Data structures in Python-

{Dictionaries}

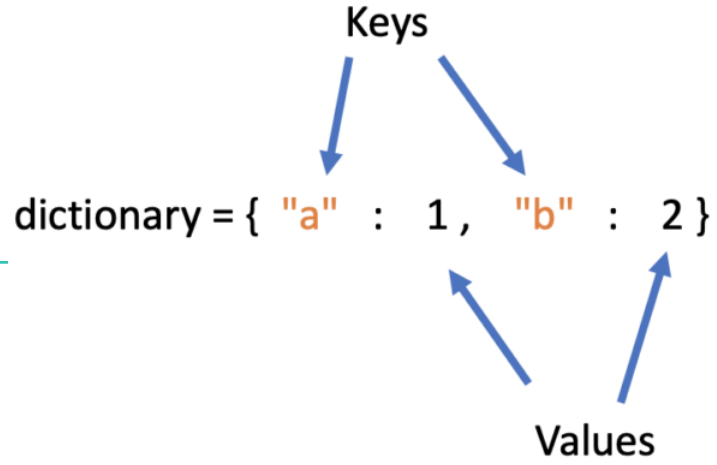
By

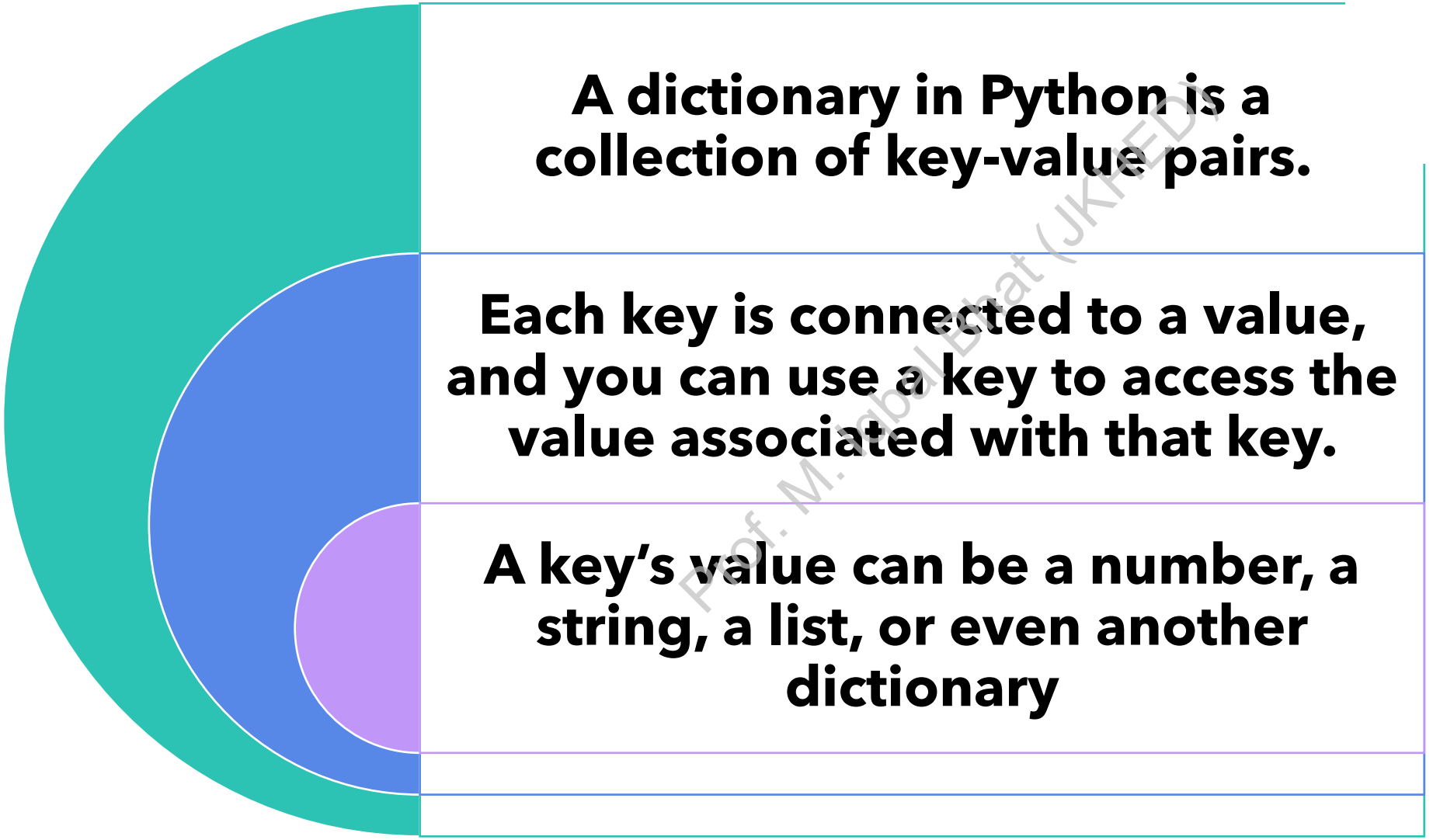
Prof. Muhammad Iqbal Bhat

Department of Higher Education

Government Degree College Beerwah

{Dictionaries} in Python:





A dictionary in Python is a collection of key-value pairs.

Each key is connected to a value, and you can use a key to access the value associated with that key.

A key's value can be a number, a string, a list, or even another dictionary

Creating and modifying {Dictionaries}:

In Python, a dictionary is wrapped in braces ({}) with a series of key-value pairs inside the braces, as shown in the earlier

alien_0 = {'color': 'green', 'points': 5}

Accessing Values in a Dictionary: To get the value associated with a key, give the name of the dictionary and then place the key inside a set of square brackets, as shown here: **print(alien_0['color'])**

Dictionaries are dynamic structures, and you can add new key-value pairs to a dictionary at any time. To add a new key-value pair, you would give the name of the dictionary followed by the new key in square brackets, along with the new value.

alien_0['x_position'] = 0

alien_0['y_position'] = 25

Modifying values in a {dictionary}:

To modify a value in a dictionary, give the name of the dictionary with the key in square brackets and then the new value you want associated with that key.

```
alien_0['color'] = 'yellow'
```

Removing Key-Value pair: We can use del statement to completely remove a key-value pair from dictionary

```
del alien_0['points']
```

Accessing values in a {dictionary}:

Using keys in square brackets to retrieve the value you're interested in from a dictionary might cause one potential problem: if the key you ask for doesn't exist, you'll get an error

```
alien_0 = {'color': 'green', 'speed': 'slow'}

print(alien_0['points'])    #Error
```

For dictionaries specifically, you can use the get() method to set a default value that will be returned if the requested key doesn't exist.

```
point_value = alien_0.get('points', 'No point value assigned.')
print(point_value)
```

Looping Through Elements of a Dictionary

- Looping through keys:

```
# Creating a dictionary
my_dict = {"name": "Alice", "age": 25, "city": "New York"}

# Looping through keys
for key in my_dict:
    print(key)

# Output: name
#         age
#         city
```

Looping Through Elements of a Dictionary

- Looping through values:

```
# Creating a dictionary
my_dict = {"name": "Alice", "age": 25, "city": "New York"}

# Looping through values
for value in my_dict.values():
    print(value)

# Output: Alice
#         25
#         New York
```

Looping Through Elements of a Dictionary

- Looping through both key and values:

```
# Creating a dictionary
my_dict = {"name": "Alice", "age": 25, "city": "New York"}

# Looping through keys and values
for key, value in my_dict.items():
    print(key, value)

# Output: name Alice
#         age 25
#         city New York
```

Looping Through Elements of a Dictionary

- Looping through a subset of keys:

```
# Creating a dictionary
my_dict = {"name": "Alice", "age": 25, "city": "New York"}

# Looping through selected keys
for key in my_dict.keys():
    if key != "city":
        print(key)

# Output: name
#         age
```

Looping Through Elements of a Dictionary

- Looping through a subset of values:

```
# Creating a dictionary
my_dict = {"name": "Alice", "age": 25, "city": "New York"}

# Looping through selected values
for value in my_dict.values():
    if isinstance(value, str):
        print(value)

# Output: Alice
#         New York
```

Looping Through Elements of a Dictionary

- Looping Through a Dictionary's Keys in a Particular Order:

```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'rust',
'phil': 'python',
}
for name in sorted(favorite_languages.keys()):
    print(f"{name.title()}, thank you for taking the poll.")
```

A List in a Dictionary:

```
# Store information about a pizza being ordered.
pizza = {
    'crust': 'thick',
    'toppings': ['mushrooms', 'extra cheese'],
}
# Summarize the order.
1 print(f"You ordered a {pizza['crust']}-crust pizza "
"with the following toppings:")
2 for topping in pizza['toppings']:
print(f"\t{topping}")
```

A List in a Dictionary:

```
favorite_languages = {
    'jen': ['python', 'rust'],
    'sarah': ['c'],
    'edward': ['rust', 'go'],
    'phil': ['python', 'haskell'],
}
for name, languages in favorite_languages.items():
    print(f"\n{name.title()}'s favorite languages are:")
    for language in languages:
        print(f"\t{language.title()}")
```

A Dictionary in a Dictionary:

```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
    },
    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
    },
    }
for username, user_info in users.items():
    print(f"\nUsername: {username}")
    full_name = f"{user_info['first']} {user_info['last']}"
    location = user_info['location']
    print(f"\tFull name: {full_name.title()}")
    print(f"\tLocation: {location.title()}")
```

Common methods for dictionary manipulation

clear(): Removes all key-value pairs from the dictionary..

copy()): Returns a shallow copy of the dictionary.

fromkeys(seq, val): Returns a new dictionary with keys from and values set to values set to val

get(key, default=None): Returns the value associated with , or if is not found.

pop(key, default=None): Removes the key-value pair with from the dictionary and returns the valueof the set are present in the other set, and False otherwise

update(other): Returns a view object that contains values of the dictionary.

Examples:

```
# Creating a dictionary
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}

# Using some of the methods on the dictionary
my_dict.clear()
new_dict = my_dict.copy()
new_dict = dict.fromkeys(['a', 'b', 'c'], 0)
value = my_dict.get('name', 'Unknown')
key_value_pairs = my_dict.items()
keys = my_dict.keys()
value = my_dict.pop('name', 'Unknown')
key, value = my_dict.popitem()
value = my_dict.setdefault('name', 'Unknown')
my_dict.update({'name': 'Bob', 'age': 30})
values = my_dict.values()
```

Uses of {Dictionaries}

1. Storing and accessing key-value pairs

```
# Example: Storing information about a person
person = {'name': 'Alice', 'age': 30, 'gender': 'female'}
print(person['name'])  # Output: 'Alice'
```

2. Counting occurrences of items

```
# Example: Counting the occurrences of words in a sentence
sentence = "The quick brown fox jumps over the lazy dog"
word_counts = {}
for word in sentence.split():
    if word not in word_counts:
        word_counts[word] = 1
    else:
        word_counts[word] += 1
print(word_counts)  # Output: {'The': 1, 'quick': 1, 'brown': 1,
'fox': 1, 'jumps': 1, 'over': 1, 'the': 1, 'lazy': 1, 'dog': 1}
```

3. Grouping and organizing data:

```
# Example: Grouping students by grade level
students = [
    {'name': 'Alice', 'grade': 12},
    {'name': 'Bob', 'grade': 10},
    {'name': 'Charlie', 'grade': 11},
    {'name': 'Dave', 'grade': 12},
    {'name': 'Emily', 'grade': 10}
]
students_by_grade = {}
for student in students:
    if student['grade'] not in students_by_grade:
        students_by_grade[student['grade']] = []
    students_by_grade[student['grade']].append(student['name'])
print(students_by_grade)  # Output: {12: ['Alice', 'Dave'], 10:
['Bob', 'Emily'], 11: ['Charlie']}
```

4. Creating lookup tables

```
# Example: Creating a dictionary to look up
phone numbers by name
phonebook = {'Alice': '555-1234', 'Bob': '555-
5678', 'Charlie': '555-9012'}
print(phonebook['Alice'])  # Output: '555-1234'
```

5. Building complex data structures:

```
# Example: Building a graph using dictionaries
graph = {
    'A': ['B', 'C'],
    'B': ['C', 'D'],
    'C': ['D'],
    'D': ['C'],
    'E': ['F'],
    'F': ['C']
}
```

Conclusion:

Dictionaries are a fundamental data structure in Python that allow you to store and access data using key-value pairs.

Dictionary keys must be unique and immutable, while values can be of any data type.

There are many built-in methods and functions available for working with dictionaries in Python, such as keys(), values(), items(), and get().

Dictionaries can be used in many different ways, such as counting occurrences of items, grouping and organizing data, creating lookup tables, and building complex data structures.

When using dictionaries, it is important to be mindful of their performance characteristics, as certain operations can be slower than others depending on the size of the dictionary and the specific use case.

Questions?