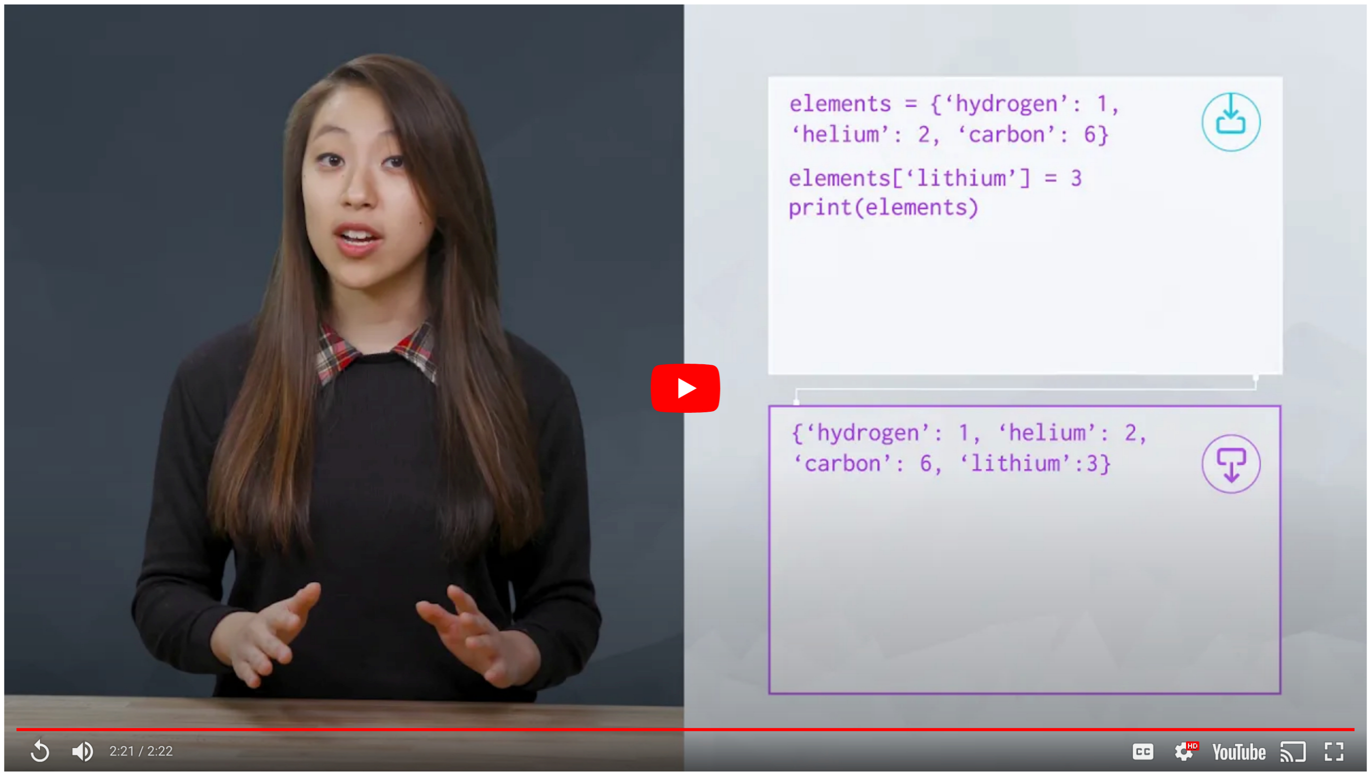
Dictionaries and Identity Operators



```
elements = {'hydrogen': 1,
'helium': 2, 'carbon': 6}
elements['lithium'] = 3
print(elements)
```

```
{'hydrogen': 1, 'helium': 2,
'carbon': 6, 'lithium':3}
```

Dictionaries and Identity Operators

Dictionaries

A **dictionary** is a mutable data type that stores mappings of unique **keys** to **values**.
Here's a dictionary that stores elements and their atomic numbers.

```
elements = {"hydrogen": 1, "helium": 2, "carbon": 6}
```

In general, dictionaries look like key-value pairs, separated by commas:

```
{key1:value1, key2:value2, key3:value3, key4:value4, ...}
```

Dictionaries are mutable, but their keys need to be any immutable type, like strings, integers, or tuples. It's not even necessary for every key in a dictionary to have the same type! For example, the following dictionary is perfectly valid:

```
random_dict = {"abc": 1, 5: "hello"}
```

This dictionary has two keys: "abc" and 5. The first key has a string type, and the second key has an integer type. And the dictionary has two values: 1 and "hello".

We can look up values in the dictionary using square brackets "[]" around the key, like :

`dict_name[key]`.

For example, in our random dictionary above, the value for `random_dict["abc"]` is 1, and the value for `random_dict[5]` is "hello".

In our elements dictionary above, we could print out the atomic number mapped to helium like this:

```
print(elements["helium"])
```

This would print out 2.

We can also insert a new element into a dictionary as in this example:

```
elements["lithium"] = 3
```

If we then executed `print(elements)`, the output would be:

{'hydrogen': 1, 'carbon': 6, 'helium': 2, 'lithium': 3}

This illustrates how dictionaries are mutable.

What if we try to look up a key that is not in our dictionary, using the square brackets, like `elements['dilithium']`? This will give you a "KeyError".

We can check whether a key is in a dictionary the same way we check whether an element is in a list or set, using the `in` keyword. Dictionaries have a related method that's also useful, `get`. `get` looks up values in a dictionary, but unlike square brackets*,* `get` returns None (or a default value of your choice) if the key isn't found.

```
print("carbon" in elements)
print(elements.get("dilithium"))
```

This would output:

```
True
None
```

"carbon" is in the dictionary, so True is printed. "dilithium" isn't in our dictionary so None is returned by `get` and then printed. So if you expect lookups to sometimes fail, `get` might be a better tool than normal square bracket lookups, because errors can crash your program.

Identity Operators

Keyword	**Operator**
`is`	evaluates if both sides have the same identity
`is not`	evaluates if both sides have different identities

You can check if a key returned None with the `is` operator. You can check for the opposite using `is not`.

```
n = elements.get("dilithium")
print(n is None)
print(n is not None)
```

This would output:

```
True
False
```

Previous Next Give Page Feedback