😊 Dictionaries in Python

A **dictionary** in Python is a built-in data structure that stores data in **key-value pairs**. It is **unordered**, **mutable**, and allows for fast lookups.

Characteristics of Dictionaries

1. **Key-Value Pairs** – Each element consists of a unique key and an associated value.
2. **Unordered** – The order of items is not guaranteed (before Python 3.7, but since then, insertion order is preserved).
3. **Mutable** – Values can be modified, added, or removed.
4. **Keys Must Be Unique** – Duplicate keys are not allowed.
5. **Keys Must Be Immutable** – Keys must be of immutable data types like strings, numbers, or tuples.

Basic Dictionary Operations

- **Creating a Dictionary**: Defined using curly braces {} with key-value pairs separated by colons.
- **Accessing Values**: Values are accessed using keys.
- **Modifying Values**: Values can be updated using the key.
- **Adding Items**: New key-value pairs can be inserted.
- **Removing Items**: Methods like pop(), del, and popitem() are used.
- **Iterating Through a Dictionary**: Loops can be used to traverse keys, values, or both.

Common Dictionary Methods

- keys() – Returns all keys.
- values() – Returns all values.
- items() – Returns key-value pairs.
- get(key, default) – Retrieves a value safely.
- update(dict) – Merges two dictionaries.
- clear() – Empties the dictionary.

Dictionaries are widely used for efficient data retrieval and structured storage.

😄 To begin with the Lab:

1. A Python dictionary consists of a key and then an associated value. That value can be almost any Python object.
2. As you can see below, we defined a dictionary, then we get the value from one of them.

```
[1]: # Make a dictionary with {} and : to signify a key and a value
     my_dict = {'key1':'value1','key2':'value2'}
```

```
[2]: # Call values by their key
     my_dict['key2']
```

```
[2]: 'value2'
```

3. It's important to note that dictionaries are very flexible in the data types they can hold.

```
[3]: my_dict = {'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}
```

```
[4]: # Let's call items from the dictionary
     my_dict['key3']
```

```
[4]: ['item0', 'item1', 'item2']
```

```
[5]: # Can call an index on that value
     my_dict['key3'][0]
```

```
[5]: 'item0'
```

```
[6]: # Can then even call methods on that value
     my_dict['key3'][0].upper()
```

```
[6]: 'ITEM0'
```

4. We can affect the values of a key as well. To do so we get the value of key 1 and then we made the value of key 1 zero by subtracting it.

```
[7]: my_dict['key1']
```

```
[7]: 123
```

```
[8]: # Subtract 123 from the value
     my_dict['key1'] = my_dict['key1'] - 123
```

```
[9]: #Check
     my_dict['key1']
```

```
[9]: 0
```

5. A quick note, Python has a built-in method of doing a self **subtraction or addition** (or multiplication or division). We could have also used **+= or -=** for the above statement.

[10]:
```
# Set the object equal to itself minus 123
my_dict['key1'] -= 123
my_dict['key1']
```

[10]:
```
-123
```

6. We can also create keys by assignment. For instance, if we started with an empty dictionary, we could continually add to it.

[11]:
```
# Create a new dictionary
d = {}
```

[12]:
```
# Create a new key through assignment
d['animal'] = 'Dog'
```

[13]:
```
# Can do this with any object
d['answer'] = 42
```

[14]:
```
#Show
d
```

[14]:
```
{'animal': 'Dog', 'answer': 42}
```

7. Now we will understand what nesting with dictionaries is; you're starting to see how powerful Python is with its flexibility of nesting objects and calling methods on them.

[15]:
```
# Dictionary nested inside a dictionary nested inside a dictionary
d = {'key1':{'nestkey':{'subnestkey':'value'}}}
```

```
That's a quite the inception of dictionaries! Let's see how we can grab that value:
```

[16]:
```
# Keep calling the keys
d['key1']['nestkey']['subnestkey']
```

[16]:
```
'value'
```

8. Below are some more examples of dictionary methods.

[17]:
```
# Create a typical dictionary
d = {'key1':1,'key2':2,'key3':3}
```

•••

[18]:
```
# Method to return a list of all keys
d.keys()
```

[18]:
```
dict_keys(['key1', 'key2', 'key3'])
```

[19]:
```
# Method to grab all values
d.values()
```

[19]:
```
dict_values([1, 2, 3])
```

[20]:
```
# Method to return tuples of all items  (we'll learn about tuples soon)
d.items()
```

[20]:
```
dict_items([('key1', 1), ('key2', 2), ('key3', 3)])
```