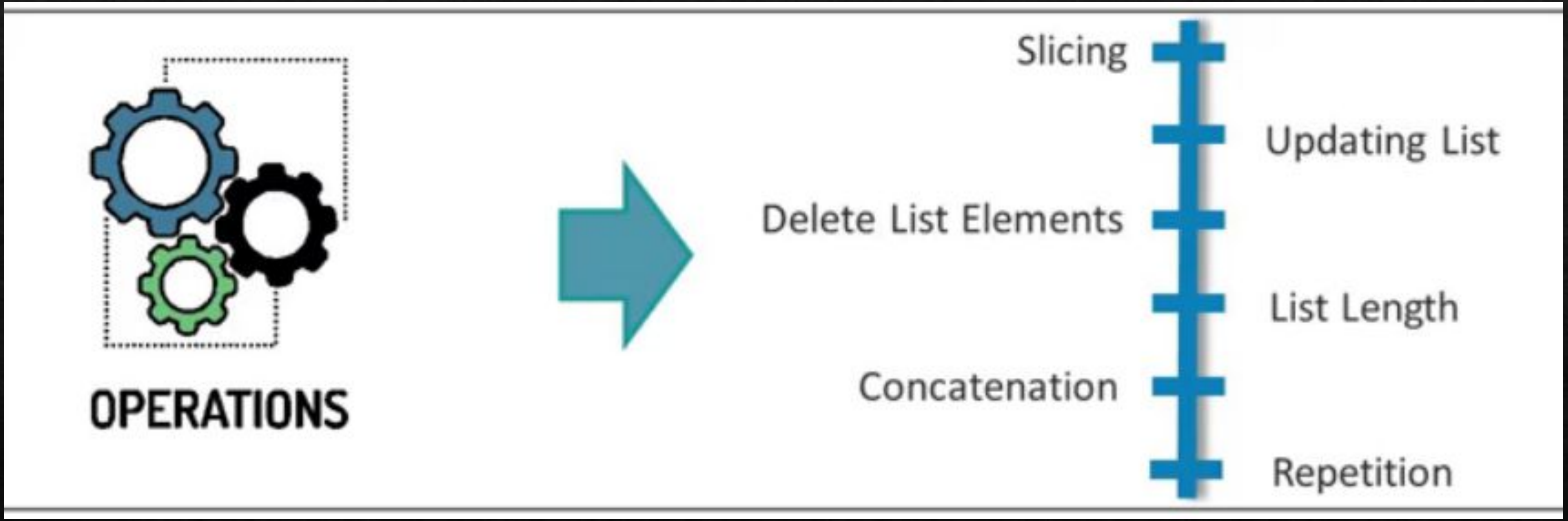
Introduction to Python

Noble Xavier

Sequence Operations

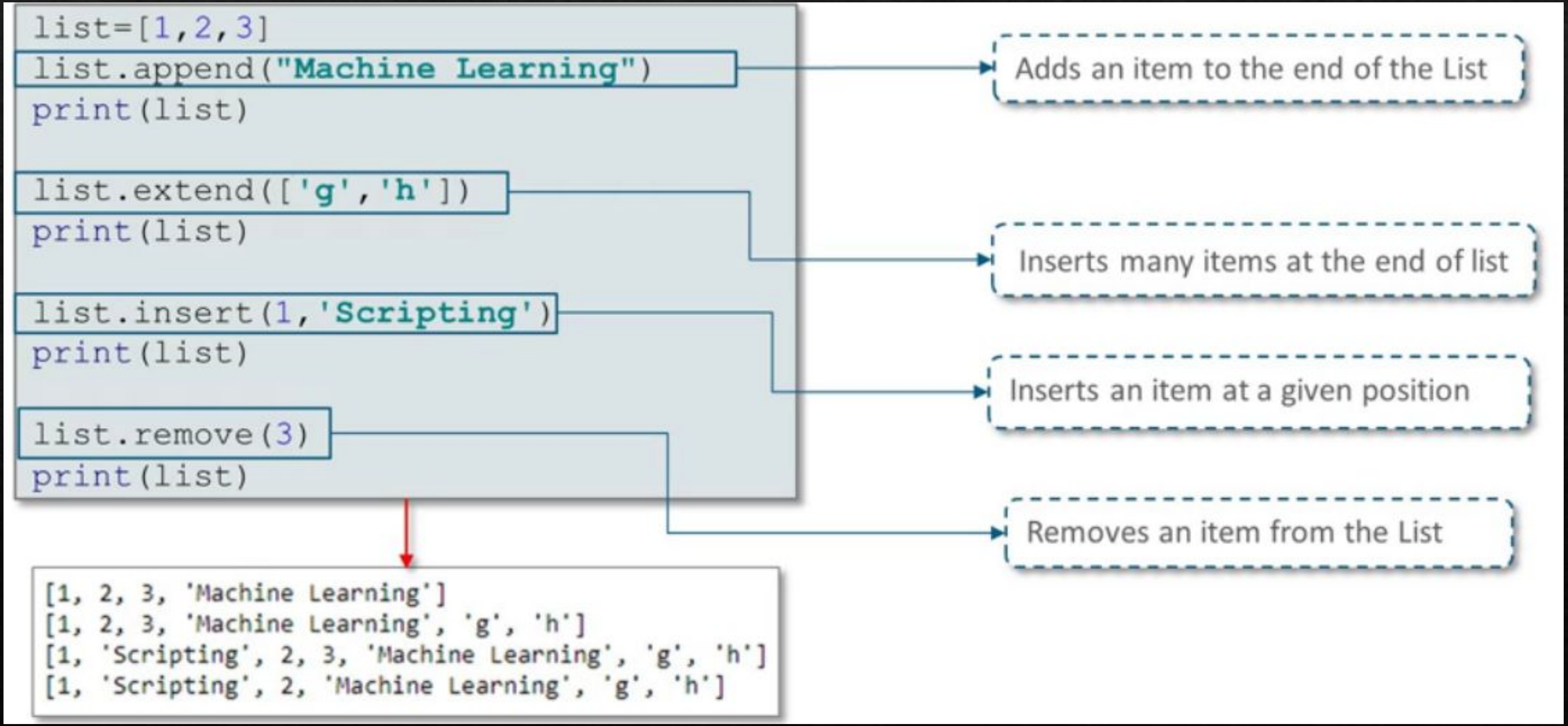
- Concatenation
- Repetition
- Membership testing
- Slicing
- Indexing

List



OPERATIONS
Slicing
Updating List
Delete List Elements
List Length
Concatenation
Repetition

List Operations

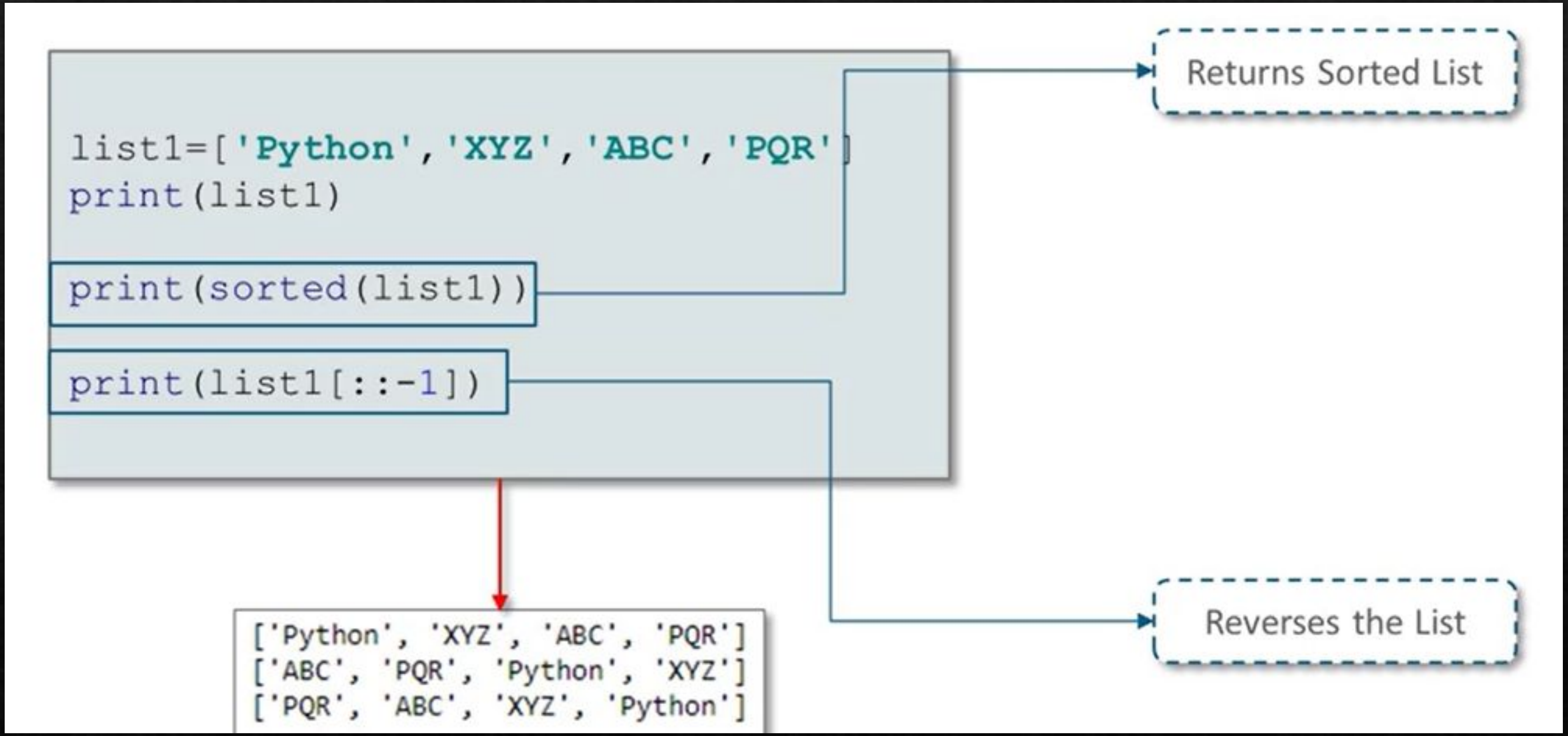


```
list=[1,2,3]
list.append("Machine Learning")
print(list)

list.extend(['g','h'])
print(list)

list.insert(1,'Scripting')
print(list)

list.remove(3)
print(list)
```

Adds an item to the end of the List
Inserts many items at the end of list
Inserts an item at a given position
Removes an item from the List

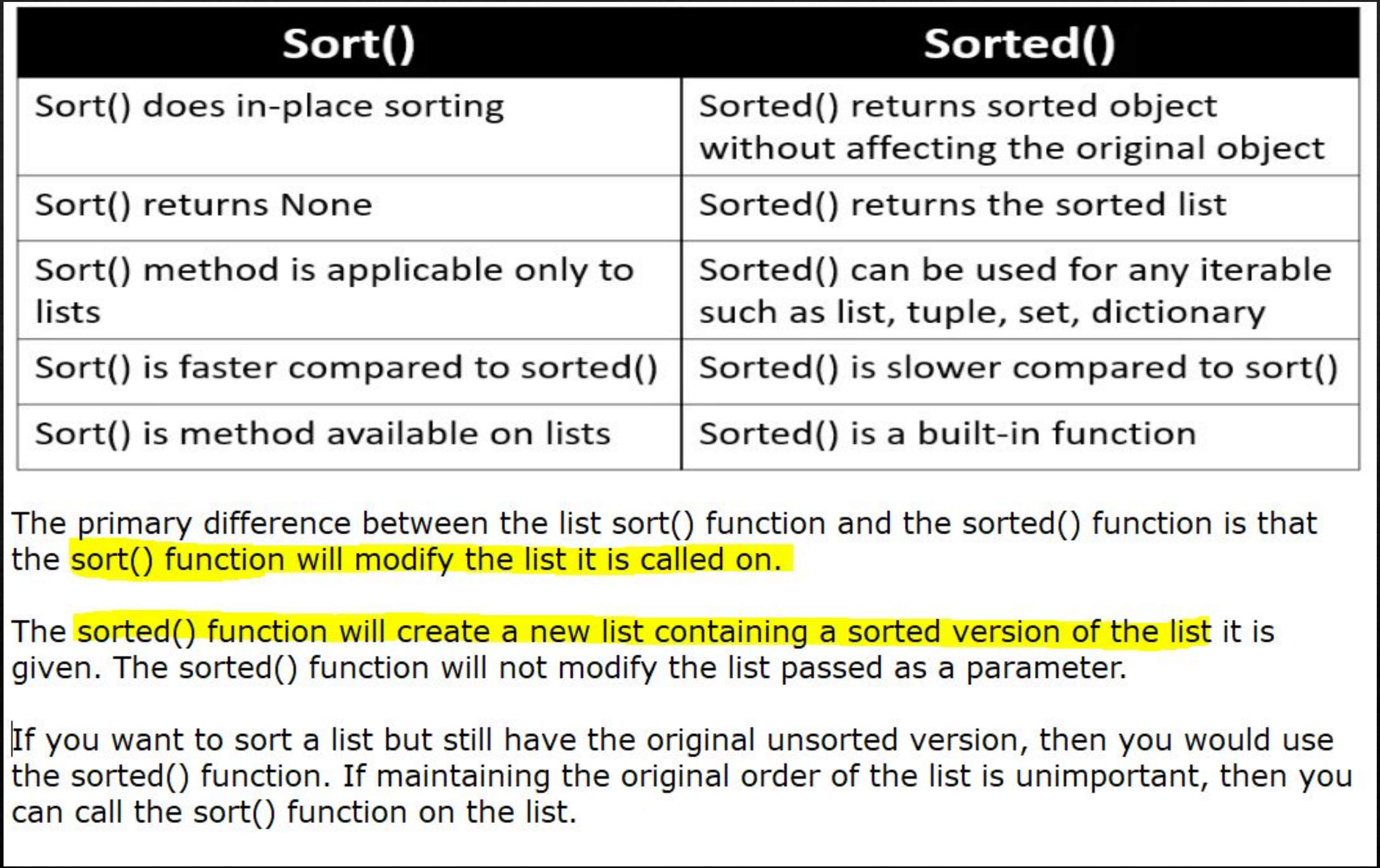
```
[1, 2, 3, 'Machine Learning']
[1, 2, 3, 'Machine Learning', 'g', 'h']
[1, 'Scripting', 2, 3, 'Machine Learning', 'g', 'h']
[1, 'Scripting', 2, 'Machine Learning', 'g', 'h']
```

List Operations



```
list1=['Python','XYZ','ABC','PQR']
print(list1)

print(sorted(list1))

print(list1[::-1])
```

```
['Python', 'XYZ', 'ABC', 'PQR']
['ABC', 'PQR', 'Python', 'XYZ']
['PQR', 'ABC', 'XYZ', 'Python']
```



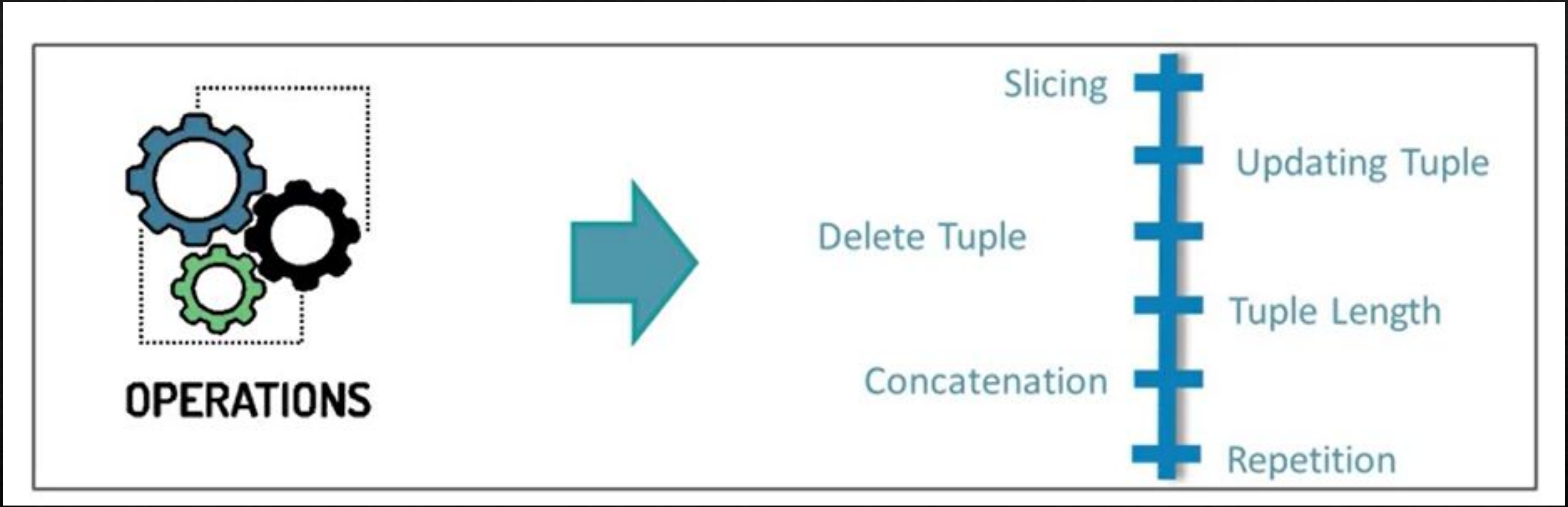
Sort()	Sorted()
Sort() does in-place sorting	Sorted() returns sorted object without affecting the original object
Sort() returns None	Sorted() returns the sorted list
Sort() method is applicable only to lists	Sorted() can be used for any iterable such as list, tuple, set, dictionary
Sort() is faster compared to sorted()	Sorted() is slower compared to sort()
Sort() is method available on lists	Sorted() is a built-in function

The primary difference between the list sort() function and the sorted() function is that the sort() function will modify the list it is called on.

The sorted() function will create a new list containing a sorted version of the list it is given. The sorted() function will not modify the list passed as a parameter.

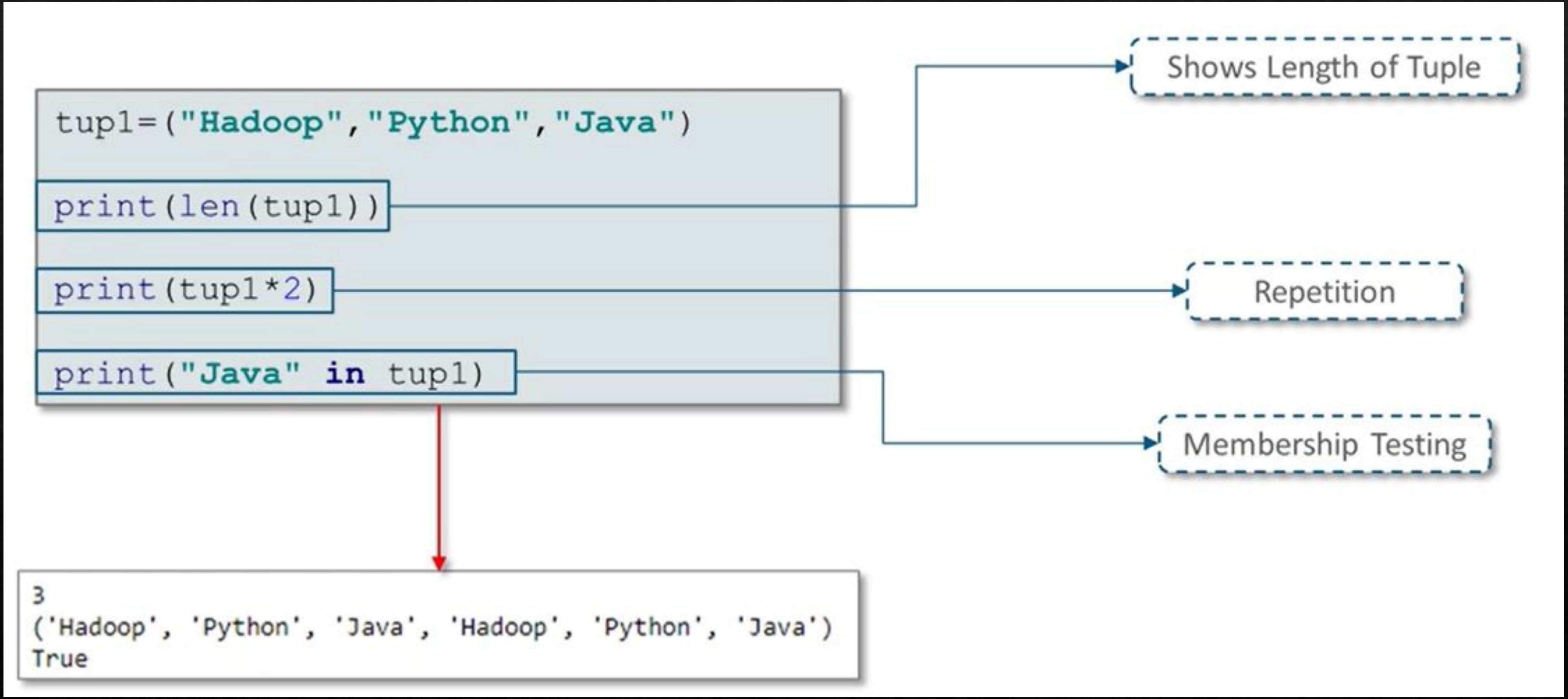
If you want to sort a list but still have the original unsorted version, then you would use the sorted() function. If maintaining the original order of the list is unimportant, then you can call the sort() function on the list.

Tuples



Adv:
- Faster
- Values can’t be changed

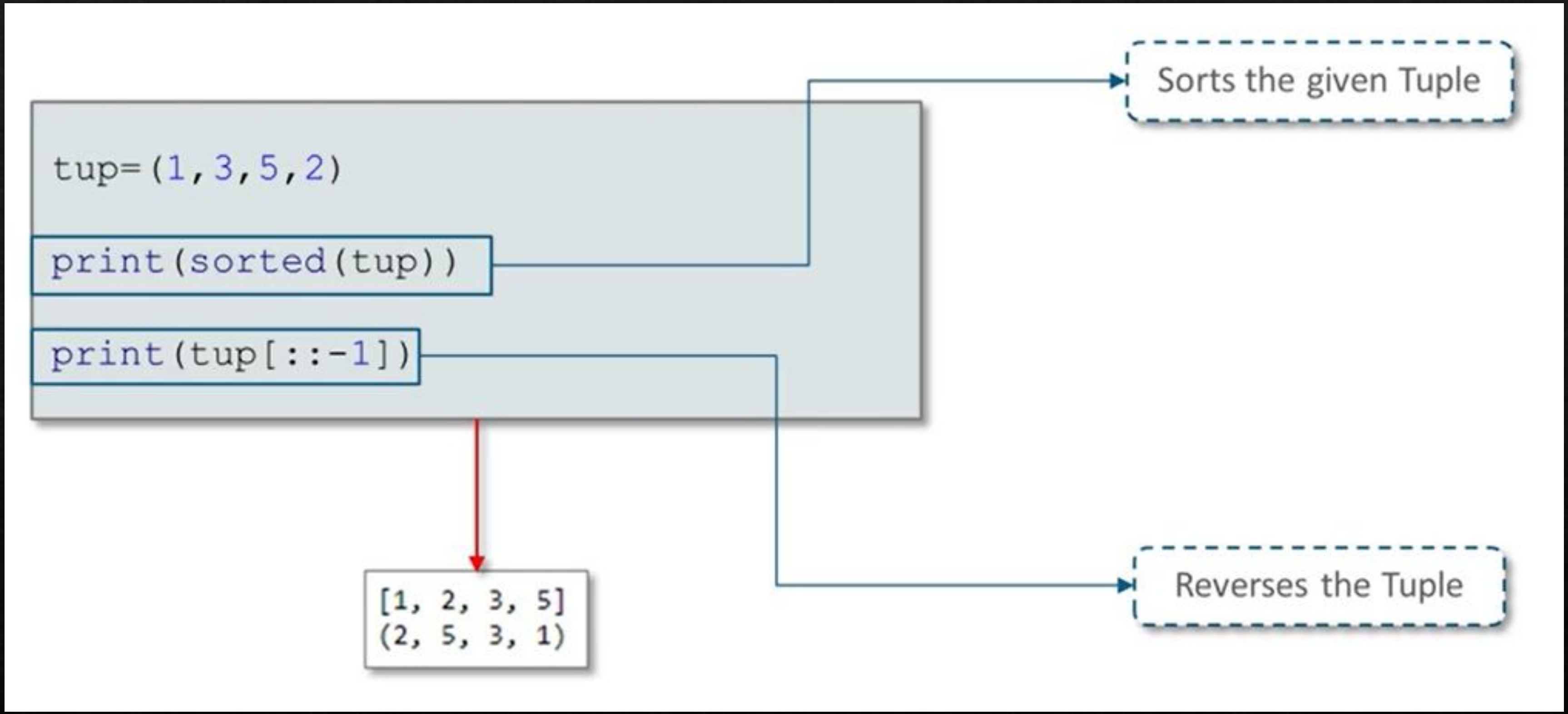
Tuple



```
tup1=("Hadoop","Python","Java")

print(len(tup1))

print(tup1*2)

print("Java" in tup1)
```

```
3
('Hadoop', 'Python', 'Java', 'Hadoop', 'Python', 'Java')
True
```

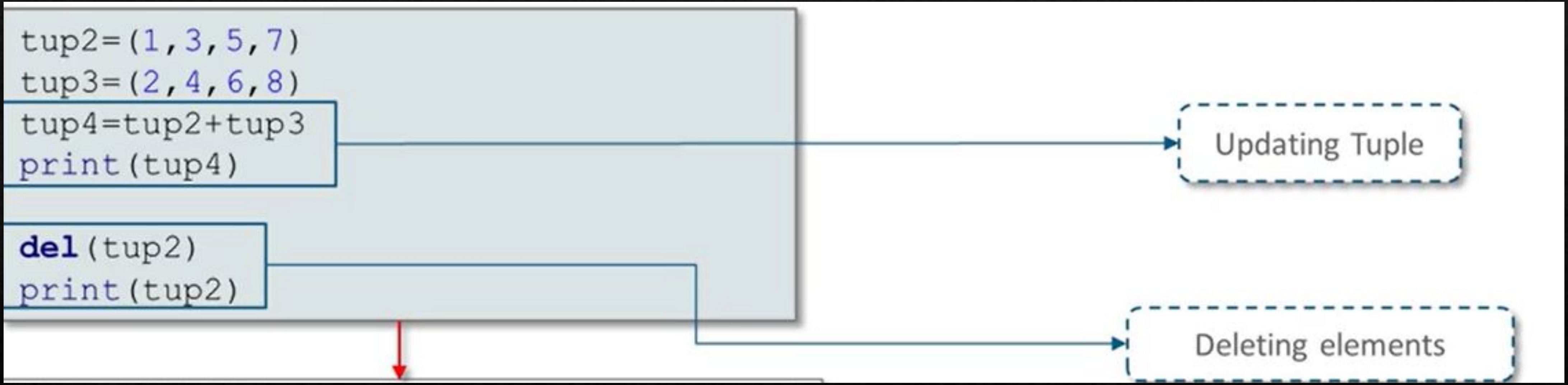
Tuple



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

print(sorted(tup))

print(tup[::-1])
```

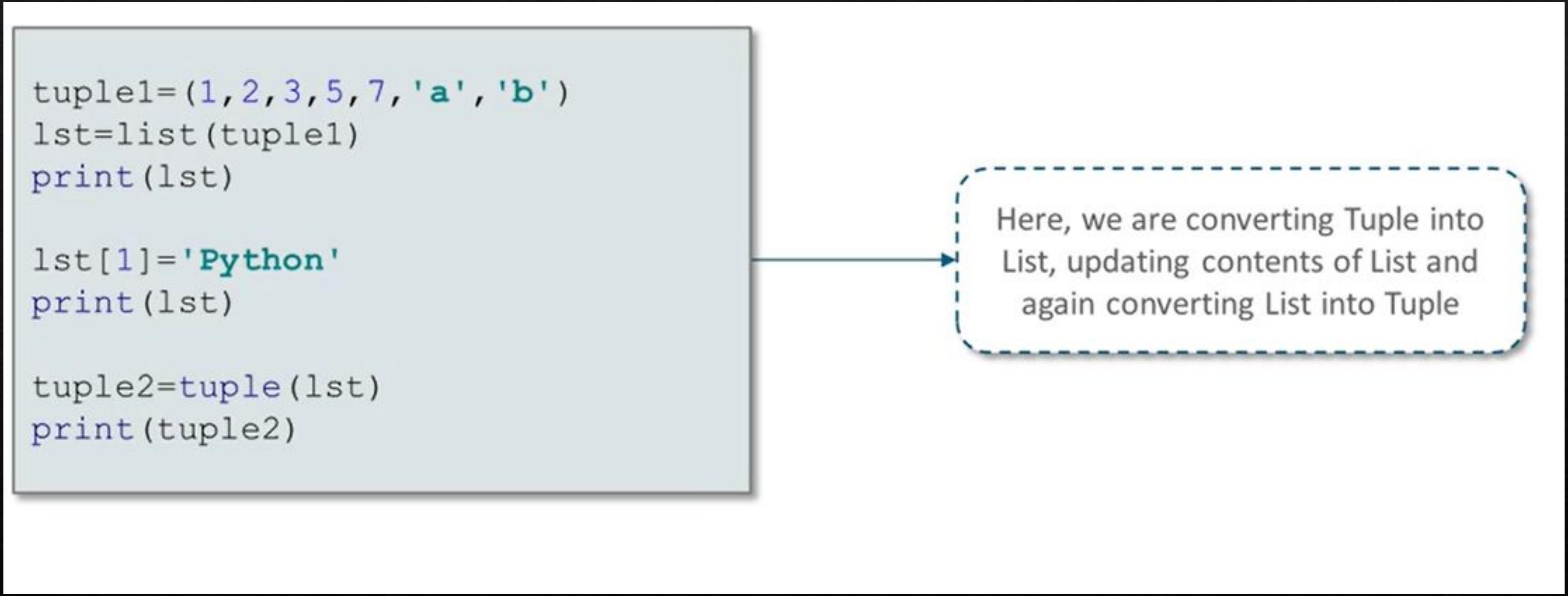
```
[1, 2, 3, 5]
(2, 5, 3, 1)
```

Tuple



```
tup2=(1,3,5,7)
tup3=(2,4,6,8)
tup4=tup2+tup3
print(tup4)

del(tup2)
print(tup2)
```

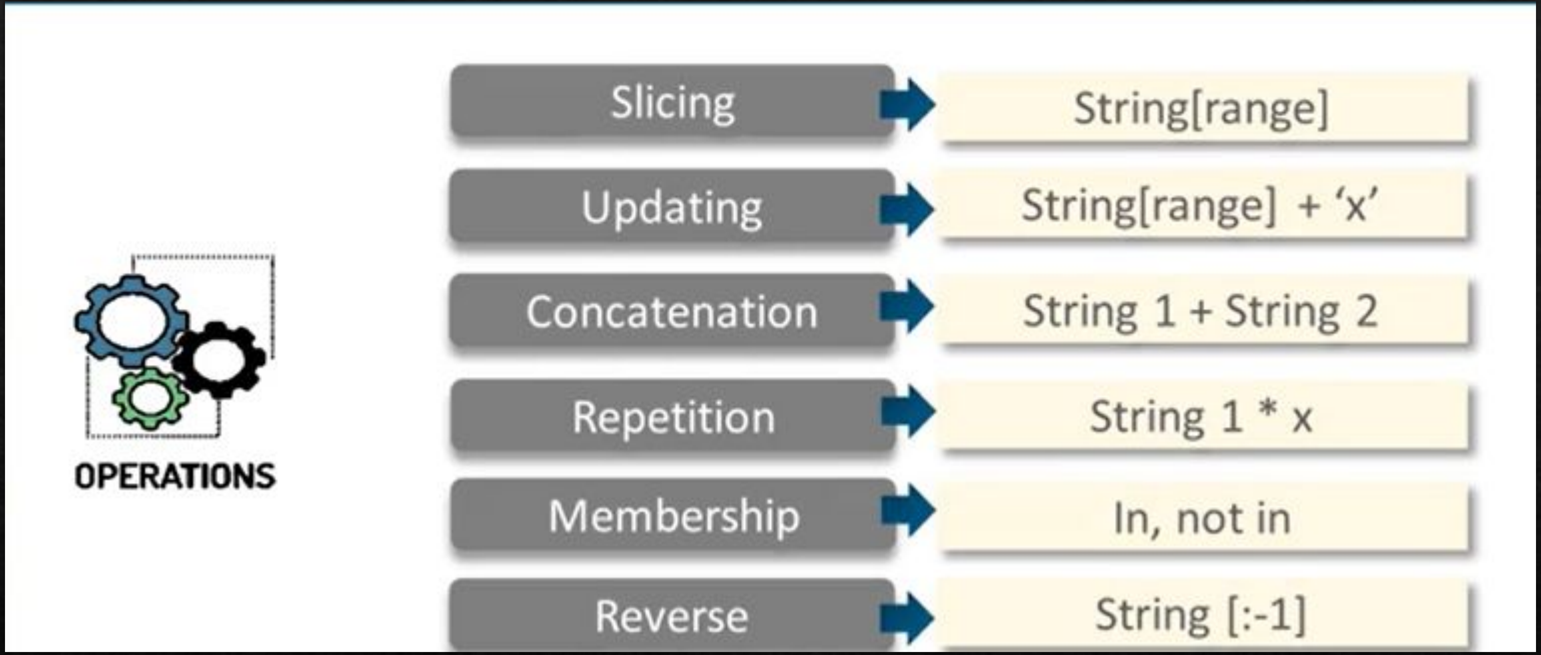
Converting list to tuple



```
tuple1=(1,2,3,5,7,'a','b')
lst=list(tuple1)
print(lst)

lst[1]='Python'
print(lst)

tuple2=tuple(lst)
print(tuple2)
```

Here, we are converting Tuple into List, updating contents of List and again converting List into Tuple

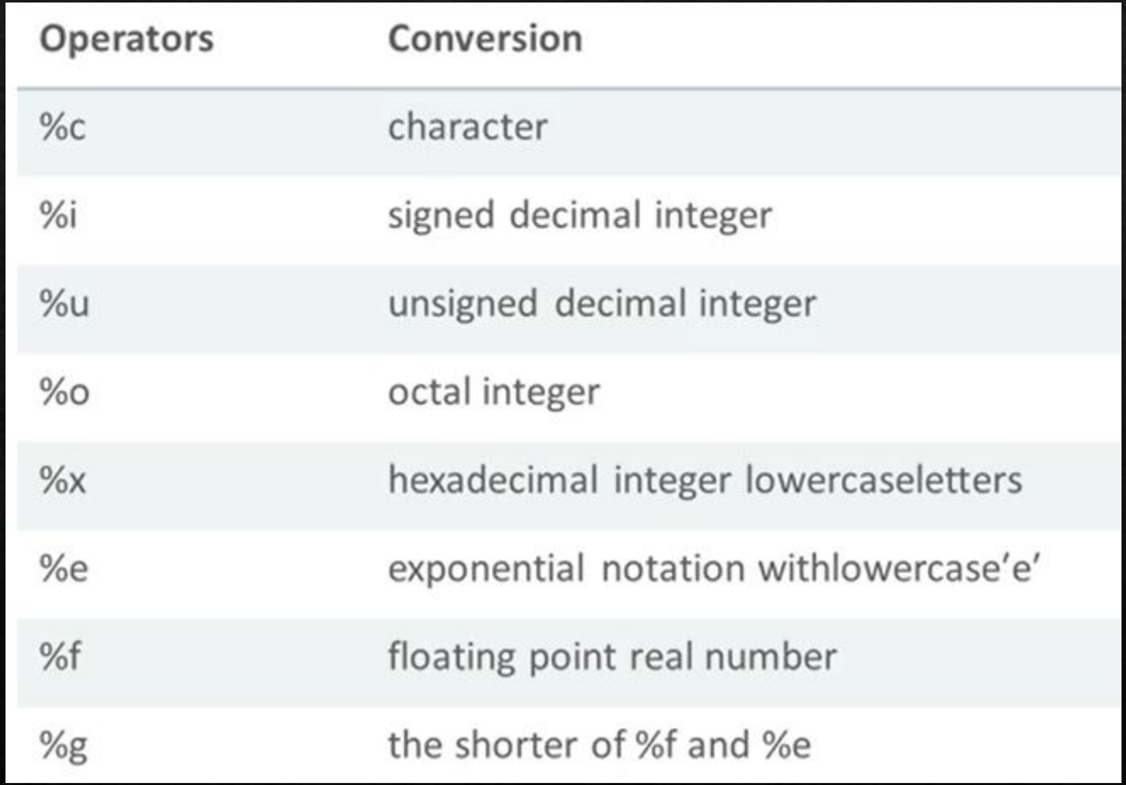
String



Slicing	String[range]
Updating	String[range] + 'x'
Concatenation	String 1 + String 2
Repetition	String 1 * x
Membership	In, not in
Reverse	String [:-1]

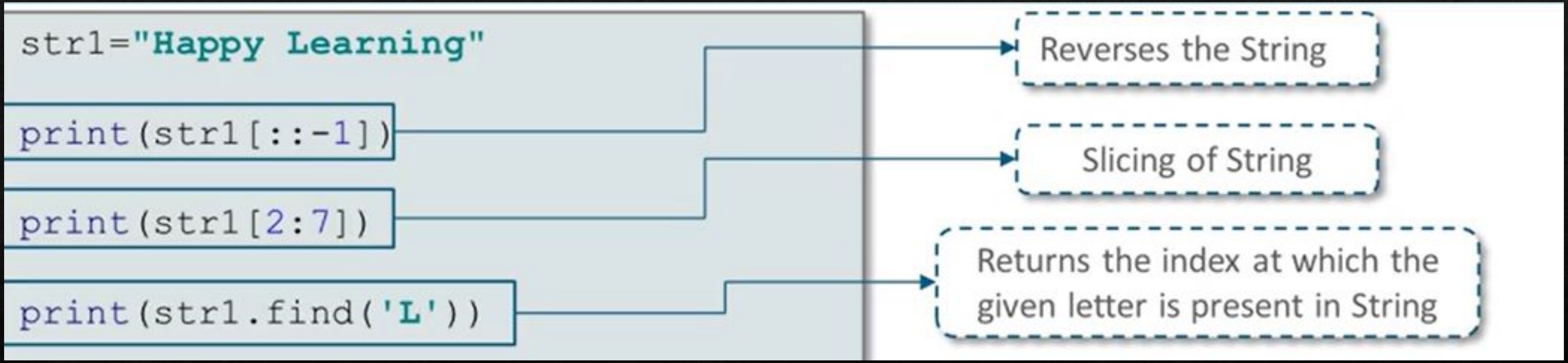
OPERATIONS

String Formatting



Operators	Conversion
%c	character
%i	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer lowercaseletters
%e	exponential notation withlowercase’e’
%f	floating point real number
%g	the shorter of %f and %e

String Operations

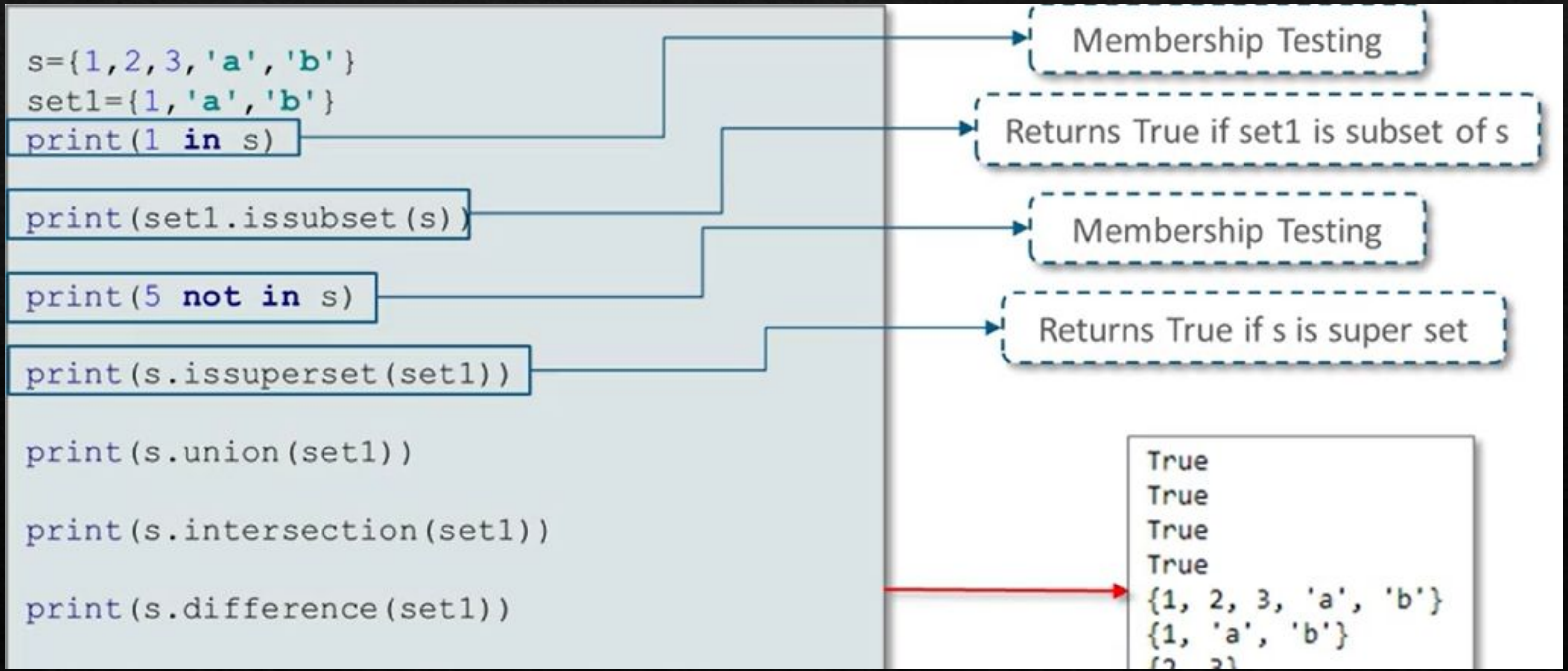


```
str1="Happy Learning"

print(str1[::-1])

print(str1[2:7])

print(str1.find('L'))
```

Reverses the String
Slicing of String
Returns the index at which the given letter is present in String

Set

- Un ordered collection of Unique items, separated by {}
- Used to collect unique strings and int
- Operations
- Union (|)
- Intersection (&)
- Difference (-)

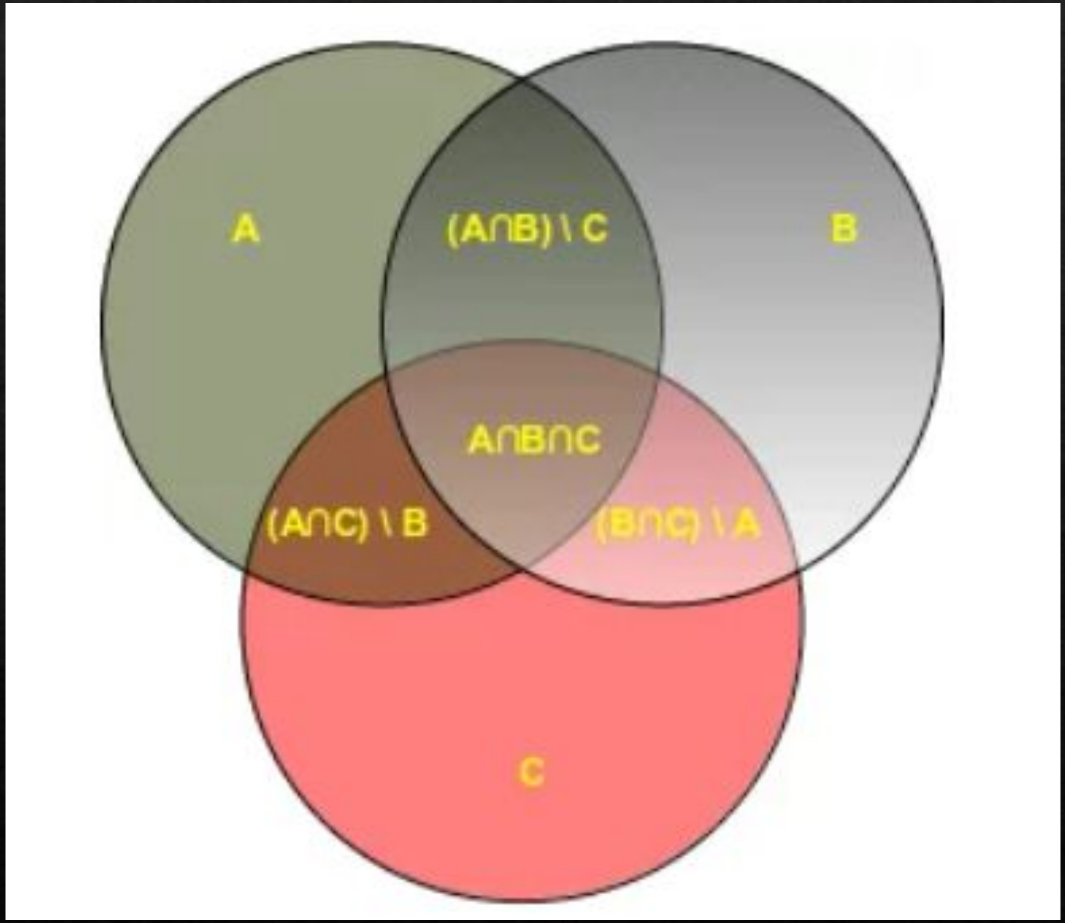


```
s={1,2,3,'a','b'}
set1={1,'a','b'}
print(1 in s)

print(set1.issubset(s))

print(5 not in s)

print(s.issuperset(set1))

print(s.union(set1))

print(s.intersection(set1))

print(s.difference(set1))
```

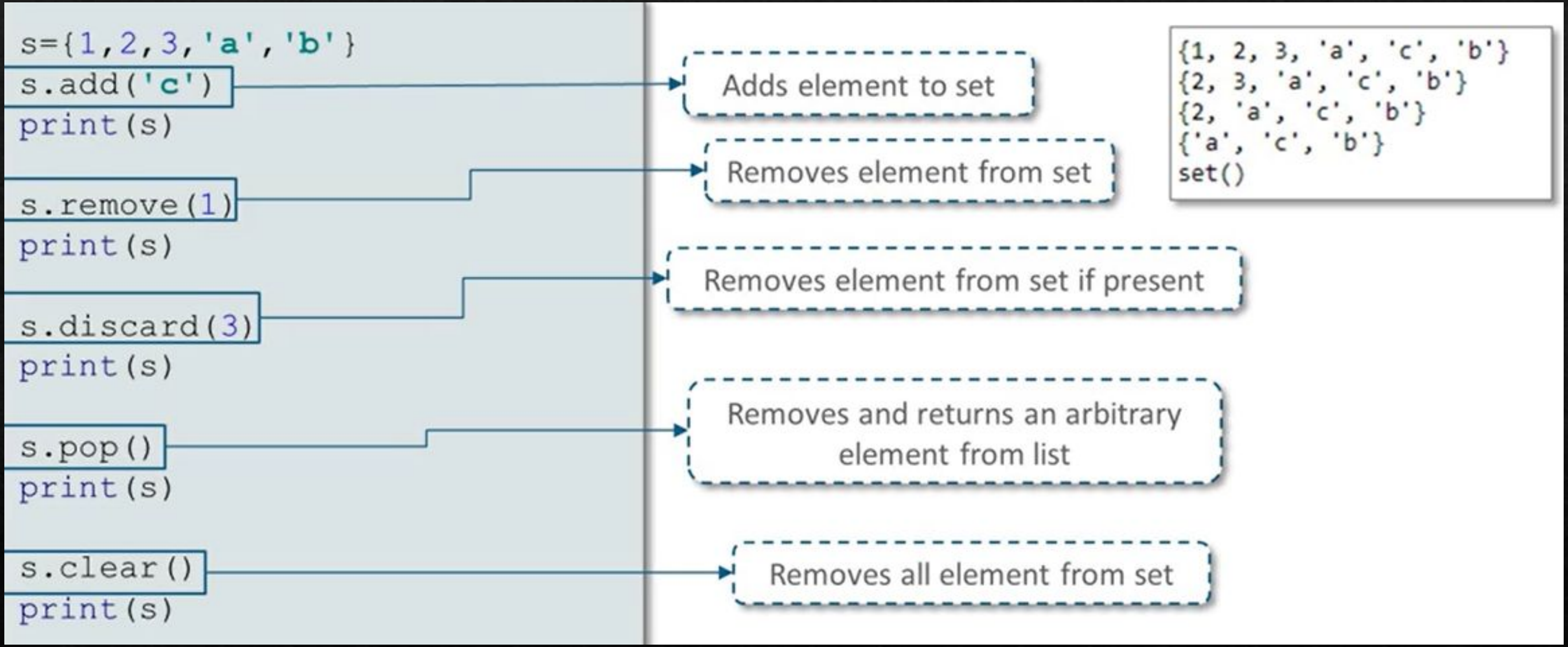
Membership Testing
Returns True if set1 is subset of s
Membership Testing
Returns True if s is super set

```
True
True
True
True
{1, 2, 3, 'a', 'b'}
{1, 'a', 'b'}
```

Set



Set Operations



```
s={1,2,3,'a','b'}
s.add('c')
print(s)

s.remove(1)
print(s)

s.discard(3)
print(s)

s.pop()
print(s)

s.clear()
print(s)
```

Adds element to set
Removes element from set
Removes element from set if present
Removes and returns an arbitrary element from list
Removes all element from set

```
{1, 2, 3, 'a', 'c', 'b'}
{2, 3, 'a', 'c', 'b'}
{2, 'a', 'c', 'b'}
{'a', 'c', 'b'}
set()
```

Remove – Error if the element not present
Discard – No error if not present

Frozen Set

Frozen means unmoving or fixed.

The **frozenset()** is an inbuilt function in python that takes an iterable object as input and makes them immutable. It simply freezes the iterable objects and makes them unchangeable.

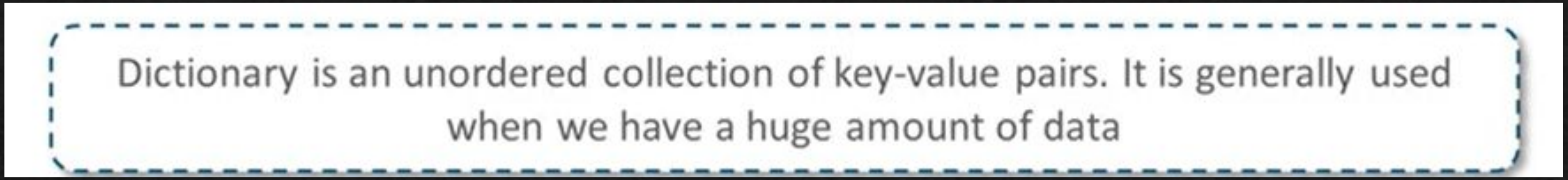
Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. That is once you created the set, it becomes immutable. Frozenset is also called a read-only set.

Frozenset Vs Set

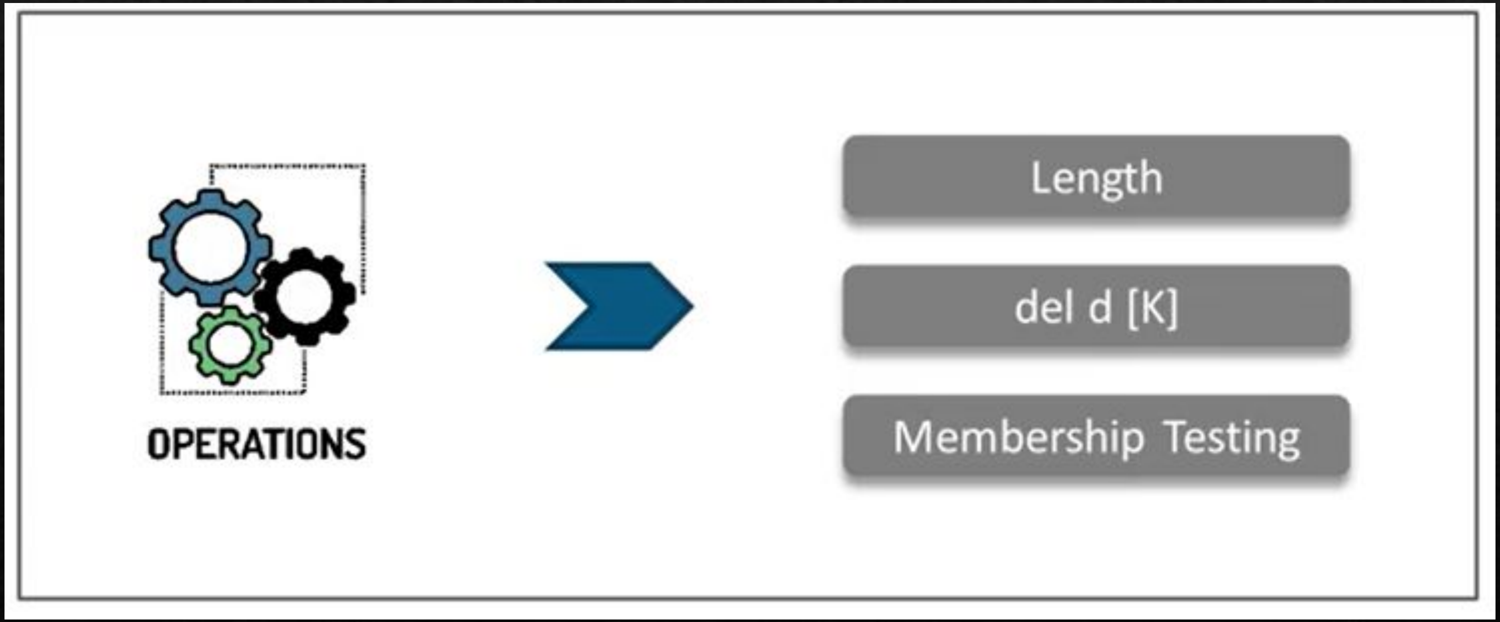
Set is a most basic level datatype, It supports all the method operations of the set such as add(), remove(), and so on.

The Frozen set is immutable, it does not support any operations like add(), remove(), and so on.

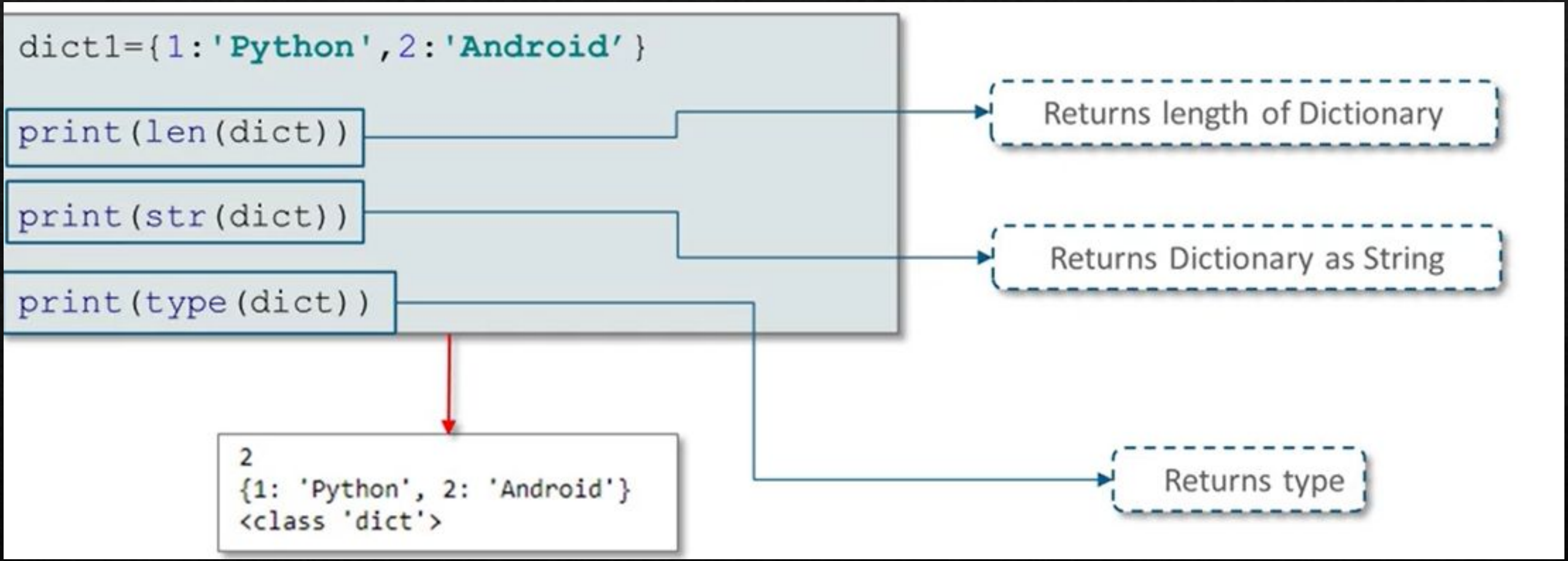
Dictionaries



Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data



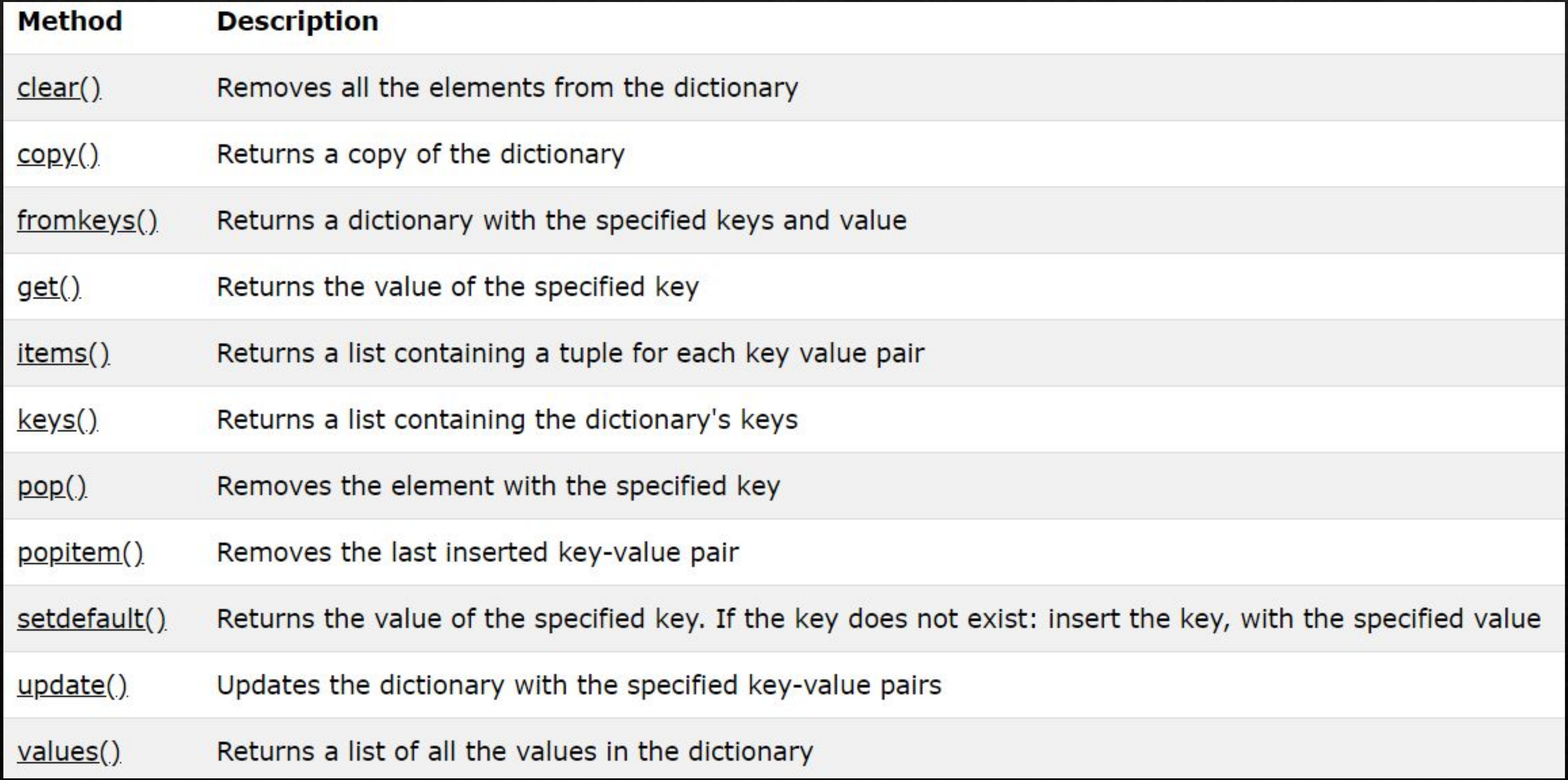
Dictionaries



```
dict1={1:'Python',2:'Android'}

print(len(dict))
print(str(dict))
print(type(dict))
```

```
2
{1: 'Python', 2: 'Android'}
<class 'dict'>
```

Dictionary methods



Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary