Python Programming Basics

Agenda

01 Data Types

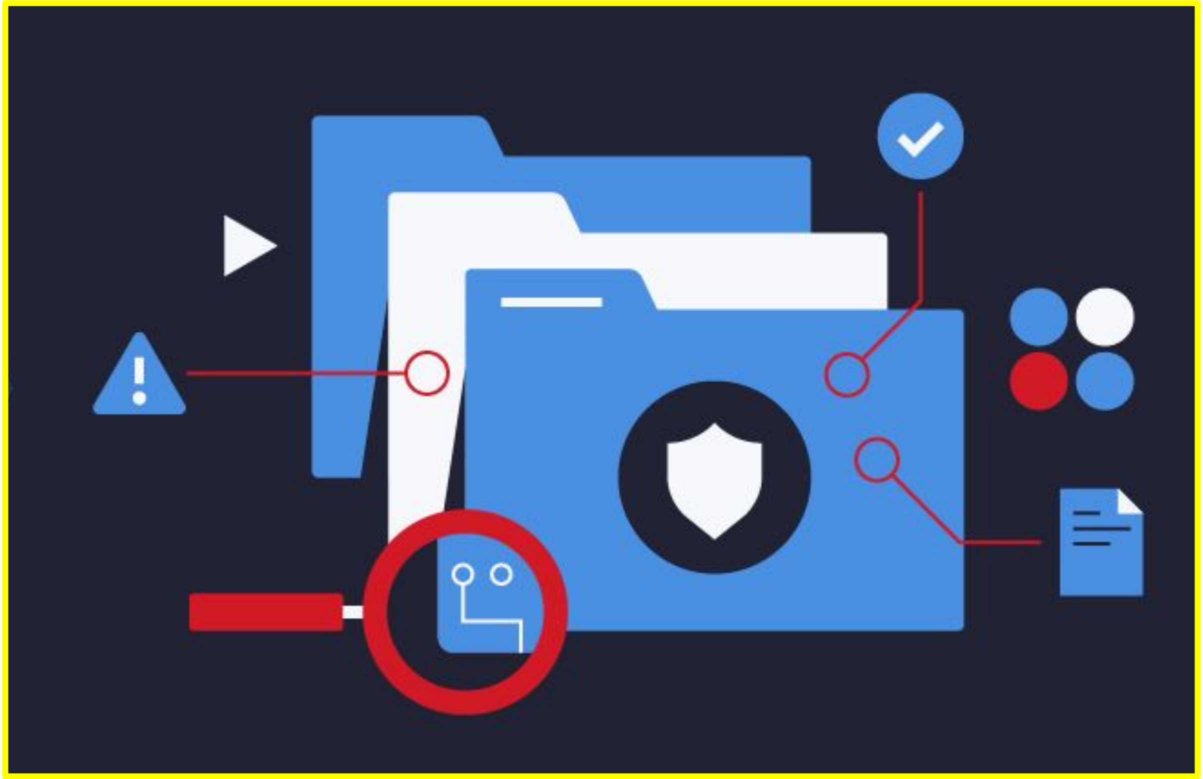
02 Mutable VS Immutable Data Types

03 Slicing in Python

Data Types

What is a Data Type?

The **classification** or **categorization** of **data elements** is referred to as **Data Types**. It represents the kind of value that tells what operations can be performed on a particular data.



What is a Data Type?

Since everything in **Python programming** is an **object**, **data types** are actually **classes**, and **variables** are the **instances of these classes**.

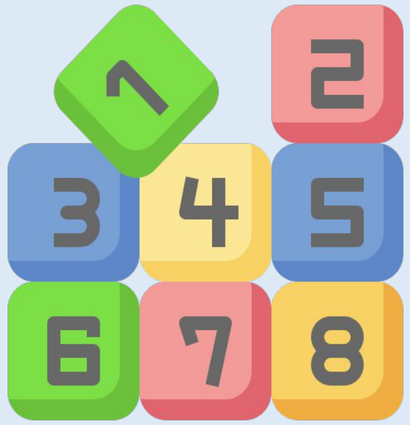
What is a Data Type?

1. Numeric Data Types
2. Sequence Types
3. Boolean Data Types

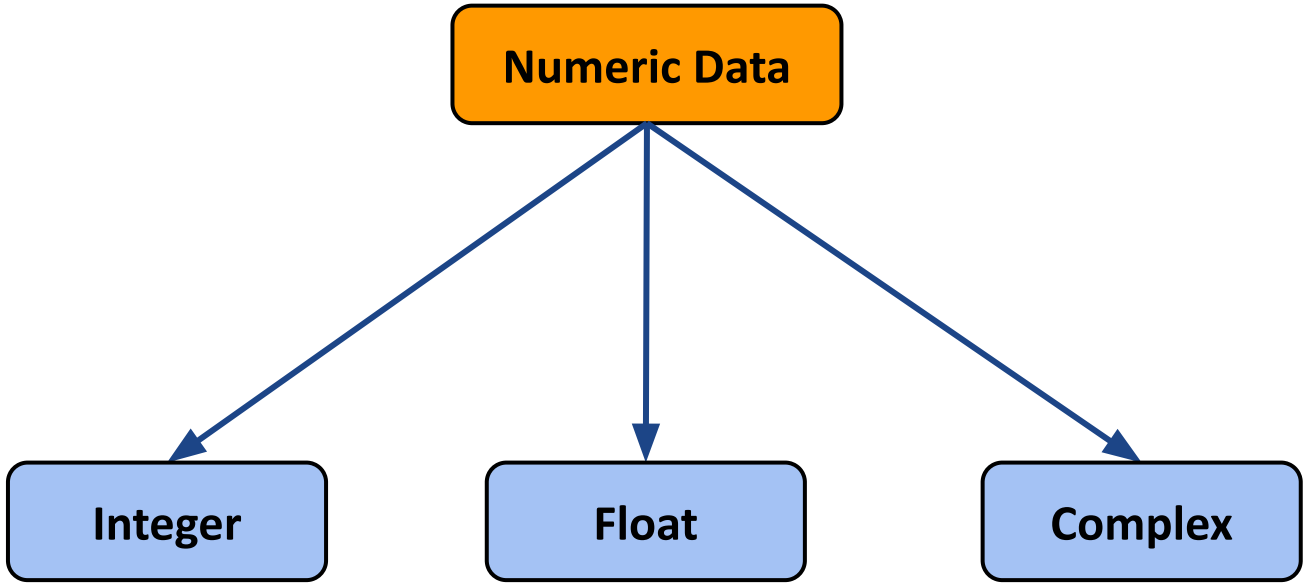
Numeric Data Types

Numeric Data Types

Numeric Data Types

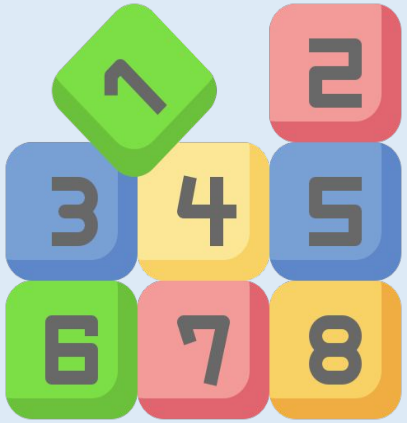


In Python, **numeric data type** represent the **data which has numeric value**.



Numeric Data Types

Numeric Data Types

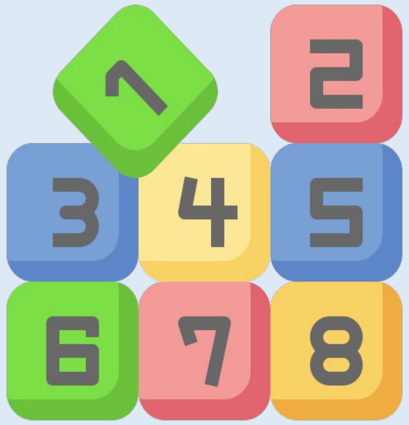


Integer

1. Integer values are represented by **'int'** class.
1. Contains Positive or negative numbers.
1. Covers whole numbers. (unsupported for decimal or fractional numbers)
1. There is no limit to how long an integer value can be.

Numeric Data Types

Numeric Data Types



Float

1. Float values are represented by **‘float’** class.
1. Real numbers with floating representation.
1. Specified by decimal points.

Numeric Data Types

Numeric Data Types

Complex

1. Complex numbers are represented by ‘complex’ class.

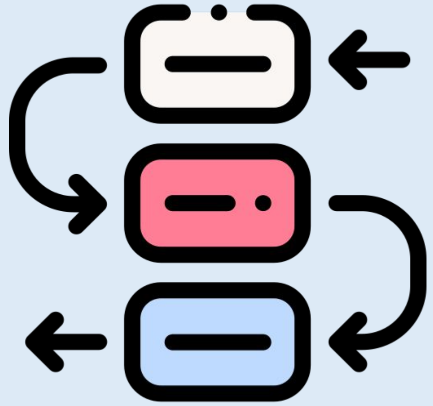
1. It is specified as (real part) + (imaginary part)j.

1. Example, c = 2 + 4j.

Sequence Data Types

Sequence Data Types

Sequence Data Types



In Python, **sequence** is the **ordered collection** of similar or different data types.

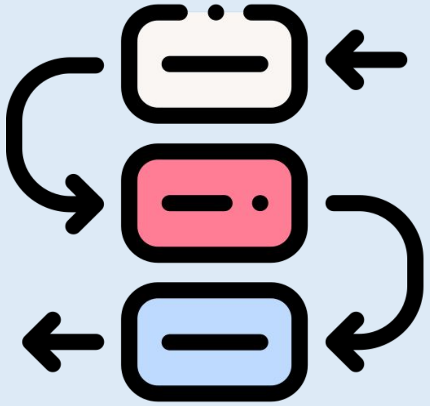
Sequence Data Types

01 String	02 List
03 Tuple	04 Set
05 Dictionary	

String

String

Sequence Data Types



String

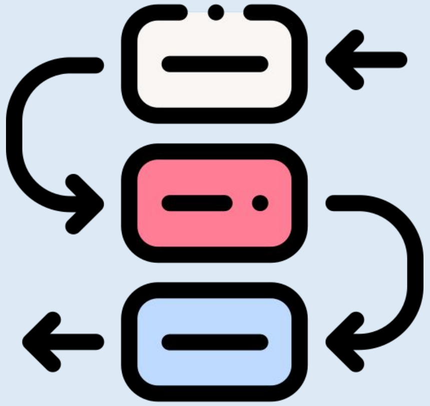
1. A string is a collection of one or more characters put in a single quote, double-quote or triple quote.

1. In python there is no character data type, a character is a string of length one.

1. Initialization: String1 = 'Intellipaat'

String

Sequence Data Types



String Methods

1. count() - Returns occurrences of specified character

1. split() -Splits the string using specified separator

1. join() - Connects different string objects

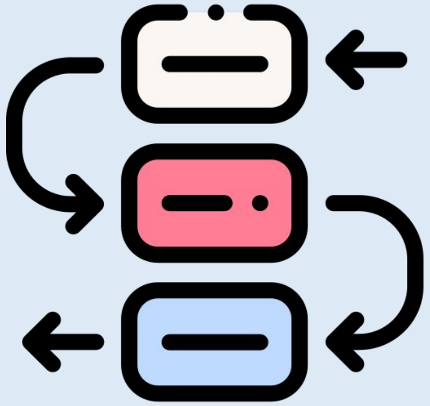
1. find() - searches the string for specified value

1. replace() - Returns string by replacing specified character with another character

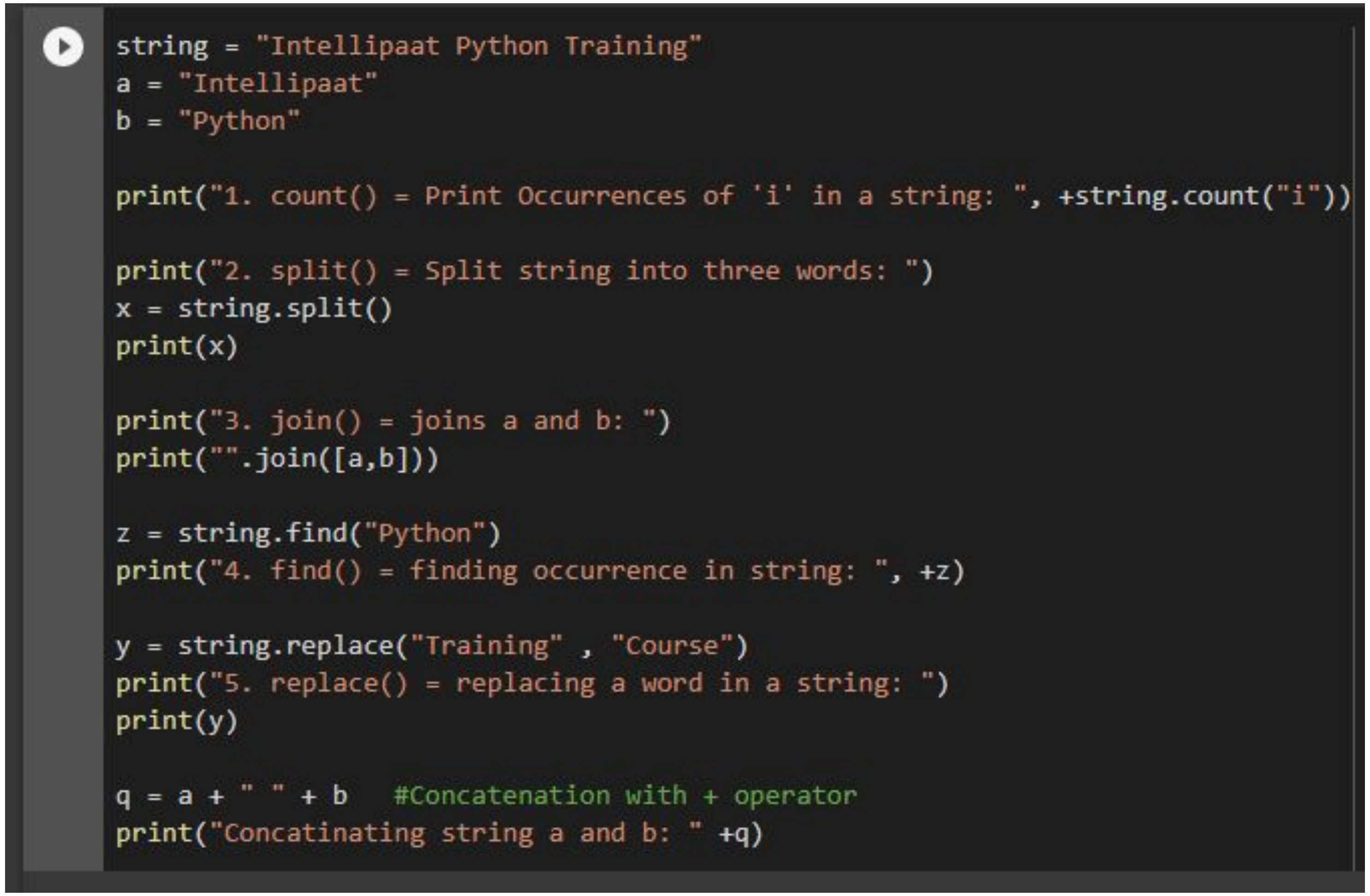
1. String Concatenation In Python - combining two different strings

String

Sequence Data Types



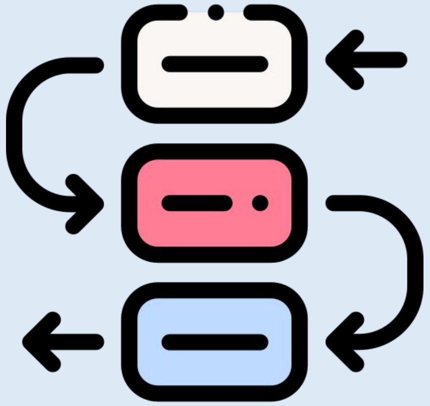
Implementing String Methods



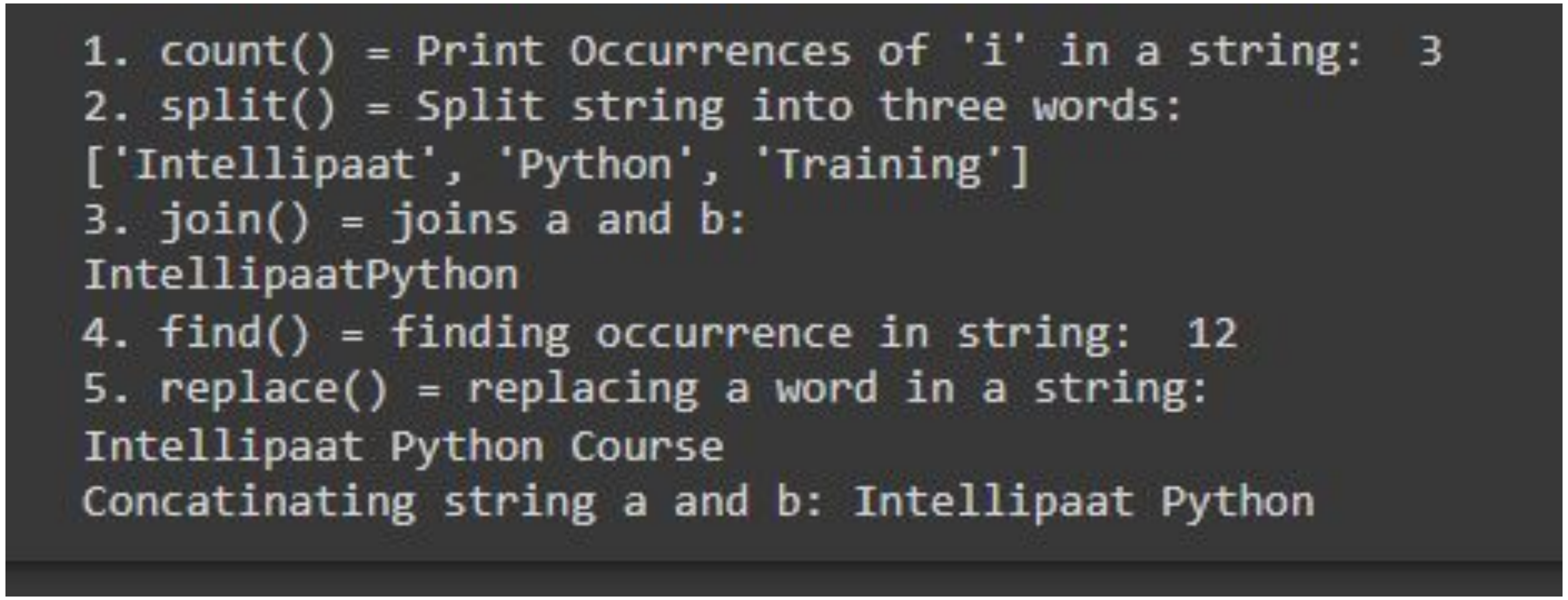
```
string = "Intellipaat Python Training"
a = "Intellipaat"
b = "Python"

print("1. count() = Print Occurrences of 'i' in a string: ", +string.count("i"))

print("2. split() = Split string into three words: ")
x = string.split()
print(x)

print("3. join() = joins a and b: ")
print("".join([a,b]))

z = string.find("Python")
print("4. find() = finding occurrence in string: ", +z)

y = string.replace("Training" , "Course")
print("5. replace() = replacing a word in a string: ")
print(y)

q = a + " " + b   #Concatenation with + operator
print("Concatinating string a and b: " +q)
```

String

Sequence Data Types



Result

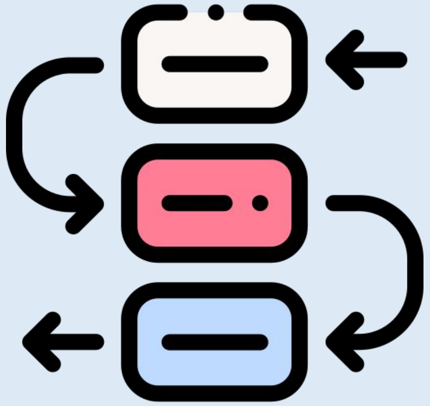


```
1. count() = Print Occurrences of 'i' in a string:  3
2. split() = Split string into three words:
['Intellipaat', 'Python', 'Training']
3. join() = joins a and b:
IntellipaatPython
4. find() = finding occurrence in string:  12
5. replace() = replacing a word in a string:
Intellipaat Python Course
Concatinating string a and b: Intellipaat Python
```

List

List

Sequence Data Types



List

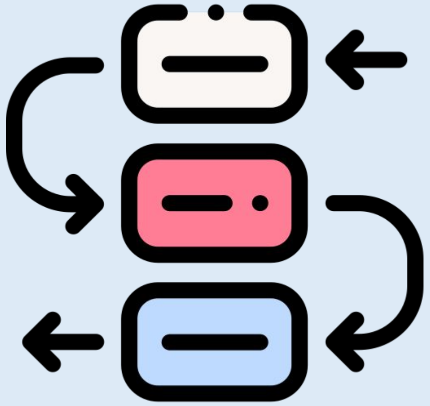
1. Lists are just like the arrays, declared in other languages which is a ordered collection of data.

1. Python lists support multiple data types. Hence they are more flexible.

1. Initialization: List1 = [0, 2, 3]

List

Sequence Data Types

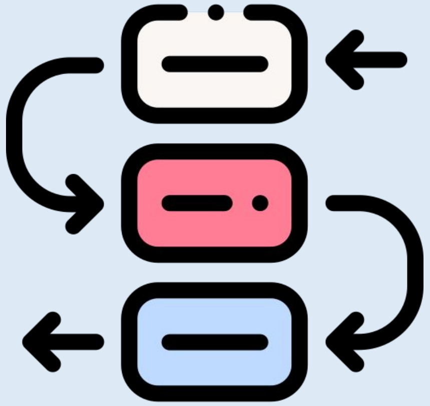


List Methods

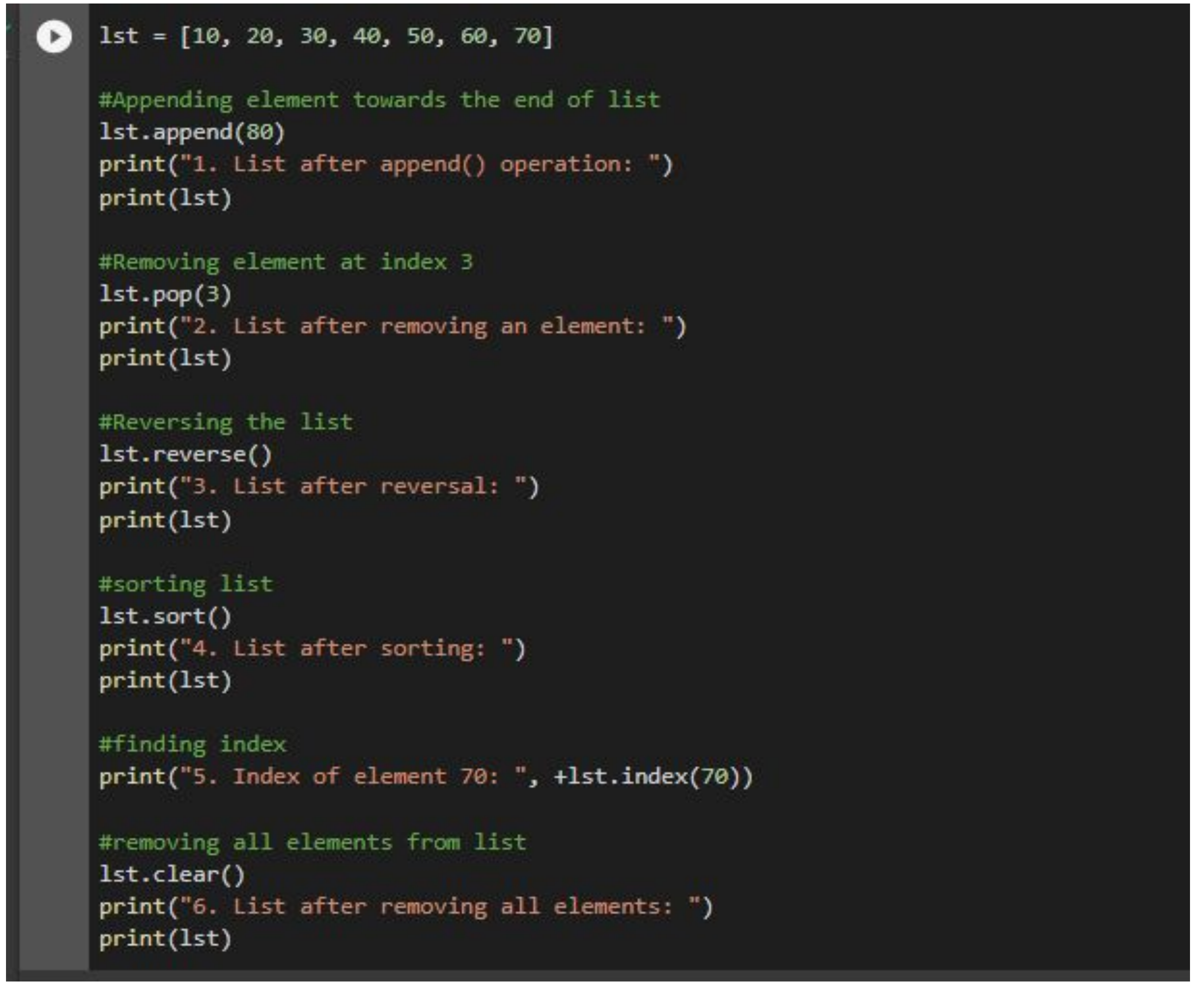
1. append() - Adds element at the end of list
1. pop() - Removes element from specified position
1. reverse() - Reverses the Python List
1. sort() - Sorts elements of python list
1. index() - Returns index of specified value
1. clear() - Removes all elements from the list

List

Sequence Data Types



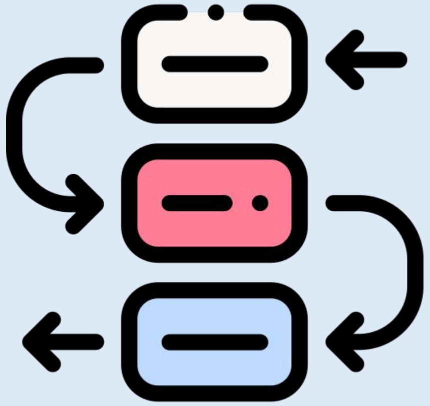
Implementing List Methods



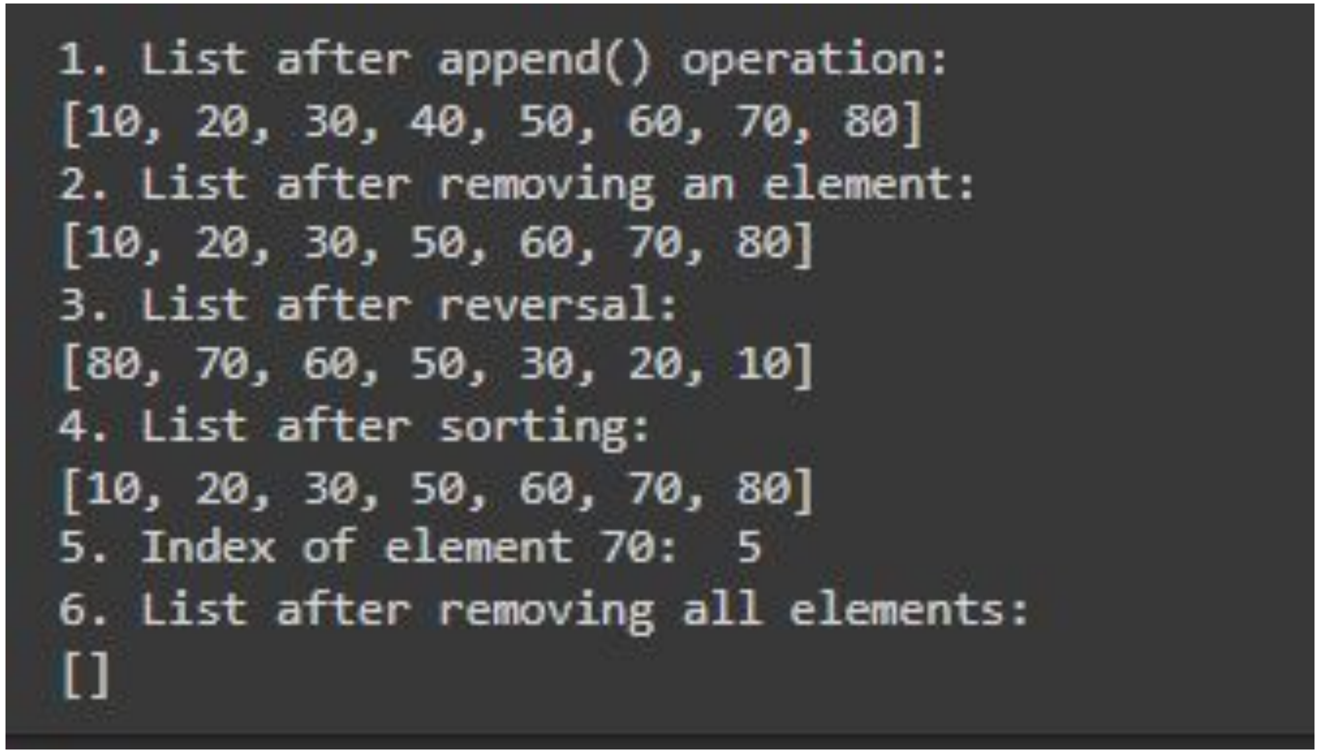
```
lst = [10, 20, 30, 40, 50, 60, 70]

#Appending element towards the end of list
lst.append(80)
print("1. List after append() operation: ")
print(lst)

#Removing element at index 3
lst.pop(3)
print("2. List after removing an element: ")
print(lst)

#Reversing the list
lst.reverse()
print("3. List after reversal: ")
print(lst)

#sorting list
lst.sort()
print("4. List after sorting: ")
print(lst)

#finding index
print("5. Index of element 70: ", +lst.index(70))

#removing all elements from list
lst.clear()
print("6. List after removing all elements: ")
print(lst)
```

List

Sequence Data Types

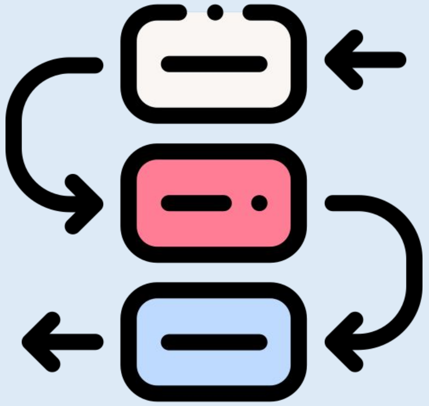


Result



```
1. List after append() operation:
[10, 20, 30, 40, 50, 60, 70, 80]
2. List after removing an element:
[10, 20, 30, 50, 60, 70, 80]
3. List after reversal:
[80, 70, 60, 50, 30, 20, 10]
4. List after sorting:
[10, 20, 30, 50, 60, 70, 80]
5. Index of element 70:  5
6. List after removing all elements:
[]
```

Sequence Data Types

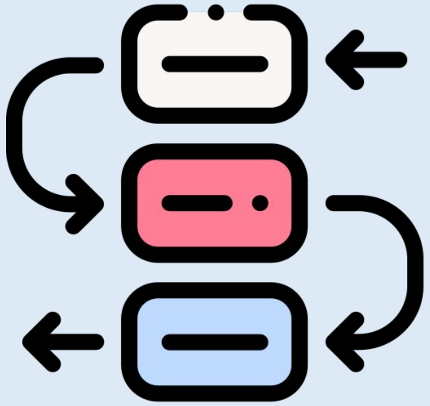


More List Methods

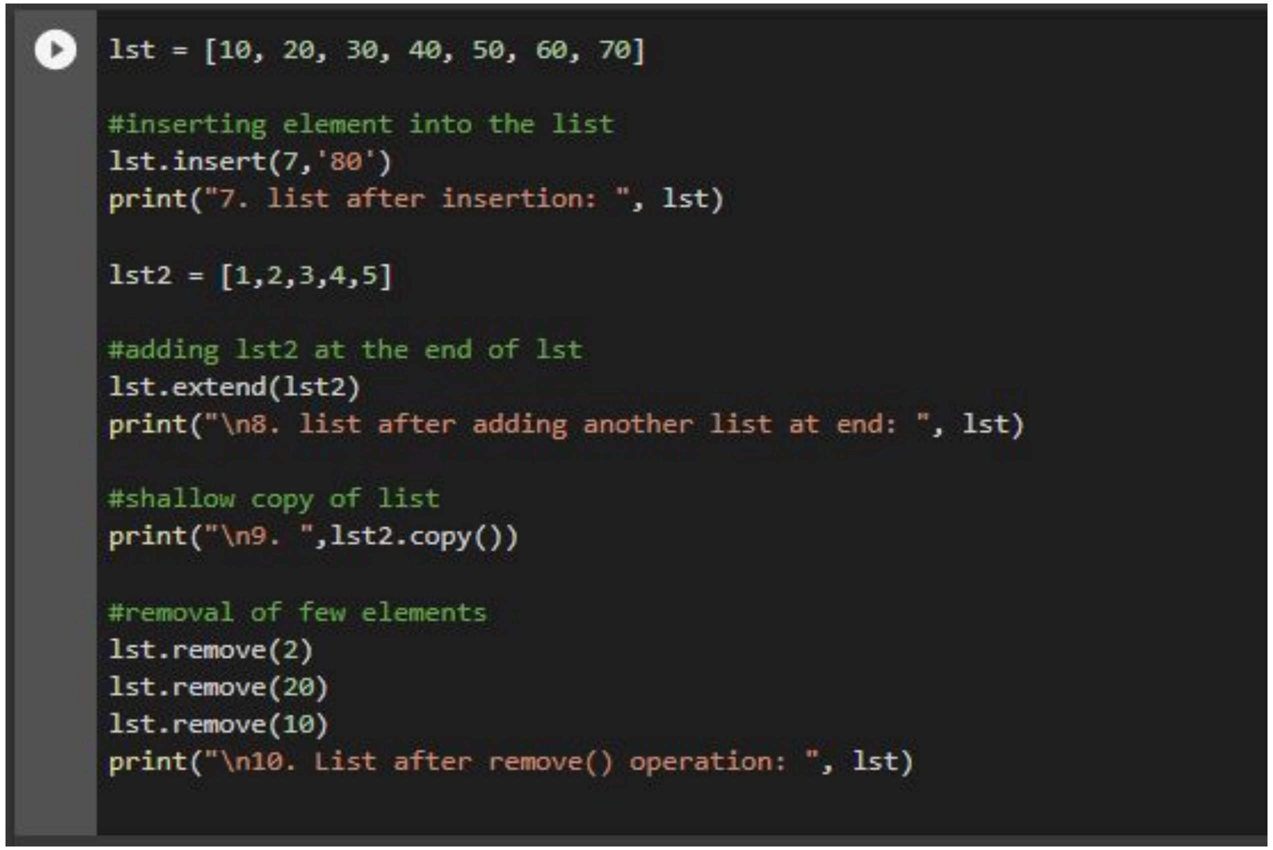
7. insert(): Inserts an elements at specified position
8. extend(): Adds contents of List2 to the end of List1
9. copy(): Returns a shallow copy of list
10. remove(): Removes specified element

List

Sequence Data Types



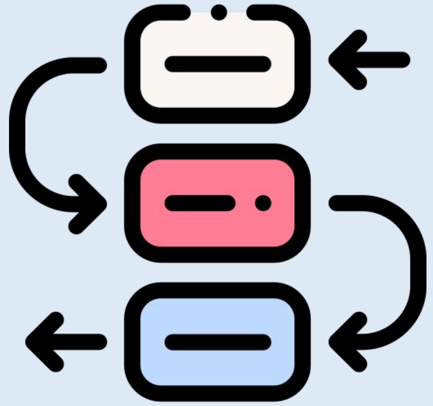
Implementing List Methods



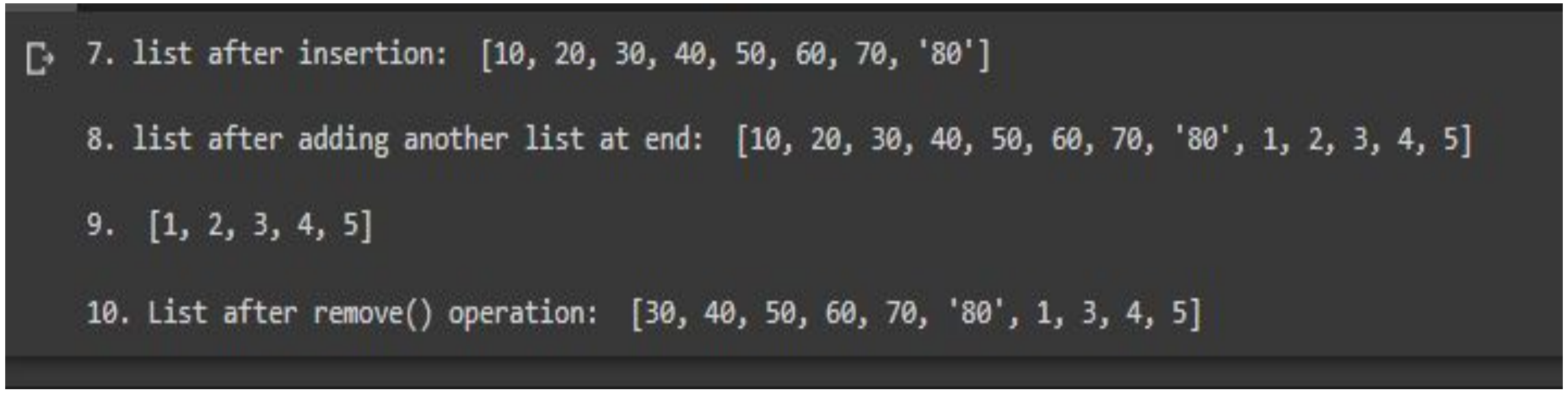
```
lst = [10, 20, 30, 40, 50, 60, 70]

#inserting element into the list
lst.insert(7,'80')
print("7. list after insertion: ", lst)

lst2 = [1,2,3,4,5]

#adding lst2 at the end of lst
lst.extend(lst2)
print("\n8. list after adding another list at end: ", lst)

#shallow copy of list
print("\n9. ",lst2.copy())

#removal of few elements
lst.remove(2)
lst.remove(20)
lst.remove(10)
print("\n10. List after remove() operation: ", lst)
```

List

Sequence Data Types



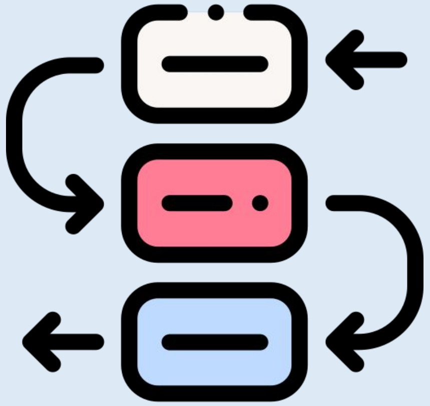
Result



```
7. list after insertion:  [10, 20, 30, 40, 50, 60, 70, '80']

8. list after adding another list at end:  [10, 20, 30, 40, 50, 60, 70, '80', 1, 2, 3, 4, 5]

9.  [1, 2, 3, 4, 5]

10. List after remove() operation:  [30, 40, 50, 60, 70, '80', 1, 3, 4, 5]
```

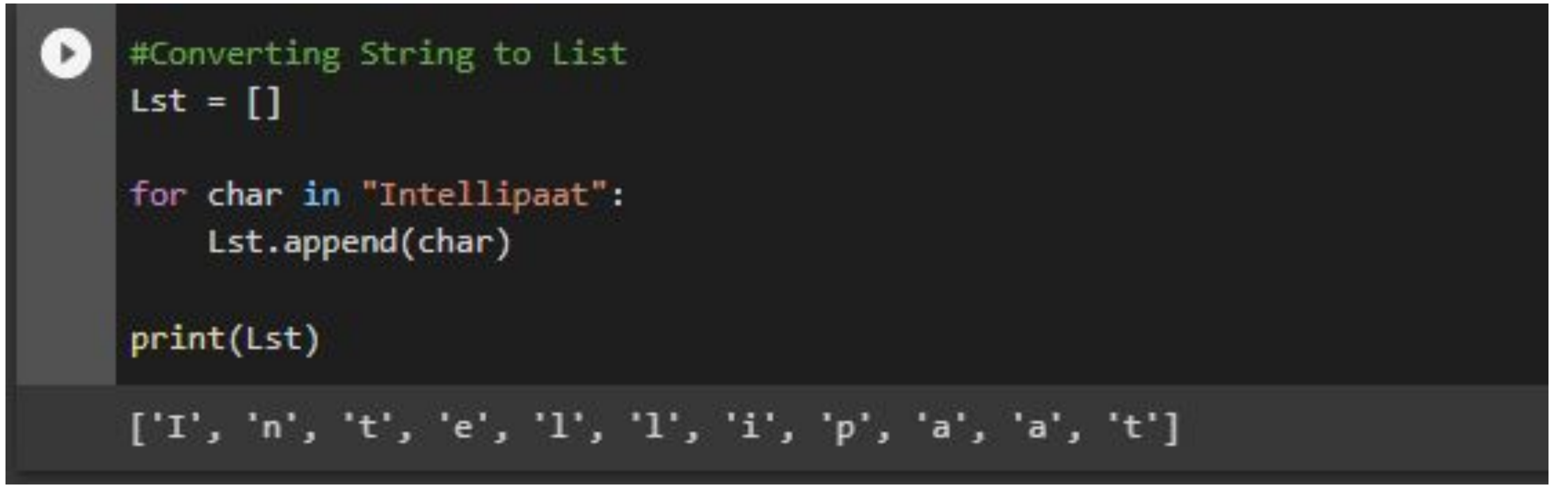
List

Sequence Data Types



List Comprehension:

List comprehensions are used for creating new lists from other iterables like tuples, strings, arrays, lists, etc.



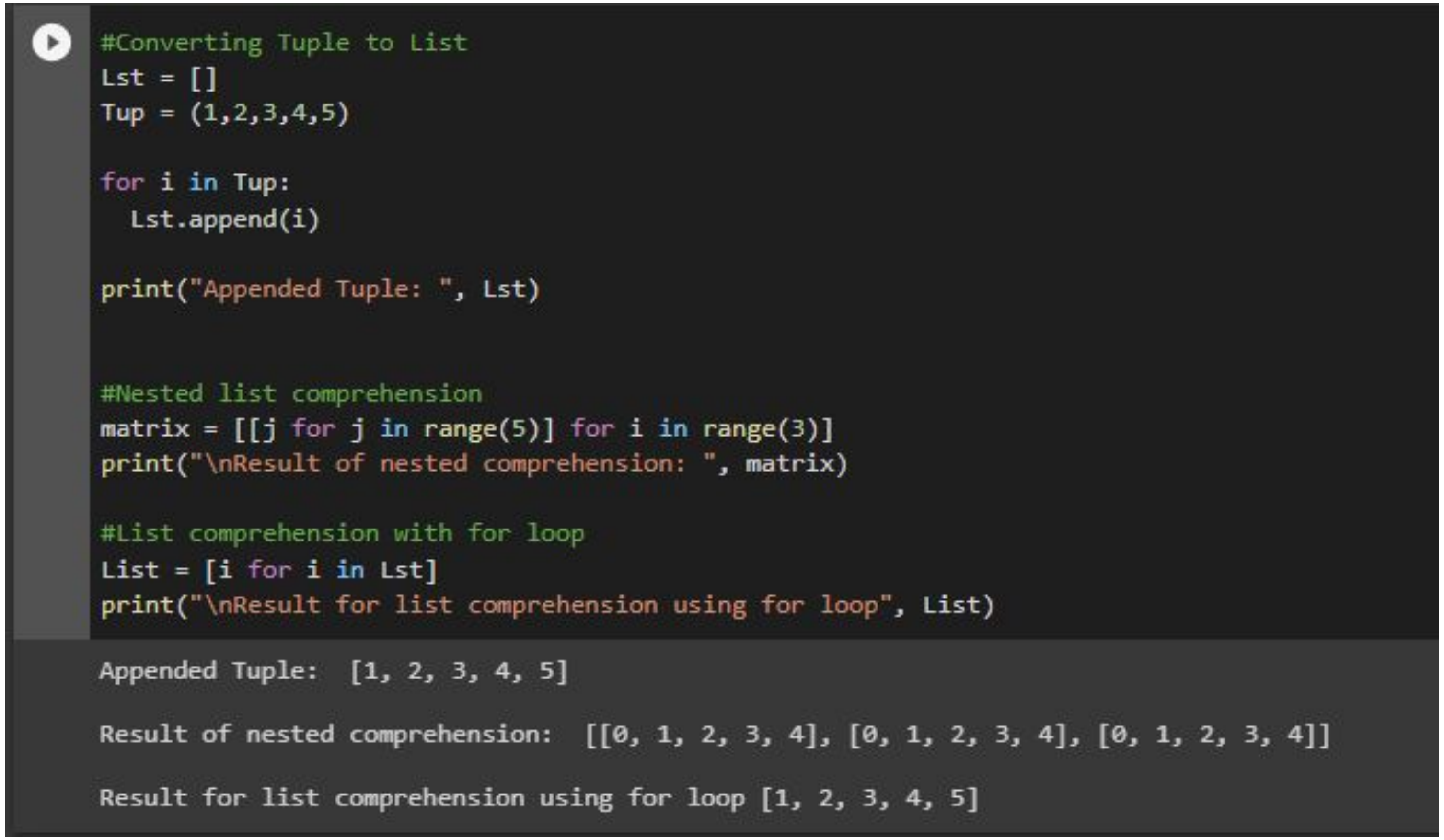
```
#Converting String to List
Lst = []

for char in "Intellipaat":
    Lst.append(char)

print(Lst)
```

```
['I', 'n', 't', 'e', 'l', 'l', 'i', 'p', 'a', 'a', 't']
```

Example of List Comprehension: String to List

List Comprehension Demo 2:

Sequence Data Types



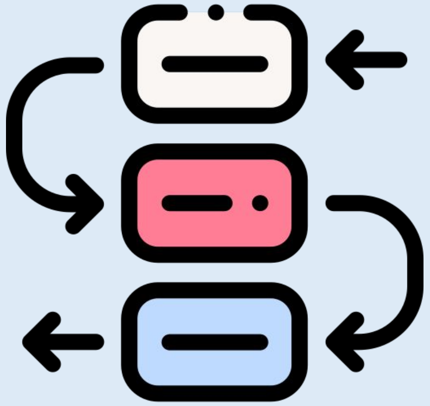
```
#Converting Tuple to List
Lst = []
Tup = (1,2,3,4,5)

for i in Tup:
  Lst.append(i)

print("Appended Tuple: ", Lst)


#Nested list comprehension
matrix = [[j for j in range(5)] for i in range(3)]
print("\nResult of nested comprehension: ", matrix)

#List comprehension with for loop
List = [i for i in Lst]
print("\nResult for list comprehension using for loop", List)
```

```
Appended Tuple:  [1, 2, 3, 4, 5]

Result of nested comprehension:  [[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

Result for list comprehension using for loop [1, 2, 3, 4, 5]
```

Sequence Indexing

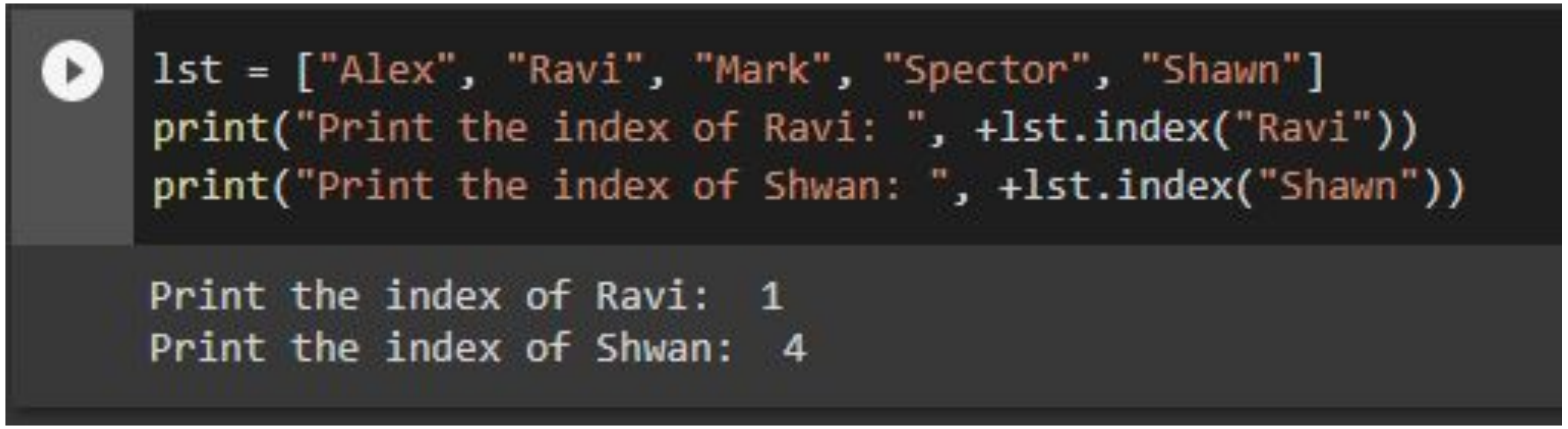
Sequence Indexing

Sequence Data Types



Indexing in Python is a way to refer the **individual items** within an **iterable** by its **position**.

Let's try to understand what index is with the help of an example.

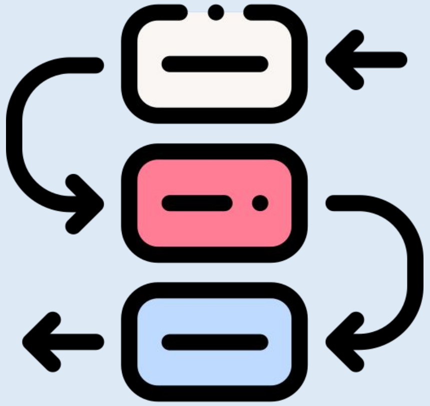


```
lst = ["Alex", "Ravi", "Mark", "Spector", "Shawn"]
print("Print the index of Ravi: ", +lst.index("Ravi"))
print("Print the index of Shwan: ", +lst.index("Shawn"))
```

```
Print the index of Ravi:  1
Print the index of Shwan:  4
```

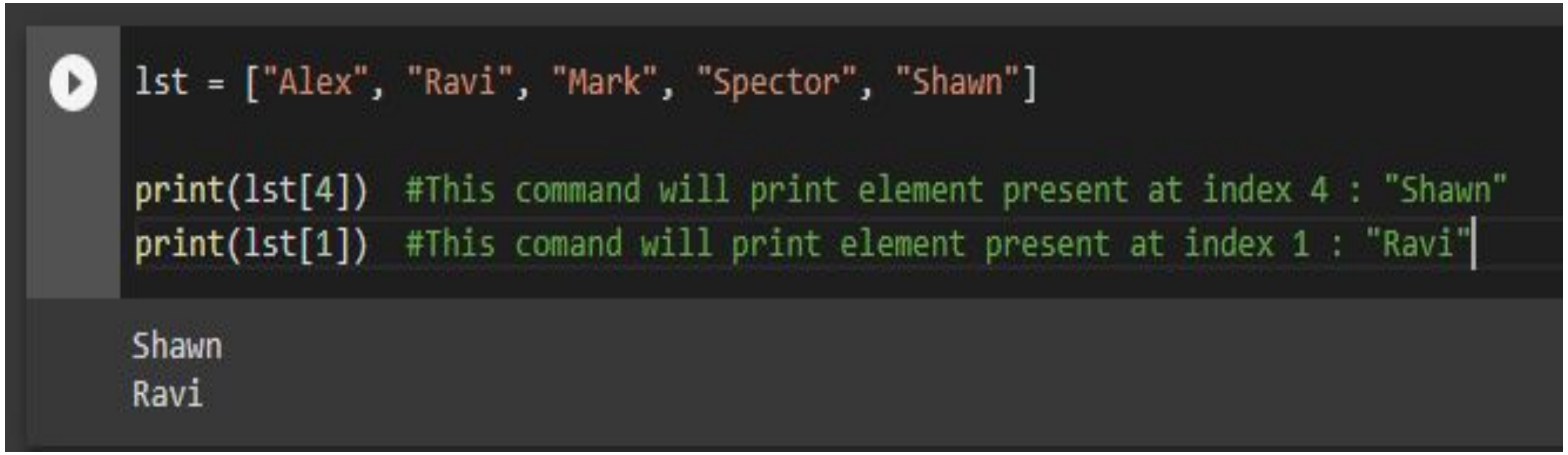
Sequence Indexing

Sequence Data Types



With the use of index we can access the element present inside a sequence.

Consider the example of List Indexing given below.



```
lst = ["Alex", "Ravi", "Mark", "Spector", "Shawn"]

print(lst[4])  #This command will print element present at index 4 : "Shawn"
print(lst[1])  #This comand will print element present at index 1 : "Ravi"
```

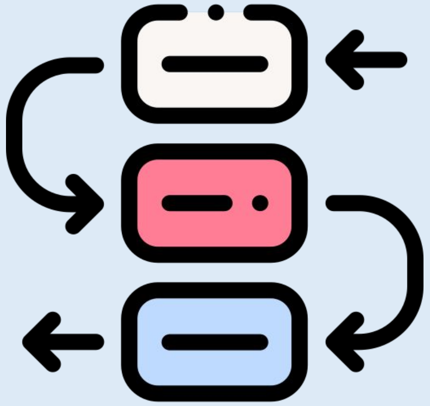
```
Shawn
Ravi
```

This type of indexing is known as positive indexing.

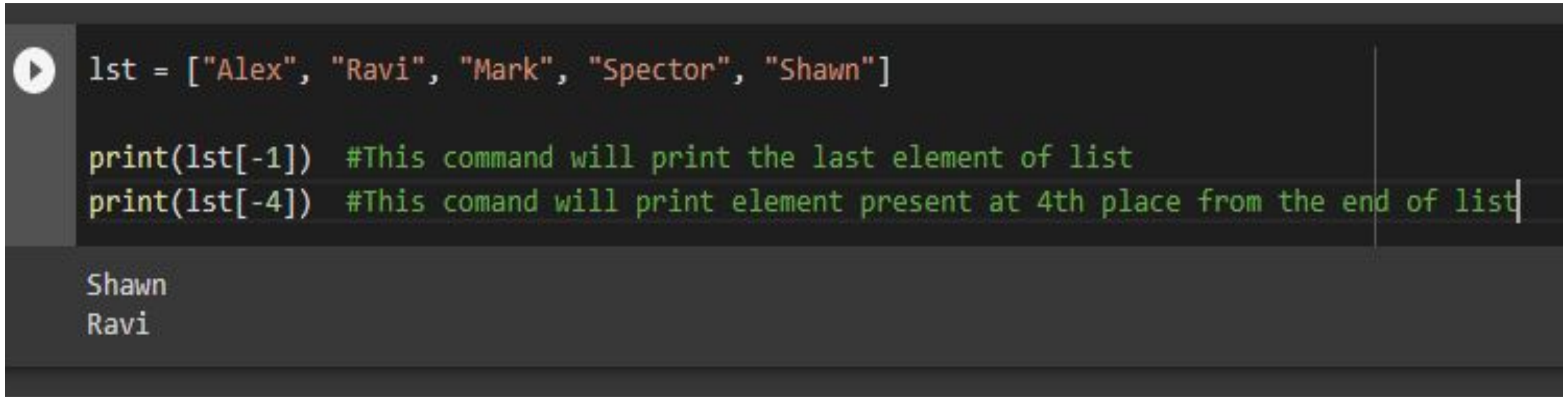
Sequence Indexing

We can access the elements in reversed order by using convention of **'-ve'** sign.

Sequence Data Types



Consider the example of **Negative List Indexing** given below.



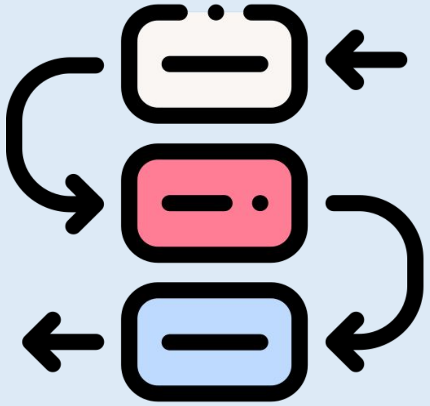
```
lst = ["Alex", "Ravi", "Mark", "Spector", "Shawn"]

print(lst[-1])  #This command will print the last element of list
print(lst[-4])  #This comand will print element present at 4th place from the end of list
```

```
Shawn
Ravi
```

Tuple

Tuple

Sequence Data Types



Tuple

1. Just like list, tuple is also an ordered collection of Python objects.

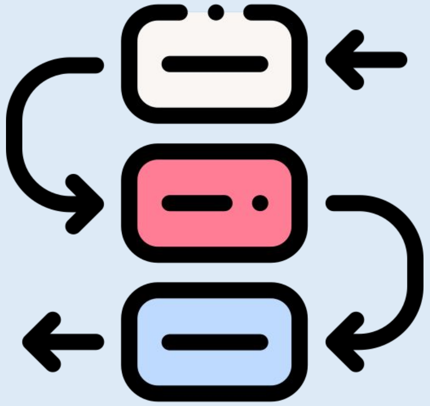
1. Tuples are immutable, which means they cannot be modified once they are created.

1. In Python, tuples are created by placing a sequence of values separated by 'comma'.

1. Initialization: Tuple = (1, 2, 3)

Tuple

Sequence Data Types

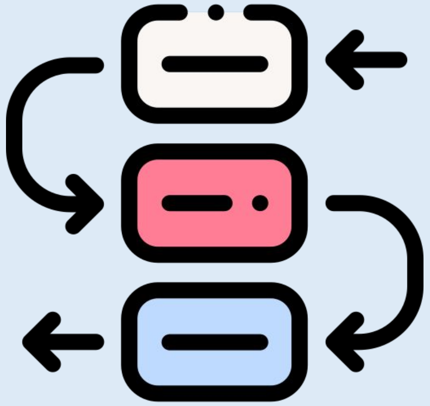


Tuple Methods

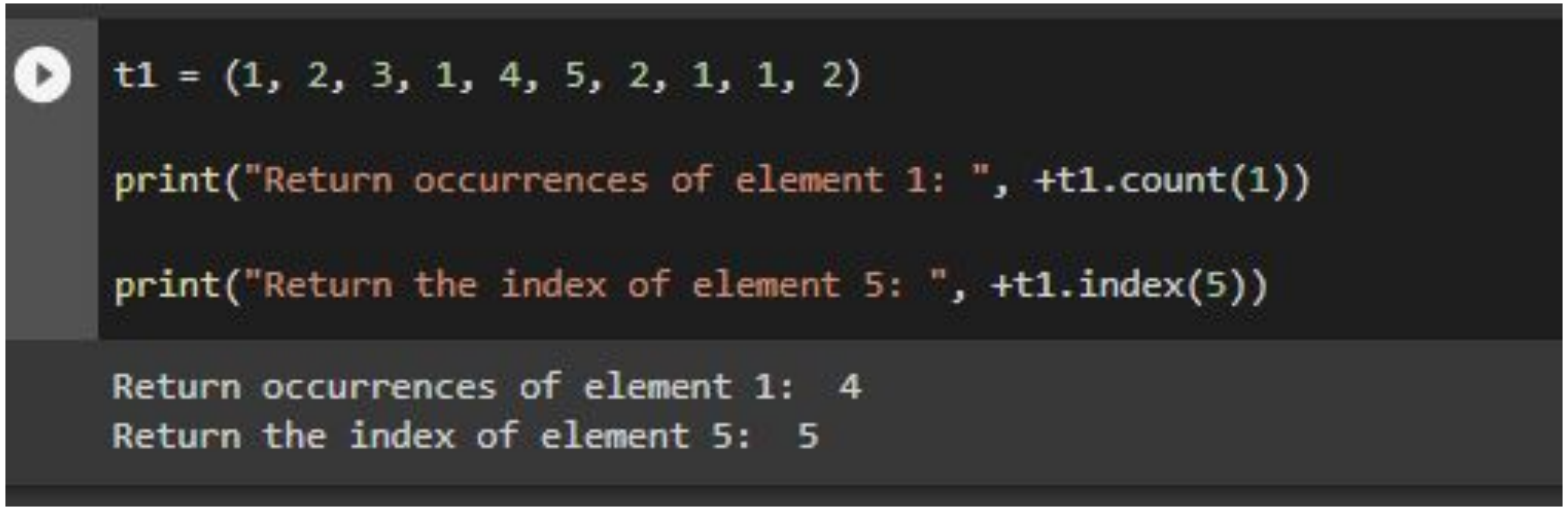
1. count() - Gives the count of the specified element.
2. index() - Gives the index of the first occurrence of a specified element.

Tuple

Sequence Data Types



Tuple Demo



```
t1 = (1, 2, 3, 1, 4, 5, 2, 1, 1, 2)

print("Return occurrences of element 1: ", +t1.count(1))

print("Return the index of element 5: ", +t1.index(5))
```

```
Return occurrences of element 1:  4
Return the index of element 5:  5
```

Set Data Type

Set Data Types

Set Data Types

Set is an **unordered collection of data types** in Python that is **iterable**, **changeable**, and **contains no duplicate elements**.

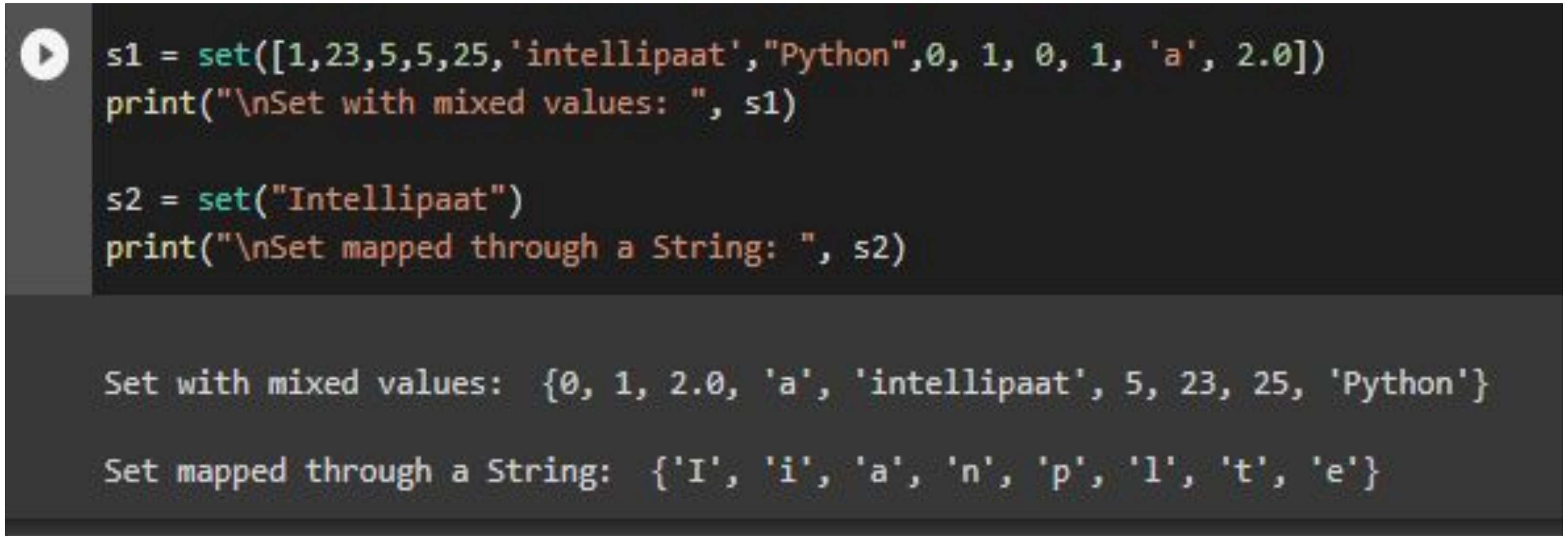
Set can be created by using python **built in function set()**.

The order of elements in set is completely **undefined**.

Set Data Types

Set Data Types

The order of elements in set is completely **undefined**.



```
s1 = set([1,23,5,5,25,'intellipaat',"Python",0, 1, 0, 1, 'a', 2.0])
print("\nSet with mixed values: ", s1)

s2 = set("Intellipaat")
print("\nSet mapped through a String: ", s2)
```

```
Set with mixed values:  {0, 1, 2.0, 'a', 'intellipaat', 5, 23, 25, 'Python'}

Set mapped through a String:  {'I', 'i', 'a', 'n', 'p', 'l', 't', 'e'}
```

Program with two different Set Creation Methods

Set Data Types

Set Data Types

Set Methods

1. add(): Adds a given element to a set

1. clear(): Removes all elements from the set

1. remove(): Removes element from set

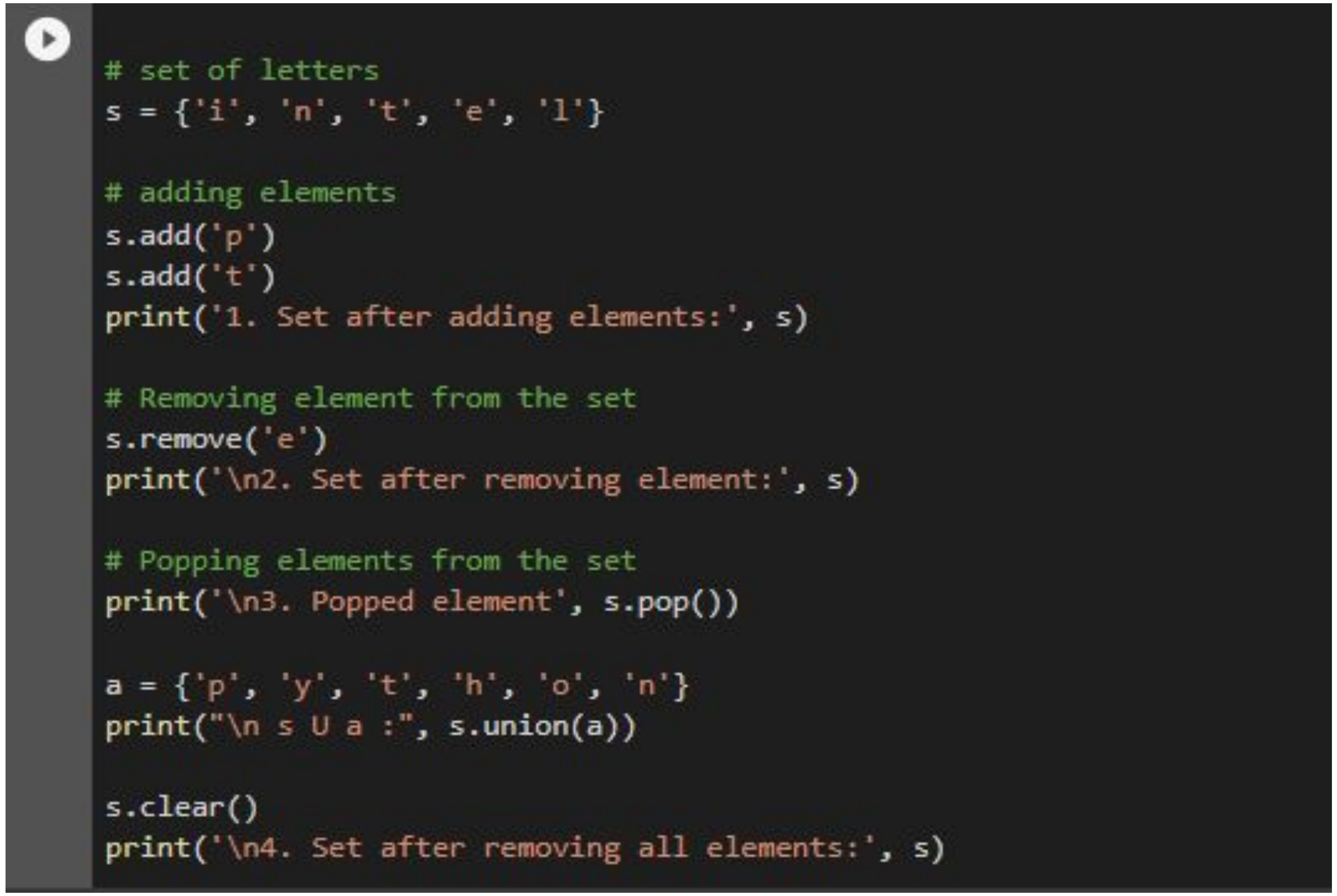
1. pop(): Returns and removes a random element from the set

1. union(): Returns a set that has the union of all sets

Set Data Types

Set Data Types

Set Method Demo

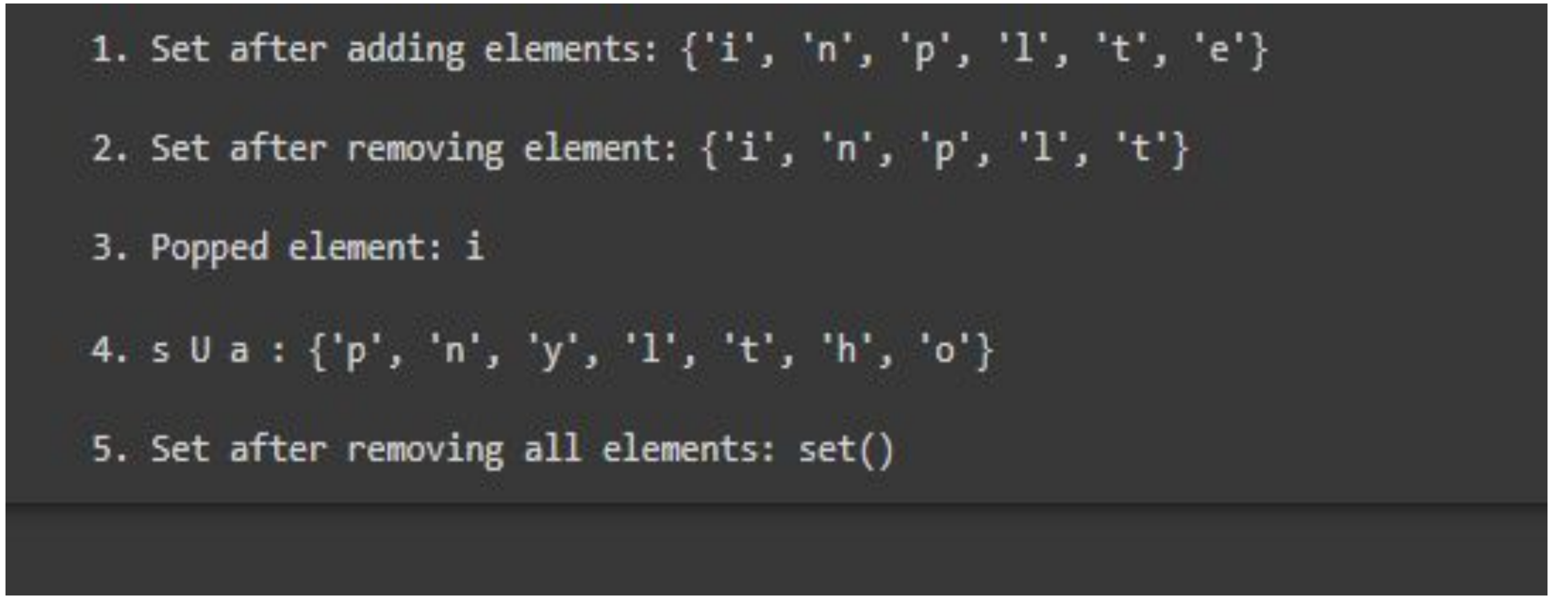


```
# set of letters
s = {'i', 'n', 't', 'e', 'l'}

# adding elements
s.add('p')
s.add('t')
print('1. Set after adding elements:', s)

# Removing element from the set
s.remove('e')
print('\n2. Set after removing element:', s)

# Popping elements from the set
print('\n3. Popped element', s.pop())

a = {'p', 'y', 't', 'h', 'o', 'n'}
print("\n s U a :", s.union(a))

s.clear()
print('\n4. Set after removing all elements:', s)
```

Set Data Types

Set Data Types

Result



```
1. Set after adding elements: {'i', 'n', 'p', 'l', 't', 'e'}

2. Set after removing element: {'i', 'n', 'p', 'l', 't'}

3. Popped element: i

4. s U a : {'p', 'n', 'y', 'l', 't', 'h', 'o'}

5. Set after removing all elements: set()
```

Set Data Types

Set Data Types

Set Operations

1. intersection(): Returns common elements of both sets

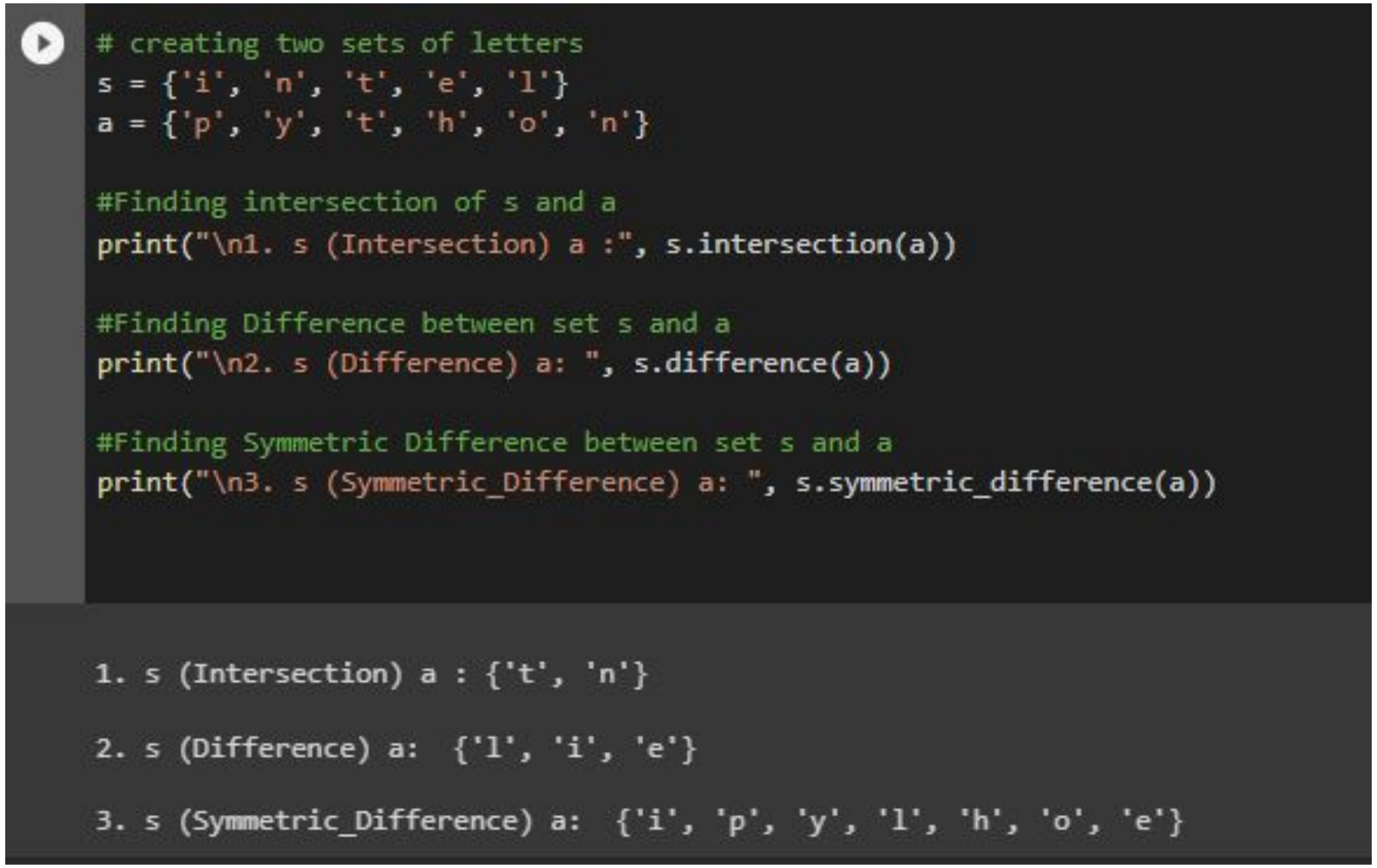
1. difference(): Returns set of elements that is present in first set but not in second

1. symmetric_difference(): Returns set of all the elements that are either in the first set or the second set but not in both

Set Data Types

Set Data Types

Set Operations Demo



```
# creating two sets of letters
s = {'i', 'n', 't', 'e', 'l'}
a = {'p', 'y', 't', 'h', 'o', 'n'}

#Finding intersection of s and a
print("\n1. s (Intersection) a :", s.intersection(a))

#Finding Difference between set s and a
print("\n2. s (Difference) a: ", s.difference(a))

#Finding Symmetric Difference between set s and a
print("\n3. s (Symmetric_Difference) a: ", s.symmetric_difference(a))
```

```
1. s (Intersection) a : {'t', 'n'}

2. s (Difference) a:  {'l', 'i', 'e'}

3. s (Symmetric_Difference) a:  {'i', 'p', 'y', 'l', 'h', 'o', 'e'}
```

Set Data Types

Set Data Types

Set Joins

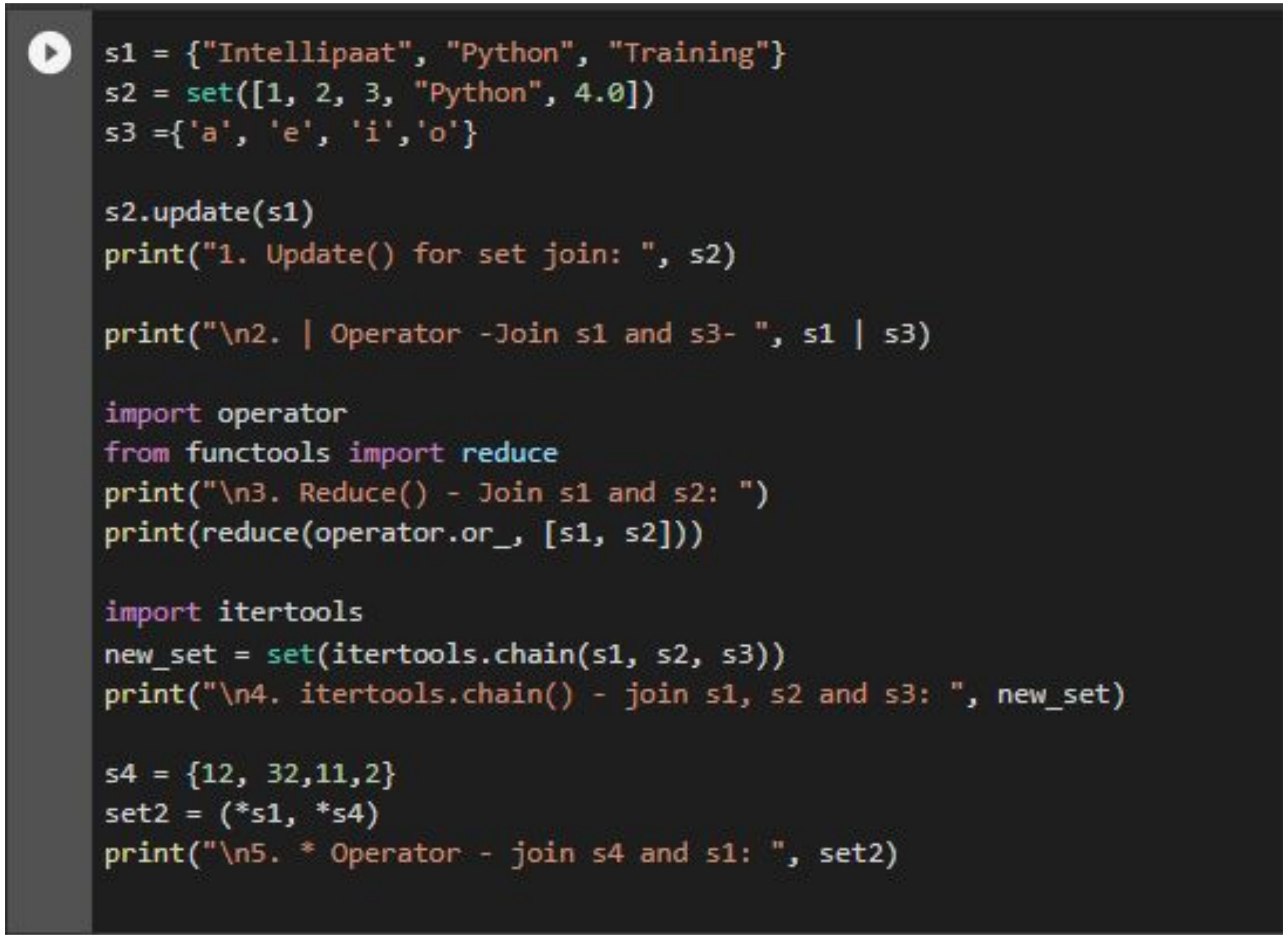
In python, the merging of two or more sets is achievable. Let's learn more about methods used to achieve this merging.

1. update(): Inserts all items from one set to other
1. '|' operator: This is union operator which joins two or more different elements
1. reduce(): Returns bitwise or of two sets
1. itertools.chain(): Joins two distinct objects
1. * operator: unpacking operator for joining sets

Set Data Types

Set Data Types

Implementing Set Joins

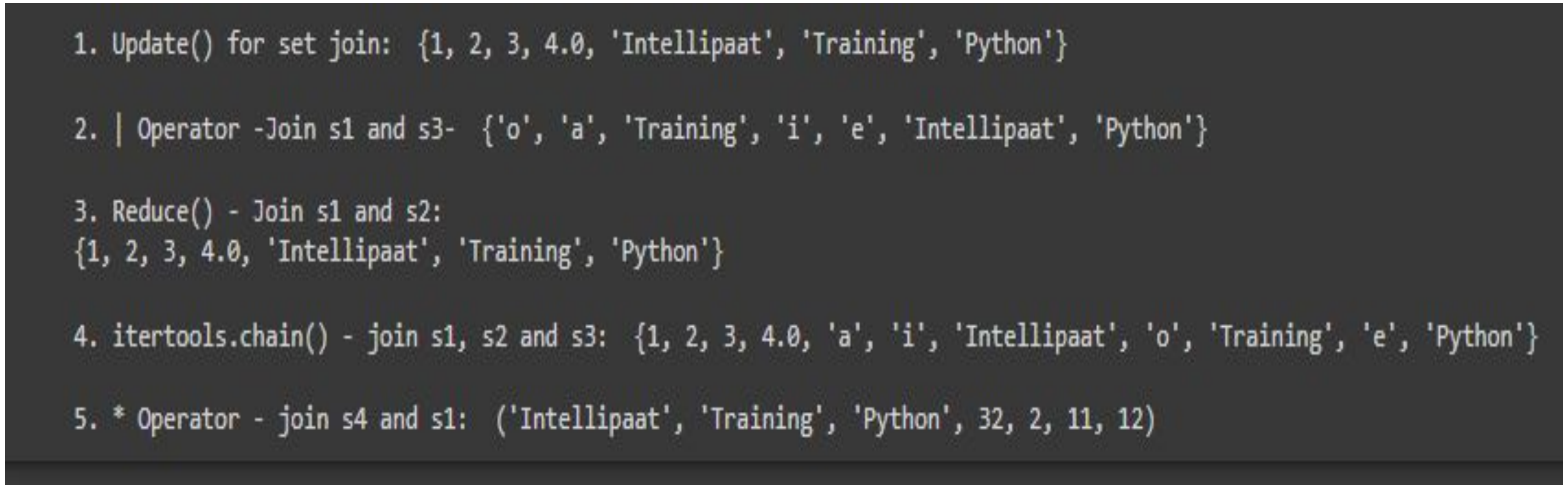


```
s1 = {"Intellipaat", "Python", "Training"}
s2 = set([1, 2, 3, "Python", 4.0])
s3 ={'a', 'e', 'i','o'}

s2.update(s1)
print("1. Update() for set join: ", s2)

print("\n2. | Operator -Join s1 and s3- ", s1 | s3)

import operator
from functools import reduce
print("\n3. Reduce() - Join s1 and s2: ")
print(reduce(operator.or_, [s1, s2]))

import itertools
new_set = set(itertools.chain(s1, s2, s3))
print("\n4. itertools.chain() - join s1, s2 and s3: ", new_set)

s4 = {12, 32,11,2}
set2 = (*s1, *s4)
print("\n5. * Operator - join s4 and s1: ", set2)
```

Set Data Types

Set Data Types

Result



```
1. Update() for set join:  {1, 2, 3, 4.0, 'Intellipaat', 'Training', 'Python'}

2. | Operator -Join s1 and s3-  {'o', 'a', 'Training', 'i', 'e', 'Intellipaat', 'Python'}

3. Reduce() - Join s1 and s2:
{1, 2, 3, 4.0, 'Intellipaat', 'Training', 'Python'}

4. itertools.chain() - join s1, s2 and s3:  {1, 2, 3, 4.0, 'a', 'i', 'Intellipaat', 'o', 'Training', 'e', 'Python'}

5. * Operator - join s4 and s1:  ('Intellipaat', 'Training', 'Python', 32, 2, 11, 12)
```

Dictionary Data Type

Dictionary Data Type

Dictionary Data Types

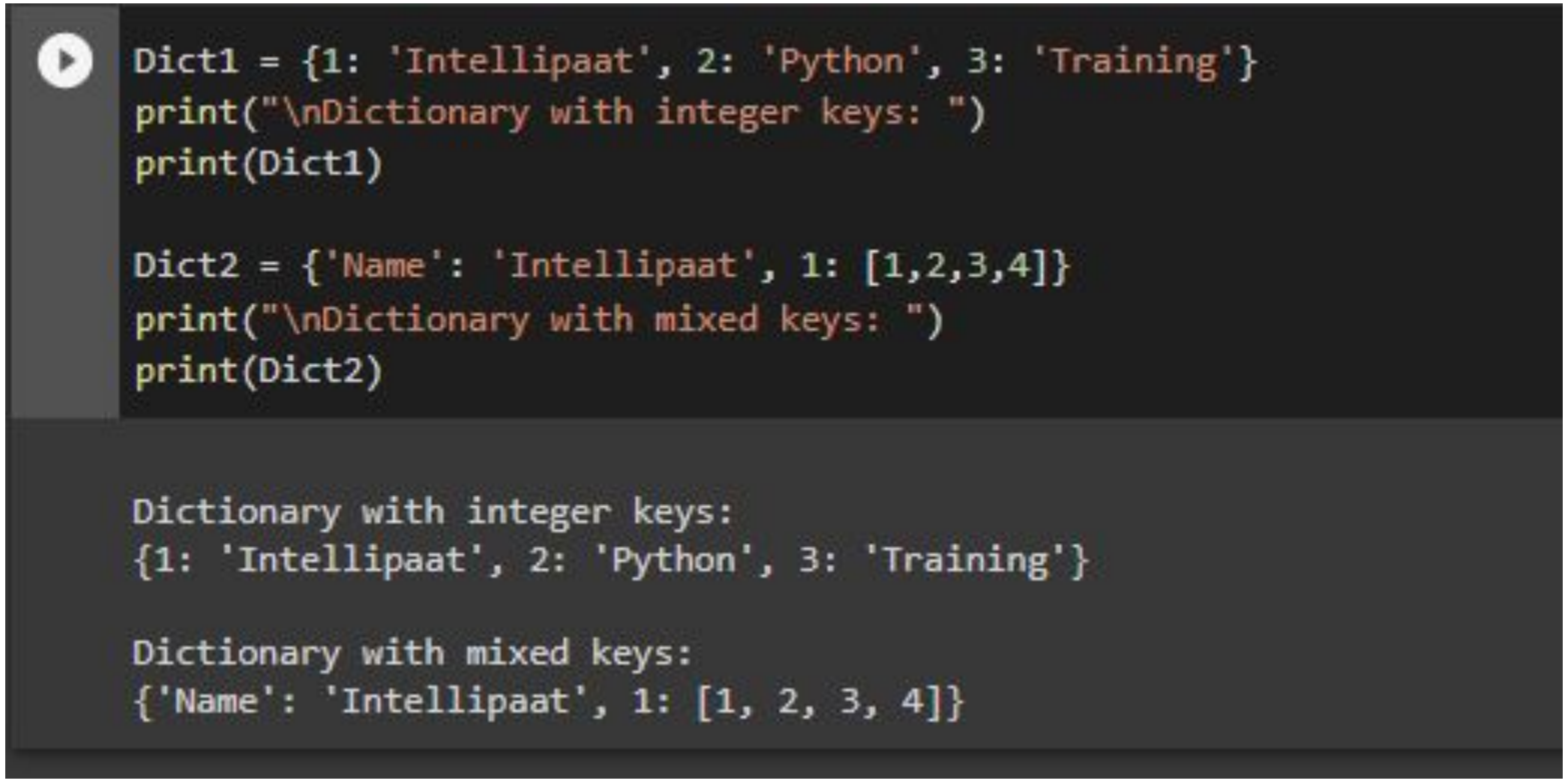
Dictionaries are Python's implementation of an **associative array**, which is a **data structure.** A dictionary is a **cluster** of **key-value pairs.**

In Python, a Dictionary can be created by placing a sequence of elements within **curly {} braces**, **separated by ‘comma’**.

A dictionary's values can be of **any datatype** and can be **replicated**, however **keys cannot be copied and must be immutable**.

Dictionary Data Type

Dictionary Data Types



```
Dict1 = {1: 'Intellipaat', 2: 'Python', 3: 'Training'}
print("\nDictionary with integer keys: ")
print(Dict1)

Dict2 = {'Name': 'Intellipaat', 1: [1,2,3,4]}
print("\nDictionary with mixed keys: ")
print(Dict2)
```

```
Dictionary with integer keys:
{1: 'Intellipaat', 2: 'Python', 3: 'Training'}

Dictionary with mixed keys:
{'Name': 'Intellipaat', 1: [1, 2, 3, 4]}
```

Program to Create Dictionaries

Dictionary Data Type

Dictionary Data Types

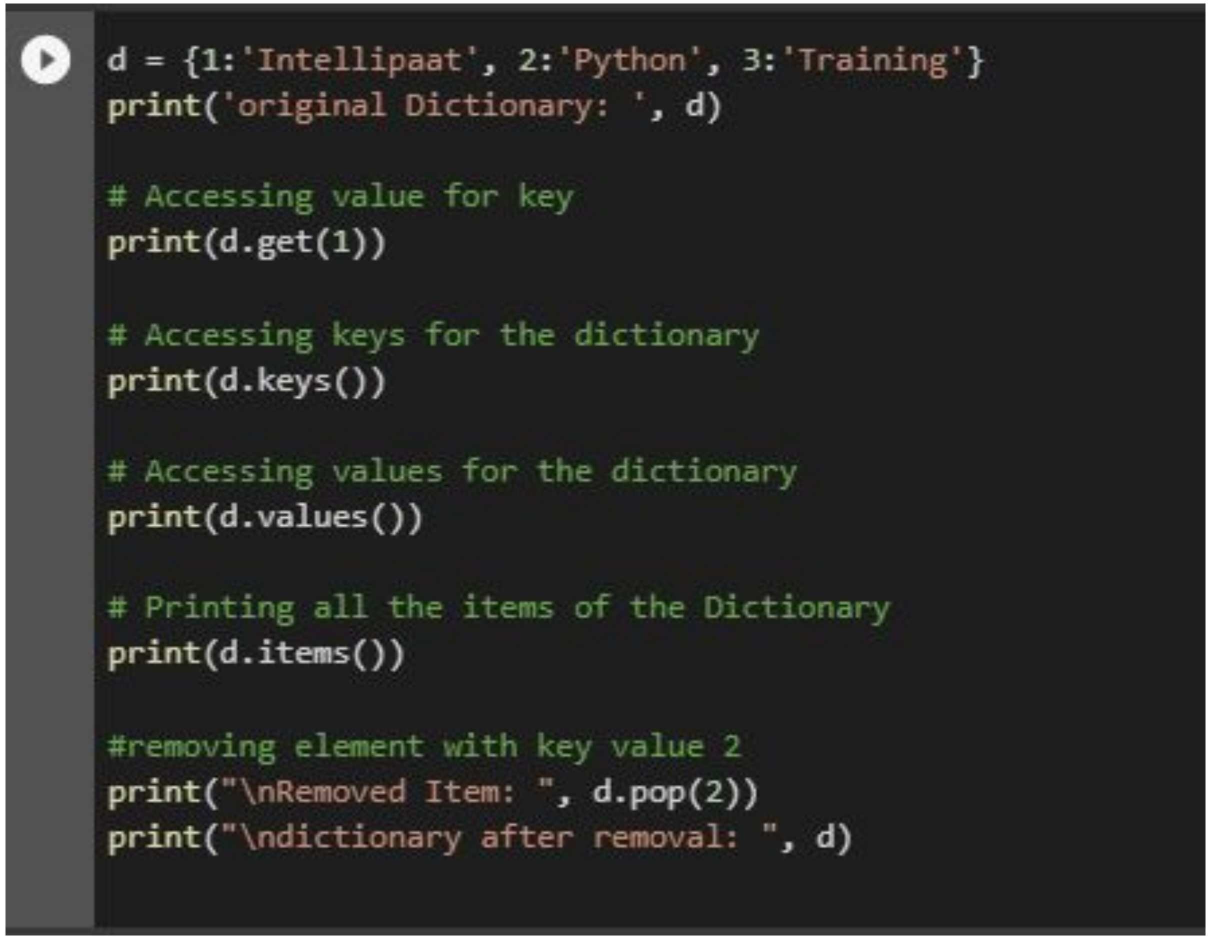
Dictionary Methods

1. get(): Returns the value for the given key
1. keys(): Returns a view object that displays a list of all the keys in the dictionary in order of insertion
1. values(): Returns a list of all the values available in a given dictionary
1. items(): Returns the list with all dictionary keys with values
1. pop() - Returns and removes element with given key

Dictionary Data Type

Dictionary Data Types

Implementing Dictionary Methods

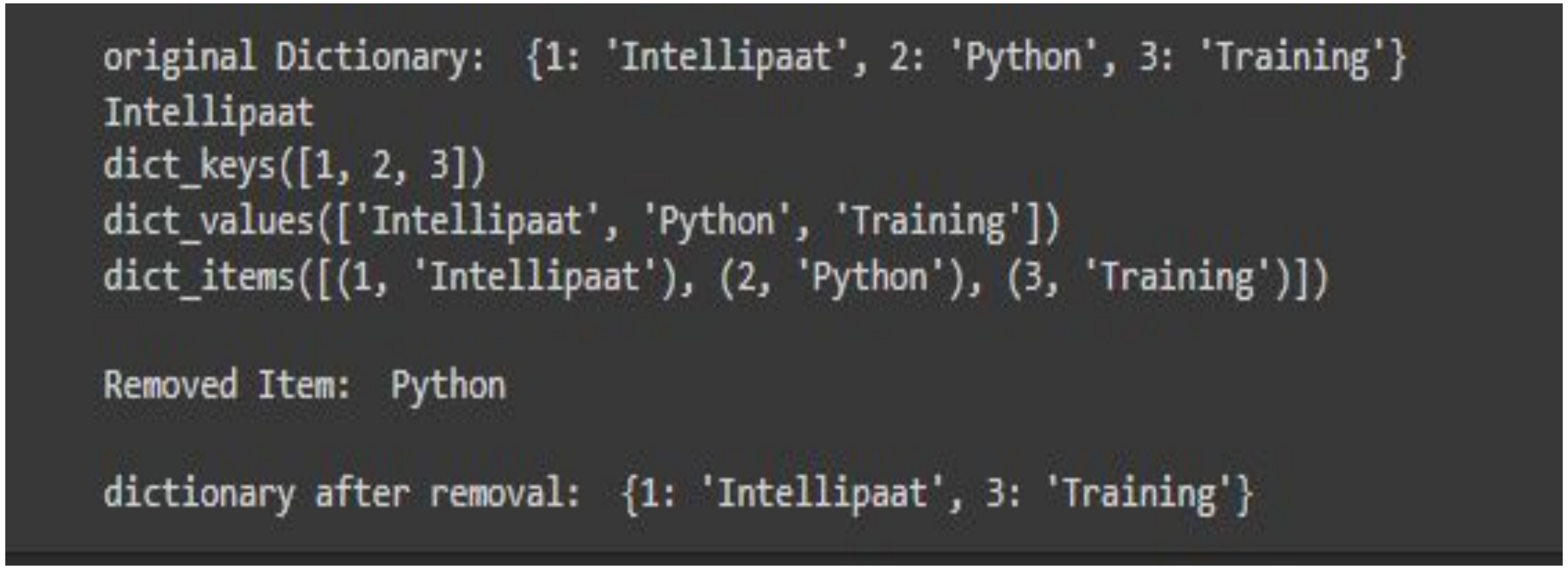


```
d = {1:'Intellipaat', 2:'Python', 3:'Training'}
print('original Dictionary: ', d)

# Accessing value for key
print(d.get(1))

# Accessing keys for the dictionary
print(d.keys())

# Accessing values for the dictionary
print(d.values())

# Printing all the items of the Dictionary
print(d.items())

#removing element with key value 2
print("\nRemoved Item: ", d.pop(2))
print("\ndictionary after removal: ", d)
```

Dictionary Data Type

Dictionary Data Types

Result



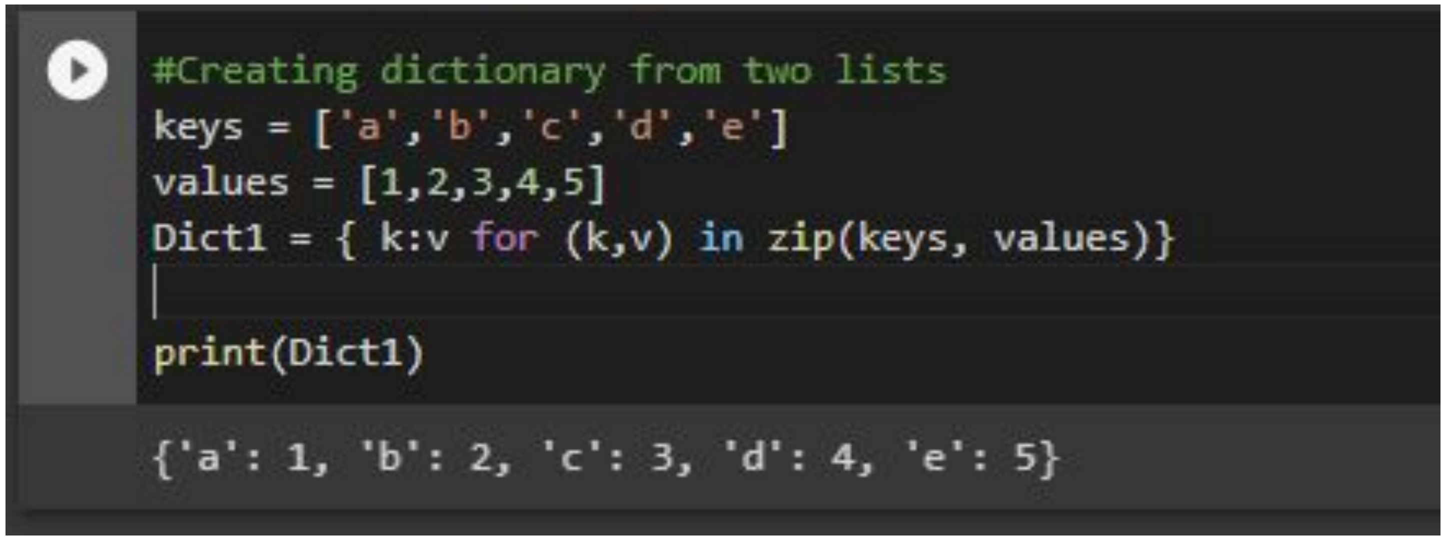
```
original Dictionary:  {1: 'Intellipaat', 2: 'Python', 3: 'Training'}
Intellipaat
dict_keys([1, 2, 3])
dict_values(['Intellipaat', 'Python', 'Training'])
dict_items([(1, 'Intellipaat'), (2, 'Python'), (3, 'Training')])

Removed Item:  Python

dictionary after removal:  {1: 'Intellipaat', 3: 'Training'}
```

Dictionary Data Type

Dictionary Data Types

Dictionary Comprehension:

Dictionary comprehensions are used for creating new dictionaries from other iterables.



```
#Creating dictionary from two lists
keys = ['a','b','c','d','e']
values = [1,2,3,4,5]
Dict1 = { k:v for (k,v) in zip(keys, values)}

print(Dict1)
```

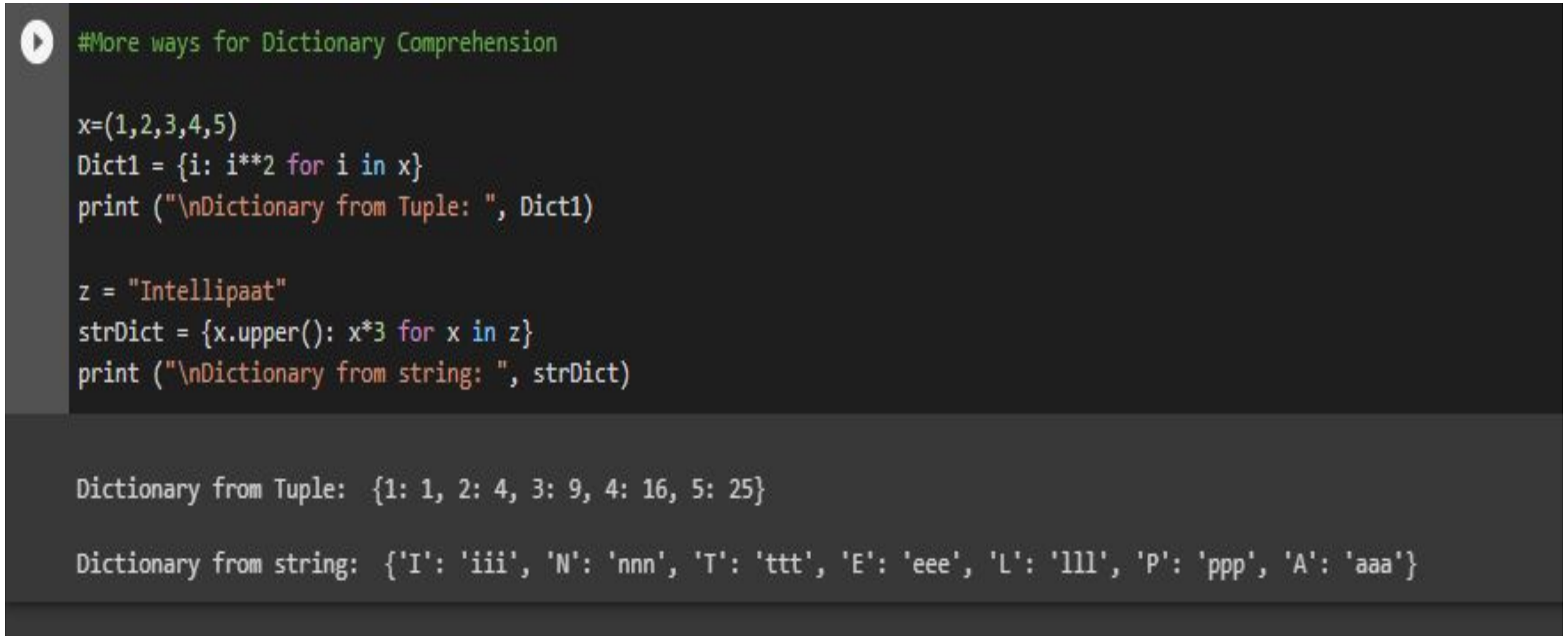
```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

Example of List Comprehension: String to List

Dictionary Data Type

Dictionary Data Types

Dictionary Comprehension Demo:



```
#More ways for Dictionary Comprehension

x=(1,2,3,4,5)
Dict1 = {i: i**2 for i in x}
print ("\nDictionary from Tuple: ", Dict1)

z = "Intellipaat"
strDict = {x.upper(): x*3 for x in z}
print ("\nDictionary from string: ", strDict)
```

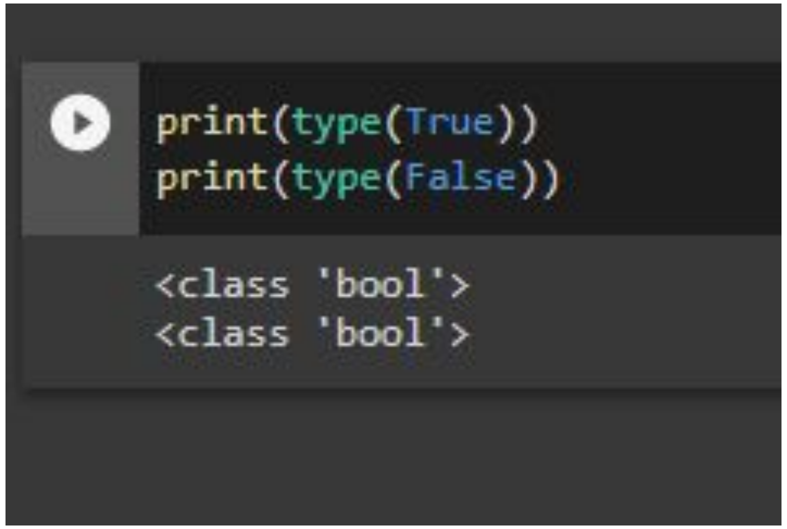
```
Dictionary from Tuple:  {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

Dictionary from string:  {'I': 'iii', 'N': 'nnn', 'T': 'ttt', 'E': 'eee', 'L': 'lll', 'P': 'ppp', 'A': 'aaa'}
```

Boolean Data Type

Boolean Data Type

Boolean Data Types

Boolean is the Data type with two built-in values, **True** or **False**.



```
print(type(True))
print(type(False))
```

```
<class 'bool'>
<class 'bool'>
```

Program to check the type of True and False Keywords in Python

Mutable Vs Immutable Data Types

Mutable Vs Immutable Data Types

Every variable in python holds an **instance of an object**. Whenever an object is instantiated, it is assigned a **unique object id**.

After generation of Object ID at the **runtime**, **object's data type cannot be changed**. However, **it's state** can be **changed** only if it is **MUTABLE.**

From these two points we can say that the object whose value can be changed is called mutable and the object whose value cannot be changed is called Immutable.

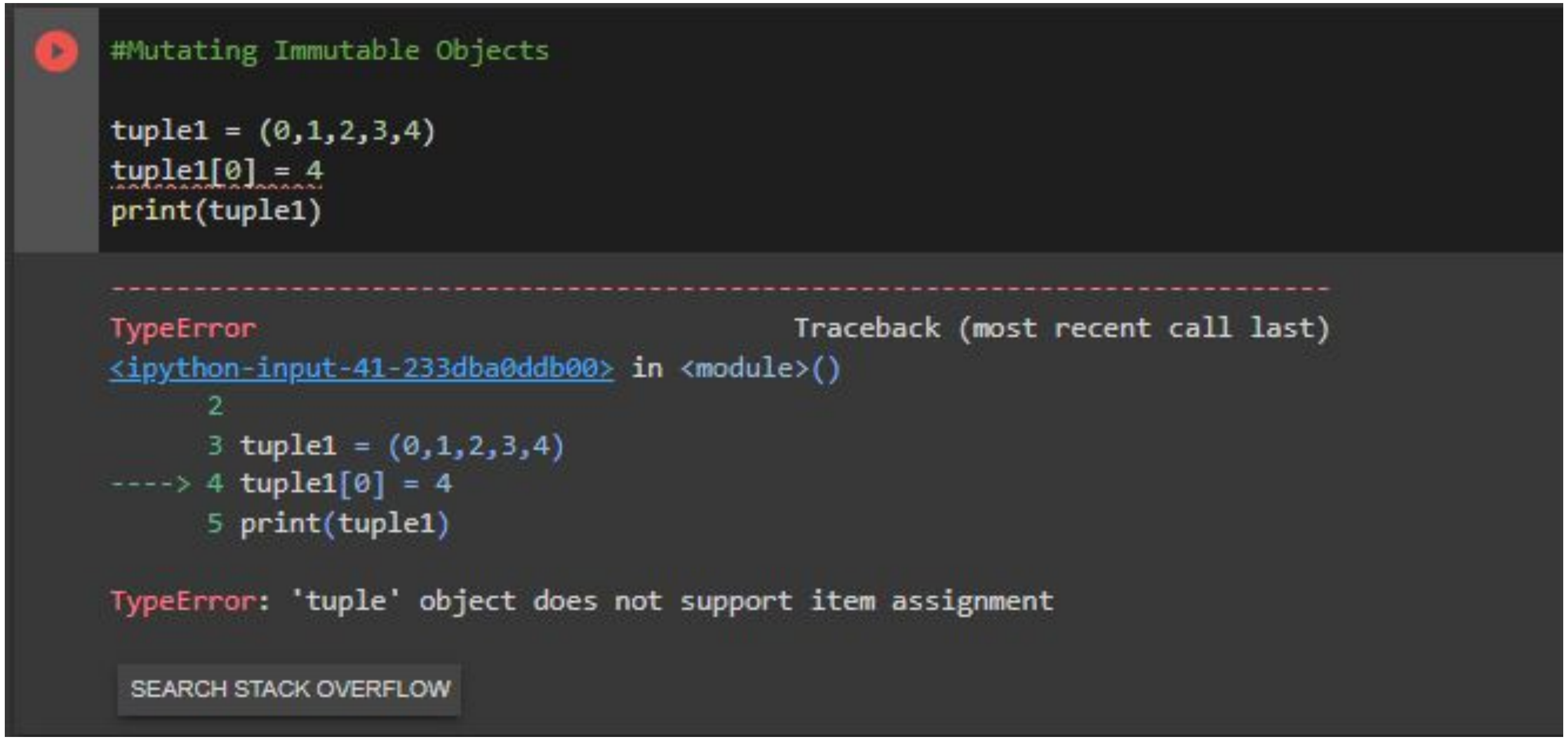
Mutable Vs Immutable Data Types

Mutable Data Types - List, Dictionary and Set

Immutable Data Types - int, float, boolean, tuple and string

Mutable Vs Immutable Data Types

What will happen if we try to mutate Immutable object?

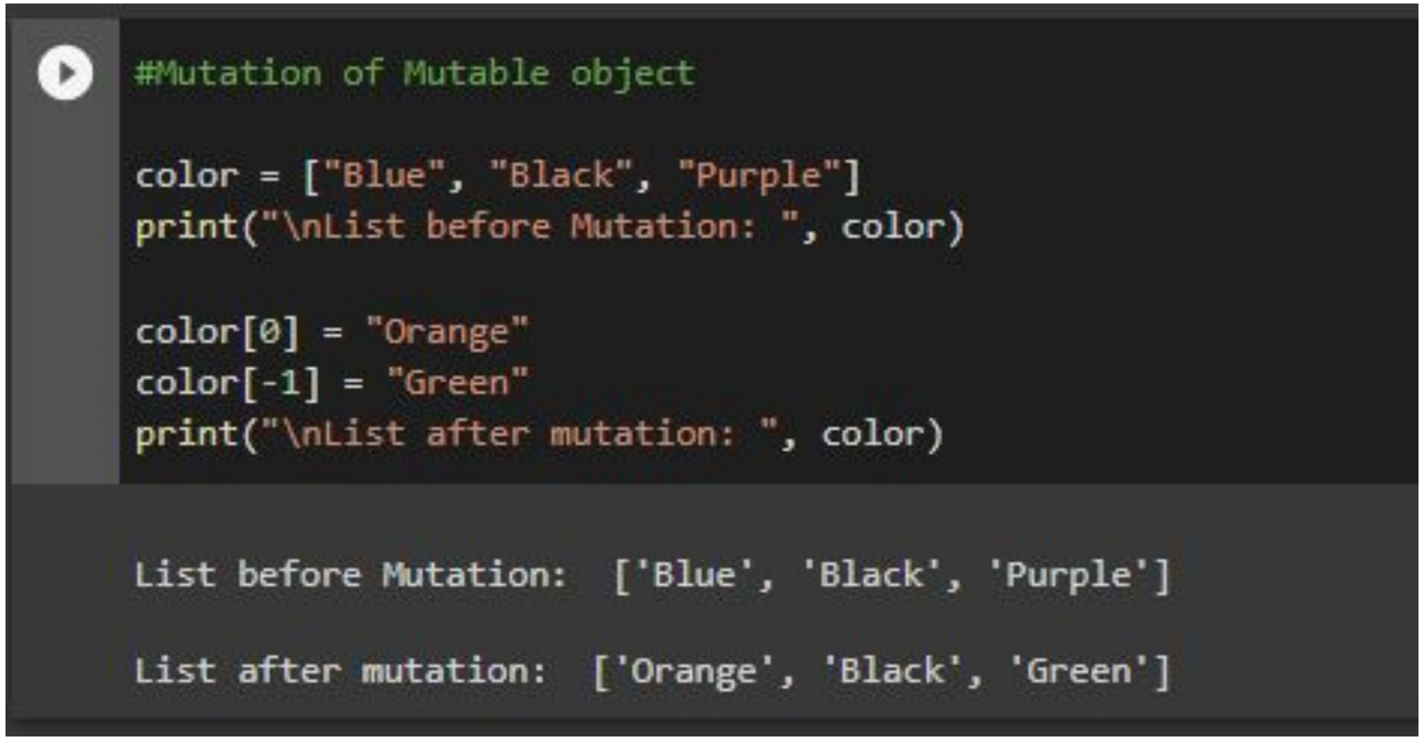


```
#Mutating Immutable Objects

tuple1 = (0,1,2,3,4)
tuple1[0] = 4
print(tuple1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-41-233dba0ddb00> in <module>()
      2
      3 tuple1 = (0,1,2,3,4)
----> 4 tuple1[0] = 4
      5 print(tuple1)

TypeError: 'tuple' object does not support item assignment
```

Mutable Vs Immutable Data Types

What will happen if we try to mutate Mutable object?



```
#Mutation of Mutable object

color = ["Blue", "Black", "Purple"]
print("\nList before Mutation: ", color)

color[0] = "Orange"
color[-1] = "Green"
print("\nList after mutation: ", color)
```

```
List before Mutation:  ['Blue', 'Black', 'Purple']

List after mutation:  ['Orange', 'Black', 'Green']
```

Result : Mutation will occur

Slicing in Python

Slicing in Python

Slicing is a Python feature that allows you to **access specific parts of a sequence**.

In slicing, we create a **subsequence**, which is essentially a **sequence that exists within another sequence**.

Slicing in Python

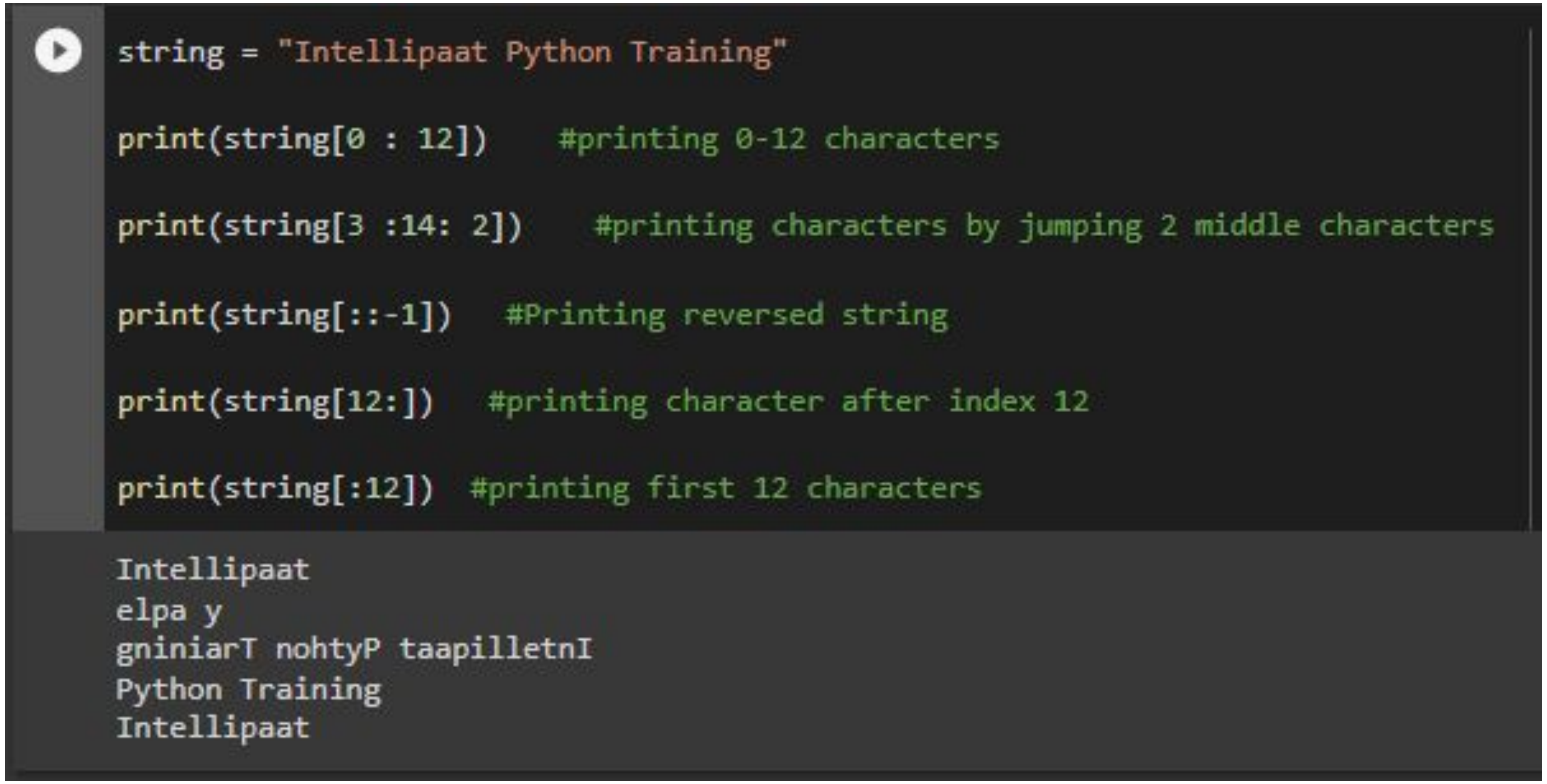
1. String Slicing:

```
String[start : end]

String[start : end : step]
```

Let’s perform few slicing operations on a String.

Slicing in Python

1. String Slicing:

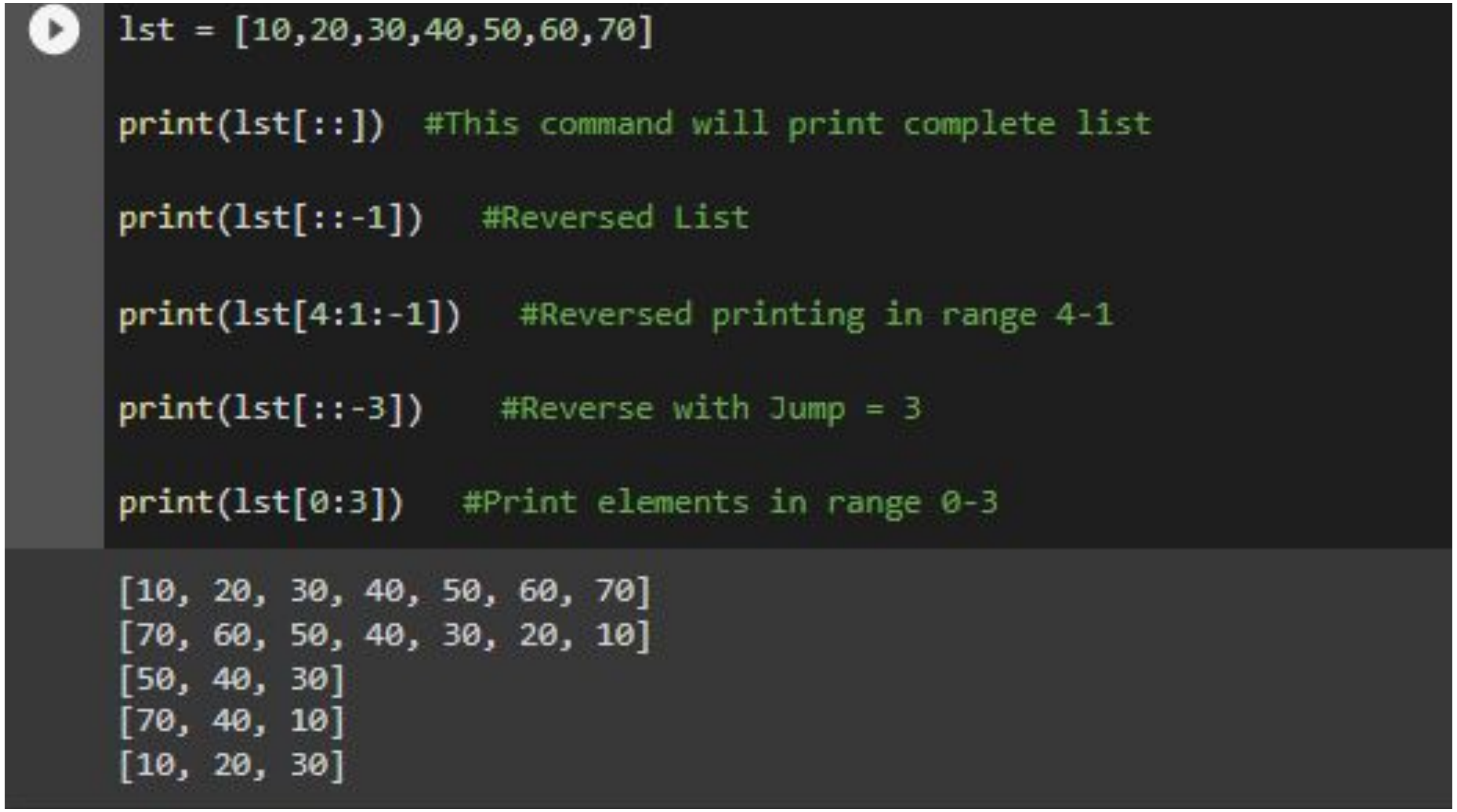


```
string = "Intellipaat Python Training"

print(string[0 : 12])    #printing 0-12 characters

print(string[3 :14: 2])    #printing characters by jumping 2 middle characters

print(string[::-1])   #Printing reversed string

print(string[12:])   #printing character after index 12

print(string[:12])  #printing first 12 characters
```

```
Intellipaat
elpa y
gniniarT nohtyP taapilletnI
Python Training
Intellipaat
```

Slicing in Python

2. List Slicing:



```
lst = [10,20,30,40,50,60,70]

print(lst[::])  #This command will print complete list

print(lst[::-1])   #Reversed List

print(lst[4:1:-1])   #Reversed printing in range 4-1

print(lst[::-3])    #Reverse with Jump = 3

print(lst[0:3])   #Print elements in range 0-3
```

```
[10, 20, 30, 40, 50, 60, 70]
[70, 60, 50, 40, 30, 20, 10]
[50, 40, 30]
[70, 40, 10]
[10, 20, 30]
```

Thank You

India: +91-7022374614

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor