**Explain Lists in Python.Explain how to access lists and its operation.**

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:

**Creating Lists:**

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

Output: [‘apple’, ‘banana’, ‘cherry’]

**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:

**List Items - Data Types**

List items can be of any data type:

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

A list can contain different data types:

list1 = ["abc", 34, True, 40, "male"]

**type():**

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

Ex:

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

Output: <class ‘list’>

**List Length:**

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

Print the number of items in the list:

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

Output: 3

**Accessing values in list:**

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

Print the second item of the list:

Ex:

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

Output:banana

**Negative Indexing:**

Negative indexing means start from the end

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

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.

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

Output:['cherry', 'orange', '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:

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

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

**Check if Item Exists:**

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

Check if "apple" is present in the list:

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

Output:  
Yes, 'apple' is in the fruits list

**List Operations:**

**Change Item Value:**

To change the value of a specific item, refer to the index number:

Change the second item:

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

Output: ['apple', 'blackcurrant', 'cherry']

**Change a Range of Item Values:**

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]  
thislist[1:3] = ["blackcurrant", "watermelon"]  
print(thislist)

Output: ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']

**Insert Items:**

To insert a new list item, without replacing any of the existing values, we can use the **insert()** method.

The insert() method inserts an item at the specified index:

Insert "watermelon" as the third item:

thislist = ["apple", "banana", "cherry"]  
thislist.insert(2, "watermelon")  
print(thislist)

Output: ['apple', 'banana', 'watermelon', 'cherry']

**Add List Items:**

For x in thislist Items

To add an item to the end of the list, use the **append()** method:

Using the append() method to append an item:

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

Output: ['apple', 'banana', 'cherry', 'orange']

**Extend List:**

To append elements from another list to the current list, use the **extend()** method.

Add the elements of tropical to thislist:

thislist = ["apple", "banana", "cherry"]  
tropical = ["mango", "pineapple", "papaya"]  
thislist.extend(tropical)  
print(thislist)

Output: ['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

**Add Any Iterable:**

The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

Add elements of a tuple to a list:

thislist = ["apple", "banana", "cherry"]  
thistuple = ("kiwi", "orange")  
thislist.extend(thistuple)  
print(thislist)

Output: ['apple', 'banana', 'cherry', 'kiwi', 'orange']

**Remove List Items:**

The **remove()** method removes the specified item.

Remove "banana":

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

Output: ['apple', 'cherry']

**Remove Specified Index:**

The **pop()** method removes the specified index.

Remove the second item:

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

Output: ['apple', 'cherry']

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

Remove the last item:

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

Output: ['apple', 'banana']

**The del keyword** can also delete the list completely.

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.

Clear the list content:

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

Output:[]

**Loop Through a List:[travesing a list]**

You can loop through the list items by **using a for loop**:

Print all items in the list, one by one:

thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
  print(x)

output:

apple

banana

cherry

Print all items, **using a while loop** to go through all the index numbers

thislist = ["apple", "banana", "cherry"]  
i = 0  
while i < len(thislist):  
  print(thislist[i])  
  i = i + 1

output:

apple

banana

cherry

**Looping Using List Comprehension:**

List Comprehension offers the shortest syntax for looping through lists:

A short hand for loop that will print all items in a list:

thislist = ["apple", "banana", "cherry"]  
# print(x) for x in thislist

output:

apple

banana

cherry

**using the range() function to create an iterable:**

newlist = [x for x in range(10)]

output: [0,1,2,3,4,5,6,7,8,9]

**Sort Lists:**

List objects have a **sort()** method that will sort the list alphanumerically, ascending, by default:

Sort the list alphabetically:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]  
thislist.sort()  
print(thislist)

output: ['banana', 'kiwi', 'mango', 'orange', 'pineapple']

**Sort Descending:**

To sort descending, use the keyword argument reverse = True:

Sort the list descending:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]  
thislist.sort(reverse = True)  
print(thislist)

output: ['pineapple', 'orange', 'mango', 'kiwi', 'banana']

**Copy a List (list aliasing) (cloning a list):**

Make a copy of a list with the **copy()** method:

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

output: ['apple', 'banana', 'cherry']

**Make a copy of a list with the list() method:**

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

output: ['apple', 'banana', 'cherry']

**Join Two Lists:**

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Join two list:

list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]  
list3 = list1 + list2  
print(list3)

output: ['a', 'b', 'c', 1, 2, 3]

**Python slice() Function:**

The **slice()** function returns a slice object.

A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g. slice only every other item.

**Syntax : slice(start, end, step)**

a = ("a", "b", "c", "d", "e", "f", "g", "h")  
x = slice(2)  
print(a[x])

output: (’a’,’b’)

a = ("a", "b", "c", "d", "e", "f", "g", "h")

x = slice(3, 5)

print(a[x])

output: (’d’,’e’)

**Passing a List as an Argument:**

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

def my\_function(food):  
   for x in food:  
     print(x)  
fruits = ["apple", "banana", "cherry"]  
  
my\_function(fruits)

output: apple

banana

cherry

**List Methods:**

Python has a set of built-in methods that you can use on lists.

|  |  |
| --- | --- |
| Method | Description |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

**What in tuple?Explain how to access tuple.describe the operations on 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-** items have a defined order, and that order will not change.

**Unchangeable -** we cannot change, add or remove items after the tuple has been created.

**allow duplicates.**

Tuples are written with round brackets.

**Create a Tuple:**

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

output: (‘apple’,’banana’,’cherry’)

**Access Tuple Items:**

You can access tuple items by referring to the index number, inside square brackets:

**Print the second item in the tuple:**

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

output:banana

**Negative Indexing:**

Negative indexing means start from the end.

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

Print the last item of the tuple:

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

output: Cherry

**Range of Indexes**

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:5]) # Return the third, fourth, and fifth item:

print(thistuple[:4]) # the range will go on to the end of the list:

print(thistuple[2:]) # from the beginning to,

**Check if Item Exists:**

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

Check if "apple" is present in the tuple:

thistuple = ("apple", "banana", "cherry")  
if "apple" in thistuple:  
   print("Yes, 'apple' is in the fruits tuple")

**Change Tuple Values:**

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.

But there are some workarounds.

You can convert the tuple into a list, change the list, and convert the list back into a tuple.

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

output: ("apple", "kiwi", "cherry")

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.

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)

output: ('apple', 'banana', 'cherry', 'orange')

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:

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

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

output: ('apple', 'banana', 'cherry', 'orange')

**Remove Items:**

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:

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)

output: ('banana', 'cherry')

**you can delete the tuple completely:**

The**del keyword** can delete the tuple completely:

thistuple = ("apple", "banana", "cherry")  
del thistuple  
print(thistuple) #this will raise an error because the tuple no longer exists

**Unpacking a Tuple:**

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

*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":*

Unpacking a tuple:

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

output: apple

banana

cherry

**Using Asterisk\*:**

If the number of variables is less than the number of values, you can add an \* to the variable name and the values will be assigned to the variable as a list:

Assign the rest of the values as a list called "red":

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")  
(green, yellow, \*red) = fruits  
print(green)  
print(yellow)  
print(red)

output:

apple

banana

['cherry', 'strawberry', 'raspberry']

**Loop Through a Tuple:[Traversing a Tuple]**

You can loop through the tuple items by **using a for loop**.

Iterate through the items and print the values:

thistuple = ("apple", "banana", "cherry")  
for x in thistuple:  
  print(x)

output: apple

banana

cherry

**Join Two Tuples:**

To join two or more tuples you can use the + operator:

Join two tuples:

tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)  
tuple3 = tuple1 + tuple2  
print(tuple3)

output: ('a', 'b', 'c', 1, 2, 3)

**Multiply Tuples:**

If you want to multiply the content of a tuple a given number of times, you can use **the \* operator**.

Multiply the fruits tuple by 2:

fruits = ("apple", "banana", "cherry")  
mytuple = fruits \* 2  
print(mytuple)

output: ('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')

**Tuple Methods:**

Python has two built-in methods that you can use on tuples.

|  |  |
| --- | --- |
| Method | Description |
| [count()](https://www.w3schools.com/python/ref_tuple_count.asp) | Returns the number of times a specified value occurs in a tuple |
| [index()](https://www.w3schools.com/python/ref_tuple_index.asp) | Searches the tuple for a specified value and returns the position of where it was found |

**Define Dictionary.Explain accessing dictionary and Operations on dictionary.**

Dictionaries are used to store data values in **key:value pairs.**

A dictionary is a **collection** ,which is

* + - ordered,
    - changeable
    - do not allow duplicates.

Dictionaries are written with curly brackets

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict["brand"])

output: Ford

Duplicate values will overwrite existing values(no duplicates)

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964,  
  "year": 2020  
}  
print(thisdict)

output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

**Dictionary Length:**

Print the number of items in the dictionary:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964,

"year": 2020

}

print(len(thisdict))

output: 3

**Dictionary Items - Data Types**

The values in dictionary items can be of any data type:

thisdict = {  
  "brand": "Ford",  
  "electric": False,  
  "year": 1964,  
  "colors": ["red", "white", "blue"]  
}

output: {'brand': 'Ford', 'electric': False, 'year': 1964, 'colors': ['red', 'white', 'blue']}

**type():**

dictionaries are defined as objects with the **data type 'dict'.**

print(type(thisdict))

output: <class ‘dict’>

**Accessing Items:**

You can access the items of a dictionary by referring to its key name, inside square brackets

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
x = thisdict["model"]

output: Mustang

**get():**

this method is used to access the elements present in the dictionary.

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = thisdict.get("model")

print(x)

output: Mustang

**Get Keys:**

The **keys()** method will return a list of all the keys in the dictionary.

Add a new item to the original dictionary

car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = car.keys()

print(x) #before the change

car["color"] = "white" # Add a new item to the original dictionary

print(x) #after the change

output : dict\_keys(['brand', 'model', 'year'])

dict\_keys(['brand', 'model', 'year', 'color'])

**Get Values:**

The **values()** method will return a list of all the values in the dictionary.

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = thisdict.values()

print(x)

output: dict\_values(['Ford', 'Mustang', 1964])

**Get Items:**

The **items()** method will return each item in a dictionary.

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = thisdict.items()

print(x)

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

**Check if Key Exists:**

To determine if a specified key is present in a dictionary use the in keyword:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

**Change Values:**

You can change the value of a specific item by referring to its key name:

Change the "year" to 2018:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["year"] = 2018

output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2018}

**Adding Items:**

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)

output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

**Update Dictionary:**

The**update()** method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.update({"year": 2020})

output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

**Removing Items:**

There are several methods to remove items from a dictionary:

The **pop()** method removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

output: {'brand': 'Ford', 'year': 1964}

The **popitem()** method removes the last inserted item

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.popitem()  
print(thisdict)

output: {'brand': 'Ford', 'model': 'Mustang'}

The **del keyword** removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict["model"]  
print(thisdict)

output: {'brand': 'Ford', 'year': 1964}

The **del keyword** can also delete the dictionary completely:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict  
print(thisdict) #this will cause an error because "thisdict" no longer exists.

The **clear()** method empties the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.clear()  
print(thisdict)

output: {}

**Loop Through a Dictionary:[Traversing a Dictionary]**

You can loop through a dictionary by **using a for loop**.

When looping through a dictionary, the return value are the keys of the dictionary,

Print all key names in the dictionary, one by one:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

for x in thisdict:

print(x)

output: brand

model

year

You can also use the**values()** method to return values of a dictionary:

for x in thisdict.values():  
  print(x)

output: Ford

Mustang

1964

You can use the **keys()** method to return the keys of a dictionary:

for x in thisdict.keys():  
  print(x)

output: brand

model

year

Loop through both *keys* and *values*, by using the**items()**method:

for x, y in thisdict.items():  
  print(x, y)

output: brand Ford

model Mustang

year 1964

**Copy a Dictionary:**

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method **copy().**

Make a copy of a dictionary with the copy() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)

output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

**Make a copy of a dictionary with the dict() function:**

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)

**Nested Dictionaries:**

A dictionary can contain dictionaries, this is called nested dictionaries.

Create a dictionary that contain three dictionaries:

myfamily = {  
  "child1" : {  
    "name" : "Emil",  
    "year" : 2004  
  },  
  "child2" : {  
    "name" : "Tobias",  
    "year" : 2007  
  },  
  "child3" : {  
    "name" : "Linus",  
    "year" : 2011  
  }  
}

Or, if you want to add three dictionaries into a new dictionary:

child1 = {  
  "name" : "Emil",  
  "year" : 2004  
}  
child2 = {  
  "name" : "Tobias",  
  "year" : 2007  
}  
child3 = {  
  "name" : "Linus",  
  "year" : 2011  
}  
  
myfamily = {  
  "child1" : child1,  
  "child2" : child2,  
  "child3" : child3  
}

output: {'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}

**Dictionary Methods**

Python has a set of built-in methods that you can use on dictionaries.

|  |  |
| --- | --- |
| Method | Description |
| [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary |
| [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary |
| [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and value |
| [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key |
| [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair |
| [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys |
| [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key |
| [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair |
| [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs |
| [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary |