Chapter 3 Lists, Tuples, Sets and Dictionaries

❖ There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered and changeable. No duplicate members.

- (List) .
- (Tuple) .
- (Set) , .
- (Dictionary) .

3.1 Lists and operations on Lists

List: It is used to store the sequence of various types of data.

Create Python Lists

✓ In Python, a list is created by placing elements inside square brackets [], separated by commas.

Example:

```
y = [3, 4, 5]
print(y)
```

Output:

```
[1, 2, 3]
```

✓ A list can have any number of items and they may be of different types (integer, float, string, etc.).

Example:

```
a = []
a = [8, "Harsh", 5.7,4]
print(a)
```

Output:

```
[8, Harsh, 5.7,4]
```

List operations:

✓ operations include
1) indexing,
2) slicing,
3) adding,
4) multiplying,
5) checking for membership.

use + operator to combine two lists which is called concatenation.

✓ The * operator repeats a list for the given number of times.

Access List Elements

✓ There are various ways in which we can access the elements of a list.

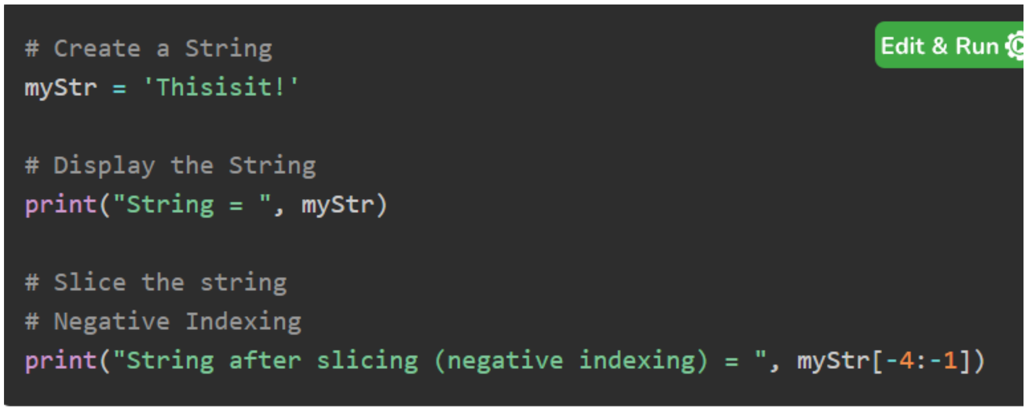
List Index

✓ We can use the index operator [ ] to access an item in a list. In Python, indices start at 0. So, a list having 5 elements will have an index from 0 to 4.

```
x = ['S', 'L', 'T', 'I',
'E','T']
                                  # first character: h
print(x[0])
print(x[2])                        # third character: l

print(x[4])                        # first character: o

# Nested List
x_list = ["Happy", [2, 0, 1, 3]]

print(x_list[0][1])                       #a

print(x_list[1][3])                       #3
```

Negative indexing

✓ Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.



```
# Create a String
myStr = 'Thisisit!'

# Display the String
print("String = ", myStr)

# Slice the string
# Negative Indexing
print("String after slicing (negative indexing) = ", myStr[-4:-1])
```

List Slicing

- ✓ We can access a range of items in a list by using the slicing operator.
- ✓ We can get sublist of the list using syntax:

List(start:stop:step)

Where start –starting index of the list Stop

—last index of the list

Step—skip the nth element within start and stop.

x=[2,4,6,8] print(x[:]) #Output: [2, 4, 6, 8] print(x[1:]) # Output: [4, 6, 8] print(x[:2]) #Output: [2, 4] print(x[1:3]) #Output: [4, 6]	x=[2,4,6,8] print(x[-1]) #Output:8 print(x[-3:]) #Output: [4, 6, 8] print(x[:-2]) #Output: [2, 4] print(x[-3:-1]) #Output: [4, 6]

```
x = ['h','e','l','l','0','w','o','r','l','d']

print(x[2:5])          # elements from index 2 to index 4 : ['l', 'l', '0']

print(x[5:])           # elements from index 5 to end: ['w', 'o', 'r', 'l', 'd']

print(x[:])           # elements beginning to end:['h', 'e', 'l', 'l', '0', 'w', 'o', 'r', 'l', 'd']
```

```
x=[10,20,30,40,50,60,70]
print(x[1:6:2])                    #Out put: [20, 40, 60]
```

Add/Change List Elements

- ✓ Lists are mutable, meaning their elements can be changed unlike string or tuple.
- ✓ We can use the assignment operator = to change an item or a range of items.

```
x = [2, 4, 6, 8]

x[0] = 1                    # change the 1st item    : [1, 4, 6, 8]
print(x)

x[1:4] = [3, 5, 7]          # change 2nd to 4th items : [1, 3, 5, 7]

print(x)
```

Append( ) and extend( ):

- ✓ We can add one item to a list using the append() method or add several items using the extend() method.

```
x = [1, 3, 5]

x.append(7)
print(x)                        Output: [1, 3, 5, 7]

x.extend([9, 11, 13])
print(x)                        Output: [1, 3, 5, 7, 9, 11, 13]
```

Insert( )

- ✓ For inserting in the list

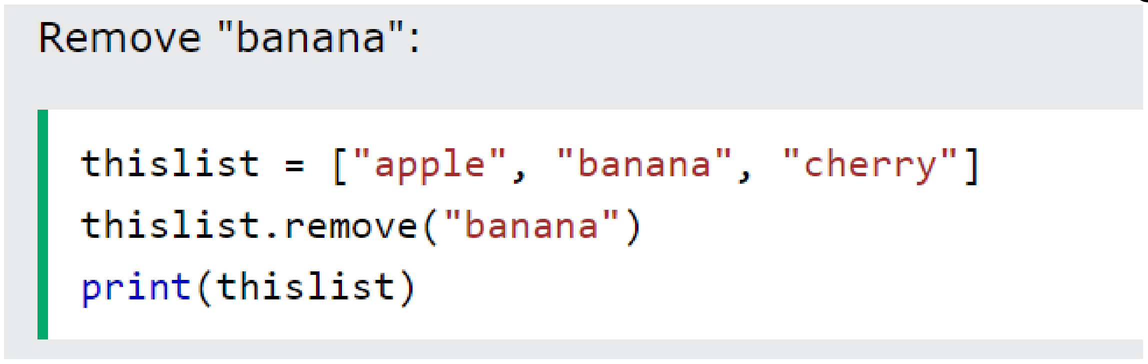
```
x = [1, 9]
x.insert(1,3)

print(x)                        Output: [1, 3, 9]

x[2:2] = [5, 7]

print(x)                        Output: [1, 3, 5, 7, 9]
```

Delete

- ✓ For deleting from the list

Remove( ) and pop( )

- ✓ We can use remove() to remove the given item or pop() to remove an item at the given index.



Remove "banana":

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

```
x = ['d','i','s','c','u','s','s']
x.remove('p')

print(x)                          # Output: ['d', 'i', 'c', 'u', 's', 's']

print(x.pop(1))

print(x)                          # Output:

print(x.pop())

print(x)                        # Output: ['r', 'b', 'l', 'e']

x.clear()
```

```
print(x)                        # Output: []
```

Iterating Through a List

✓ Using a for loop we can iterate through each item in a list.

```
for name in ['Yesha','Harsh','Kunal']:
        print(name)
Output:
          Yesha
          Harsh
          Kunal
```

Reverse a List: The reverse () method reverses the elements of the list.

x = ['h', 'e', 'l', 'l', 'o']
x.reverse()
print(x)
	Output:
	['o', 'l', 'l', 'e', 'h']

Sort List: The sort() method sorts the items of a list in ascending or descending order.

Python List Built in Function:

Function	Description	Example
len(list)	It is used to calculate the length of the list.	x=[10,20,30,40,50,60,70] print(len(x)) #Output:7
max(list)	It returns the maximum element of the list	x=[10,20,30,40,50,60,70] print(max(x)) #Output:70
min(list)	It returns the maximum element of the list	x=[10,20,30,40,50,60,70] print(min(x)) #Output: 10

list(seq)	It converts any sequence to the list.	str="abcd"
		print(type(str)) # <class 'str'>
		s=list(str)
		print(type(s)) # <class 'list'>

3.2 Tuples and operations on Tuples

Tuple: A tuple in Python is similar to a list.
The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

Characteristics of Tuple:

✓ It is **ordered**.
✓ It allows **duplicate** members.

It is **immutable** type(We cannot **modify** its element after it is created)

✓ Elements of Tuple can access by index.
✓ To use a tuple, you must declare it first. It is declare using ( ) brackets and separate values with commas.
✓ A tuple can also be created without using parentheses. This is known as tuple packing

Creating a Tuple:

We can construct tuple in many ways:

```
X=()        #no item tuple
X=(1,2,3)
X=tuple(list1)
X=1,2,3,4
```

Example:

Example 1:	Example 2:

>>> x=(1,2,3)
>>> print(x)
Output: (1, 2, 3)
>>> x
Output: (1, 2, 3)
	>>> x=()
	>>> x
	Output: ()

Example 3:
>>> x=[4,5,66,9]
>>> y=tuple(x) #type cast list to tuple
>>> y
Output: (4, 5, 66, 9)
	Example 4:
	>>> x=1,2,3,4
	>>> x
	Output: (1, 2, 3, 4)

Example 5: # tuple with mixed datatypes >>>x=(2,"SLTIET",3.14) >>>print(x) Output: (2, 'SLTIET', 3.14)	**Example 6: nested tuple** >>>x=("hello",['a','b','c'],(4,5,6)) >>>x Output: ('hello', ['a', 'b', 'c'], (4, 5, 6))

✓ A tuple without using parentheses is known as tuple packing.

```
x = 2,4.16,"world"
print(x)                    # (1,3.14,'hello')
a, b, c = x                # tuple unpacking #
print(a)                    1
print(b)                    #3.14
print(c)                    # hello
```

Operations on Tuples

✓ Access tuple items(indexing, negative indexing and Slicing)
✓ Change tuple items
✓ Loop through a tuple
✓ Count()
✓ Length()

Access tuple items(Indexing):

✓ We can use the index operator [] to access an item in a tuple, where the index starts from 0.
✓ So, a tuple having 6 elements will have indices from 0 to 5.

Example:
>>> x=('S','L','T','I','E','T')
>>> print(x[2])
 	
Output: T
	Example:
	x = ('SLTIET', [8, 4, 6], (1, 2, 3)) # nested tuple
	print(x[0][3]) # nested index : l
	print(x[1][1]) # nested index : 4
	
	Output:
	I
	4

Negative Indexing: Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```
x = ('S', 'L', 'T', 'I', 'E','T')
print(x[-1])
print(x[-5])
print(x[-6])
print(x[-7])
Output:
T
L
S
Traceback (most recent call last):
File "<string>", line 5, in <module>
```

Slicing
We can access a range of items in a tuple by using the slicing operator colon:

```
x=(2,4,6,8)
print(x[:])          #Output: (2, 4, 6, 8)
print(x[1:])         # Output: (4, 6, 8)
print(x[:2])          #Output:  (2, 4)
print(x[1:3])         #Output:  (4, 6)
```

```
x=(2,4,6,8)
print(x[-1])          #Output:8
print(x[-3:])        #Output: (4, 6, 8)
print(x[:-2])        #Output: (2, 4)
print(x[-3:-1])       #Output: (4, 6)
```

```
x = ('S','L','T','I','E','T')

print(x[1:4])
print(x[:-7])
print(x[7:])
print(x[:])

Output:
('L', 'T', 'I')
()
()
('S','L','T','I','E','T')
```

Change tuple items: Tuples are unchangeable / immutable.

```
>>> x=(1,3,8,'4',9,11,34,55,664,728)
>>> x[1]=10
```

```
Output:
Traceback (most recent call last):
File "<pyshell#41>", line 1, in <module> x[1]=10
```

Example:

```
x = (4, 2, 3, [6, 5])
x[3][0] = 9
print(x) #here it got changed as the updated is the list

x = ('S', 'L', 'T', 'I', 'E',’T’)
print(x)
```

Output:

```
(4, 2, 3, [9, 5])
('S', 'L', 'T', 'I', 'E',’T’)
```

Loop through a tuple(Iteration): We can loop the values of tuple using for loop

```
>>> x=5,4,7,8,3,'aa'
>>> for i in x:
        print(i)
Output:
        5
        4
        7
        8
        3
        aa
```

Length (): To know the number of items or values present in a tuple, we use len().

```
>>> x=(1,2,3,4,5,6,2,10,2,11,12,2)
>>> y=len(x)
>>> print(y)
Output: 12
```

Concatenation: We can use + operator to combine two tuples. This is called concatenation.

- ✓ We can also repeat the elements in a tuple for a given number of times using the * operator.
- ✓ Both + and * operations result in a new tuple.

Example:

```
print((1, 2, 3) + (4, 5, 6))
print(("Repeat",) * 3)
```

Output:

```
(1, 2, 3, 4, 5, 6)
('Repeat', 'Repeat', 'Repeat')
```

Deleting a Tuple: we cannot change the elements in a tuple.

Example:

```
x = (‘Hello’,’How are you?’,3)
del x
print(x)
Output:
Traceback (most recent call last):
  File "<string>", line 8, in <module>
NameError: name 'x' is not defined
```

Tuple Methods

```
x = ('s', 'l', 't', 'i', 'e')
print(x.count('s'))
print(x.index('l'))
```

Output:

```
1
1
```

Python List Built in Function:

Function	Description	Example
len(tuple)	It is used to calculate the length of the tuple.	x=(70,90,30,40,50,60,10) print(len(x)) #Output:7
max(tuple)	It returns the maximum element of the tuple	x=(10,20,30,40,50,60,70) print(max(x)) #Output:70
min(tuple)	It returns the maximum element of the tuple	x=(0,20,30,40,50,60,7) print(min(x)) #Output: 0

tuple (seq)	It converts any sequence to the tuple.	str="abcd"
		print(type(str)) # <class 'str'>
		s=tuple(str)
		print(type(s)) # <class ‘tuple’>

Where to use tuple?

Tuple data is constant and must not be changed so when there is a requirement of read only data tuple is preferable.

Tuple can simulate a dictionary without key. For example:
[(10,"abc",22), (20,"def",34), (30,"xyz",56)]

3.3 Sets and operations on Sets

A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).

Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.

Characteristics of Set:

Sets are unordered.
Set element is unique. Duplicate elements are not allowed.
Set are immutable. (Cannot change value)
There is no index in set. So they don't support indexing or slicing operator.
The set are used for mathematical operation like union, intersection, difference etc.

Creating Python Sets

A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function.

It can have any number of items and they may be of different types (integer, float, tuple, string etc.).

To make a set without any elements, we use the set() function without any argument.

Example:

Example 1:
s = {1, 2, 3}
print(s)
 	
Output:
{1, 2, 3}
	Example 2:
	# set of mixed datatypes
	x = {1.0, "SLTIET", (1, 2, 3)}
	print(x)
	
	Output:
	{1.0, (1, 2, 3), 'SLTIET'}

Example 3:
 	
# set cannot have duplicates 	
x = {1, 2, 3, 4, 3, 2}
print(x)
 	
Output:
{1, 2, 3, 4}
	Example 4:
	
	x = set([1, 2, 3, 2])
	print(x)
	
	Output:
	{1, 2, 3}

Example 5: x= set() print(type(x)) Output: <class 'set'>	

Access value in a Set: We cannot access individual values in a set. We can only access all the elements together.

```
x = {1, 2, 3, 4, 3, 2}
for i in x:
    print(I,end=' ')
```

Output: 1 2 3 4

Modifying a set in Python

We can add a single element using the add() method, and multiple elements using the update() method. The update() method can take tuples, lists, strings orother sets as its argument. In all cases, duplicates are avoided.

x = {1, 3}
print(x)
 	
x.add(2)
print(x)
 	
x.update([2, 3, 4])
print(x)
 	
x.update([4, 5], {1, 6, 8})
print(x)
	Output:
	{1, 3}
	
	{1, 2, 3}
	
	{1, 2, 3, 4}
	
	{1, 2, 3, 4, 5, 6, 8}

Removing elements from a set

- ✓ A particular item can be removed from a set using the methods discard() and remove().
- ✓ The only difference between the two is that **the discard() function leaves a set unchanged if the element is not present in the set**. On the other hand, the **remove() function will raise an error** in such a condition (if element is not present in the set).

x = {1, 3, 4, 5, 6}
print(x)
 	
x.discard(4)
print(x)
 	
x.remove(6)
print(x)
 	
x.discard(2)
print(x)
 	
x.remove(2)
	Output:
	{1, 3, 4, 5, 6}
	
	{1, 3, 5, 6}
	
	{1, 3, 5}
	
	{1, 3, 5}
	
	Traceback (most recent call last):
	File "<string>", line 28, in <module>
	KeyError: 2

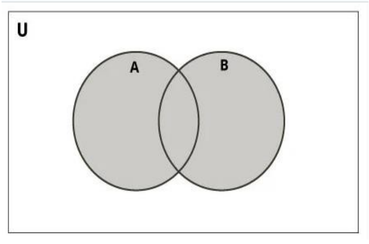
- ✓ Similarly, we can remove and return an item using the pop() method.
- ✓ Since set is an unordered data type, there is no way of determining which item will be popped.
- ✓ We can also remove all the items from a set using the clear() method.

x = set("abcdefgh")
print(x)
 	
print(x.pop())
 	
print(x.pop())
 	
print(x)
x.clear()
print(x)
	Output:
	{'d', 'c', 'b', 'a', 'f', 'h', 'e', 'g'}
	
	d
	
	c
	
	{'b', 'a', 'f', 'h', 'e', 'g'}
	set()

Python Set Operations

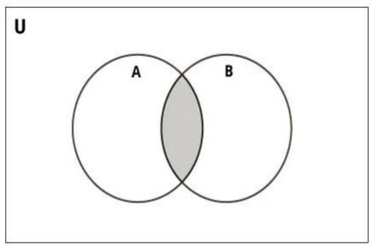
- ✓ Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

Set Union: Union of A and B is a set of all elements from both sets. Union is performed using | operator. Same can be accomplished using the union() method.

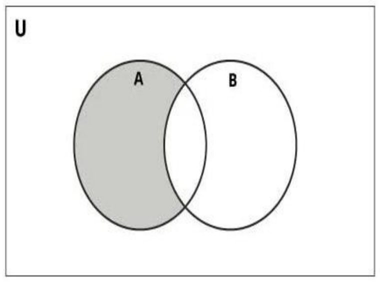
	A = {1, 2, 3, 4, 5}
	B = {4, 5, 6, 7, 8}
	print(A | B)
	
	Output:
	{1, 2, 3, 4, 5, 6, 7, 8}
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		print(A.union(B))
		print(B.union(A))
		
		Output:
		{1, 2, 3, 4, 5, 6, 7, 8}
		{1, 2, 3, 4, 5, 6, 7, 8}

Set Intersection: Intersection of A and B is a set of elements that are common in both the sets.

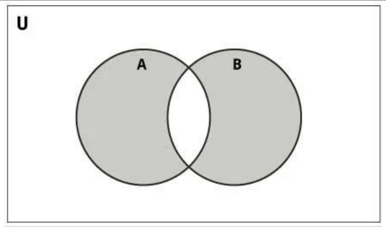
Intersection is performed using & operator. Same can be accomplished using the intersection() method.

	A = {1, 2, 3, 4, 5}
	B = {4, 5, 6, 7, 8}
	print(A & B)
	
	Output: {4, 5}
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		print(A.intersection(B))
		print(B.intersection(A))
		
		Output:
		{4, 5}
		{4, 5}

Set Difference: Difference of the set B from set A(A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of elements in B but not in A. Difference is performed using - operator. Same can be accomplished using the difference() method.

	A = {1, 2, 3, 4, 5}
	B = {4, 5, 6, 7, 8}
	print(A - B)
	
	Output: {1,2,3}
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		print(A.difference(B))
		print(B.difference(A))
		
		Output:
		{1,2,3}
		{6,7,8}

Set Symmetric Difference: Symmetric Difference of A and B is a set of elements in A and B but not in both (excluding the intersection).Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric_difference().

	A = {1, 2, 3, 4, 5}
	B = {4, 5, 6, 7, 8}
	print(A ^ B)
	
	Output:{1,2,3,6,7,8}
		A = {1, 2, 3, 4, 5}
		B = {4, 5, 6, 7, 8}
		print(A.symmetric_difference(B))
		print(B.symmetric_difference(A))
		
		Output: {1,2,3,6,7,8}
		{1,2,3,6,7,8}

Set Membership Test: We can test if an item exists in a set or not, using the in keyword.

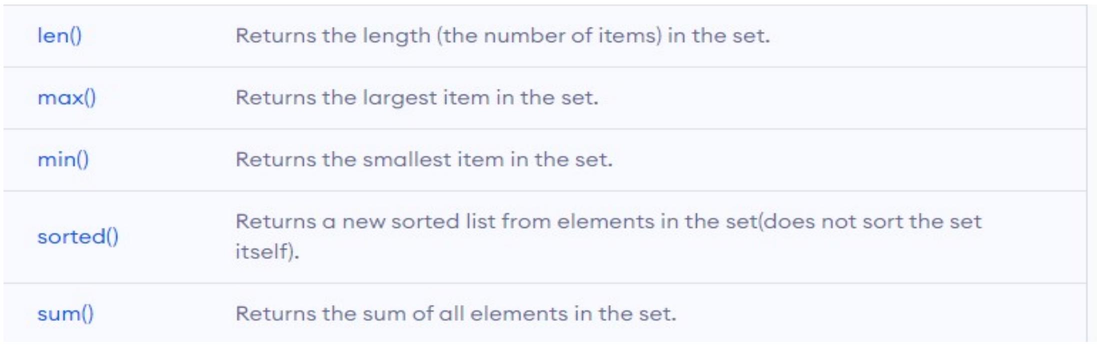
x = set("apple")
 	
print('a' in x)
 	
print('p' not in x)
	Output:
	True
	False

Iterating Through a Set: We can iterate through each item in a set using a for loop.
Example:

for letter in set("apple"): print(letter)	Outp ut: a p p l

	e

Built-in Functions with Set



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.

3.4 Dictionaries and operations on Dictionaries

Python dictionary is an ordered collection of items. Each item of a dictionary has a key/value pair.

Characteristics of Dictionary:

- It is used to store data in a key-value pair format.
- It is mutable / changeable.
- Duplicate values are not allowed.
- Key must be a single element.
- Value can be of any type such as list, tuple, integer etc.

Creating Python Dictionary: By using {} we can create dictionary.
An item has a key and a corresponding value that is expressed as a pair (key: value). We can also create a dictionary using the **built-in dict() function**.

	Output

x = {}
print(x)
	{}

x = {1: 'abc', 2: 'def'} # dictionary with integer keys
print(x)
	{1: 'abc', 2: 'def'}

x = {'name': 'abc', 1: 'def'}
print(x)
	{'name': 'abc', 1: 'def'}

x = dict({1:'abc', 2:'def'})   # using dict()
print(x)
	{1: 'abc', 2: 'def'}

x = dict([(1,'abc'), (2,'def')])	{1: 'abc', 2: 'def'}

Accessing Elements from Dictionary:

While indexing is used with other data types to access values, a dictionary uses keys. Keys can be used either inside square brackets [] or with the get() method.

x = {'name': 'abc', 'rollno': 1}
 	
print(x['name'])
print(x.get('rollno'))
print(x.get('address'))   # Output None
 	
print(x['address']) # Trying to access keys which doesn't exist throws error
	Output
	
	abc
	1
	None
	
	Traceback (most recent call last):
	  File "<string>", line 9, in <module>
	KeyError: 'address'

Changing and Adding Dictionary elements:

- ✓ We can add new items or change the value of existing items using an **assignment operator**.
- ✓ If the key is already present, then the existing value gets updated. In case the key is not present, a new (key: value) pair is added to the dictionary.

x = {'name': 'abc', 'rollno': 1}
print(x)
 	
x['rollno'] = 20   # update value
print(x)
 	
x['address'] = 'xyz'   # adding value
print(x)
	Output
	{'name': 'abc', 'rollno': 1}
	
	{'name': 'abc', 'rollno': 20}
	
	{'name': 'abc', 'rollno': 20, 'address': 'xyz'}

Removing elements from Dictionary:

- ✓ We can remove a particular item in a dictionary by using the **pop() method**. This method removes an item with the provided key and returns the value.
- ✓ The **popitem()** method can be used to remove and return an arbitrary (key, value) item pair from the dictionary.
- ✓ All the items can be removed at once, using the **clear()** method.
- ✓ We can also use the **del keyword** to remove individual items or the entire dictionary itself.

# Example for pop() 	
my_dict = {'a': 1, 'b': 2, 'c': 3}
value = my_dict.pop('b')
print(value) # Output: 2

# Example for popitem() 	
my_dict = {'a': 1, 'b': 2, 'c': 3}
key, value = my_dict.popitem()
print(key, value) # Output: ('c', 3)

# Example for clear() 	
my_dict = {'a': 1, 'b': 2, 'c': 3}
my_dict.clear()
print(my_dict)
# Output: {} 	

# Example for del to delete the entire dictionary 	
my_dict = {'a': 1, 'b': 2, 'c': 3}
del my_dict
# Raises 	
NameError if
you try to
access
my_dict
beyond this
point

Credits/Refrences for the content:

https://geeksforgeeks.org/
https://w3schools.com/python/
https://www.python.org/
https://vpmpce.wordpress.com/