PYTHON TUTORIAL FOR BEGINNERS

Source: www.youtube.com/@RishabhMishraOfficial

Chapter - 19

Dictionary in Python

- What is a Dictionary
- Create Dictionary
- Access Dictionary Values
- Dictionary Methods
- Dictionary – Add, Modify & Remove Items
- Dictionary Iteration
- Nested Dictionary
- Dictionary Comprehensions

Dictionary in Python

A **dictionary** is a data structure in Python that stores data in **key-value pairs**. Dictionary items (key – value pair) are ordered, changeable, and do not allow duplicates.

- **Key:** Must be unique and immutable (strings, numbers, or tuples).
- **Value:** Can be any data type and does not need to be unique.

Example: Simple dictionary with three key-value pairs

```
student = {
      1: "Class-X",
    "name": "Madhav",
    "age": 20
}
```

Create Dictionary in Python

Method-1: We create a dictionary using **curly braces {}** and separating keys and values with a **colon**.

```
# Syntax
my_dict =
{"key1": "value1", "key2": "value2", "key3": "value3", …}
```

```
# Empty dictionary
empty_dict = {}

# Dictionary with data
        cohort = {
            "course": "Python",
            "instructor": "Rishabh Mishra",
            "level": "Beginner"
        }
```

Method-2: Using dict() constructor

Pass key-value pairs as keyword arguments to dict()

```
person = dict(name="Madhav", age=20, city="Mathura")
print(person)

# Output: {'name': 'Madhav', 'age': 20, 'city':
'Mathura'}
```

Method-3: Using a List of Tuples

Pass a list of tuples, where each tuple contains a key-value pair.

```
student = dict([("name", "Madhav"), ("age", 20),
("grade", "A")])
print(student)

# Output: {'name': 'Madhav', 'age': 20, 'grade': 'A'}
```

Access Dictionary Values

Access dictionary values by using the **key** name inside **square brackets**.

Example:

```
        student = {
             1: "Class-X",
            "name": "Madhav",
            "age": 20
        }
```

```
# print value based on respective key-names
print(student["name"])  # Output: Madhav
print(student["age"])  # Output: 20
```

Dictionary Methods

Python provides several **built-in methods** to use on dictionary.

Methods	Description
values()	Returns a list of all values in the dictionary
fromkeys()	Returns a dictionary with specified keys and value
get()	Returns value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
update()	Updates the dictionary with the specified key-value pairs
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary

Here are a few useful methods:

- **.keys():** Returns all keys in the dictionary.
- **.values():** Returns all values in the dictionary.
- **.items():** Returns all key-value pairs.
- **.get():** Returns value for a key (with an optional default if key is missing).

Examples

```
print(student.keys())       # All keys
print(student.values())     # All values
print(student.items())      # All key-value pairs
print(student.get("name")) # Safe way to access a value
```

Dictionary – Add, Modify & Remove Items

1. Add or **Modify Item:** Use **assign-operator '='** to add/modify items in a dictionary.

```
# Adding a new key-value pair
student["email"] = "madhav@example.com"

# Modifying an existing value
student["age"] = 25
```

2. Remove Item: Use **del** or **.pop()** to remove items from a dictionary.

```
# Remove with del
del student["age"]

# Remove with pop() and store the removed value
email = student.pop("email")
print(email)  # Output: madhav@example.com
```

Dictionary Iterations

A dictionary can be iterated using **for loop**. We can loop through dictionaries by keys, values, or both.

```
# Loop through keys
    for key in student:
        print(key)
# Loop through values
    for value in student:
        print(student[value])
# Loop through values: using values() method
    for value in student.values():
        print(value)
# Loop through both keys and values
    for key, value in student.items():
        print(key, value)
```

Nested Dictionary

Dictionaries can contain other dictionaries, which is useful for storing more complex data.

```
# nested dictionaries
    students = {
        "student1": {
            "name": "Madhav",
            "age": 20,
            "grade": "A"
        },
        "student2": {
            "name": "Keshav",
            "age": 21,
            "grade": "B"
        }
    }
    print(students["student1"]["name"])  # Output: Madhav
```

Dictionary Comprehension

A dictionary comprehension allows you to create dictionaries in a concise way.

```
# Syntax:
new_dict =
{key_expression: value_expression for item in iterable if
condition}

# Example: Creating a dictionary with square numbers
squares = {x: x * x for x in range(1, 6)}
print(squares)

# Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Dictionary Common Use Cases

- **User Profiles in Web Applications:** Store user details like name, email, etc.
- **Product Inventory Management:** Keep track of stock levels for products in an e-commerce system.
- **API Responses**: Parse JSON data returned from APIs (e.g., weather data).
- **Grouping Data:** Organize data into categories. Example: grouped = {"fruits": ["apple", "banana"], "veggies": ["carrot"]}
- **Caching:** Store computed results to reuse and improve performance. Example: cache = {"factorial_5": 120}
- **Switch/Lookup Tables:** Simulate switch-case for decision-making.

```
# Example:
actions = {"start": start_fn, "stop": stop_fn}
actions["start"]()
```

Python Tutorial Playlist: **Click Here**

https://www.youtube.com/playlist?list=PLdOKnrf8EcP384Ilxra4UlK9BDJGwawg9