Data Engineering

Name: Ramireddy Preethi

Batch: Python Batch 2

Dictionary:

Dictionary are mutable, unordered collection with elements in the form of a key : value pairs that associate keys to values.

With list, tuples and strings we need to know the index of individual element to access but with Dictionaries user need to search value based on key.

Characteristics:

1) Unordered: Dictionaries do not maintain order (before Python 3.7). In Python 3.7 and later, dictionaries maintain insertion order.

2) Mutable: You can add, update, or delete key-value pairs in a dictionary.

3) Keys are unique: If you add a duplicate key, the most recent value for that key will overwrite the previous one.

Example:

```
my_dict = {"a": 1, "b": 2}

my_dict["a"] = 3  # The key "a" already exists, so the value 1 is replaced by 3

print(my_dict)  # Output: {'a': 3, 'b': 2}
```

4) Keys must be immutable: Keys can be of any immutable data type, such as strings, numbers, or tuples. However, values can be of any data type.

Example:

1) Dictionary with immutable keys

```
my_dict = { "name": "Alice",          # String key

            42: "The answer",         # Integer key

           (1, 2): "Tuple as key"     # Tuple key

           }

print(my_dict)  # Output: {'name': 'Alice', 42: 'The answer', (1, 2): 'Tuple as key'}
```

2) Dictionary with mutable keys

```
my_dict = {

        [1, 2, 3]: "List as key"    # This will raise a TypeError

        }
```

5) Indexing and Slicing concepts are not applicable

Creating Dictionaries:

1) If you want to create an empty dictionary, we use the following approach:

d = { }

print(type(d)) **# Output: <class ‘dict’>**

2) dict()

d = dict()

print(type(d)) **# Output: <class ‘dict’>**

# to add entries into dictionaries

d[100]=“preethi"

d[200]=“pooja"

d[300]="sreeja"

print(d) **# output:{100: ‘preethi, 200: ‘pooja', 300: 'sreeja'}**

3) If we know data in advance

d={100:‘preethi' ,200:‘varun', 300:‘pooja'}

print(d) **#Output: {100: ‘preethi', 200: ‘varun', 300: ‘pooja'}**

Accessing data from dictionary:

We can access data by using keys

Syntax: d [key]

1. Accessing Values by Key

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

Accessing a value by key

print(my_dict["name"]) **# Output: Alice**

print(my_dict["country"]) **# This will raise a KeyError if "country" is not in the dictionary**

print(my_dict) **# {"name": "Alice", "age": 25, "city": "New York"}**

2) Using get() Method:

get() method is safer way to access values. It allows you to specify a default value if key doesnot exist,which helps to avoid KeyError.

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

print(my_dict.get("name"))   **# Output: Alice**

print(my_dict.get("country", "USA")) **# Output: USA (default value if "country" key is missing)**

3) Accessing All Keys with Keys():

The Keys() method returns a object containing all keys in dictionary.

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

print(my_dict.keys()) **# Output: dict_keys(['name', 'age', 'city']**

keys_list = list(my_dict.keys())

print(keys_list) **# Output: ['name', 'age', 'city']**

4) Accessing All Values with Values():

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

print(my_dict.values()) **# Output: dict_values(['Alice', 25, 'New York'])**

values_list = list(my_dict.values())

print(values_list) **# Output: ['Alice', 25, 'New York']**

5) Accessing All Key-Value Pairs with items():

Items() method returns a view of key-value pair as tuple.

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

```
print(my_dict.items())  # Output: dict_items([('name', 'Alice'), ('age', 25), ('city', 'New York')])

# Convert to a list of tuples

items_list = list(my_dict.items())

print(items_list)  # Output: [('name', 'Alice'), ('age', 25), ('city', 'New York')]

for key,val in items_list:

   print(key,val) # name Alice

                  age  25

                  city New York
```

Adding and updating Elements to Dictionary:

1) Adding a Single Key-Value Pair

```
my_dict = {"name": "Alice", "age": 25}

my_dict["city"] = "New York“   # Adding a new key-value pair

print(my_dict) # Output: {'name': 'Alice', 'age': 25, 'city': 'New York'}

 my_dict["age"] = 26  # if key already exist,value will be updated

 print(my_dict) # Output: {'name': 'Alice', 'age': 26, 'city': 'New York‘}
```

2) Using Update() Method: to add multiple Key-value Pairs

```
my_dict = {"name": "Alice", "age": 25}

my_dict.update({"country": "USA", "email": "alice@example.com"})

print(my_dict)

# Output: {'name': 'Alice', 'age': 26, 'country': 'USA', 'email': 'alice@example.com'}
```

3) Using setdefault() to Add with Default Value:

Adds a key-value pair if key not exists otherwise no change

Example:

```
my_dict = {"name": "Alice", "age": 25}

my_dict.setdefault("city", "New York") # Adding a key with a default value if it doesn't exist

print(my_dict) # Output: {'name': 'Alice', 'age': 25, 'city': 'New York'}
```

my_dict.setdefault("age", 30) # Attempting to add an existing key with setdefault (no change)

print(my_dict) **# Output: {'name': 'Alice', 'age': 25, 'city': 'New York'}**

Deleting Elements from Dictionary:

1) Using del:

Del allows us to delete specific key and its value, if key doesnot exist raise KeyError.

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

del my_dict["age"]

print(my_dict) **# Output: {'name': 'Alice', 'city': 'New York'}**

# Attempting to delete a non-existent key raises KeyError

# del my_dict["country"] **# this will raise KeyError**

2) pop():

Removes key and returns value, if not exist key Error

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

age = my_dict.pop("age")

print(age)   **# Output: 25**

print(my_dict)   **# Output: {'name': 'Alice', 'city': 'New York'}**

# Popping a non-existent key with a default value

country = my_dict.pop("country", "Not Found")

print(country)   **# Output: Not Found**

3) popitem(): Removes and returns last inserted key-value pair from the dictionary as tuple.

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

item = my_dict.popitem() # Popping the last inserted key-value pair print(item)   **# Output: ('city', 'New York')**

print(my_dict)   **# Output: {'name': 'Alice', 'age': 25}**

# If the dictionary is empty, popitem() raises KeyError

my_dict.clear() # this will make the dictionary empty

my_dict.popitem() **# Raises KeyError**

4) clear(): Removes all key-value pairs and make dictionary empty

Example:

my_dict = {"name": "Alice", "age": 25, "city": "New York"}

my_dict.clear() # Clearing the entire dictionary

print(my_dict)   **# Output: {}**

Methods:

copy(): creates new copy of dictionary with same values but changes to copied dictionary doesnot effect original dictionary.

Example:

my_dict = {"name": "Alice", "age": 25}

copy_dict = my_dict.copy()

copy_dict["age"] = 30 # Modifying the copied dictionary

print(my_dict)   **# Output: {'name': 'Alice', 'age': 25} (original remains unchanged)**

print(copy_dict)   **# Output: {'name': 'Alice', 'age': 30} (copy is modified)**

fromkeys(): creates a new dictionary with specified key and default values.

Its default value is **None.**

Example:

# Creating a dictionary with specified keys and a default value of None

keys = ['name', 'age', 'city']

new_dict = dict.fromkeys(keys)

print(new_dict) **# Output: {'name': None, 'age': None, 'city': None}**

Example:

```
# Creating a dictionary with specified keys and a default value of 0
keys = ['a', 'b', 'c']
new_dict = dict.fromkeys(keys, 0)
print(new_dict)  # Output: {'a': 0, 'b': 0, 'c': 0}
```

Dictionary Comprehensions:

Allows us to create a new dictionary in concise way.

Syntax:

{key_expression: value_expression for item in iterable if condition}

Example:

```
# Creating a dictionary of squares
squares = {x: x**2 for x in range(5)}
print(squares)   # Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```