Study Guide: Dictionary Methods

This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

In the Dictionary segment, you learned about the properties of the Python dictionary data type, how dictionaries differ from lists, how to iterate over the contents of a dictionary, and how to use dictionaries with lists and strings.

Knowledge

Python dictionaries are used to organize elements into collections. Dictionaries include one or more keys, with one or more values associated with each key.

Syntax

```
my_dictionary = {keyA:value1,value2, keyB:value3,value4}
```

Operations

- **len(dictionary)** - Returns the number of items in a dictionary.
- **for key in dictionary** - Iterates over each key in a dictionary.
- **for key, value in dictionary.items()** - Iterates over each key,value pair in a dictionary.
- **if key in dictionary** - Checks whether a key is in a dictionary.
- **dictionary[key]** - Accesses a value using the associated key from a dictionary.
- **dictionary[key] = value** - Sets a value associated with a key.
- **del dictionary[key]** - Removes a value using the associated key from a dictionary.

Methods

- **dictionary.get(key, default)** - Returns the value corresponding to a key, or the default value if the specified key is not present.
- **dictionary.keys()** - Returns a sequence containing the keys in a dictionary.
- **dictionary.values()** - Returns a sequence containing the values in a dictionary.

- **dictionary[key].append(value) -** Appends a new value for an existing key.
- **dictionary.update(other_dictionary) -** Updates a dictionary with the items from another dictionary. Existing entries are replaced; new entries are added.
- **dictionary.clear() -** Deletes all items from a dictionary.
- **dictionary.copy() -** Makes a copy of a dictionary.

Dictionaries versus Lists

Dictionaries are similar to lists, but there are a few differences:

Both dictionaries and lists:

- are used to organize elements into collections;
- are used to initialize a new dictionary or list, use empty brackets;
- can iterate through the items or elements in the collection; and
- can use a variety of methods and operations to create and change the collections, like removing and inserting items or elements.

Dictionaries only:

- are unordered sets;
- have keys that can be a variety of data types, including strings, integers, floats, tuples;.
- can access dictionary values by keys;
- use square brackets inside curly brackets { [ ] };
- use colons between the key and the value(s);
- use commas to separate each key group and each value within a key group;
- make it quicker and easier for a Python interpreter to find specific elements, as compared to a list.

Dictionary Example:

```
pet_dictionary = {"dogs": ["Yorkie", "Collie", "Bulldog"], "cats": ["Persian", "Scottish Fol


print(pet_dictionary.get("dogs", 0))
# Should print ['Yorkie', 'Collie', 'Bulldog']

```

Run

Reset

Lists only:

- are ordered sets;
- access list elements by index positions;
- require that these indices be integers;
- use square brackets [ ];
- use commas to separate each list element.

List Example:

```
pet_list  = ["Yorkie", "Collie", "Bulldog", "Persian", "Scottish Fold", "Siberian", "Angora"


print(pet_list[0:3])
# Should print ['Yorkie', 'Collie', 'Bulldog']

```

Run

Reset

Coding skills

Skill Group 1

- Iterate over the key and value pairs of a dictionary using a **for** loop with the **dictionary.items()** method to calculate the sum of the values in a dictionary.

```
# This function returns the total time, with minutes represented as
# decimals (example: 1 hour 30 minutes = 1.5), for all end user time
# spent accessing a server in a given day.


def sum_server_use_time(Server):

    # Initialize the variable as a float data type, which will be used
    # to hold the sum of the total hours and minutes of server usage by
    # end users in a day.
    total_use_time = 0.0

    # Iterate through the "Server" dictionary’s key and value items
    # using a for loop.
    for key,value in Server.items():

        # For each end user key, add the associated time value to the
        # total sum of all end user use time.
        total_use_time += Server[key]

    # Round the return value and limit to 2 decimal places.
    return round(total_use_time, 2)

FileServer = {"EndUser1": 2.25, "EndUser2": 4.5, "EndUser3": 1, "EndUser4": 3.75, "EndUser5"

print(sum_server_use_time(FileServer)) # Should print 20.1
```

Run

Reset

Skill Group 2

- Concatenate a value, a string, and the key for each item in the dictionary and append to the end of a new list[ ] using the **list.append(x)** method.

- Iterate over keys with multiple values from a dictionary using nested **for** loops with the **dictionary.items()** method.

```
# This function receives a dictionary, which contains common employee
# last names as keys, and a list of employee first names as values.
# The function generates a new list that contains each employees’ full
# name (First_name Last_Name). For example, the key "Garcia" with the
# values ["Maria", "Hugo", "Lucia"] should be converted to a list
# that contains ["Maria Garcia", "Hugo Garcia", "Lucia Garcia"].


def list_full_names(employee_dictionary):
    # Initialize the "full_names" variable as a list data type using
    # empty [] square brackets.
    full_names = []

    # The outer for loop iterates through each "last_name" key and
    # associated "first_name" values, in the "employee_dictionary" items.
    for last_name, first_names in employee_dictionary.items():

        # The inner for loop iterates over each "first_name" value in
        # the list of "first_names" for one "last_name" key at a time.
        for first_name in first_names:

            # Append the new "full_names" list with the "first_name" value
            # concatenated with a space " ", and the key "last_name".
            full_names.append(first_name+" "+last_name)

    # Return the new "full_names" list once the outer for loop has
    # completed all iterations.
    return(full_names)


print(list_full_names({"Ali": ["Muhammad", "Amir", "Malik"], "Devi": ["Ram", "Amaira"], "Che
# Should print ['Muhammad Ali', 'Amir Ali', 'Malik Ali', 'Ram Devi', 'Amaira Devi'[illegible] Feng Ch

```

Run

Reset

Skill Group 3

- Use the **dictionary[key] = value** operation to associate a value with a key in a dictionary.
- Iterate over keys with multiple values from a dictionary, using nested **for** loops and an **if**-statement, and the **dictionary.items()** method.

- Use the **dictionary[key].append(value)** method to add the key, a string, and the key for each item in the dictionary.

```
# This function receives a dictionary, which contains resource
# categories (keys) with a list of available resources (values) for a
# company’s IT Department. The resources belong to multiple categories.
# The function should reverse the keys and values to show which
# categories (values) each resource (key) belongs to.


def invert_resource_dict(resource_dictionary):
  # Initialize a "new_dictionary" variable as a dict data type using
  # empty {} curly brackets.
    new_dictionary = {}
    # The outer for loop iterates through each "resource_group" and
    # associated "resources" in the "resource_dictionary" items.
    for resource_group, resources in resource_dictionary.items():
        # The inner for loop iterates over each "resource" value in
        # the list of "resources" for one "resource_group" key at a time.
        for resource in resources:
            # The if-statement checks if the current "resource" value has
            # been appended as a key to the "new_dictionary" yet.
            if resource in new_dictionary:
                # If True, then append the "resource_group" as a value to the
                # "resource", which is now the key.
                new_dictionary[resource].append(resource_group)
            # If False (else), then add the "resource" as a new key with the
            # "resource_group" as a value for that key.
            else:
                new_dictionary[resource] = [resource_group]
    # Return the new dictionary once the outer for loop has completed
    # all iterations.
    return(new_dictionary)


print(invert_resource_dict({"Hard Drives": ["IDE HDDs", "SCSI HDDs"],
        "PC Parts":  ["IDE HDDs", "SCSI HDDs", "High-end video cards", "Basic video cards"],
# Should print {'IDE HDDs': ['Hard Drives', 'PC Parts'], 'SCSI HDDs': ['Hard Drives', 'PC Pa

```

Run

Reset

Resources

For additional information about dictionaries, please visit:

- Mapping Types — dict - Official python.org documentation for dictionary methods
- Python Dictionaries - Tutorial with interactive code blocks for practicing using dictionary methods and operations