Python List Methods

Methods	Descriptions
append()	adds an element to the end of the list
extend()	adds all elements of a list to another list
insert()	inserts an item at the defined index
remove()	removes an item from the list
pop()	returns and removes an element at the given index
clear()	removes all items from the list
index()	returns the index of the first matched item
count()	returns the count of the number of items passed as an argument
sort()	sort items in a list in ascending order
reverse()	reverse the order of items in the list
copy()	returns a shallow copy of the list

Advantages of Tuple over List

Since tuples are quite similar to lists, both of them are used in similar situations. However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:

- We generally use tuples for heterogeneous (different) data types and lists for homogeneous (similar) data types.
- Since tuples are immutable, iterating through a tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Python Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set

discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns `True` if two sets have a null intersection
issubset()	Returns `True` if another set contains this set
issuperset()	Returns `True` if this set contains another set
pop()	Removes and returns an arbitrary set element. Raises `KeyError` if the set is empty
remove()	Removes an element from the set. If the element is not a member, raises a `KeyError`
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

Built-in Functions with Set

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

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

Python Dictionary Methods

Method	Description
clear()	Removes all items from the dictionary.

copy()	Returns a shallow copy of the dictionary.
fromkeys(seq[, v])	Returns a new dictionary with keys from `seq` and value equal to `v` (defaults to `None`).
get(key[,d])	Returns the value of the `key`. If the `key` does not exist, returns `d` (defaults to `None`).
items()	Return a new object of the dictionary's items in (key, value) format.
keys()	Returns a new object of the dictionary's keys.
pop(key[,d])	Removes the item with the `key` and returns its value or `d` if `key` is not found. If `d` is not provided and the `key` is not found, it raises `KeyError`.
popitem()	Removes and returns an arbitrary item (**key, value**). Raises `KeyError` if the dictionary is empty.
setdefault(key[,d])	Returns the corresponding value if the `key` is in the dictionary. If not, inserts the `key` with a value of `d` and returns `d` (defaults to `None`).
update([other])	Updates the dictionary with the key/value pairs from `other`, overwriting existing keys.
values()	Returns a new object of the dictionary's values

Dictionary Built-in Functions

Built-in functions like `all()`, `any()`, `len()`, `cmp()`, `sorted()`, etc. are commonly used with dictionaries to perform different tasks.

Function	Description

all()	Return `True` if all keys of the dictionary are True (or if the dictionary is empty).
any()	Return `True` if any key of the dictionary is true. If the dictionary is empty, return `False`.
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries. (Not available in Python 3)
sorted()	Return a new sorted list of keys in the dictionary.