DICTIONARY METHODS

keys	*Returns the keys from a dictionary*	.keys()
values	*Returns the values from a dictionary*	.values()
items	*Returns key value pairs from a dictionary as a list of tuples*	.items()
get	*Returns a value for a given key, or an optional value if the key isn't found*	.get(*key*, *value if key not found*)
update	*Appends specified key-value pairs, including entire dictionaries*	.update (*key:value pairs*)

KEYS

The **.keys()** method returns the keys from a dictionary

```
item_details = {'skis': [249.99, 10, 'in stock'],
                'snowboard': [219.99, 0, 'sold out'],
                'goggles': [99.99, 0, 'sold out'],
                'boots': [79.99, 7, 'in stock']}

item_details.keys()
```

```
dict_keys(['skis', 'snowboard', 'goggles', 'boots'])
```

*.keys() returns a **view object** that represents the keys as a list (this is more memory efficient than creating a list)*

```
for item in item_details.keys():
    print(item)
```

```
skis
snowboard
goggles
boots
```

This view object can be iterated through, which has the same behavior as looping through the dictionary keys directly

```
key_list = list(item_details.keys())

print(key_list)
```

```
['skis', 'snowboard', 'goggles', 'boots']
```

This view object can be converted into a list or a tuple if needed

VALUES

The **.values()** method returns the values from a dictionary

```
item_details = {'skis': [249.99, 10],
                'snowboard': [219.99, 0],
                'goggles': [99.99, 0],
                'boots': [79.99, 7]}

item_details.values()
```

```
dict_values([[249.99, 10], [219.99, 0], [99.99, 0], [79.99, 7]])
```

*.values() returns a **view object** that represents the values as a list*

(this is more memory efficient than creating a list)

```
price_list = []
for attribute in item_details.values():
    price_list.append(attribute[0])

price_list
```

```
[249.99, 219.99, 99.99, 79.99]
```

This view object can be looped through as well

Here we're grabbing the first element from each of the lists returned by .values() and appending them to a new list

ITEMS

The **.items()** method returns key-value pairs from a dictionary as a list of tuples

```
item_details = {'skis': [249.99, 10],
                'snowboard': [219.99, 0],
                'goggles': [99.99, 0],
                'boots': [79.99, 7]}

item_details.items()
```

```
dict_items([('skis', [249.99, 10]), ('snowboard', [219.99, 0]),
('goggles', [99.99, 0]), ('boots', [79.99, 7])])
```

.items() returns a ***view object*** *that represents the key-value pairs as a list of tuples*

```
for key, value in item_details.items():
    print(f'The {key} costs {value[0]}.')
```

```
The skis costs 249.99.
The snowboard costs 219.99.
The goggles costs 99.99.
The boots costs 79.99.
```

You can ***unpack*** *the tuple to retrieve individual keys and values*

In this case, the variable 'key' is assigned to the key in the tuple, and 'value' is assigned to the dictionary value

ITEMS

The **.items()** method returns key-value pairs from a dictionary as a list of tuples

```
item_details = {'skis': [249.99, 10],
                'snowboard': [219.99, 0],
                'goggles': [99.99, 0],
                'boots': [79.99, 7]}

item_details.items()
```

```
dict_items([('skis', [249.99, 10]), ('snowboard', [219.99, 0]),
('goggles', [99.99, 0]), ('boots', [79.99, 7])])
```

.items() returns a ***view object*** *that represents the key-value pairs as a list of tuples*

```
for item, item_attributes in item_details.items():
    print(f'The {item} costs {item_attributes[0]}.')
```

```
The skis costs 249.99.
The snowboard costs 219.99.
The goggles costs 99.99.
The boots costs 79.99.
```

You can give these variables intuitive names, although k, v is common to represent keys and values

GET

The **.get()** method returns the values associated with a dictionary key

- It won't return a KeyError if the key isn't found
- You can specify an optional value to return if the key is not found

```
item_details = {'skis': [249.99, 10, 'in stock'],
                'snowboard': [219.99, 0, 'sold out'],
                'goggles': [99.99, 0, 'sold out'],
                'boots': [79.99, 7, 'in stock']}

item_details.get('boots')
```

```
[79.99, 7, 'in stock']
```

.get() returns the value associated with the 'boots' key

```
item_details['bindings']
```

```
KeyError: 'bindings'
```

```
item_details.get('bindings')
```

```
item_details.get('bindings', "Sorry we don't carry that item.")
```

```
"Sorry we don't carry that item."
```

The difference between using .get() and simply entering the key directly is that .get() will not return an error if they key is not found

And you can specify an optional value to return if the key is not found

- **.get(*key*, *value if key not found*)**

UPDATE

The **.update()** method appends key-value pairs to a dictionary

```
item_details = {'skis': [249.99, 10, 'in stock'],
                'snowboard': [219.99, 0, 'sold out'],
                'goggles': [99.99, 0, 'sold out'],
                'boots': [79.99, 7, 'in stock']}

item_details.update({'bindings': [139.99, 0, 'out of stock']})
item_details
```

```
{'skis': [249.99, 10, 'in stock'],
 'snowboard': [219.99, 0, 'sold out'],
 'goggles': [99.99, 0, 'sold out'],
 'boots': [79.99, 7, 'in stock'],
 'bindings': [139.99, 0, 'out of stock']}
```

.update() appends new key-value pairs to a dictionary, in this case a single pair for a key of 'bindings'

- **.update(*key:value pairs*)**

```
new_items = {'scarf': [19.99, 100, 'in stock'], 'snowpants': 'N/A'}

item_details.update(new_items)
item_details
```

```
{'skis': [249.99, 10, 'in stock'],
 'snowboard': [219.99, 0, 'sold out'],
 'goggles': [99.99, 0, 'sold out'],
 'boots': [79.99, 7, 'in stock'],
 'scarf': [19.99, 100, 'in stock'],
 'snowpants': 'N/A'}
```

This is the preferred way to ***combine dictionaries***

As a reminder, dictionary values do not need to be the same type; note that the value for 'snowpants' is 'N/A', while the values for the rest of the keys are lists