

```
S="this is the string . Strings in the python are immutable "  
print (S)
```

```
→ this is the string . Strings in the python are immutable
```

```
S= "this is the string . Strings in the python are immutable" # Sentence ko first lettter capital banuacha capatilize le  
print (S.capitalize())
```

```
→ This is the string . strings in the python are immutable
```

```
# t lai count garna count('t')  
s="this is the string. strings in the python are immutable"  
print (s.count('t'))
```

```
→ 7
```

```
s="this is the string. strings in the python are immutable"  
print(s.count('tr'))
```

```
→ 2
```

```
s="this is the string . string in the python are immutable"  
print(s.count('i'))
```

```
→ 6
```

```
name='ram'  
age=22  
s=f"this is {name}. my age is {age}"  
print(s)
```

```
→ this is ram. my age is 22
```

```
name ='ram'  
age=22  
s="this is {} and my age is{}". format(name,age)  
print(s)
```

```
→ this is ram and my age is22
```

```
# list  
x=[1,2,3,4,5]  
print(x)
```

```
→ [1, 2, 3, 4, 5]
```

```
#list  
x=[1,2,3,4,5]  
print(x,type(x))
```

```
→ [1, 2, 3, 4, 5] <class 'list'>
```

```
# order collection  
# support any type of data structure  
x=[1,2,'ram', 1,1.5,'5j',(1,2,3),{1,2,3},{'name':"ram"},[1,2,3],12]  
print(x)
```

```
→ [1, 2, 'ram', 1, 1.5, '5j', (1, 2, 3), {1, 2, 3}, {'name': 'ram'}, [1, 2, 3], 12]
```

```
fruits=['apple','banana'] # array ko 1th position ma banana cha so teslai print garauna  
fruits[1]
```

```
→ 'banana'
```

```
books=['saya', 'samar love','monsoon','mahako maha']  
books[0]
```

```
→ 'saya'
```

```
# nested loop to find the value of 4
# 4 array ko 2nd ma cha then tyo 2nd array ko ni 1st item ho so
x=[1,2,[3,4,5,6,7],8,9,10]
x[2][1]
```

↔ 4

```
# 9 find out garna ko lagi
x=[98,88,4,5,6,[6,7,8,9,45]]
x[5][3]
```

↔ 9

```
# list is the mutable object can be changed once defined
fruits=['apple','banana','mango','orange']
print(fruits)
fruits[0]='guava'
print(fruits)
# pailaa apple, bananana, mango , orange sisplay vayo tespachi easily hamley apple lai guava le replace garna sakem
```

↔ ['apple', 'banana', 'mango', 'orange']  
['guava', 'banana', 'mango', 'orange']

```
# practise
books=['saya','aashra','mahako maha','pagal basti']
print (books)
books[2]='karnali blues'
print(books)
```

↔ ['saya', 'aashra', 'mahako maha', 'pagal basti']  
['saya', 'aashra', 'karnali blues', 'pagal basti']

```
# list sanga related kura ko help ko lagi
help(list)
```

↔

```
-----  
Class methods defined here:
```

```
__class_getitem__(object, /)  
    See PEP 585
```

```
-----  
Static methods defined here:
```

```
__new__(*args, **kwargs)  
    Create and return a new object.  See help(type) for accurate signature.
```

```
-----  
Data and other attributes defined here:
```

```
__hash__ = None
```

```
# using the append command we can add the new list object  
fruits=['apple','banana','mango','orange']  
fruits.append('pineapple')  
print(fruits)
```

```
➦ ['apple', 'banana', 'mango', 'orange', 'pineapple']
```

```
#practise  
pet=['dog','cat','rabbit']  
pet.append('Monkey')  
print(pet)
```

```
➦ ['dog', 'cat', 'rabbit', 'Monkey']
```

```
# using the append command we can add the new list object=> list is added as it is  
fruits=['apple','guava','orange','pineapple']  
fruits.append(['mango','litchi','avocado'])  
print(fruits)
```

```
➦ ['apple', 'guava', 'orange', 'pineapple', ['mango', 'litchi', 'avocado']]
```

```
# using the extend command we can add the new list object by seperating the each object of the list  
#list ko kuraaa separe vayera aauna ko lagi extend use garenchhaaa plus append pani hunchhaa  
fruits=['apple','banana','orange']  
fruits.extend(['mango','guava','papaya'])  
print(fruits)
```

```
➦ ['apple', 'banana', 'orange', 'mango', 'guava', 'papaya']
```

```
# insert in any index using the index  
fruits=['apple','banana','orange']  
fruits.insert(1,'grapes')  
print(fruits)  
#1th index ma grapes add bhayo
```

```
➦ ['apple', 'grapes', 'banana', 'orange']
```

```
books=['saya','samar love','pani ko gham']  
books.insert(3,'aama')  
print(books)
```

```
➦ ['saya', 'samar love', 'pani ko gham', 'aama']
```

```
fruits=['apple','banana','orange']#use to pop out the index value  
print(fruits.pop(2))  
print(fruits)
```

```
➦ orange  
['apple', 'banana']
```

```
#tuple
#support any type of datastructure
#ordered collection
x=(1,2,3,'garima')
print(x)
```

```
(1, 2, 3, 'garima')
```

```
#tuple
#support ant kind of data structure
#ordered collection
x=(1,2,3,'garima')
print(x,type(x))
```

```
(1, 2, 3, 'garima') <class 'tuple'>
```

#tuple is the imutable datatype means its value cannot be changed once written

```
# for help
help(tuple)
```

```
__getattribute__(self, name, /)
    Return getattr(self, name).

__getitem__(self, key, /)
    Return self[key].

__getnewargs__(self, /)

__gt__(self, value, /)
    Return self>value.

__hash__(self, /)
    Return hash(self).

__iter__(self, /)
    Implement iter(self).

__le__(self, value, /)
    Return self<=value.

__len__(self, /)
    Return len(self).

__lt__(self, value, /)
    Return self<value.

__mul__(self, value, /)
    Return self*value.

__ne__(self, value, /)
    Return self!=value.

__repr__(self, /)
    Return repr(self).

__rmul__(self, value, /)
    Return value*self.

count(self, value, /)
    Return number of occurrences of value.

index(self, value, start=0, stop=9223372036854775807, /)
    Return first index of value.

    Raises ValueError if the value is not present.

-----
Class methods defined here:

__class_getitem__(object, /)
    See PEP 585

-----
Static methods defined here:

__new__(*args, **kwargs)
    Create and return a new object.  See help(type) for accurate signature.
```

```
# used to count the value of attributes
```

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

```
print(x.count(3))
```

```
➞ 2
```

```
x=(1,2,3,4,5,6,3,2,1,3,3)
```

```
print(x.index(1))
```

```
#yaha 1 ko index 0 pani ho ra 8 pani ho but first ma jun cha tehi dncha so 0
```

```
#early found index is given
```

```
➞ 0
```

```
#dictionary
```

```
person={
```

```
    'id':1,
```

```
    'name':"garima",
```

```
    'age':20,
```

```
    'salary':4656474,
```

```
    'contact':123456764553,
```

```
}
```

```
print(person)
```

```
➞ {'id': 1, 'name': 'garima', 'age': 20, 'salary': 4656474, 'contact': 123456764553}
```

```
#dictionary
```

```
person={
```

```
    'id':1,
```

```
    'name':"garima",
```

```
    'age':20,
```

```
    'salary':4656474,
```

```
    'contact':123456764553,
```

```
}
```

```
print(person['name'])
```

```
➞ garima
```

```
person={
```

```
    'id':1,
```

```
    'name':"garima",
```

```
    'age':20,
```

```
    'salary':4656474,
```

```
    'contact':123456764553,
```

```
}
```

```
# to add the email the dictionary
```

```
person['email']='gari@gmail.com'
```

```
print(person)
```

```
➞ {'id': 1, 'name': 'garima', 'age': 20, 'salary': 4656474, 'contact': 123456764553, 'email': 'gari@gmail.com'}
```

```
person={
```

```
    'id':1,
```

```
    'name':"garima",
```

```
    'age':20,
```

```
    'salary':4656474,
```

```
    'contact':123456764553,
```

```
}
```

```
del person['age'] #to delete from the dictionary
```

```
print(person)
```

```
➞ {'id': 1, 'name': 'garima', 'salary': 4656474, 'contact': 123456764553}
```

```
person={
```

```
    'id':1,
```

```
    'name':"garima",
```

```
    'age':20,
```

```
    'salary':4656474,
```

```
    'contact':123456764553,
```

```
}
```

```
print(person.get('email'))#this is the error handling technique=> here there is no email but the email is handled carefully with none message
```

```
print('helloo!!!!')
```

```
➦ None
hellooo!!!!
```

```
person={
    'id':1,
    'name':"garima",
    'age':20,
    'salary':4656474,
    'contact':123456764553,
}
print(person.get('age'))
```

```
➦ 20
```

```
person={
    'id':1,
    'name':"garima",
    'age':20,
    'salary':4656474,
    'contact':123456764553,
}
print(person.keys()) #dictionary ko keys ko lagi
```

```
➦ dict_keys(['id', 'name', 'age', 'salary', 'contact'])
```

```
person={
    'id':1,
    'name':"garima",
    'age':20,
    'salary':4656474,
    'contact':123456764553,
}
print (person.values())#dictionary ko value ko lagi
```

```
➦ dict_values([1, 'garima', 20, 4656474, 123456764553])
```

#for both the keys and the value we use items

```
person={
    'id':1,
    'name':"garima",
    'age':20,
    'salary':4656474,
    'contact':123456764553,
}
print(person.items())
```

```
➦ dict_items([('id', 1), ('name', 'garima'), ('age', 20), ('salary', 4656474), ('contact', 123456764553)])
```

```
person={
    'id':1,
    'name':"garima",
    'age':20,
    'salary':4656474,
    'contact':123456764553,
}
person.pop('name')#to delete from the dictionary
print(person)
```

```
➦ {'id': 1, 'age': 20, 'salary': 4656474, 'contact': 123456764553}
```

#to add the email to the dictionary

```
person={
    'id':1,
    'name':"garima",
    'age':20,
    'salary':4656474,
    'contact':123456764553,
}
person.update({"email":"abc@gmail.com"})
print (person)
```

```
➦ {'id': 1, 'name': 'garima', 'age': 20, 'salary': 4656474, 'contact': 123456764553, 'email': 'abc@gmail.com'}
```

```
#tuple , int float,complex (number datatypes),strings=> immutable
#list,dictionary,set => mutable
```

```
#unordered collection
s={'ram',1,4,5,7,0,(1,2,3),'shyam',1}
print(s)
```

```
➞ {0, 1, 4, 5, 7, (1, 2, 3), 'ram', 'shyam'}
```

```
#unique ordered collection
#here the o/p has declined the repetitation and makes it as single="ram"
s={'ram',1,2,3,4,'ram'}
print(s)
```

```
➞ {1, 2, 3, 4, 'ram'}
```

```
#in set only mutable data structure can be replaced
tea={'ram','hari','shyam','gita'}
coffee={'ram','darpan','garima'}
tea_or_coffee=tea| coffee
#union ko lagi | yo symbol use hunchhaa
#union ma repeated value yek patak matra aaucha
print(tea_or_coffee)
```

```
➞ {'gita', 'ram', 'darpan', 'hari', 'garima', 'shyam'}
```

```
tea={'ram','hari','gita'}
coffee={'ram','abi','hariprasad'}
tea_or_coffee=tea.union(coffee)#union of the set
print(tea_or_coffee)
```

```
➞ {'hariprasad', 'gita', 'ram', 'hari', 'abi'}
```

```
tea={'ram','hari','shyam','gita'}
coffee={'ram','darpan','garima'}
tea_or_coffee=tea & coffee
#union ko lagi & yo symbol use hunchhaa
#intersection ma common value matra aaucha
print(tea_or_coffee)
```

```
➞ {'ram'}
```

```
tea={'ram','hari','shyam','gita'}
coffee={'ram','darpan','garima'}
tea_or_coffee=tea.intersection(coffee)#for intersection
#union ko lagi & yo symbol use hunchhaa
#intersection ma common value matra aaucha
print(tea_or_coffee)
```

```
➞ {'ram'}
```

```
# difference in the set
tea={'ram','hari','shyam'}
coffee={'ram','bed prasad','ram hari'}# a ma bhako b ma navako aaune
tea_or_coffee=tea.difference(coffee)
print(tea_or_coffee)
```

```
➞ {'hari', 'shyam'}
```

```
# for help
help(set)
```

```
➞
```

```

issubset(self, other, /)
    Report whether another set contains this set.

issuperset(self, other, /)
    Report whether this set contains another set.

pop(self, /)
    Remove and return an arbitrary set element.

    Raises KeyError if the set is empty.

remove(self, object, /)
    Remove an element from a set; it must be a member.

    If the element is not a member, raise a KeyError.

symmetric_difference(self, other, /)
    Return a new set with elements in either the set or other but not both.

symmetric_difference_update(self, other, /)
    Update the set, keeping only elements found in either set, but not in both.

union(self, /, *others)
    Return a new set with elements from the set and all others.

update(self, /, *others)
    Update the set, adding elements from all others.

-----
Class methods defined here:

__class_getitem__(object, /)
    See PEP 585

-----
Static methods defined here:

__new__(*args, **kwargs)
    Create and return a new object.  See help(type) for accurate signature.

-----
Data and other attributes defined here:

__hash__ = None

```

```

# to add 33 in the set
s={'ram',1,4,5,(1,2,3),3,'ram',1}
s.add(33)
print(s)

```

```

➦ {1, 33, 3, 4, 5, 'ram', (1, 2, 3)}
```

```

#to remove the value from the set
s={'ram',1,4,5,(1,2,3),3,'ram',1}
s.discard(1)
print(s)

```

```

➦ {3, 4, 5, 'ram', (1, 2, 3)}
```

```

#to find the length of the list
len([1,2,333,4,5,6.7])

```

```

➦ 6
```

```

#to find the sum of the list
sum([1,2,3,4])

```

```

➦ 10
```

```

#to find the max of the list
max([1,2,3,4])

```

```

➦ 4
```

```

#to find the min of the list
min([1,2,3,4])

```



↔ 1

```
#sorting in ascending order  
sorted([4,3,5,7,9,2,10])
```

↔ [2, 3, 4, 5, 7, 9, 10]

```
#sorting in decending order  
sorted([4,3,5,7,9,2,10],reverse=True)
```

↔ [10, 9, 7, 5, 4, 3, 2]

Start coding or [generate](#) with AI.