Question 1:

1. Write a program to create an empty list named my_list and
2. Add numbers 5 and 9 to the list using append() method
3. Ask the user to input any number in the console and add the number to the list.
4. You can use int() to typecast string to integer if you want.
5. Create another list more_items with 3 items on it and extend the list my_list using extend method.
6. Now find the length of the list and print the length of list describing it in a sentence. you can use string formatting for better outputs.
7. Now remove the second item using pop() method and see if the item exists in the list you can print the list before and after using the pop() method.

In [1]:
```
#1
my_list=[]
```

In [2]:
```
type(my_list)
```

Out[2]:
```
list
```

In [3]:
```
#2
my_list.append(5)
my_list.append(9)
```

In [4]:
```
print(my_list)
```

```
[5, 9]
```

In [5]:
```
#mistakely, we add two 5, and from here we can also said that list accept duplicate value.
```

In [6]:
```
#3.
num=int(input("Add number of your choice: "))
my_list.append(num)
```

```
Add number of your choice: 6
```

In [7]:
```
print(my_list)
```

```
[5, 9, 6]
```

In [8]:
```
#5
more_item=[7,8,10,11]
```

In [9]:
```
my_list.extend(more_item)
```

In [10]:
```
print(my_list)
```

```
[5, 9, 6, 7, 8, 10, 11]
```

In [11]:
```
#6
length=len(my_list)
print(f'The length of my_list is {length}')
```

```
The length of my_list is 7
```

In [12]:
```
#7
print(my_list)
```

```
[5, 9, 6, 7, 8, 10, 11]
```

In [13]:
```
my_list.pop(2)
```

Out[13]:
```
6
```

In [14]:
```
print(my_list)
```

```
[5, 9, 7, 8, 10, 11]
```

Question 2:

Write a program to add 5 different wild animals to a list named wild. Example: tiger, lion, deer, bear, zebra

1. sort them in ascending using the sort() method.
2. reverse the list using the reverse() method.
3. now add 3 more animals to the list wild. Example: leopard, elephant, rhino
4. find the position of leopard using the index() method and remove it using the pop() method.
5. pop should have the index value returned using the index() method.
6. do not hard-code the position of leopard by manually counting it from the list.
7. check whether the leopard is removed from the list or not by index() method again. if the value error occurs, you have successfully removed it from the list. otherwise, try to do it again.
8. you can then comment the line that gives exception to continue to the next question.

9. Now add leopard again in the index 2 using insert() method.
10. Again, remove rhino from the list using remove() method.

```
In [15]: wild=['tiger','lion','deer','bear','zebra']
```

```
In [16]: print(wild)
```

```
['tiger', 'lion', 'deer', 'bear', 'zebra']
```

```
In [17]: type(wild)
```

```
Out[17]: list
```

```
In [18]: wild.sort()
         print("List after sorting: ", wild)
```

```
List after sorting:  ['bear', 'deer', 'lion', 'tiger', 'zebra']
```

```
In [19]: #2
         wild.reverse()
         print("List after reversing: ", wild)
```

```
List after reversing:  ['zebra', 'tiger', 'lion', 'deer', 'bear']
```

```
In [20]: print(wild)
```

```
['zebra', 'tiger', 'lion', 'deer', 'bear']
```

```
In [21]: #3
         morewild=['leopard', 'elephant', 'rhino']
         wild.extend(morewild)
         print(wild)
```

```
['zebra', 'tiger', 'lion', 'deer', 'bear', 'leopard', 'elephant', 'rhino']
```

```
In [22]: #4 #5 #6
         wild.index('leopard')
```

```
Out[22]: 5
```

```
In [23]: wild.pop(5)
```

```
Out[23]: 'leopard'
```

```
In [24]: print(wild)
```

```
['zebra', 'tiger', 'lion', 'deer', 'bear', 'elephant', 'rhino']
```

```
In [25]: #7
         wild.index("leopard")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[25], line 2
      1 #7
----> 2 wild.index("leopard")

ValueError: 'leopard' is not in list
```

```
In [ ]: #8
        # ValueError: 'leopard' is not in list
```

```
In [ ]: #9
        wild.insert(2,"leopard")
```

```
In [ ]: print(wild)
```

```
In [ ]: #10
        wild.remove("rhino")
```

```
In [ ]: print(wild)
```

Question 3:

1. Try creating a multi-dimensional list or nested list multi containing different numbers.

Example: [ [12, 52, 37], [46, 51, 16], [17, 82, 39] ]

1. Access the number 51 from the list.

2. Access the number 82 using the negative indices.

3. Find out the length of the list multi

4. Append another list [40, 61, 10] inside the list multi. The final list should be: [[12,52,37],[46,51,16],[17,82,39],[40, 61, 10]]

5. Use foreach in the list multi to print each list item to the console. Bonus: Try using nested foreach to access each item inside of the inner list Bonus: Try finding out the length of each inner list

6. Finally clear the list multi using the clear() method and verify if the list is empty or not.

```
In [ ]: #1
        list1=[[12,52,37],[46,51,16],[17,82,39]]
```

```
In [ ]: print(list1)
```

```
In [ ]: type(list1)
```

```
In [ ]: #2
        list1[1][1]           #1th row and 1th column
```

```
In [ ]: #3
        list1[-1][-2]
```

```
In [ ]: #4
        len(list1)
```

```
In [ ]: #5
        list2=[40, 61, 10]
```

```
In [ ]: list1.extend(list2)
```

```
In [ ]: print(list1)
```

```
In [ ]: #we use extend, but we can use append too and mistakely, we run list2 two times here.
```

```
In [ ]: #7
        list1.clear()
```

```
In [ ]: print(list1)
```

Question 4:

1. Write a program to create a tuple to add 5 different numbers.
2. find out the length of the tuple
3. find out the 3rd element of the tuple by accessing it through the index
4. use enumerate() function to map each element with its index and print it using for loop.

```
In [ ]: tup1=(1,2,3,4,5)  #this is normal way to create tuple
```

```
In [ ]: type(tup1)
```

```
In [ ]: #1.
        numbers=[]
        for i in range(5):
            number=int(input("Enter a number: "))
            numbers.append(number)
            tuple1=tuple(numbers)
```

```
In [ ]: print(tuple1)
```

```
In [ ]: numbers=[]
        for i in range(6,11):
            numbers.append(i)
            tuple1=tuple(numbers)
```

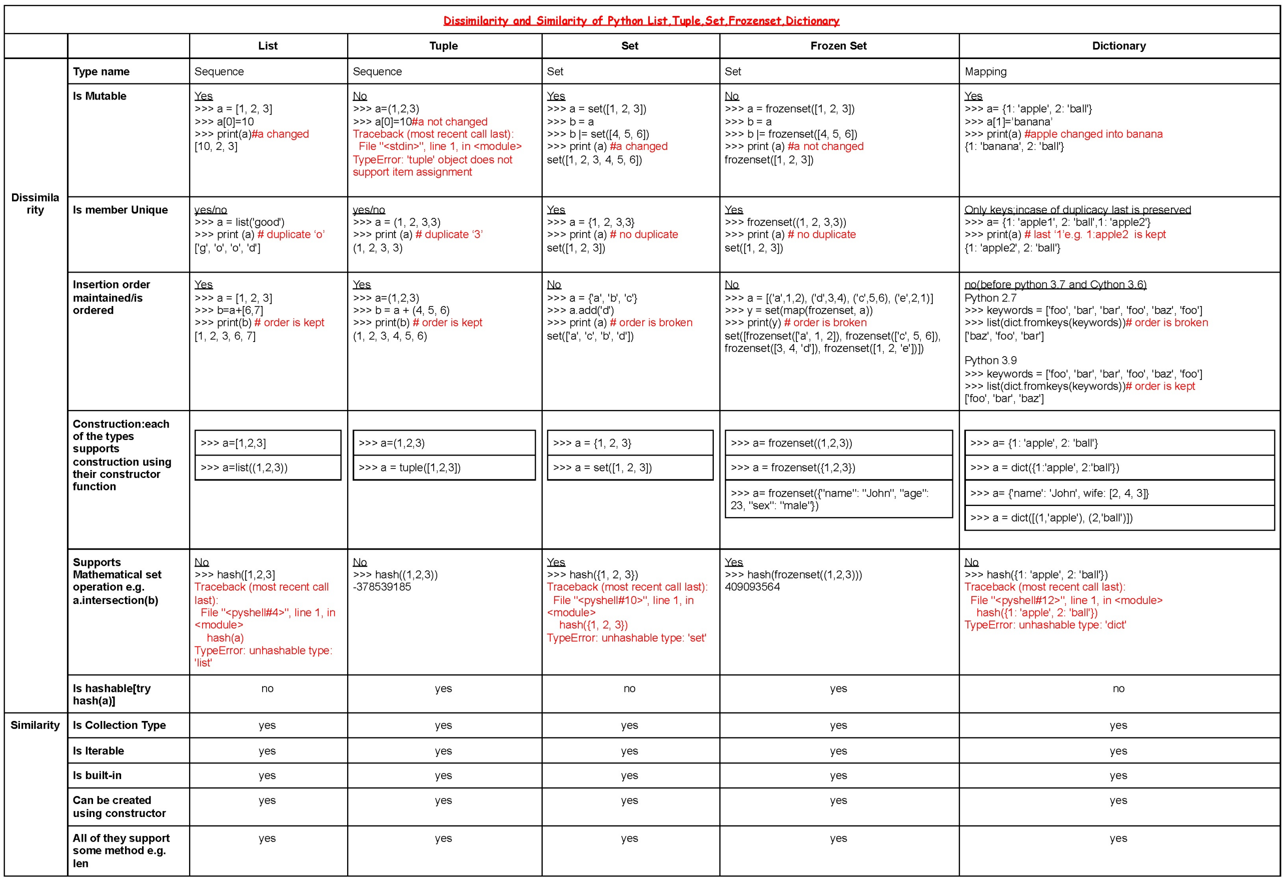
```
In [ ]: print(tuple1)
```

```
In [ ]: #2
        len(tuple1)
```

```
In [ ]: #3
        tuple1[3]
```

```
In [ ]: #4
        for i in enumerate(tuple1):
            print(i)
```

Lets again see basic difference between set, tuple, lists and dictionary:



Dissimilarity and Similarity of Python List,Tuple,Set,Frozenset,Dictionary

| | | List | Tuple | Set | Frozen Set | Dictionary |
|---|---|---|---|---|---|---|
| Dissimilarity | **Type name** | Sequence | Sequence | Set | Set | Mapping |
| | **Is Mutable** | Yes<br>>>> a = [1, 2, 3]<br>>>> a[0]=10<br>>>> print(a)#a changed<br>[10, 2, 3] | No<br>>>> a=(1,2,3)<br>>>> a[0]=10#a not changed<br>Traceback (most recent call last):<br>File "<stdin>", line 1, in <module><br>TypeError: 'tuple' object does not support item assignment | Yes<br>>>> a = set([1, 2, 3])<br>>>> b = a<br>>>> b \|= set([4, 5, 6])<br>>>> print (a) #a changed<br>set([1, 2, 3, 4, 5, 6]) | No<br>>>> a = frozenset([1, 2, 3])<br>>>> b = a<br>>>> b \|= frozenset([4, 5, 6])<br>>>> print (a) #a not changed<br>frozenset([1, 2, 3]) | Yes<br>>>> a= {1: 'apple', 2: 'ball'}<br>>>> a[1]='banana'<br>>>> print(a) #apple changed into banana<br>{1: 'banana', 2: 'ball'} |
| | **Is member Unique** | yes/no<br>>>> a = list('good')<br>>>> print (a) # duplicate 'o'<br>['g', 'o', 'o', 'd'] | yes/no<br>>>> a = (1, 2, 3,3)<br>>>> print (a) # duplicate '3'<br>(1, 2, 3, 3) | Yes<br>>>> a = {1, 2, 3,3}<br>>>> print (a) # no duplicate<br>set([1, 2, 3]) | Yes<br>>>> frozenset((1, 2, 3,3))<br>>>> print (a) # no duplicate<br>set([1, 2, 3]) | Only keys;incase of duplicacy last is preserved<br>>>> a= {1: 'apple1', 2: 'ball',1: 'apple2'}<br>>>> print(a) # last '1'e.g. 1:apple2 is kept<br>{1: 'apple2', 2: 'ball'} |
| | **Insertion order maintained/is ordered** | Yes<br>>>> a = [1, 2, 3]<br>>>> b=a+[6,7]<br>>>> print(b) # order is kept<br>[1, 2, 3, 6, 7] | Yes<br>>>> a=(1,2,3)<br>>>> b = a + (4, 5, 6)<br>>>> print(b) # order is kept<br>(1, 2, 3, 4, 5, 6) | No<br>>>> a = {'a', 'b', 'c'}<br>>>> a.add('d')<br>>>> print (a) # order is broken<br>set(['a', 'c', 'b', 'd']) | No<br>>>> a = [('a',1,2), ('d',3,4), ('c',5,6), ('e',2,1)]<br>>>> y = set(map(frozenset, a))<br>>>> print(y) # order is broken<br>set([frozenset(['a', 1, 2]), frozenset(['c', 5, 6]), frozenset([3, 4, 'd']), frozenset([1, 2, 'e'])]) | no(before python 3.7 and Cython 3.6)<br>Python 2.7<br>>>> keywords = ['foo', 'bar', 'bar', 'foo', 'baz', 'foo']<br>>>> list(dict.fromkeys(keywords))# order is broken<br>['baz', 'foo', 'bar']<br><br>Python 3.9<br>>>> keywords = ['foo', 'bar', 'bar', 'foo', 'baz', 'foo']<br>>>> list(dict.fromkeys(keywords))# order is kept<br>['foo', 'bar', 'baz'] |
| | **Construction:each of the types supports construction using their constructor function** | >>> a=[1,2,3]<br>>>> a=list((1,2,3)) | >>> a=(1,2,3)<br>>>> a = tuple([1,2,3]) | >>> a = {1, 2, 3}<br>>>> a = set([1, 2, 3]) | >>> a= frozenset((1,2,3))<br>>>> a = frozenset({1,2,3})<br>>>> a= frozenset({"name": "John", "age": 23, "sex": "male"}) | >>> a= {1: 'apple', 2: 'ball'}<br>>>> a = dict({1:'apple', 2:'ball'})<br>>>> a= {'name': 'John', wife: [2, 4, 3]}<br>>>> a = dict([(1,'apple'), (2,'ball')]) |
| | **Supports Mathematical set operation e.g. a.intersection(b)** | No<br>>>> hash([1,2,3]<br>Traceback (most recent call last):<br>File "<pyshell#4>", line 1, in <module><br>hash(a)<br>TypeError: unhashable type: 'list' | No<br>>>> hash((1,2,3))<br>-378539185 | Yes<br>>>> hash({1, 2, 3})<br>Traceback (most recent call last):<br>File "<pyshell#10>", line 1, in <module><br>hash({1, 2, 3})<br>TypeError: unhashable type: 'set' | Yes<br>>>> hash(frozenset((1,2,3)))<br>409093564 | No<br>>>> hash({1: 'apple', 2: 'ball'})<br>Traceback (most recent call last):<br>File "<pyshell#12>", line 1, in <module><br>hash({1: 'apple', 2: 'ball'})<br>TypeError: unhashable type: 'dict' |
| | **Is hashable[try hash(a)]** | no | yes | no | yes | no |
| Similarity | **Is Collection Type** | yes | yes | yes | yes | yes |
| | **Is Iterable** | yes | yes | yes | yes | yes |
| | **Is built-in** | yes | yes | yes | yes | yes |
| | **Can be created using constructor** | yes | yes | yes | yes | yes |
| | **All of they support some method e.g. len** | yes | yes | yes | yes | yes |

## Question 5:

Write a program to create a nested tuple and access different elements of the inner tuple using positive and negative indexes.

In [26]:
```python
n=((1,2,3),(4,5,6),(7,8,9))
```

In [27]:
```python
print(n)
```

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

In [28]:
```python
type(n)
```

Out[28]:
```
tuple
```

In [29]:
```python
#accessing using positive indexes, to bring output 4
n[1][0]
```

Out[29]:
```
4
```

In [30]:
```python
#accessing through nehative index to bring out same 4
n[-2][-3]
```

Out[30]:
```
4
```

## Question 6:

1. Create two different tuples t1 and t2 with different elements inside it create the next tuple t3 to add all values of t1 and t2 by destructuring or unpacking.

suppose t1 has (1, 6, 9, 4, 3) and t2 has (2, 7, 8, 3, 5) t3 must have (1, 6, 9, 4, 3, 2, 7, 8, 3, 5)

In [31]:
```python
t1=(1, 6, 9, 4, 3)
t2= (2, 7, 8, 3, 5)
```

In [32]:
```python
#two different tuple has been created but we cannot create t3 directly
#because tuple is immutable so we destructured tuple to create new tuple.
t3=list(t1)+list(t2)
```

In [33]:
```python
print(t3)
```

```
[1, 6, 9, 4, 3, 2, 7, 8, 3, 5]
```

In [34]:
```python
#now converting t3 list into tuple
t3_updated=tuple(t3)
```

```
In [35]: print(t3_updated)
```

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

Question 6:

Rich: USA, China, Japan, Germany, France, Australia, Italy Europe: Germany, France, England, Switzerland, Italy, Portugal, Sweden

Use Set methods to find out:

1. countries that are rich but not in Europe
2. countries that are in Europe but not rich
3. countries that are both rich and are in Europe
4. countries that are either rich or in Europe, but not both
5. all the countries in either of the sets. (Names must be unique)
6. see if two sets are disjoint or not
7. now remove the common countries from the rich set and check if two sets are disjoint or not. hint: use difference_update() method. for more, please refer to python documentation
8. Create 2 more sets: asian_rich: China, Japan american_rich: USA, Canada
9. Check whether asian_rich is a subset of rich or not.
10. Check whether rich is a superset of asian_rich or not.
11. Check whether american_rich is a subset of rich or not.

```
In [36]: Rich={"USA","China","Japan","Germany","France","Australia","Italy"}
         Europe={"Germany","France","England","Switzerland","Italy","Portugal","Sweden"}
         type(Rich)

Out[36]: set
```

```
In [37]: #1
         print(Rich.difference(Europe))
```

```
{'Japan', 'Australia', 'China', 'USA'}
```

```
In [38]: #2 Similar to one, here we use symbol
         print(Europe-Rich)
```

```
{'Switzerland', 'England', 'Sweden', 'Portugal'}
```

```
In [39]: #3 both rich and are in Europe
         print(Rich.intersection(Europe))
```

```
{'Italy', 'Germany', 'France'}
```

```
In [40]: #4 either rich or europe
         print(Rich.union(Europe) - Rich.intersection(Europe))
```

```
{'England', 'Australia', 'Japan', 'Portugal', 'USA', 'Switzerland', 'Sweden', 'China'}
```

```
In [41]: #5
         print(Rich.union(Europe))
```

```
{'England', 'Australia', 'Germany', 'Japan', 'Italy', 'Portugal', 'USA', 'Switzerland', 'Sweden', 'France', 'Ch
ina'}
```

```
In [ ]: #6
        print(Rich)
```

```
In [ ]: print(Europe)
```

```
In [42]: if Rich.isdisjoint(Europe):
             print("They are disjoint")
         else:
             print("They are not disjoint")
```

```
They are not disjoint
```

```
In [43]: #7
         Rich.difference_update(Europe)

         if Rich.isdisjoint(Europe):
             print("They are disjoint")
         else:
             print("They are not disjoint")
```

```
They are disjoint
```

```
In [45]: #Note difference_update will update the value in real variables too.
```

```
In [44]: #8
         asian_rich={'China', 'Japan'}
         american_rich={'USA', 'Canada'}
```

```
In [46]: #9
```

```
print(Rich)
```

```
{'Australia', 'China', 'Japan', 'USA'}
```

In [47]:
```
type(asian_rich)
```

Out[47]:
```
set
```

In [48]:
```
if asian_rich.issubset(Rich):
    print("It is a subset")
else:
    print("it is not subset")
```

```
It is a subset
```

In [49]:
```
#10
print(Rich.issuperset(asian_rich))
```

```
True
```

In [50]:
```
#11
print(american_rich.issubset(Rich))
```

```
False
```

Question 7:

1. Create a dictionary of a person that contains key value pair of

name: str age: int profession: str married: bool Set values with valid data types to each keys of the dictionary

1. Print the value of 'name' from the dictionary
2. Add the age by 10 and print the dictionary items in formatted string Eg: {name} will be {new_age} after 10 years.
3. Try getting the value of 'employed' from the dictionary.
4. If exception occurs, note it and check what exception says and finally comment the line.
5. try using get() method instead of large brackets [] in the previous question.
6. try using get() method with second parameter as False and see what is printed.

In [51]:
```
#1
person={"name":"hari","profession":"Teacher","married":False,"age":50}
```

In [52]:
```
print(person)
```

```
{'name': 'hari', 'profession': 'Teacher', 'married': False, 'age': 50}
```

In [53]:
```
type(person)
```

Out[53]:
```
dict
```

In [54]:
```
#2
person["name"]
```

Out[54]:
```
'hari'
```

In [59]:
```
#3
person["age"]=50+10
print(person)
```

```
{'name': 'hari', 'profession': 'Teacher', 'married': False, 'age': 60}
```

In [64]:
```
print(f"{person['name']} will be {'age'} years old after 10 years")
```

```
hari will be age years old after 10 years
```

In [66]:
```
#4 #5
person["employed"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[66], line 2
      1 #3
----> 2 person["employed"]

KeyError: 'employed'
```

# 6 #7

person={"name":"hari","profession":"Teacher","married":False,"age":50}

In [5]:
```
print(person)
```

```
{'name': 'hari', 'profession': 'Teacher', 'married': False, 'age': 50}
```

```
In [6]: x=person.get("name")
```

```
In [7]: print(x)
```

```
hari
```

```
In [8]: person.get("married")
```

```
Out[8]: False
```

```
In [9]: person.get("employed")
```

```
In [10]: #See the difference, get method return nothing when key is not present in dictionary,
         #but large bracket return error, example
         person["employed"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[10], line 3
      1 #See the difference, get method return nothing when key is not present in dictionary,
      2 #but large bracket return error, example
----> 3 person["employed"]

KeyError: 'employed'
```

Question 8:

create a dictionary car with 3 keys brand, model, and price.

1. Set a random value to a new key engine in the dictionary.
2. Add 3 more keys (color, no_of_seats, transmission) using update method.
3. Remove the key color from the dictionary.
4. Try using popitem() method in the dictionary and see what changes in dictionary
5. Use for loop to iterate through all keys from a dictionary.
6. Use for loop to iterate through all keys from a dictionary using keys() method.
7. Use for loop to iterate through all values from a dictionary using values() method.
8. Use for loop to iterate through all keys, values from a dictionary using items() method.

```
In [12]: car={"brand":"honda","model":"778z","price":15000}
         print(car)
```

```
{'brand': 'honda', 'model': '778z', 'price': 15000}
```

```
In [13]: #1
         import random
         car["engine"] = random.choice(["V6", "4-cylinder", "Hybrid"])
```

```
In [14]: print(car)
```

```
{'brand': 'honda', 'model': '778z', 'price': 15000, 'engine': 'Hybrid'}
```

```
In [15]: #2
         car.update({"color":"red","no_of_seats":6})
```

```
In [16]: print(car)
```

```
{'brand': 'honda', 'model': '778z', 'price': 15000, 'engine': 'Hybrid', 'color': 'red', 'no_of_seats': 6}
```

```
In [18]: #3
         car.pop("color")
```

```
Out[18]: 'red'
```

```
In [19]: print(car)
```

```
{'brand': 'honda', 'model': '778z', 'price': 15000, 'engine': 'Hybrid', 'no_of_seats': 6}
```

```
In [22]: #4
         car.popitem()
```

```
Out[22]: ('no_of_seats', 6)
```

```
In [23]: #firstly, i try to give attribute to popitem but, I got error saying popitem does not take aatribute, but when
         #any attribute, it take out last item from car dictionary.
```

```
In [24]: print(car)
```

```
{'brand': 'honda', 'model': '778z', 'price': 15000, 'engine': 'Hybrid'}
```

```
In [25]: #5
         for i in car:
             print(i)
```

```
brand
model
price
engine
```

In [26]:
```python
#6
for key in car.keys():
    print(key)
```

```
brand
model
price
engine
```

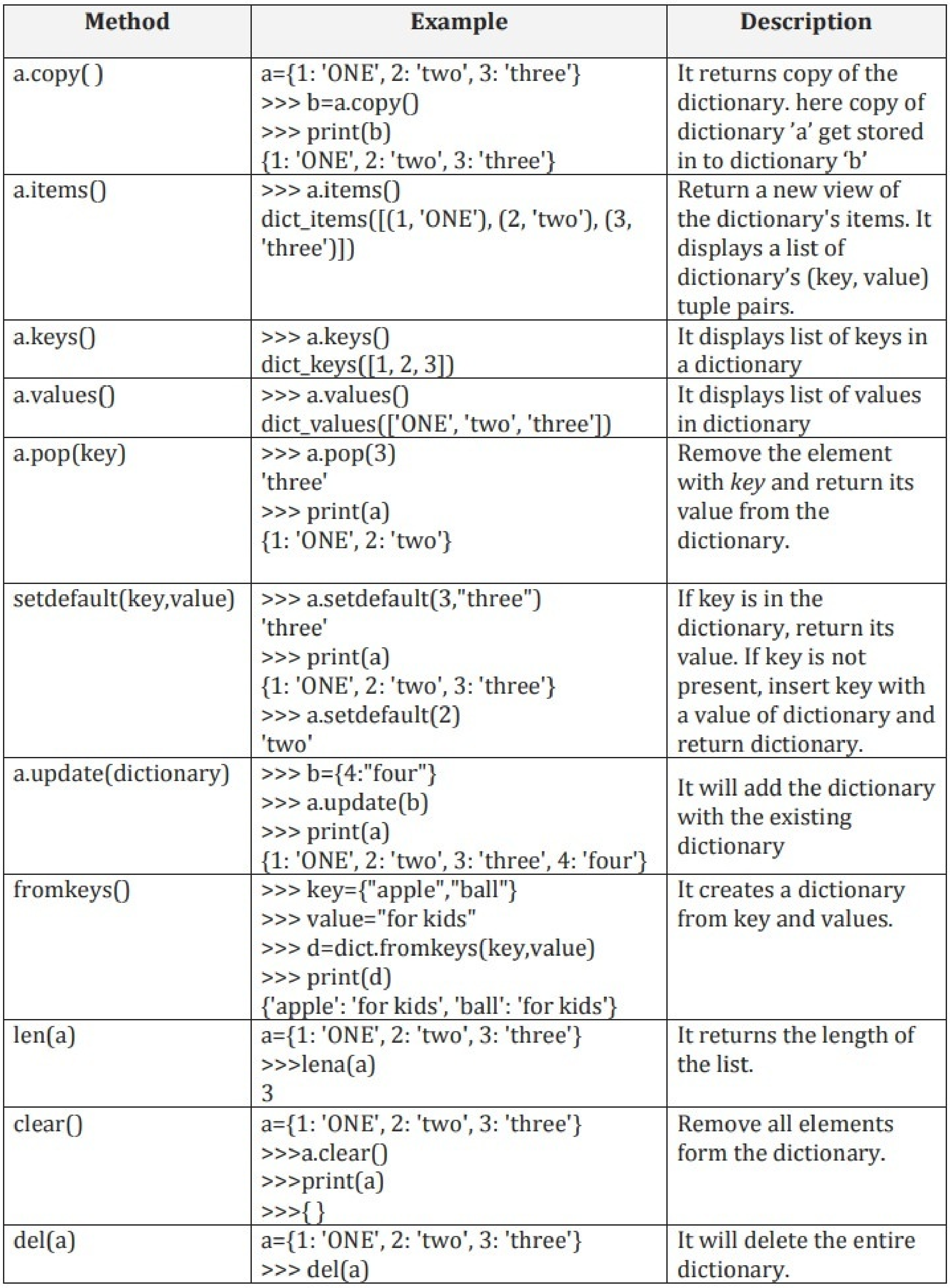
In [27]:
```python
#7
for value in car.values():
    print(value)
```

```
honda
778z
15000
Hybrid
```

In [28]:
```python
#8
for item in car.items():
    print(item)
```

```
('brand', 'honda')
('model', '778z')
('price', 15000)
('engine', 'Hybrid')
```

Lets again see: Method in Dictionary, List and Tuples (To revise)



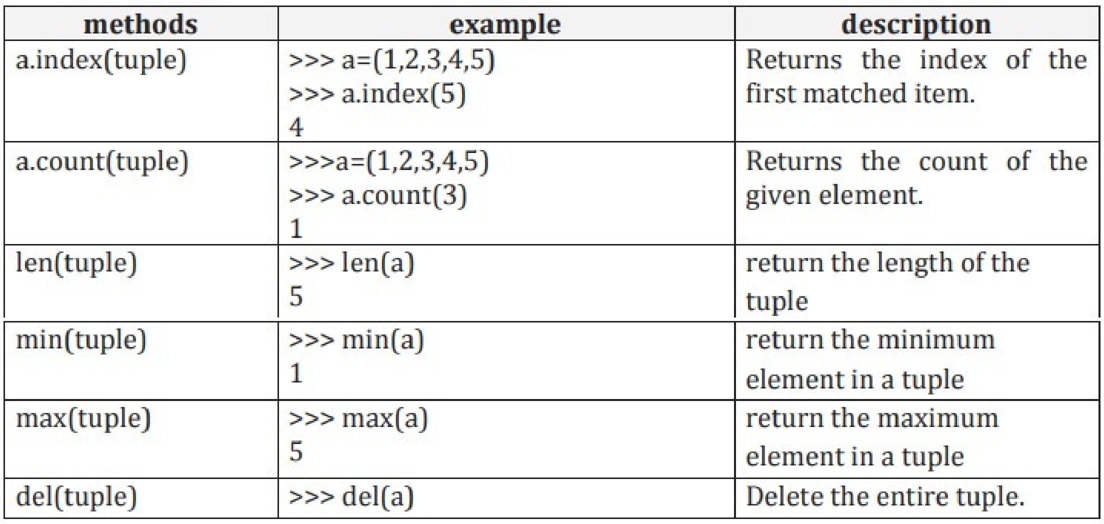
| Method | Example | Description |
|---|---|---|
| a.copy( ) | a={1: 'ONE', 2: 'two', 3: 'three'}<br>>>> b=a.copy()<br>>>> print(b)<br>{1: 'ONE', 2: 'two', 3: 'three'} | It returns copy of the dictionary. here copy of dictionary 'a' get stored in to dictionary 'b' |
| a.items() | >>> a.items()<br>dict_items([(1, 'ONE'), (2, 'two'), (3, 'three')]) | Return a new view of the dictionary's items. It displays a list of dictionary's (key, value) tuple pairs. |
| a.keys() | >>> a.keys()<br>dict_keys([1, 2, 3]) | It displays list of keys in a dictionary |
| a.values() | >>> a.values()<br>dict_values(['ONE', 'two', 'three']) | It displays list of values in dictionary |
| a.pop(key) | >>> a.pop(3)<br>'three'<br>>>> print(a)<br>{1: 'ONE', 2: 'two'} | Remove the element with *key* and return its value from the dictionary. |
| setdefault(key,value) | >>> a.setdefault(3,"three")<br>'three'<br>>>> print(a)<br>{1: 'ONE', 2: 'two', 3: 'three'}<br>>>> a.setdefault(2)<br>'two' | If key is in the dictionary, return its value. If key is not present, insert key with a value of dictionary and return dictionary. |
| a.update(dictionary) | >>> b={4:"four"}<br>>>> a.update(b)<br>>>> print(a)<br>{1: 'ONE', 2: 'two', 3: 'three', 4: 'four'} | It will add the dictionary with the existing dictionary |
| fromkeys() | >>> key={"apple","ball"}<br>>>> value="for kids"<br>>>> d=dict.fromkeys(key,value)<br>>>> print(d)<br>{'apple': 'for kids', 'ball': 'for kids'} | It creates a dictionary from key and values. |
| len(a) | a={1: 'ONE', 2: 'two', 3: 'three'}<br>>>>lena(a)<br>3 | It returns the length of the list. |
| clear() | a={1: 'ONE', 2: 'two', 3: 'three'}<br>>>>a.clear()<br>>>>print(a)<br>>>>{ } | Remove all elements form the dictionary. |
| del(a) | a={1: 'ONE', 2: 'two', 3: 'three'}<br>>>> del(a) | It will delete the entire dictionary. |

Dict method:

| | syntax | example | description |
|---|---|---|---|
| 1 | a.append(element) | >>> a=[1,2,3,4,5]<br>>>> a.append(6)<br>>>> print(a)<br>[1, 2, 3, 4, 5, 6] | Add an element to the end of the list |
| 2 | a.insert(index,element) | >>> a.insert(0,0)<br>>>> print(a)<br>[0, 1, 2, 3, 4, 5, 6] | Insert an item at the defined index |
| 3 | a.extend(b) | >>> b=[7,8,9]<br>>>> a.extend(b)<br>>>> print(a)<br>[0, 1, 2, 3, 4, 5, 6, 7, 8,9] | Add all elements of a list to the another list |
| 4 | a.index(element) | >>> a.index(8)<br>8 | Returns the index of the first matched item |
| 5 | a.sort() | >>> a.sort()<br>>>> print(a)<br>[0, 1, 2, 3, 4, 5, 6, 7, 8] | Sort items in a list in ascending order |
| 6 | a.reverse() | >>> a.reverse()<br>>>> print(a)<br>[8, 7, 6, 5, 4, 3, 2, 1, 0] | Reverse the order of items in the list |

List method:

| | | | |
|---|---|---|---|
| 7 | a.pop() | >>> a.pop()<br>0 | Removes and returns an element at the last element |
| 8 | a.pop(index) | >>> a.pop(0)<br>8 | Remove the particular element and return it. |
| 9 | a.remove(element) | >>> a.remove(1)<br>>>> print(a)<br>[7, 6, 5, 4, 3, 2] | Removes an item from the list |
| 10 | a.count(element) | >>> a.count(6)<br>1 | Returns the count of number of items passed as an argument |
| 11 | a.copy() | >>> b=a.copy()<br>>>> print(b)<br>[7, 6, 5, 4, 3, 2] | Returns a shallow copy of the list |
| 12 | len(list) | >>> len(a)<br>6 | return the length of the length |
| 13 | min(list) | >>> min(a)<br>2 | return the minimum element in a list |
| 14 | max(list) | >>> max(a)<br>7 | return the maximum element in a list. |
| 15 | a.clear() | >>> a.clear()<br>>>> print(a)<br>[ ] | Removes all items from the list. |
| 16 | del(a) | >>> del(a)<br>>>> print(a)<br>Error: name 'a' is not defined | delete the entire list. |



| methods | example | description |
|---|---|---|
| a.index(tuple) | >>> a=(1,2,3,4,5)<br>>>> a.index(5)<br>4 | Returns the index of the first matched item. |
| a.count(tuple) | >>>a=(1,2,3,4,5)<br>>>> a.count(3)<br>1 | Returns the count of the given element. |
| len(tuple) | >>> len(a)<br>5 | return the length of the tuple |
| min(tuple) | >>> min(a)<br>1 | return the minimum element in a tuple |
| max(tuple) | >>> max(a)<br>5 | return the maximum element in a tuple |
| del(tuple) | >>> del(a) | Delete the entire tuple. |

Tuple method:

```
In [ ]:
```