Sets

Site: mthree Academy
Course: C394 Foundations of SRE - Pre-course
Book: Sets

Printed by: Valerija Nikitenko
Date: Sunday, 7 July 2024, 6:43 P

Table of contents

Concept 1: Sets

The steps to create a set are similar to those used to create other data collections, such as lists and tuples. The main difference is th use of curly brackets `{ }` to define the collection as a set.

A set is an unordered and unindexed collection of unique items. Specifically:

- The values stored in a set are not indexed in any way. To retrieve an item from a set, you use the value itself, rather than a index.
- Each item in a set must be unique. You cannot include multiple items with the same value in the same set.
- The contents of a set are not ordered. You can add the items in any order you wish, and Python will typically retrieve them a different, random order.

Objectives

By the end of this lesson, you will be able to:

- Store, retrieve, and manipulate data in a set.

Example 1

In this example, we create a set of names and then retrieve the values from the set.

The order of the items in a set is random, so the names may appear in a different order than the order in which they were added.

```
names = {"Robert", "Mark", "Nancy"}
print(names) # The order of the items is random in a set, so the names may not appear in the same order.
```

Output:

```
{'Robert', 'Nancy', 'Mark'}
```

Practice 1

Create a set that contains a collection of your favorite fruits.

Concept 2: Retrieve Items from a Set

We can use a `for` loop with `in` to iterate through a set. Again, because the set is not ordered, the items may appear in a different order than how they were added to the set.

Example 2

We retrieve the names individually from the same set we used earlier.

```
names = {"Robert", "Mark", "Nancy"}
for name in names:
    print(name)
```

Output:

```
Robert
Nancy
Mark
```

Practice 2

Using the set of fruits you created in the previous exercise, retrieve the items from the set so that each item appears on a separate line in the output.

Concept 3: Add Items to a Set

We can use the `add()` method to add new items to a set.

Example 3

We create a set with three names and then add three more names to the set.

```
names = {"Robert", "Mark", "Nancy"}
print(names)

names.add("Mary")
names.add("Alice")
names.add("Bob")
print(names)
```

Output:

```
{'Robert', 'Nancy', 'Mark'}
{'Mark', 'Alice', 'Mary', 'Bob', 'Robert', 'Nancy'}
```

Practice 4

Create a set with at least three fruit items in it and print it. Then add at least three more items to the set and print the updated set.

Concept 4: Empty Sets

We can use `set()` to create an empty set. This can be useful if we need to stage an empty set that we intend to add data to at a later time.

Example 4

In this example, we create an empty set and then add three names to it.

```
names_set = set()
print(names_set)

names_set.add("Mary")
names_set.add("Alice")
names_set.add("Bob")
print(names_set)
```

Output:

```
set()
{'Alice', 'Bob', 'Mary'}
```

Practice 4a

Use the `add()` method to convert the names list into a set called `names_set`.

Use the following steps:

- Create a new empty set.
- Add the items from the list to the new set.
- Print the set when it is complete.

```
names = ["Robert", "Mark", "Nick", "Jenny"] # Do not change this
print(names)

# Your code here

```

Practice 4b

Fix the errors in the following code. Do not remove any existing instructions or add new instructions.

```
names = {"Robert", "Mark", "Nick"Jenny"}
print(names)

another_set = set(1)
another_set \= names
print(another_set)
```

Concept 5: Uniqueness

Each item in a set must be unique within that set. Duplicate items will be ignored.

Example 5

We create another set of names, with some of the names duplicated during input. When we print it, we see that only distinct values are stored in the set.

```
names = {"Robert", "Mark", "Nancy", "Robert", "Mark", "Jenny", "Robert"}
print(names)
```

Output:

```
{'Nancy', 'Mark', 'Jenny', 'Robert'}
```

Practice 5

Convert the `names` list into a set called `names_set` and print the set.

```
names = ["Robert", "Mark", "Nancy", "Robert", "Mark", "Jenny", "Robert"] # Do not change this line of code
print(names)

# Your code here
```

Concept 6: Search Items in a Set

We can use the `in` operator to check if an element exists in a set. This operation returns a boolean value: `True` if the value exists in the set and `False` if it does not.

Example 6

We start with a set of state abbreviations.

```
states_abbrev = {"AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA",
                 "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
                 "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
                 "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
                 "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"}

print("DC" in states_abbrev) # This will return True if the item is in the set and False
otherwise
print("PR" in states_abbrev)
print("CA" in states_abbrev)
```

Output:

```
False
False
True
```

Practice 6

Create a program that performs the following steps:

- Prompts the user for a state abbreviation.
- Returns a success message if the user entered a valid state abbreviation.
- Returns an error message if the user entered an invalid state abbreviation.

The final version of the code should support both uppercase and lowercase input. For example, the user should be able to input "FL "Fl," "fL," or "fl" and get the same response.

```
states_abbrev = {"AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA",
                 "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
                 "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
                 "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
                 "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"}

# Your code here
```

Concept 7: Add Multiple Items to a Set

We can use the `update()` method to add multiple items to an existing set.

Example 7

We create two sets of food items: one with four items and the other with two.

We then use `update()` to add the shorter set to the longer set and display the updated set.

```
food = {"pasta", "burger", "hotdog", "pizza"}
print(food)

other_food = {"taco", "burrito"}

food.update(other_food)
print(food)
```

Output:

```
{'hotdog', 'pasta', 'pizza', 'burger'}
{'taco', 'hotdog', 'burger', 'burrito', 'pizza', 'pasta'}
```

Practice 7

Create a script that keeps asking the user for input until the user enters "quit" (uppercase or lowercase).

- Each input value should be added to a set.
- When the user enters "quit", the program should display all distinct values entered by the user, with each word on a separate line.

Concept 8: Calculate the Length of a Set

We can use the `len()` method to compute the number of elements in a set.

Example 8

We create a new set and then use `len()` to display the number of items in the set.

```
food =  {"pasta", "burger", "hotdog", "pizza"}
print(food)
print(len(food))
```

Output:

```
{'hotdog', 'pasta', 'pizza', 'burger'}
4
```

Practice 8

Create a program that performs the following steps:

- Prompts the user for an integer.
- Asks the user to enter that many words.
- Displays the set of words provided by the user.

For example, if the user enters 5, the program will ask the user for 5 different words that it will store in a set.

When the user finishes entering the items of the set, the program displays the set to the user.

Concept 9: Delete Items from a Set

We can use the `remove()` or `discard()` methods to delete elements from a set.

These methods behave differently:

- discard() will **not** raise an error if the item to remove does not exist.
- remove() will raise an error if the item to remove does not exist.

See the example below.

Example 9

We start by creating a set of food items.

We then use `discard()` to delete one item and `remove()` to delete another item.

When we repeat the `discard()` and `remove()` steps, the final `remove()` step throws an error because that item no longer exists in the set.

```
food = {"pasta", "burger", "hotdog", "pizza"}
print(food)

food.discard("pasta")
print(food)

food.remove("burger")
print(food)

# This will NOT throw an error even though pasta doesn't exist in the set anymore.
food.discard("pasta")

# This WILL throw an error because pasta doesn't exist in the set anymore.
food.remove("pasta")
```

Output:

```
{'hotdog', 'pasta', 'pizza', 'burger'}
{'hotdog', 'pizza', 'burger'}
{'hotdog', 'pizza'}


---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-35-753fa67537fd> in <module>
     12
     13 #this WILL throw an error because pasta doesn't exist in the set anymore.
---> 14 food.remove("pasta")


KeyError: 'pasta'
```

Practice 9

Complete the code below, using the `in` operator and the `remove()` method, to create a script that will remove items from a set.

- If the item exists, remove the item and display the current set with the item removed.
- If the item does not exist, display an appropriate, user-friendly message and list the items currently in the set.

The program should not throw an error if the item doesn't exist. This means that the script should check if the item exists in the set and remove the item only if it is included in the current set.

```
food = {"pasta", "burger", "hotdog", "pizza"}

# The remove shouldn't throw an error.
food.remove("taco")
```

Concept 10: Clear a Set

We can use the `clear()` method to empty a set.

This is useful if we want to completely update an existing set with new data.

Example 10

In this example, we create a set of food items, clear the set, and then update the set with new items.

```
food = {"pasta", "burger", "hotdog", "pizza"}
print(food)

food.clear() # This will empty the food set.
print(food)
```

Output:

```
{'hotdog', 'pasta', 'pizza', 'burger'}
set()
```

Practice 10

Update the following code to include a user prompt.

- If the user inputs a string that corresponds to the variable name of one of the sets below, clear the set using the `clear()` method.
- If there is no set that matches the user input, display an error message that the set doesn't exist.

```
shake_1 = {"banana", "blueberry", "spinach"}
shake_2 = {"strawberry", "pistachio", "cocoa powder"}
shake_3 = {"kiwi", "banana", "peanut butter"}

# Your code here
```

Concept 11: Pop Items in a Set

We can use the `pop()` method to return and remove the last element in a set.

Because sets are unordered, the item returned by the `pop` method is random.

The `pop()` method does not take an argument, which means that you cannot use it to remove a specific item from the set.

Example 11

We create a set of four food items and use `pop()` to identify and remove one of those items.

Note that while "pizza" is the last item added to the new set, it may not correspond to the popped item, because sets are unordered.

```
food = {"pasta", "burger", "hotdog", "pizza"}
item = food.pop() # Item is not necessarily "pizza" because sets are unordered.
print(item)
print(food)
```

Output:

```
hotdog
{'pasta', 'pizza', 'burger'}
```

Practice 11

Create a script that performs the following steps:

- Displays the set to the user along with the number of items in the set.
- If there are no items in the set, displays an output message to that effect and ends the script.
- Asks the user if they want to remove an item from the set.
- If the user says yes:
    - Verifies that there is at least one item in the set.
    - Removes a random item from the set.
    - Displays the updated set to the user.
    - Prompts the user to remove another item.
- If the user says no, ends the script.

You cannot use the `clear()`, `remove()`, or `discard()` methods in your solution.

```
food = {"pasta", "burger", "hotdog", "pizza"}

# Your code here
```

Concept 12: Delete a Set

We can use the `del` keyword to completely delete a set and its contents.

Example 12

Here, we create a set of food items, print it out, and then delete it.

Printing a non-existent set will throw an error.

```
food = {"pasta", "burger", "hotdog", "pizza"}
print(food)

del food    # Delete the set
print(food) # This will throw an error because the set a doesn't exist anymore
```

Output:

```
{'hotdog', 'pasta', 'pizza', 'burger'}



---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-38-36e6a08655fd> in <module>
      3
      4 del food #delete the set
----> 5 print(food) #this will throw an error because the set a doesn't exist anymore


NameError: name 'food' is not defined
```

Practice 12

Complete the code below to ask the user for input.

- If the user inputs a string that corresponds to the variable name of one of the sets below, delete that set using `del`.
- If there is no set with that name, display an error message that the set doesn't exist.

```
shake_1 = {"banana", "blueberry", "spinach"}
shake_2 = {"strawberry", "pistachio", "cocoa powder"}
shake_3 = {"kiwi", "banana", "peanut butter"}

# Your code here
```

Concept 13: Determine the Difference Between Sets

We can use the `difference()` method to compare two sets and return a set containing the items that appear in the first set but do not appear in the second set.

Example 13

In the following example, notice that both sets include one item that is not in the other set. The `difference()` method only looks fo items in the first set that do not appear in the second set, and it ignores other items in the second set.

```
shake_1 = {"kiwi", "banana", "peanut butter"}
shake_2 = {"banana", "kiwi", "spinach"}
shake_3 = shake_1.difference(shake_2) # This set will contain the difference of shake_1 and
shake_2

print(shake_1)
print(shake_2)
print(shake_3)
```

Output:

```
{'kiwi', 'banana', 'peanut butter'}
{'spinach', 'banana, 'kiwi'}
{'peanut butter'}
```

Practice 13

Create a script that performs the following steps:

- Prompts the user for the abbreviation of a state they have visited and stores the value in a new set.
- Repeats the prompt and continues storing values in the same set until the user enters "done" (in uppercase or lowercase).
- After the user enters "done", displays a list of abbreviations for the states that the user has not visited yet, using the `states_abbrev` set below and the `difference()` method.

Tip: It's always a good idea to tell the user how to get out of a loop!

```
states_abbrev = {"AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA",
                 "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
                 "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
                 "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
                 "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"}

# Your code here
```

Concept 14: Intersect Sets

We can use the `intersection()` method to compute the intersection of two or more sets.

The result includes only the values that both sets have in common.

Example 14

In the following example, notice that there is at least one item that appears in both sets.

We use `intersection()` to identify the shared items.

```
shake_1 = {"kiwi", "banana", "peanut butter"}
shake_2 = {"banana", "kiwi", "spinach"}
shake_3 = shake_1.intersection(shake_2) # This set will contain the intersection of shake_1
and shake_2

print(shake_1)
print(shake_2)
print(shake_3)
```

Output:

```
{'kiwi', 'banana', 'peanut butter'}
{'banana', 'kiwi', 'spinach'}
{'banana', 'kiwi'}
```

Practice 14

The following code includes a set of state abbreviations representing states that Mary has visited. Update the code to include the following steps:

- Prompts the user for the abbreviation of a state they have visited and stores the value in a new set.
- Repeats the prompt and continues storing values in the same set until the user enters "done" (in uppercase or lowercase).
- Displays a list of the states that both the user and Mary have visited, or displays a meaningful message if the user has not visited any of the states that Mary has visited.

```
mary_states = {"AZ", "CA", "FL", "GA", "IN",
               "KY", "MA", "NV", "NY", "NC",
               "PA", "SC", "TN"}

# Your code here
```

Concept 15: Combine Sets

We can use the `union()` method to compute the union of two or more sets.

The result is a new set of items that exist in at least one of the sets.

Example 15

In the following example, we create three sets of food items and then combine all three sets into a new set.

Notice that "kiwi" and "banana" appear in two sets, but each of those values appears only once in the results.

```
shake_1 = {"kiwi", "banana", "peanut butter"}
shake_2 = {"banana", "kiwi", "spinach"}
shake_3 = {"orange", "apple", "almonds"}

# The union method combines two or more sets. We can add as many sets as needed.
shake_4 = shake_1.union(shake_2, shake_3)

print(shake_4)
```

Output:

```
{'banana', 'spinach', 'peanut butter', 'almonds', 'kiwi', 'orange', 'apple'}
```

Practice 15

The following code includes a set of the states that Mary has visited. Update the code to perform the following steps:

- Prompts the user for the abbreviation of a state they have visited and stores the value in a new set.
- Repeats the prompt and continues storing values in the same set until the user enters "done" (in uppercase or lowercase).
- Displays a list of the states that either the user or Mary has visited.

```
mary_states = {"AZ", "CA", "FL", "GA", "IN",
               "KY", "MA", "NV", "NY", "NC",
               "PA", "SC", "TN"}

# Your code here
```