Session 4

List

Python Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). In simple language, a list is a collection of things, enclosed in [ ] and separated by commas.

The list is a sequence data type which is used to store the collection of data.

If we want to represent a group of values as a single entity where insertion order is preserved and duplicates are allowed we should go for a list.

Example:

```
Var = ["I","LOVE","NEPAL"]
print(Var)
```

Output:

```
['I', 'LOVE', 'NEPAL']
```

Operations that can be performed in the Lists:

```
Var = ["I","LOVE","NEPAL"]
print(Var)
print(type(Var))


# 1. Create a list
my_list = [1, 2, 3, 4, 5, "Nepal", True, 5+2j]

# 2. Access elements
print("First element:", my_list[0])   # Access the first element
print("Last element:", my_list[-1])   # Access the last element

# 3. Modify an element
```

```
my_list[2] = 10   # Modify the third element
print("Modified list:", my_list)

# 4. Append an element
my_list.append(6)   # Add an element to the end of the list
print("List after appending:", my_list)

# 5. Remove an element
my_list.remove(4)   # Remove the element '4'
print("List after removing 4:", my_list)

# 6. Length of the list
print("Length of the list:", len(my_list))

# 7. Insert an element at a specific index
my_list.insert(1, 20)  # Insert '20' at index 1
print("List after insertion:", my_list)


# 9. Reverse the list
my_list.reverse()
print("Reversed list:", my_list)


# 10. Slicing
print(my_list[2:5])
```

Output:

```
[Running] python -u "g:\INTERN\03-Day3\lists.py"
['I', 'LOVE', 'NEPAL']
<class 'list'>
First element: 1
Last element: (5+2j)
Modified list: [1, 2, 10, 4, 5, 'Nepal', True, (5+2j)]
List after appending: [1, 2, 10, 4, 5, 'Nepal', True, (5+2j), 6]
List after removing 4: [1, 2, 10, 5, 'Nepal', True, (5+2j), 6]
Length of the list: 8
List after insertion: [1, 20, 2, 10, 5, 'Nepal', True, (5+2j), 6]
Reversed list: [6, (5+2j), True, 'Nepal', 5, 10, 2, 20, 1]
[True, 'Nepal', 5]
```

Tuple:

Python Tuple is a collection of objects separated by commas. In some ways, a tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

Example:

```
var = ("My", "name", "is","Khan")
print(var)
print(type(var))
```

Output

```
('My', 'name', 'is', 'Khan')
<class 'tuple'>
```

Operations in Python Tuple

```
var = ("My", "name", "is","Khan")
print(var)
print(type(var))
# 1. Create a tuple
my_tuple = (1, 2, 3, 4, 5, "Nepal", True, 5+2j)
print("Original tuple:", my_tuple)

# 2. Access elements
print("First element:", my_tuple[0])   # Access the first element
print("Last element:", my_tuple[-1])   # Access the last element

# 3. Find length of tuple
print("Length of the tuple:", len(my_tuple))

# 4. Count occurrences of an element
print("Count of 3 in tuple:", my_tuple.count(3))
```

```
# 5. Find index of an element
print("Index of element 4:", my_tuple.index(4))
```

Output:

```
('My', 'name', 'is', 'Khan')
<class 'tuple'>
Original tuple: (1, 2, 3, 4, 5, 'Nepal', True, (5+2j))
First element: 1
Last element: (5+2j)
Length of the tuple: 8
Count of 3 in tuple: 1
Index of element 4: 3
```

Note:

- Tuples are immutable, meaning their elements cannot be changed directly after creation.
- To modify a tuple, it must be converted to a list, the changes made, and then converted back to a tuple if needed.

Set:

A Set in Python programming is an unordered collection data type that is iterable and has no duplicate elements. While sets are mutable, meaning we can add or remove elements after their creation, the individual elements within the set must be immutable and cannot be changed directly. A set cannot have duplicate values.

Operations in Sets:

```
# 1. Create a set
my_set = {1, 2, 3, 4, 5}
print("Original set:", my_set)

# 2. Add an element to the set
my_set.add(6)   # Add element 6
print("Set after adding 6:", my_set)

# 3. Remove an element from the set
```

```python
my_set.remove(4)   # Remove element 4 (raises error if element doesn't exist)
print("Set after removing 4:", my_set)

# 4. Check if an element exists in the set
print("Is 3 in the set?", 3 in my_set)

# 5. Get the length of the set
print("Length of the set:", len(my_set))

# 6. Use set operations (union, intersection, difference)
another_set = {3, 5, 7, 8}

# Union: combines both sets
union_set = my_set.union(another_set)
print("Union of sets:", union_set)

# Intersection: common elements between both sets
intersection_set = my_set.intersection(another_set)
print("Intersection of sets:", intersection_set)

# Difference: elements in my_set but not in another_set
difference_set = my_set.difference(another_set)
print("Difference of sets:", difference_set)

# 7. Clear the set
my_set.clear()  # Removes all elements
print("Set after clearing:", my_set)
```

Output:

```
Original set: {1, 2, 3, 4, 5}
Set after adding 6: {1, 2, 3, 4, 5, 6}
Set after removing 4: {1, 2, 3, 5, 6}
Is 3 in the set? True
Length of the set: 5
Union of sets: {1, 2, 3, 5, 6, 7, 8}
Intersection of sets: {3, 5}
Difference of sets: {1, 2, 6}
Set after clearing: set()
```

Frozen sets:

Frozen sets in Python are immutable objects that only support methods and operators that produce a result without affecting the frozen set or sets to which they are applied. It can be done with the frozenset() method in Python.

While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

If no parameters are passed, it returns an empty frozenset.

Example:

```
s = {10,20,30}
print(type(s))
fs = frozenset(s)
print(type(fs))
```

Output:

```
<class 'set'>
<class 'frozenset'>
```

Dictionary in Python

In Python, a dictionary is a built-in data type that stores data in key-value pairs. It is an unordered, mutable, and indexed collection. Each key in a dictionary is unique and maps to a value. Dictionaries are often used to store data that is related, such as information associated with a specific entity or object, where you can quickly retrieve a value based on its key.
Example:

```
capitals = {"Bagmati":"Hetauda", "Koshi":"Biratnagar", "Karnali":"Surkhet",
"Gandaki":"Pokhara"}
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
print(type(capitals))
print(type(numbers))
print(capitals)
```

Output:

```
<class 'dict'>
<class 'dict'>
{'Bagmati': 'Hetauda', 'Koshi': 'Biratnagar', 'Karnali': 'Surkhet', 'Gandaki': 'Pokhara'}
```

Range

The Python range() function returns a sequence of numbers, in a given range. The most common use of it is to iterate sequences on a sequence of numbers using Python loops.

Syntax: range(start, stop, step)

Parameter :

- start: [optional] start value of the sequence
- stop: next value after the end value of the sequence
- step: [optional] integer value, denoting the difference between any two numbers in the sequence

Return : Returns an object that represents a sequence of numbers

Example:

```
# printing first 6
# whole number
for i in range(6):
    print(i, end=" ")
print()


Output:
0 1 2 3 4 5
```

None Keyword in Python

None is used to define a null value or Null object in Python. It is not the same as an empty string, a False, or a zero. It is a data type of the class NoneType object.

Referring to the null object in Python

Assigning a value of None to a variable is one way to reset it to its original, empty state.

Example:

```
a = None
# checking it's value
if a is None:
    print("a has a value of None")
else:
    print("a has a value")

Output:

a has a value of None
```