* **Set** is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
* [**Dictionary**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\*\* and changeable. No duplicate members.

## List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

### **Example**

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

## Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

## Allow Duplicates

Since lists are indexed, lists can have items with the same value:

### **Example**

Lists allow duplicate values:

thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)

## List Length

To determine how many items a list has, use the len() function:

### **Example**

Print the number of items in the list:

thislist = ["apple", "banana", "cherry"]  
print(len(thislist))

## List Items - Data Types

List items can be of any data type:

### **Example**

String, int and boolean data types:

list1 = ["apple", "banana", "cherry"]  
list2 = [1, 5, 7, 9, 3]  
list3 = [True, False, False]

A list can contain different data types:

### **Example**

A list with strings, integers and boolean values:

list1 = ["abc", 34, True, 40, "male"]

## type()

From Python's perspective, lists are defined as objects with the data type 'list':

<class 'list'>

### **Example**

What is the data type of a list?

mylist = ["apple", "banana", "cherry"]  
print(type(mylist))

## The list() Constructor

It is also possible to use the list() constructor when creating a new list.

### **Example**

Using the list() constructor to make a List:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)

## Access Items

List items are indexed and you can access them by referring to the index number:

### **Example**

Print the second item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[1])

### **Negative Indexing**

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

### **Example**

Print the last item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])

### **Range of Indexes**

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

### **Example**

Return the third, fourth, and fifth item:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])

**Note:** The search will start at index 2 (included) and end at index 5 (not included).

By leaving out the start value, the range will start at the first item:

### **Example**

This example returns the items from the beginning to, but NOT including, "kiwi":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[:4])

By leaving out the end value, the range will go on to the end of the list:

### **Example**

This example returns the items from "cherry" to the end:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:])

### **Range of Negative Indexes**

Specify negative indexes if you want to start the search from the end of the list:

### **Example**

This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[-4:-1])

## Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

### **Example**

Check if "apple" is present in the list:

thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
  print("Yes, 'apple' is in the fruits list")

## Remove Specified Item

The remove() method removes the specified item.

### **Example**

Remove "banana":

thislist = ["apple", "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)

## Remove Specified Index

The pop() method removes the specified index.

### **Example**

Remove the second item:

thislist = ["apple", "banana", "cherry"]  
thislist.pop(1)  
print(thislist)

If you do not specify the index, the pop() method removes the last item.

### **Example**

Remove the last item:

thislist = ["apple", "banana", "cherry"]  
thislist.pop()  
print(thislist)

The del keyword also removes the specified index:

### **Example**

Remove the first item:

thislist = ["apple", "banana", "cherry"]  
del thislist[0]  
print(thislist)

The del keyword can also delete the list completely.

### **Example**

Delete the entire list:

thislist = ["apple", "banana", "cherry"]  
del thislist

## Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

### **Example**

Clear the list content:

thislist = ["apple", "banana", "cherry"]  
thislist.clear()  
print(thislist)

# **Python Tuples**

## Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

### **Example**

Create a Tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

## Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

## Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

## Allow Duplicates

Since tuples are indexed, they can have items with the same value:

### **Example**

Tuples allow duplicate values:

thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)

## Tuple Length

To determine how many items a tuple has, use the len() function:

### **Example**

Print the number of items in the tuple:

thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))

## Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

### **Example**

One item tuple, remember the comma:

thistuple = ("apple",)  
print(type(thistuple))  
  
#NOT a tuple  
thistuple = ("apple")  
print(type(thistuple))

## Tuple Items - Data Types

Tuple items can be of any data type:

### **Example**

String, int and boolean data types:

tuple1 = ("apple", "banana", "cherry")  
tuple2 = (1, 5, 7, 9, 3)  
tuple3 = (True, False, False)

A tuple can contain different data types:

### **Example**

A tuple with strings, integers and boolean values:

tuple1 = ("abc", 34, True, 40, "male")

## The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

### **Example**

Using the tuple() method to make a tuple:

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets  
print(thistuple)

## Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

### **Example**

Convert the tuple into a list to be able to change it:

x = ("apple", "banana", "cherry")  
y = list(x)  
y[1] = "kiwi"  
x = tuple(y)  
  
print(x)

## Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. **Convert into a list**: Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

### **Example**

Convert the tuple into a list, add "orange", and convert it back into a tuple:

thistuple = ("apple", "banana", "cherry")  
y = list(thistuple)  
y.append("orange")  
thistuple = tuple(y)

2. **Add tuple to a tuple**. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

### **Example**

Create a new tuple with the value "orange", and add that tuple:

thistuple = ("apple", "banana", "cherry")  
y = ("orange",)  
thistuple += y  
  
print(thistuple)

## Remove Items

**Note:** You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

### **Example**

Convert the tuple into a list, remove "apple", and convert it back into a tuple:

thistuple = ("apple", "banana", "cherry")  
y = list(thistuple)  
y.remove("apple")  
thistuple = tuple(y)

## Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

### **Example**

Packing a tuple:

fruits = ("apple", "banana", "cherry")

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

### **Example**

Unpacking a tuple:

fruits = ("apple", "banana", "cherry")  
  
(green, yellow, red) = fruits  
  
print(green)  
print(yellow)  
print(red)

**Note:** The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

# Python Sets

In Python, a **Set**is an unordered collection of data types that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.

### Creating a Set

Sets can be created by using the built-in **set()** function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a ‘comma’.

***Note:****A set cannot have mutable elements like a list or dictionary, as it is mutable.*

* Python3

|  |
| --- |
| # Python program to demonstrate  # Creation of Set in Python    # Creating a Set  set1 = set()  print("Initial blank Set: ")  print(set1)    # Creating a Set with  # the use of a String  set1 = set("GeeksForGeeks")  print("\nSet with the use of String: ")  print(set1)    # Creating a Set with  # the use of Constructor  # (Using object to Store String)  String = 'GeeksForGeeks'  set1 = set(String)  print("\nSet with the use of an Object: " )  print(set1)    # Creating a Set with  # the use of a List  set1 = set(["Geeks", "For", "Geeks"])  print("\nSet with the use of List: ")  print(set1) |

**Output**

Initial blank Set:

set()

Set with the use of String:

{'e', 'r', 'G', 's', 'F', 'k', 'o'}

Set with the use of an Object:

{'e', 'r', 'G', 's', 'F', 'k', 'o'}

Set with the use of List:

{'Geeks', 'For'}

A set contains only unique elements but at the time of set creation, multiple duplicate values can also be passed. Order of elements in a set is undefined and is unchangeable. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

* Python3

|  |
| --- |
| # Creating a Set with  # a List of Numbers  # (Having duplicate values)  set1 = set([1, 2, 4, 4, 3, 3, 3, 6, 5])  print("\nSet with the use of Numbers: ")  print(set1)    # Creating a Set with  # a mixed type of values  # (Having numbers and strings)  set1 = set([1, 2, 'Geeks', 4, 'For', 6, 'Geeks'])  print("\nSet with the use of Mixed Values")  print(set1) |

**Output**

Set with the use of Numbers:

{1, 2, 3, 4, 5, 6}

Set with the use of Mixed Values

{1, 2, 'For', 4, 6, 'Geeks'}

### Creating a set with another method

* Python3

|  |
| --- |
| # Another Method to create sets in Python3    # Set containing numbers  my\_set = {1, 2, 3}    print(my\_set)    # This code is contributed by sarajadhav12052009 |

**Output**

{1, 2, 3}

### Adding Elements to a Set

#### Using add() method

Elements can be added to the Set by using the built-in **add()** function. Only one element at a time can be added to the set by using add() method, loops are used to add multiple elements at a time with the use of add() method.

***Note:****Lists cannot be added to a set as elements because Lists are not hashable whereas Tuples can be added because tuples are immutable and hence Hashable.*

* Python3

|  |
| --- |
| # Python program to demonstrate  # Addition of elements in a Set    # Creating a Set  set1 = set()  print("Initial blank Set: ")  print(set1)    # Adding element and tuple to the Set  set1.add(8)  set1.add(9)  set1.add((6, 7))  print("\nSet after Addition of Three elements: ")  print(set1)    # Adding elements to the Set  # using Iterator  for i in range(1, 6):      set1.add(i)  print("\nSet after Addition of elements from 1-5: ")  print(set1) |

**Output**

Initial blank Set:

set()

Set after Addition of Three elements:

{8, 9, (6, 7)}

Set after Addition of elements from 1-5:

{1, 2, 3, (6, 7), 4, 5, 8, 9}

#### Using update() method

For the addition of two or more elements Update() method is used. The update() method accepts lists, strings, tuples as well as other sets as its arguments. In all of these cases, duplicate elements are avoided.

* Python3

|  |
| --- |
| # Python program to demonstrate  # Addition of elements in a Set    # Addition of elements to the Set  # using Update function  set1 = set([4, 5, (6, 7)])  set1.update([10, 11])  print("\nSet after Addition of elements using Update: ")  print(set1) |

**Output**

Set after Addition of elements using Update:

{4, 5, (6, 7), 10, 11}

### Accessing a Set

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

|  |
| --- |
| # Python program to demonstrate  # Accessing of elements in a set    # Creating a set  set1 = set(["Geeks", "For", "Geeks"])  print("\nInitial set")  print(set1)    # Accessing element using  # for loop  print("\nElements of set: ")  for i in set1:      print(i, end=" ")    # Checking the element  # using in keyword  print("Geeks" in set1) |

**Output**

Initial set

{'For', 'Geeks'}

Elements of set:

For Geeks True

### Removing elements from the Set

### Using remove() method or discard() method:

Elements can be removed from the Set by using the built-in remove() function but a KeyError arises if the element doesn’t exist in the set. To remove elements from a set without KeyError, use discard(), if the element doesn’t exist in the set, it remains unchanged.

* Python3

|  |
| --- |
| # Python program to demonstrate  # Deletion of elements in a Set    # Creating a Set  set1 = set([1, 2, 3, 4, 5, 6,              7, 8, 9, 10, 11, 12])  print("Initial Set: ")  print(set1)    # Removing elements from Set  # using Remove() method  set1.remove(5)  set1.remove(6)  print("\nSet after Removal of two elements: ")  print(set1)    # Removing elements from Set  # using Discard() method  set1.discard(8)  set1.discard(9)  print("\nSet after Discarding two elements: ")  print(set1)    # Removing elements from Set  # using iterator method  for i in range(1, 5):      set1.remove(i)  print("\nSet after Removing a range of elements: ")  print(set1) |

**Output**

Initial Set:

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Set after Removal of two elements:

{1, 2, 3, 4, 7, 8, 9, 10, 11, 12}

Set after Discarding two elements:

{1, 2, 3, 4, 7, 10, 11, 12}

Set after Removing a range of elements:

{7, 10, 11, 12}

### Using pop() method:

Pop() function can also be used to remove and return an element from the set, but it removes only the last element of the set.

***Note:****If the set is unordered then there’s no such way to determine which element is popped by using the pop() function.*

* Python3

|  |
| --- |
| # Python program to demonstrate  # Deletion of elements in a Set    # Creating a Set  set1 = set([1, 2, 3, 4, 5, 6,              7, 8, 9, 10, 11, 12])  print("Initial Set: ")  print(set1)    # Removing element from the  # Set using the pop() method  set1.pop()  print("\nSet after popping an element: ")  print(set1) |

**Output**

Initial Set:

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Set after popping an element:

{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

### Using clear() method:

To remove all the elements from the set, clear() function is used.

* Python3

|  |
| --- |
| #Creating a set  set1 = set([1,2,3,4,5])  print("\n Initial set: ")  print(set1)      # Removing all the elements from  # Set using clear() method  set1.clear()  print("\nSet after clearing all the elements: ")  print(set1) |

**Output**

Initial set:

{1, 2, 3, 4, 5}

Set after clearing all the elements:

set()

**Frozen sets** in Python are immutable objects that only support methods and operators that produce a result without affecting the frozen set or sets to which they are applied. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

If no parameters are passed, it returns an empty frozenset.

* Python3

|  |
| --- |
| # Python program to demonstrate  # working of a FrozenSet    # Creating a Set  String = ('G', 'e', 'e', 'k', 's', 'F', 'o', 'r')    Fset1 = frozenset(String)  print("The FrozenSet is: ")  print(Fset1)    # To print Empty Frozen Set  # No parameter is passed  print("\nEmpty FrozenSet: ")  print(frozenset()) |

**Output**

The FrozenSet is:

frozenset({'o', 'G', 'e', 's', 'r', 'F', 'k'})

Empty FrozenSet:

frozenset()

### Typecasting Objects into sets

* Python3

|  |
| --- |
| # Typecasting Objects in Python3 into sets    # Typecasting list into set  my\_list = [1, 2, 3, 3, 4, 5, 5, 6, 2]  my\_set = set(my\_list)  print("my\_list as a set: ", my\_set)    # Typecasting string into set  my\_str = "GeeksforGeeks"  my\_set1 = set(my\_str)  print("my\_str as a set: ", my\_set1)    # Typecasting dictionary into set  my\_dict = {1: "One", 2: "Two", 3: "Three"}  my\_set2 = set(my\_dict)  print("my\_dict as a set: ", my\_set2)    # This code is contributed by sarajadhav12052009 |

**Output**

my\_list as a set: {1, 2, 3, 4, 5, 6}

my\_str as a set: {'f', 'G', 'r', 'o', 's', 'k', 'e'}

my\_dict as a set: {1, 2, 3}

### Set Methods

| Function | Description |
| --- | --- |
| [add()](https://www.geeksforgeeks.org/set-add-python/) | Adds an element to a set |
| [remove()](https://www.geeksforgeeks.org/python-remove-discard-sets/) | Removes an element from a set. If the element is not present in the set, raise a KeyError |
| [clear()](https://www.geeksforgeeks.org/set-clear-python/) | Removes all elements form a set |
| [copy()](https://www.geeksforgeeks.org/set-copy-python/) | Returns a shallow copy of a set |
| [pop()](https://www.geeksforgeeks.org/python-set-pop/) | Removes and returns an arbitrary set element. Raise KeyError if the set is empty |
| [update()](https://www.geeksforgeeks.org/python-set-update/) | Updates a set with the union of itself and others |
| [union()](https://www.geeksforgeeks.org/union-function-python/) | Returns the union of sets in a new set |
| [difference()](https://www.geeksforgeeks.org/python-set-difference/) | Returns the difference of two or more sets as a new set |
| [difference\_update()](https://www.geeksforgeeks.org/python-set-difference_update/) | Removes all elements of another set from this set |
| [discard()](https://www.geeksforgeeks.org/python-remove-discard-sets/) | Removes an element from set if it is a member. (Do nothing if the element is not in set) |
| [intersection()](https://www.geeksforgeeks.org/intersection-function-python/) | Returns the intersection of two sets as a new set |
| intersection\_update() | Updates the set with the intersection of itself and another |
| [isdisjoint()](https://www.geeksforgeeks.org/isdisjoint-function-python/) | Returns True if two sets have a null intersection |
| [issubset()](https://www.geeksforgeeks.org/issubset-in-python/) | Returns True if another set contains this set |
| [issuperset()](https://www.geeksforgeeks.org/issuperset-in-python/) | Returns True if this set contains another set |
| [symmetric\_difference()](https://www.geeksforgeeks.org/python-set-symmetric_difference-2/) | Returns the symmetric difference of two sets as a new set |
| [symmetric\_difference\_update()](https://www.geeksforgeeks.org/python-set-symmetric_difference_update/) | Updates a set with the symmetric difference of itself and another |

# Python Dictionary

**Dictionary in Python** is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.

## Example of Dictionary in Python

Dictionary holds **key:value** pair. Key-Value is provided in the dictionary to make it more optimized.

|  |
| --- |
| Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}  print(Dict) |

**Output:**

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

## Creating a Dictionary

In [Python](https://www.geeksforgeeks.org/python-programming-language/), a dictionary can be created by placing a sequence of elements within curly **{}** braces, separated by ‘comma’. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its **Key:value**. Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be *immutable*.

**Note –**Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.

* Python3

|  |
| --- |
| # Creating a Dictionary  # with Integer Keys  Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}  print("\nDictionary with the use of Integer Keys: ")  print(Dict)    # Creating a Dictionary  # with Mixed keys  Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}  print("\nDictionary with the use of Mixed Keys: ")  print(Dict) |

**Output:**

Dictionary with the use of Integer Keys:

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with the use of Mixed Keys:

{'Name': 'Geeks', 1: [1, 2, 3, 4]}

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to curly braces{}.

|  |
| --- |
| # Creating an empty Dictionary  Dict = {}  print("Empty Dictionary: ")  print(Dict)    # Creating a Dictionary  # with dict() method  Dict = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})  print("\nDictionary with the use of dict(): ")  print(Dict)    # Creating a Dictionary  # with each item as a Pair  Dict = dict([(1, 'Geeks'), (2, 'For')])  print("\nDictionary with each item as a pair: ")  print(Dict) |

**Output:**

Empty Dictionary:

{}

Dictionary with the use of dict():

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with each item as a pair:

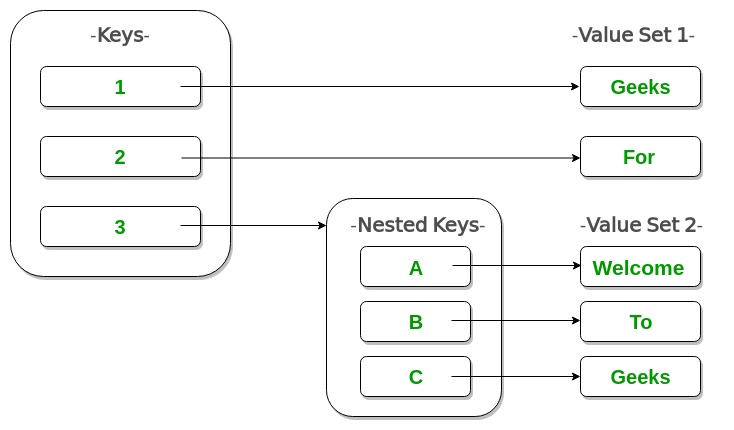
{1: 'Geeks', 2: 'For'}

#### Complexities for Creating a Dictionary:

***Time complexity:****O(len(dict))*

***Space complexity:****O(n)*

## Nested Dictionary



|  |
| --- |
| # Creating a Nested Dictionary  # as shown in the below image  Dict = {1: 'Geeks', 2: 'For',          3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}    print(Dict) |

**Output:**

{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}

## Adding elements to a Dictionary

Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = ‘Value’. Updating an existing value in a Dictionary can be done by using the built-in **update()** method. Nested key values can also be added to an existing Dictionary.

**Note-** While adding a value, if the key-value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

* Python3

|  |
| --- |
| # Creating an empty Dictionary  Dict = {}  print("Empty Dictionary: ")  print(Dict)    # Adding elements one at a time  Dict[0] = 'Geeks'  Dict[2] = 'For'  Dict[3] = 1  print("\nDictionary after adding 3 elements: ")  print(Dict)    # Adding set of values  # to a single Key  Dict['Value\_set'] = 2, 3, 4  print("\nDictionary after adding 3 elements: ")  print(Dict)    # Updating existing Key's Value  Dict[2] = 'Welcome'  print("\nUpdated key value: ")  print(Dict)    # Adding Nested Key value to Dictionary  Dict[5] = {'Nested': {'1': 'Life', '2': 'Geeks'}}  print("\nAdding a Nested Key: ")  print(Dict) |

**Output:**

Empty Dictionary:

{}

Dictionary after adding 3 elements:

{0: 'Geeks', 2: 'For', 3: 1}

Dictionary after adding 3 elements:

{0: 'Geeks', 2: 'For', 3: 1, 'Value\_set': (2, 3, 4)}

Updated key value:

{0: 'Geeks', 2: 'Welcome', 3: 1, 'Value\_set': (2, 3, 4)}

Adding a Nested Key:

{0: 'Geeks', 2: 'Welcome', 3: 1, 'Value\_set': (2, 3, 4), 5:

{'Nested': {'1': 'Life', '2': 'Geeks'}}}

#### Complexities for Adding elements in a Dictionary:

**Time complexity:** O(1)/O(n)

**Space complexity:** O(1)

## Accessing elements of a Dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets.

* Python3

|  |
| --- |
| # Python program to demonstrate  # accessing a element from a Dictionary    # Creating a Dictionary  Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}    # accessing a element using key  print("Accessing a element using key:")  print(Dict['name'])    # accessing a element using key  print("Accessing a element using key:")  print(Dict[1]) |

**Output:**

Accessing a element using key:

For

Accessing a element using key:

Geeks

There is also a method called [**get()**](https://www.geeksforgeeks.org/get-method-dictionaries-python/) that will also help in accessing the element from a dictionary. This method accepts key as argument and returns the value.

#### Complexities for Accessing elements in a Dictionary:

**Time complexity:** O(1)

**Space complexity:** O(1)

* Python3

|  |
| --- |
| # Creating a Dictionary  Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}    # accessing a element using get()  # method  print("Accessing a element using get:")  print(Dict.get(3)) |

**Output:**

Accessing a element using get:

Geeks

## Accessing an element of a nested dictionary

In order to access the value of any key in the nested dictionary, use indexing [] syntax.

* Python3

|  |
| --- |
| # Creating a Dictionary  Dict = {'Dict1': {1: 'Geeks'},          'Dict2': {'Name': 'For'}}    # Accessing element using key  print(Dict['Dict1'])  print(Dict['Dict1'][1])  print(Dict['Dict2']['Name']) |

**Output:**

{1: 'Geeks'}

Geeks

For

## ****Dictionary methods****

* [**clear()**](https://www.geeksforgeeks.org/python-dictionary-clear/)**–**Remove all the elements from the dictionary
* [**copy()**](https://www.geeksforgeeks.org/python-dictionary-copy/)**–**Returns a copy of the dictionary
* [**get()**](https://www.geeksforgeeks.org/get-method-dictionaries-python/)**–**Returns the value of specified key
* [**items()**](https://www.geeksforgeeks.org/python-dictionary-items-method/)**–**Returns a list containing a tuple for each key value pair
* [**keys()**](https://www.geeksforgeeks.org/python-dictionary-keys-method/)**–**Returns a list containing dictionary’s keys
* [**pop()**](https://www.geeksforgeeks.org/python-dictionary-pop-method/)**–** Remove the element with specified key
* [**popitem()**](https://www.geeksforgeeks.org/python-dictionary-popitem-method/)**–**Removes the last inserted key-value pair
* [**update()**](https://www.geeksforgeeks.org/python-dictionary-update-method/)**–**Updates dictionary with specified key-value pairs
* [**values()**](https://www.geeksforgeeks.org/python-dictionary-values/)**–** Returns a list of all the values of dictionary
* Python3

|  |
| --- |
| # demo for all dictionary methods  dict1 = {1: "Python", 2: "Java", 3: "Ruby", 4: "Scala"}    # copy() method  dict2 = dict1.copy()  print(dict2)    # clear() method  dict1.clear()  print(dict1)    # get() method  print(dict2.get(1))    # items() method  print(dict2.items())    # keys() method  print(dict2.keys())    # pop() method  dict2.pop(4)  print(dict2)    # popitem() method  dict2.popitem()  print(dict2)    # update() method  dict2.update({3: "Scala"})  print(dict2)    # values() method  print(dict2.values()) |

**Output:**

{1: 'Python', 2: 'Java', 3: 'Ruby', 4: 'Scala'}

{}

Python

dict\_items([(1, 'Python'), (2, 'Java'), (3, 'Ruby'), (4, 'Scala')])

dict\_keys([1, 2, 3, 4])

{1: 'Python', 2: 'Java', 3: 'Ruby'}

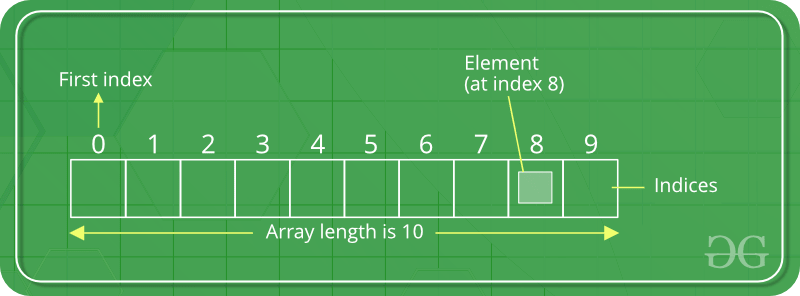
{1: 'Python', 2: 'Java'}

{1: 'Python', 2: 'Java', 3: 'Scala'}

dict\_values(['Python', 'Java', 'Scala'])

# Python Arrays

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).  
For simplicity, we can think of an array a fleet of stairs where on each step is placed a value (let’s say one of your friends). Here, you can identify the location of any of your friends by simply knowing the count of the step they are on. Array can be handled in Python by a module named **array**. They can be useful when we have to manipulate only a specific data type values. A user can treat [lists](https://www.geeksforgeeks.org/python-list/) as arrays. However, user cannot constraint the type of elements stored in a list. If you create arrays using the **array** module, all elements of the array must be of the same type. 



#### Creating a Array

Array in Python can be created by importing array module. **array(*data\_type*, *value\_list*)** is used to create an array with data type and value list specified in its arguments. 

* Python3

|  |
| --- |
| # Python program to demonstrate  # Creation of Array    # importing "array" for array creations  import array as arr    # creating an array with integer type  a = arr.array('i', [1, 2, 3])    # printing original array  print ("The new created array is : ", end =" ")  for i in range (0, 3):      print (a[i], end =" ")  print()    # creating an array with float type  b = arr.array('d', [2.5, 3.2, 3.3])    # printing original array  print ("The new created array is : ", end =" ")  for i in range (0, 3):      print (b[i], end =" ") |

**Output :**

The new created array is : 1 2 3

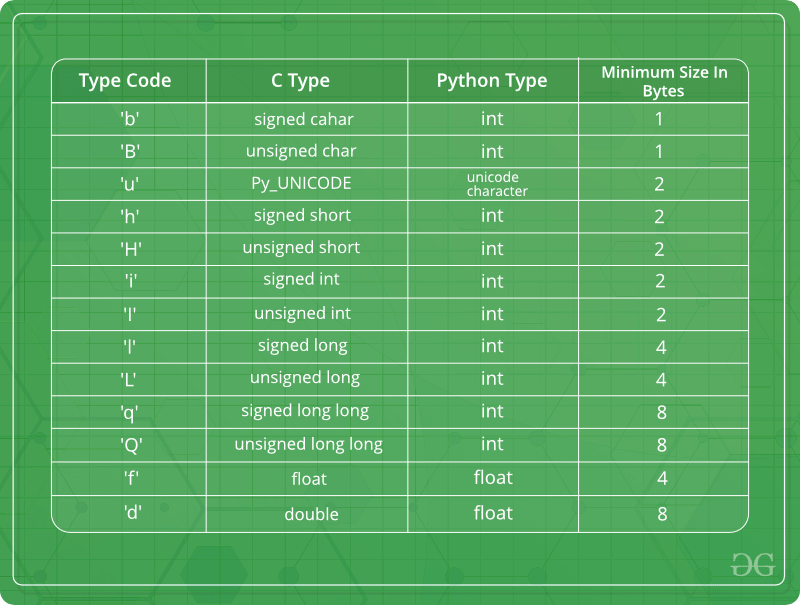
The new created array is : 2.5 3.2 3.3

### **Complexities for Creation of Arrays:**

**Time Complexity:**O(1)

**Auxiliary Space:**O(n)

Some of the data types are mentioned below which will help in creating an array of different data types. 



#### Adding Elements to a Array

Elements can be added to the Array by using built-in [insert()](https://www.geeksforgeeks.org/python-list-insert/) function. Insert is used to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array. [append()](https://www.geeksforgeeks.org/append-extend-python/) is also used to add the value mentioned in its arguments at the end of the array. 

* Python3

|  |
| --- |
| # Python program to demonstrate  # Adding Elements to a Array    # importing "array" for array creations  import array as arr    # array with int type  a = arr.array('i', [1, 2, 3])      print ("Array before insertion : ", end =" ")  for i in range (0, 3):      print (a[i], end =" ")  print()    # inserting array using  # insert() function  a.insert(1, 4)    print ("Array after insertion : ", end =" ")  for i in (a):      print (i, end =" ")  print()    # array with float type  b = arr.array('d', [2.5, 3.2, 3.3])    print ("Array before insertion : ", end =" ")  for i in range (0, 3):      print (b[i], end =" ")  print()    # adding an element using append()  b.append(4.4)    print ("Array after insertion : ", end =" ")  for i in (b):      print (i, end =" ")  print() |

**Output :**

Array before insertion : 1 2 3

Array after insertion : 1 4 2 3

Array before insertion : 2.5 3.2 3.3

Array after insertion : 2.5 3.2 3.3 4.4

### Complexities for Adding elements to the Arrays:

**Time Complexity:** O(1)/O(n) ( O(1) – for inserting elements at the end of the array, O(n) – for inserting elements at the beginning of the array and to the full array

**Auxiliary Space:** O(1)

#### Accessing elements from the Array

In order to access the array items refer to the index number. Use the index operator [ ] to access an item in a array. The index must be an integer. 

* Python3

|  |
| --- |
| # Python program to demonstrate  # accessing of element from list    # importing array module  import array as arr    # array with int type  a = arr.array('i', [1, 2, 3, 4, 5, 6])    # accessing element of array  print("Access element is: ", a[0])    # accessing element of array  print("Access element is: ", a[3])    # array with float type  b = arr.array('d', [2.5, 3.2, 3.3])    # accessing element of array  print("Access element is: ", b[1])    # accessing element of array  print("Access element is: ", b[2]) |

**Output :**

Access element is: 1

Access element is: 4

Access element is: 3.2

Access element is: 3.3

### Complexities for accessing elements in the Arrays:

**Time Complexity:** O(1)

**Auxiliary Space:** O(1)

#### Removing Elements from the Array

Elements can be removed from the array by using built-in [remove()](https://www.geeksforgeeks.org/python-list-remove/) function but an Error arises if element doesn’t exist in the set. Remove() method only removes one element at a time, to remove range of elements, iterator is used. [pop()](https://www.geeksforgeeks.org/python-list-pop/) function can also be used to remove and return an element from the array, but by default it removes only the last element of the array, to remove element from a specific position of the array, index of the element is passed as an argument to the pop() method.  
**Note –**Remove method in List will only remove the first occurrence of the searched element. 

* Python3

|  |
| --- |
| # Python program to demonstrate  # Removal of elements in a Array    # importing "array" for array operations  import array    # initializing array with array values  # initializes array with signed integers  arr = array.array('i', [1, 2, 3, 1, 5])    # printing original array  print ("The new created array is : ", end ="")  for i in range (0, 5):      print (arr[i], end =" ")    print ("\r")    # using pop() to remove element at 2nd position  print ("The popped element is : ", end ="")  print (arr.pop(2))    # printing array after popping  print ("The array after popping is : ", end ="")  for i in range (0, 4):      print (arr[i], end =" ")    print("\r")    # using remove() to remove 1st occurrence of 1  arr.remove(1)    # printing array after removing  print ("The array after removing is : ", end ="")  for i in range (0, 3):      print (arr[i], end =" ") |

**Output:**

The new created array is : 1 2 3 1 5

The popped element is : 3

The array after popping is : 1 2 1 5

The array after removing is : 2 1 5

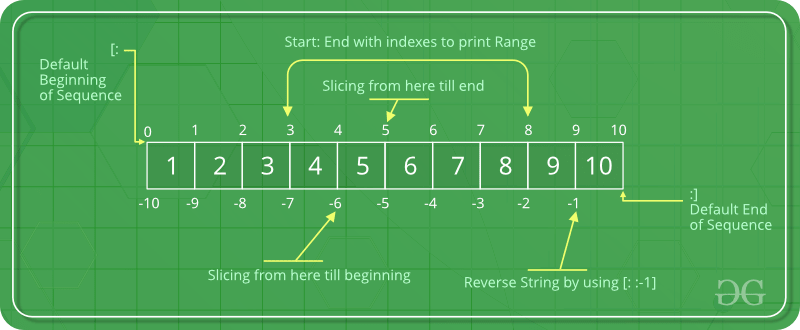
### Complexities for Removing elements in the Arrays:

**Time Complexity:** O(1)/O(n) ( O(1) – for removing elements at the end of the array, O(n) – for removing elements at the beginning of the array and to the full array

**Auxiliary Space:** O(1)

#### Slicing of a Array

In Python array, there are multiple ways to print the whole array with all the elements, but to print a specific range of elements from the array, we use [Slice operation](https://www.geeksforgeeks.org/python-list-comprehension-and-slicing/). Slice operation is performed on array with the use of colon(:). To print elements from beginning to a range use [:Index], to print elements from end use [:-Index], to print elements from specific Index till the end use [Index:], to print elements within a range, use [Start Index:End Index] and to print whole List with the use of slicing operation, use [:]. Further, to print whole array in reverse order, use [::-1]. 



* Python3

|  |
| --- |
| # Python program to demonstrate  # slicing of elements in a Array    # importing array module  import array as arr    # creating a list  l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]    a = arr.array('i', l)  print("Initial Array: ")  for i in (a):      print(i, end =" ")    # Print elements of a range  # using Slice operation  Sliced\_array = a[3:8]  print("\nSlicing elements in a range 3-8: ")  print(Sliced\_array)    # Print elements from a  # pre-defined point to end  Sliced\_array = a[5:]  print("\nElements sliced from 5th "        "element till the end: ")  print(Sliced\_array)    # Printing elements from  # beginning till end  Sliced\_array = a[:]  print("\nPrinting all elements using slice operation: ")  print(Sliced\_array) |

**Output**

Initial Array:

1 2 3 4 5 6 7 8 9 10

Slicing elements in a range 3-8:

array('i', [4, 5, 6, 7, 8])

Elements sliced from 5th element till the end:

array('i', [6, 7, 8, 9, 10])

Printing all elements using slice operation:

array('i', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

#### Searching element in a Array

In order to search an element in the array we use a python in-built [index()](https://www.geeksforgeeks.org/python-list-index/) method. This function returns the index of the first occurrence of value mentioned in arguments. 

* Python3

|  |
| --- |
| # Python code to demonstrate  # searching an element in array      # importing array module  import array    # initializing array with array values  # initializes array with signed integers  arr = array.array('i', [1, 2, 3, 1, 2, 5])    # printing original array  print ("The new created array is : ", end ="")  for i in range (0, 6):      print (arr[i], end =" ")    print ("\r")    # using index() to print index of 1st occurrence of 2  print ("The index of 1st occurrence of 2 is : ", end ="")  print (arr.index(2))    # using index() to print index of 1st occurrence of 1  print ("The index of 1st occurrence of 1 is : ", end ="")  print (arr.index(1)) |

**Output:** 

The new created array is : 1 2 3 1 2 5

The index of 1st occurrence of 2 is : 1

The index of 1st occurrence of 1 is : 0

### Complexities for searching elements in the Arrays:

**Time Complexity:** O(n)

**Auxiliary Space:**O(1)

#### Updating Elements in a Array

In order to update an element in the array we simply reassign a new value to the desired index we want to update. 

* Python3

|  |
| --- |
| # Python code to demonstrate  # how to update an element in array    # importing array module  import array    # initializing array with array values  # initializes array with signed integers  arr = array.array('i', [1, 2, 3, 1, 2, 5])    # printing original array  print ("Array before updation : ", end ="")  for i in range (0, 6):      print (arr[i], end =" ")    print ("\r")    # updating a element in a array  arr[2] = 6  print("Array after updation : ", end ="")  for i in range (0, 6):      print (arr[i], end =" ")  print()    # updating a element in a array  arr[4] = 8  print("Array after updation : ", end ="")  for i in range (0, 6):      print (arr[i], end =" ") |

**Output:**

Array before updation : 1 2 3 1 2 5

Array after updation : 1 2 6 1 2 5

Array after updation : 1 2 6 1 8 5

### Complexities for updating elements in the Arrays:

**Time Complexity:** O(n)

**Auxiliary Space:** O(1)

References:

https://www.geeksforgeeks.org/python-sets/?ref=lbp

https://www.geeksforgeeks.org/python-dictionary/?ref=lbp