Part 1: Squared Numbers Dictionary

```python
squared_num = {x: x**2 for x in range(6)}
```

- A **dictionary comprehension** creates:

  ```python
  {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
  ```

  Each key is `x`, and its value is `x**2` (x squared).

Then, you accessed individual items:

```python
squared_num[0] → 0 squared_num[5] → 25 squared_num[3] → 9 squared_num[2] → 4
```

Then you cleared the dictionary:

```python
squared_num.clear()
```

- Now `squared_num` is `{}`, an empty dictionary.

🗂️ Part 2: Using `dict.fromkeys()`

Step 1: Creating a list of keys

```python
keys = ["Masala", "Ginger", "Lemon"]
```

Now `keys` looks like:

```python
```

```
['Masala', 'Ginger', 'Lemon']
```

Step 2: Creating a dictionary from keys with the same default value

```python
default_value = "Delicious" new_dict = dict.fromkeys(keys, default_value)
```

This creates:

```python
{'Masala': 'Delicious', 'Ginger': 'Delicious', 'Lemon': 'Delicious'}
```

🧠 **Important Note**: All values are `'Delicious'`, but each key has its **own copy of the string** (since strings are immutable).

❗ Step 3: Using `dict.fromkeys()` without specifying a value

```python
new_dict = dict.fromkeys(keys)
```

If you don't provide a value, it defaults to `None`:

```python
{'Masala': None, 'Ginger': None, 'Lemon': None}
```

But in this case, you ran:

```python
new_dict = dict.fromkeys(keys, keys)
```

This sets the **same list (`keys`) as the value for each key**:

```python
{ 'Masala': ['Masala', 'Ginger', 'Lemon'], 'Ginger': ['Masala', 'Ginger', 'Lemon'],
```

```
 'Lemon': ['Masala', 'Ginger', 'Lemon'] }
```

🧠 **Important Warning**:

- All three keys **share the same list**.
- If you change `new_dict["Masala"][0] = "Spicy"` it would also change the list for the other two keys.

🔁 Summary

Code	What It Does
`dict.fromkeys(keys, "value")`	Creates a dictionary with each key having the same value
`dict.fromkeys(keys, keys)`	Each key points to the **same list**, which can be risky
`.clear()`	Empties a dictionary