Python Dictionary

Python dictionary is a data structure that stores the value in key: value pairs.

[1] How to Create a Dictionary

Dictionary can be created by placing a sequence of elements within curly {} braces, separated by a 'comma'.

```
d1 = {1: 'parmar', 2: 'harshil', 3: 'ganpatbhai'}

print(d1)

# create dictionary using dict() constructor

d2 = dict(a = "parmar", b = "harshil", c = "ganpatbhai")

print(d2)
```

Output

```
{1: 'parmar', 2: 'harshil', 3: 'ganpatbhai'}

{'a': 'parmar', 'b': 'harshil', 'c': 'ganpatbhai'}
```

Accessing Dictionary Items

We can access a value from a dictionary by using the **key** within square brackets or get() method.

```
d = { "name": "Harshil", 1: "Python", (1, 2): [1,2,4] }

# Access using key

print(d["name"])

# Access using get()

print(d.get("name"))
```

Output

```
Harshil

Harshil
```

[2] Read/Adding and Updating Dictionary Items

We can add new key-value pairs or update existing keys by using assignment.

```
d = {1: 'P', 2: 'H', 3: 'G'}

# Read/Adding a new key-value pair

d["age"] = 22

# Updating an existing value

d[1] = "Python dict"

print(d)
```

Output

```
{1: 'Python dict', 2: 'H', 3: 'G', 'age': 22}
```

[3] Removing Dictionary Items

We can remove items from dictionary using the following methods:

- **del**: Removes an item by key.
- **pop():** Removes an item by key and returns its value.
- **clear():** Empties the dictionary.
- **popitem():** Removes and returns the last key-value pair

```
d = {1: 'Parmar', 2: 'Harshil', 3: 'Ganpatbhai', 'age':22}

# Using del to remove an item

del d["age"]

print(d)

# Using pop() to remove an item and return the value

val = d.pop(1)

print(val)

# Using popitem to removes and returns

# the last key-value pair.

key, val = d.popitem()

print(f"Key: {key}, Value: {val}")

# Clear all items from the dictionary

d.clear()
```

```
print(d)
```

Output

```
{1: 'Parmar', 2: 'Harshil', 3: 'Ganpatbhai'}

Parmar

Key: 3, Value: Ganpatbhai

{}
```

Python Sets

Python set is an unordered collection of multiple items having different datatypes. In Python, sets are mutable, unindexed and do not contain duplicates. The order of elements in a set is not preserved and can change.

- Can store None values.
- Implemented using hash tables internally.
- Do not implement interfaces like Serializable or Cloneable.
- Python sets are not inherently thread-safe; synchronization is needed if used across threads.

[1] **Creating a Set in Python**

In Python, the most basic and efficient method for creating a set is using curly braces.

```
set1 = {1, 2, 3, 4}

print(set1)
```

Output - {1, 2, 3, 4}

[1] Create Using the set() function

Python Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a 'comma'.

Note: *A Python set cannot contain mutable types such as lists or dictionaries, because they are unhashable.*

```
set1 = set()

print(set1)
```

```
set1 = set("PHG")

print(set1)

# Creating a Set with the use of a List

set1 = set(["Parmar", "Harshil", "Ganpatbhai"])

print(set1)

# Creating a Set with the use of a tuple

tup = ("Parmar", "Harshil", "Ganpatbhai")

print(set(tup))

# Creating a Set with the use of a dictionary

d = {"Parmar": 1, "Harshil": 2, "Ganpatbhai": 3}

print(set(d))
```

Output

```
set()

{'P', 'H', 'G'}

{'Parmar', 'Harshil', 'Ganpatbhai'}

{'Parmar', 'Harshil', 'Ganpatbhai'}

{'Parmar', 'Harshil', 'Ganpatbhai'}
```

[2] **Accessing/Read a Set in Python**

We can loop through a set to access set items as set is unindexed and do not support accessing elements by indexing. Also we can use in keyword which is membership operator to check if an item exists in a set.

```
set1 = set(["Parmar", "Harshil", "Ganpatbhai."])

# Accessing element using For loop

for i in set1:

    print(i, end=" ")

# Checking the element# using in keyword
```

print("Harshil" in set1)

Output -

Parmar

Harshil

Ganpatbhai.

True

[3] **Update a Set in Python**

Add single element = add()

numbers.add(5)

print(numbers) **output** - {1, 2, 5}

Add multiple elements = update()

numbers.update([6, 7, 8])

print(numbers) **output** - {1, 2, 5, 6, 7, 8}

Update using another set

numbers.update({9, 10})

print(numbers) **output** - {1, 2, 5, 6, 7, 8, 9, 10}

[4] **Removing Elements from the Set in Python**

We can remove an element from a set in Python using several methods: remove(), discard() and pop(). Each method works slightly differently :

- Using remove() Method or discard() Method
- Using pop() Method
- Using clear() Method

Using remove() Method or discard() Method

remove() method removes a specified element from the set. If the element is not present in the set, it raises a KeyError. discard() method also removes a specified element from the set. Unlike remove(), if the element is not found, it does not raise an error.

```
# Using Remove Method
set1 = {1, 2, 3, 4, 5}
set1.remove(3)
print(set1)
# Attempting to remove an element that does not exist
try:
   set1.remove(10)
except KeyError as e:
   print("Error:", e)
# Using discard() Method
set1.discard(4)
print(set1)
# Attempting to discard an element that does not exist
set1.discard(10)  # No error raised
print(set1)
```

Output

```
{1, 2, 4, 5}
Error: 10
{1, 2, 5}
{1, 2, 5}
```

Using pop() Method

pop() method removes and returns an arbitrary element from the set. This means we don't know which element will be removed. If the set is empty, it raises a KeyError.

Note: *If the set is unordered then there's no such way to determine which element is popped by using the pop() function.*

```
set1 = {1, 2, 3, 4, 5}
val = set1.pop()
print(val)
```

```
print(set1)

# Using pop on an empty set

set1.clear()  # Clear the set to make it empty

try:

   set1.pop()

except KeyError as e:

   print("Error:", e)
```

Output

```
1

{2, 3, 4, 5}

Error: 'pop from an empty set'
```

Using clear() Method

clear() method removes all elements from the set, leaving it empty.

```
set1 = {1, 2, 3, 4, 5}

set1.clear()

print(set1)
```

Output

```
set()
```

Python Lists

In Python, a list is a built-in data structure that can hold an ordered collection of items. Unlike arrays in some languages, Python lists are very flexible:

- Can contain duplicate items
- **Mutable:** items can be modified, replaced, or removed
- **Ordered:** maintains the order in which items are added
- **Index-based:** items are accessed using their position (starting from 0)
- Can store mixed data types (integers, strings, booleans, even other lists)

[1] **Creating a List**

Lists can be created in several ways, such as using square brackets, the list() constructor or by repeating elements. Let's look at each method one by one with example:

1. Using Square Brackets

We use square brackets [] to create a list directly.

```
a = [1, 2, 3, 4, 5] # List of integers

b = ['apple', 'banana', 'cherry'] # List of strings

c = [1, 'hello', 3.14, True] # Mixed data types

print(a)

print(b)

print(c)
```

Output

```
[1, 2, 3, 4, 5]

['apple', 'banana', 'cherry']

[1, 'hello', 3.14, True]
```

2. Using list() Constructor

We can also create a list by passing an iterable (like a tuple, string or another list) to the list() function.

```
a = list((1, 2, 3, 'apple', 4.5))

print(a)

b = list("GFG")

print(b)
```

Output

```
[1, 2, 3, 'apple', 4.5]

['G', 'F', 'G']
```

3. Creating List with Repeated Elements

We can use the multiplication operator * to create a list with repeated items.

```
a = [2] * 5

b = [0] * 7

print(a)

print(b)
```

Output

```
[2, 2, 2, 2, 2]

[0, 0, 0, 0, 0, 0, 0]
```

[2] **Accessing/Read List Elements**

Elements in a list are accessed using indexing. Python indexes start at 0, so a[0] gives the first element. Negative indexes allow access from the end (e.g., -1 gives the last element).

```
a = [10, 20, 30, 40, 50]

print(a[0])

print(a[-1])

print(a[1:4])   # elements from index 1 to 3
```

Output

```
10

50

[20, 30, 40]
```

[3] **Adding/Update Elements into List**

Adding Elements into List

We can add elements to a list using the following methods:

- **append():** Adds an element at the end of the list.
- **extend():** Adds multiple elements to the end of the list.
- **insert():** Adds an element at a specific position.
- **clear():** removes all items.

```
a = []
```

```
a.append(10)

print("After append(10):", a)

a.insert(0, 5)

print("After insert(0, 5):", a)

a.extend([15, 20, 25])

print("After extend([15, 20, 25]):", a)

a.clear()

print("After clear():", a)
```

Output

```
After append(10): [10]

After insert(0, 5): [5, 10]

After extend([15, 20, 25]): [5, 10, 15, 20, 25]

After clear(): []
```

Updating Elements into List

Since lists are mutable, we can update elements by accessing them via their index.

```
a = [10, 20, 30, 40, 50]

a[1] = 25

print(a)
```

Output

```
[10, 25, 30, 40, 50]
```

[4] Removing Elements from List

We can remove elements from a list using:

- **remove():** Removes the first occurrence of an element.
- **pop():** Removes the element at a specific index or the last element if no index is specified.
- **del statement:** Deletes an element at a specified index.

```
a = [10, 20, 30, 40, 50]
```

```
a.remove(30)

print("After remove(30):", a)

popped_val = a.pop(1)

print("Popped element:", popped_val)

print("After pop(1):", a)

del a[0]

print("After del a[0]:", a)
```

Output

```
After remove(30): [10, 20, 40, 50]

Popped element: 20

After pop(1): [10, 40, 50]

After del a[0]: [40, 50]
```