Programming Languages

CS 315

Homework 1

Kutay Tire

22001787

Section 2

1. Dart

Initialization

```
var dictionaryOfDart = {
   'Assyria' : 'Nineveh',
   'Babylonia' : 'Babylon',
   'Sumeria' : 'Ur',
   'Persia' : 'Susa'
 };
```

Output

Since it is just the initialization process, there is no output yet.

Explanation

This code segment initializes a map in Dart that consists of 4 elements.

Getting a Value

```
print("Getting the value of 'Sumeria':");
var getValue = dictionaryOfDart['Sumeria'];
print("Value of 'Assyria' is: '$getValue'");
```

Output

Getting the value of ‘Sumeria’:

Value of ‘Sumeria’ is: ‘Ur’

Explanation

This code segment gets the associated value of ‘Sumeria’ which is the key and prints the value.

Adding a new Element

```
print('\n');
print("Adding a new element 'Egypt':");
dictionaryOfDart.addAll({'Egypt' : 'Thebes'});
print(dictionaryOfDart);
print('\n');
```

Output

Adding a new element 'Egypt':

{Assyria: Nineveh, Babylonia: Babylon, Sumeria: Ur, Persia: Susa, Egypt: Thebes}

Explanation

This code segment adds a new kay-value pair which is {‘Egypt’: ‘Thebes’}. Then, the new associative array is printed.

Removing an Element

```
print("Removing the element 'Persia':");
var removedElement =
dictionaryOfDart.remove('Persia');
 if (removedElement != null) {
    print("’Persia’ is successfully removed.”);}
else {
   print("’Persia’ does not exist.”);}
print(dictionaryOfDart);
print('\n');
```

Output

Removing the element 'Persia':

‘Persia’ is successfully removed.

{Assyria: Nineveh, Babylonia: Babylon, Sumeria: Ur, Egypt: Thebes}

Explanation

This code segment first removes the value assocaited with the given key. If the removed value is something other than null, it prints a success message.

Modifying an Element

```
print("Modifying the element 'Assyria': ");

dictionaryOfDart['Assyria'] = 'Assur';

print(dictionaryOfDart);

print('\n');
```

Output

```
Modifying the element 'Assyria':

{Assyria: Assur, Babylonia: Babylon, Sumeria: Ur, Egypt: Thebes}
```

Explanation

This code segment modifies the assocaited value for the key ‘Assyria’ from ‘Nineveh’ to ‘Assur’ and prints the new associative array.

Searching for a Key

```
print("Searching for the 'Babylonia' key:");

if (dictionaryOfDart.containsKey('Babylonia')){

    var matchingValue = dictionaryOfDart['Babylonia'];

    print ("Key is successfully found with the value '$matchingValue'.\n"); }

else{

   print('Key was not found\n');}

print("Searching for the 'Rome' key:");

if(dictionaryOfDart.containsKey('Rome')){

   var matchingValue = dictionaryOfDart['Rome'];

   print("Key is successfully found with the value '$matchingValue'.\n"); }

else{

   print('Key was not found.\n'); }
```

Output

```
Searching for the 'Babylonia' key:

Key is successfully found with the value 'Babylon'.

Searching for the 'Rome' key:

Key was not found.
```

Explanation

This code segment first searches for the key ‘Babylonia’ and since it is present, prints a success message. Then, it searches for the key ‘Rome’ and since it is not present, prints an error message. The search is done with **containsKey()** method of Dart.

Searching for a Value

```
print("Searching for the 'Thebes' value:");

if (dictionaryOfDart.containsValue('Thebes')){

   print("'Thebes' is successfully found.\n");}

else{

    print("'Thebes' was not found as a value.\n");

print("Searching for the 'Athens' value:");

if(dictionaryOfDart.containsValue('Athens')){

   print("'Athens' is successfully found.\n"); }

else{

   print("'Athens' was not found as a value.\n");}
```

Output

```
Searching for the 'Thebes' value:

'Thebes' is successfully found.


Searching for the 'Athens' value:

'Athens' was not found as a value.
```

Explanation

This code segment searches for the values ‘Thebes’ and ‘Athens’ in the assocaiteve array and prints the corresponding message based on the outcome. The search is done via **containsValue()** method of Dart.

Looping through Array

```
print("Looping through the array: ");

void foo(key, value) {

  print('$key: $value');}

dictionaryOfDart.forEach((key, value) {

  foo(key, value);});
```

Output

```
Looping through the array:

Assyria: Assur

Babylonia: Babylon

Sumeria: Ur

Egypt: Thebes
```

Explanation

This code segment defines a function foo() that prints key-value pairs. Then, it calls the function foo() in a loop to print each pair. Looping is done with **forEach()** method.

2. JavaScript

Initialization

```
var javascriptDictionary = {

   'Assyria' : 'Nineveh',

   'Babylonia' : 'Babylon',

   'Sumeria' : 'Ur',

   'Persia' : 'Susa'};
```

Output

Since it is just the initialization process, there is no output yet.

Explanation

This code segment initializes JavaScript associative array.

Getting a Value

```
document.write("<br\>");

document.write("Getting the value of
'Sumeria': <br\>");

var value1 = javascriptDictionary['Sumeria'];

document.write("The value of Sumeria is: ",
value1, "<br\>");

document.write("<br\>");
```

Output

```
Getting the value of 'Sumeria':
The value of Sumeria is: Ur
```

Explanation

This code segment gets the associated value of 'Sumeria' which is the key in JavaScript.

Adding a new Element

```
document.write("Adding a new element
'Egypt': <br\>");

javascriptDictionary['Egypt'] = "Thebes";

for(var key in javascriptDictionary) {

   foo(key, javascriptDictionary[key]);}

document.write("<br\>");
```

Output

```
Adding a new element 'Egypt':
Assyria: Nineveh
Babylonia: Babylon
Sumeria: Ur
Persia: Susa
Egypt: Thebes
```

Explanation

This code segment adds a new kay-value pair which is {'Egypt': 'Thebes'} and then prints the new array with foo() function defined in "Looping through Array" section.

Removing an Element

```
document.write("Removing the element
'Persia': <br\>");

delete javascriptDictionary['Persia'];

document.write("'Persia' is successfully
removed. <br\>");

for(var key in javascriptDictionary) {

   foo(key, javascriptDictionary[key]);}

document.write("<br\>")
```

Output

```
Removing the element 'Persia':
'Persia' is successfully removed.
```

```
Assyria: Nineveh
Babylonia: Babylon
Sumeria: Ur
Egypt: Thebes
```

Explanation

This code segment first removes the element ‘Persia’ from the array in JavaScript with the delete key word. Then, prints the new array.

Modifying an Element

```
document.write("Modifying the element 'Assyria': <br\>");

javascriptDictionary['Assyria'] = 'Assur';

for(var key in javascriptDictionary) {

   foo(key, javascriptDictionary[key]);}

document.write("<br\>");
```

Output

```
Modifying the element 'Assyria':
Assyria: Assur
Babylonia: Babylon
Sumeria: Ur
Egypt: Thebes
```

Explanation

This code segment modifies the assocaited value for the key ‘Assyria’ from ‘Nineveh’ to ‘Assur’ and prints the new associative array.

Searching for a Key

```
document.write("Searching for the 'Babylonia' key: <br\>");

if
(javascriptDictionary.hasOwnProperty("Babylonia")) {

          var matchingValue =
          javascriptDictionary['Babylonia'];

          document.write("The key is
          successfully found with the value " ,
          matchingValue, "<br\>"); }

else {

   document.write("Key was not found.
   <br\>");}

document.write("Searching for the 'Rome' key: <br\>");

if
(javascriptDictionary.hasOwnProperty("Rome")) {

          var matchingValue =
          javascriptDictionary['Rome'];

          document.write("The key is
          successfully found with the value " ,
          matchingValue, "<br\>");}

else {

   document.write("Key was not found.
   <br\>");}
```

Output

```
Searching for the 'Babylonia' key:
The key is successfully found with the
value Babylon
Searching for the 'Rome' key:
Key was not found
```

Explanation

This code segment first searches for the key ‘Babylonia’ and since it is present, prints a success message with its associated value. Then, it searches for the key ‘Rome’ and since it is not present, prints an error message.

Searching for a Value

```
document.write("<br\>");

document.write("Searching for the 'Thebes' value: <br\>");

foundThebes = false;

for (var key in javascriptDictionary) {

   if (javascriptDictionary[key] == 'Thebes') {

         foundThebes = true;

         break;}}

if (foundThebes) {

  document.write("'Thebes' is successfully found as a value. <br\>");}

else {

  document.write("'Thebes' was not found as a value. <br\>");}

document.write("Searching for the 'Athens' value: <br\>");

foundAthens = false;

for (var key in javascriptDictionary) {

  if (javascriptDictionary[key] == 'Athens') {

         foundAthens = true;

         break;}}

if (foundAthens) {

  document.write("'Athens' is successfully found as a value. <br\>");}

else {

  document.write("'Athens' was not found as a value. <br\>");}
```

Output

```
Searching for the 'Thebes' value:
'Thebes' is successfully found as a value.
Searching for the 'Athens' value:
'Athens' was not found as a value
```

Explanation

This code segment searches for the values 'Thebes' and 'Athens' in the assocaiteve array and prints the corresponding message based on the outcome in JavaScript.

Looping through Array

```
document.write("<br\> Looping through the array: <br\>");

function foo(key, value) {

         document.write(key, ": ", value, "<br\>");}

for(var key in javascriptDictionary) {

         foo(key, javascriptDictionary[key]);}
```

Output

```
Looping through the array:
Assyria: Assur
Babylonia: Babylon
Sumeria: Ur
Egypt: Thebes
```

Explanation

This code segment defines a function foo() that prints key-value pairs. Then, it calls the function foo() in a loop to print each pair.

3. Lua

Initialization

```
luaDictionary = {}

luaDictionary["Assyria"] = "Nineveh"

luaDictionary["Babylonia"] = "Babylon"

luaDictionary["Sumeria"] = "Ur"

luaDictionary["Persia"] = "Susa
```

Output

Since it is just the initialization process, there is no output yet.

Explanation

This code segment initializes associative array in Lua.

Getting a Value

```
print("\nGetting the value of 'Sumeria'")

local value1 = luaDictionary["Sumeria"]

print("The value of Sumeria is:", value1)
```

Output

```
Getting the value of 'Sumeria'

The value of Sumeria is:        Ur
```

Explanation

This code segment gets the associated value of 'Sumeria' which is the key in Lua. It is assigned to a local variable called value1 and then is printed.

Adding a new Element

```
print("\nAdding the new element 'Egypt': ")

luaDictionary['Egypt'] = "Thebes"

for key, value in pairs(luaDictionary) do

    foo(key, value)

end
```

Output

```
Adding the new element 'Egypt':

Egypt:          Thebes

Sumeria:        Ur

Assyria:        Nineveh

Babylonia:      Babylon

Persia:         Susa
```

Explanation

This code segment adds a new kay-value pair which is {'Egypt': 'Thebes'} and then prints the new array with foo() function in "Looping through Array" section.

Removing an Element

```
print("\nRemoving the element 'Persia': ")

luaDictionary['Persia'] = nil

for key, value in pairs(luaDictionary) do

    foo(key, value)

end
```

Output

```
Removing the element 'Persia':

Egypt:          Thebes

Babylonia:      Babylon

Sumeria:        Ur

Assyria:        Nineveh
```

Explanation

This code segment first removes the value ‘Persia’ from the array in Lua by assigning it to a **nil** value. Then, it prints the array to see the changes.

Modifying an Element

```
print("\nModifying the value of 'Assyria': ")
luaDictionary['Assyria'] = "Assur"
for key, value in pairs(luaDictionary) do
    foo(key, value)
end
```

Output

```
Modifying the value of 'Assyria':
Babylonia:    Babylon
Egypt:        Thebes
Sumeria:      Ur
Assyria:      Assur
```

Explanation

This code segment modifies the assocaited value for the key ‘Assyria’ from ‘Nineveh’ to ‘Assur’ and prints the new associative array.

Searching for a Key

```
print("\nSearching for the 'Babylonia' key:")
local foundKey = false
for key, value in pairs (luaDictionary) do
   if key == 'Babylonia' then
     foundKey = true
     break
   end
end
if foundKey then
  print("The key is successfully found.\n")
else
  print("Key was not found.\n")
end
print("\nSearching for the 'Rome' key:")
local foundKey2 = false
for key, value in pairs (luaDictionary) do
   if key == 'Rome' then
     foundKey2 = true
     break
   end
end
if foundKey2 then
  print("The key is successfully found.\n")
else
  print("Key was not found.\n")
end
```

Output

```
Searching for the 'Babylonia' key:
The key is successfully found.

Searching for the 'Rome' key:
Key was not found.
```

Explanation

This code segment first searches for the keys ‘Babylonia’ and ‘Rome’. First, it creates a boolean value to indicate whether the key is found or not. After iterating through the associative array, if the key is present, foundKey becomes true and the loop is exited. Based on the values of foundKey and foundKey2, necessary output messages are displayed.

Searching for a Value

```
print("\nSearching for the 'Thebes' value:")

local foundValue = false

for key, value in pairs (luaDictionary) do

     if value == 'Thebes' then

       foundValue = true

       break

    end

end

if foundValue then

    print("The value is successfully found.\n")

else

    print("Value was not found.\n")

end

print("\nSearching for the 'Athens' value:")

local foundValue2 = false

for key, value in pairs (luaDictionary) do

      if value == 'Athens' then

        foundValue2 = true

        break

     end

end

if foundValue2 then

    print("The value is successfully found.\n")

else

    print("Value was not found.\n")

end
```

Output

```
Searching for the 'Thebes' value:

The value is successfully found.

Searching for the 'Athens' value:

Value was not found.
```

Explanation

This code segment first searches for the values ‘Thebes’ and ‘Athens’. First, it creates a boolean value to indicate whether the value is found or not. After iterating through the associative array, if the value is present, foundValue becomes true and the loop is exited. Based on the values of foundValue and foundValue2, necessary output messages are displayed.

Looping through Array

```
function foo(key, value)

   print(key, ":", value)

end

print("Looping through the array: ")

for key, value in pairs(luaDictionary) do

        foo(key ,value)

end
```

Output

```
Looping through the array:

Sumeria:      Ur

Egypt:        Thebes

Assyria:      Assur

Babylonia:    Babylon
```

Explanation

This code segment defines a function foo() that prints key-value pairs. Then, it calls the function foo() in a loop to print each pair.

4. PHP

Initialization

```
$phpDictionary = array("Assyria" =>
"Nineveh", "Babylonia" => "Babylon",
"Sumeria" => "Ur", "Persia" => "Susa");
```

Output

Since it is just the initialization process, there is no output yet.

Explanation

This code segment initializes associative array in PHP.

Getting a Value

```
echo "Getting the value of Sumeria: <br>";

$value1 = $phpDictionary["Sumeria"];

echo "The value of 'Sumeria' is " . $value1.
"<br>";
```

Output

```
Getting the value of Sumeria:
The value of 'Sumeria' is Ur
```

Explanation

This code segment gets the associated value of 'Sumeria' which is the key in PHP. It is assigned to a local variable called value1 and then printed with the help of **echo**.

Adding a new Element

```
echo "<br>Adding the new element 'Egypt':
<br>";

$phpDictionary['Egypt'] = "Thebes";

foreach ($phpDictionary as $key => $value) {

        foo($key, $value);}
```

Output

```
Adding the new element 'Egypt':
Assyria: Nineveh
Babylonia: Babylon
Sumeria: Ur
Persia: Susa
Egypt: Thebes
```

Explanation

This code segment adds a new kay-value pair which is {'Egypt' => 'Thebes'} and then prints the new array with foo() function in "Looping through Array" section.

Removing an Element

```
echo "<br> Removing the element 'Persia':
<br>";

unset ($phpDictionary['Persia']);

foreach ($phpDictionary as $key => $value) {

        foo($key, $value);}
```

Output

```
Removing the element 'Persia':
Assyria: Nineveh
Babylonia: Babylon
Sumeria: Ur
Egypt: Thebes
```

Explanation

This code segment first removes the value 'Persia' from the array in PHP by the help of **unset** method which removes the entered variable. Then, it prints the array to see the changes.

Modifying an Element

```
echo "<br>Modifying the element 'Assyria':
<br>";
```

```
$phpDictionary['Assyria'] = "Assur";

foreach ($phpDictionary as $key => $value) {

        foo($key, $value);}
```

Output

```
Modifying the element 'Assyria':
Assyria: Assur
Babylonia: Babylon
Sumeria: Ur
Egypt: Thebes
```

Explanation

This code segment modifies the assocaited value for the key ‘Assyria’ from ‘Nineveh’ to ‘Assur’ and prints the new associative array with foreach loop and calling the foo() function.

Searching for a Key

```
echo "<br>Searching for 'Babylonia' key:
<br>";

if (array_key_exists('Babylonia',
$phpDictionary))

  echo "'Babylonia' key is successfully
  found.<br>";

else

  echo "The key was not found.<br>";

echo "<br>Searching for 'Rome' key: <br>";

if (array_key_exists('Rome', $phpDictionary))

  echo "'Rome' key is successfully
  found.<br>";

else

  echo "The key was not found.<br>";
```

Output

```
Searching for 'Babylonia' key:
'Babylonia' key is successfully found.

Searching for 'Rome' key:
The key was not found.
```

Explanation

This code segment first searches for the keys ‘Babylonia’ and ‘Rome’. It is achieved easily as PHP has a built-in method **array_key_exists** that returns true if the key is present in the list.

Searching for a Value

```
echo "<br>Searching for 'Thebes' value: <br>";

if (array_search('Thebes', $phpDictionary))

  echo "'Thebes' value is successfully
  found.<br>";

else

  echo "The value was not found.<br>";

echo "<br>Searching for 'Athens' value: <br>";

if (array_search('Athens', $phpDictionary))

  echo "'Athens' value is successfully
found.<br>";

else

  echo "The value was not found.<br>";
```

Output

```
Searching for 'Thebes' value:
'Thebes' value is successfully found.

Searching for 'Athens' value:
The value was not found.
```

Explanation

This code segment first searches for the values ‘Thebes’ and ‘Athens’. Similarly, as PHP has a built in method called **array_search**, the existence of a value is determined easily.

Looping through Array

```
function foo($key, $value){

    echo $key . ": " . $value. "<br>";}

echo "<br> Looping through the array: <br>";

foreach ($phpDictionary as $key => $value) {

        foo($key, $value);}
```

Output

```
Looping through the array:
Assyria: Assur
Babylonia: Babylon
Sumeria: Ur
Egypt: Thebes
```

Explanation

This code segment defines a function foo() that prints key-value pairs. Then, it calls the function foo() in a foreach loop to print each pair.

5. Python

Initialization

```
pythonDictionary = {
   'Assyria' : 'Nineveh',
   'Babylonia' : 'Babylon',
   'Sumeria' : 'Ur',
   'Persia' : 'Susa'}
```

Output

Since it is just the initialization process, there is no output yet.

Explanation

This code segment initializes associative array in Python.

Getting a Value

```
print("Getting the value of 'Sumeria'")
val1 = pythonDictionary['Sumeria']
print("The value of 'Sumeria' is: ", val1)
print("\nGetting the value of 'Persia'")
val2 = pythonDictionary.get('Persia')
print("The value of 'Persia' is: ", val2)
```

Output

```
Getting the value of 'Sumeria'
The value of 'Sumeria' is:  Ur

Getting the value of 'Persia'
The value of 'Persia' is:  Susa
```

Explanation

This code segment gets the associated value of ‘Sumeria’ and ‘Persia’ by two different methods. In the first one, value is directly accessed with the proper key and in the second one, a **get** method is used.

Adding a new Element

```
print("\nAdding a new element 'Egypt': ")
pythonDictionary['Egypt'] = "Thebes"
print(pythonDictionary)
```

Output

```
Adding a new element 'Egypt':
{'Assyria': 'Nineveh', 'Babylonia': 'Babylon', 'Sumeria': 'Ur', 'Persia': 'Susa', 'Egypt': 'Thebes'}
```

Explanation

This code segment adds a new kay-value pair which is {‘Egypt’: ‘Thebes’} and then prints the new array as Python supports printing of dictionaries.

Removing an Element

```
print("\nRemoving the element 'Persia': ")
removedElement = pythonDictionary.pop('Persia')
print(pythonDictionary)
```

Output

```
Removing the element 'Persia':
{'Assyria': 'Nineveh', 'Babylonia': 'Babylon', 'Sumeria': 'Ur', 'Egypt': 'Thebes'}
```

Explanation

This code segment first removes the value ‘Persia’ from the dictionary in Python by the help of **pop** method. Then, it prints the array to see the changes.

Modifying an Element

```
print("\nModifying the element 'Assyria': ")

pythonDictionary['Assyria'] = "Assur"

print(pythonDictionary)
```

Output

```
Modifying the element 'Assyria':

{'Assyria': 'Assur', 'Babylonia': 'Babylon', 'Sumeria': 'Ur', 'Egypt': 'Thebes'}
```

Explanation

This code segment modifies the assocaited value for the key ‘Assyria’ from ‘Nineveh’ to ‘Assur’ and prints the new associative array.

Searching for a Key

```
print("\nSearching for 'Babylonia' key: ")

if "Babylonia" in pythonDictionary:

   print("'Babylonia' key is found with the value: ", pythonDictionary['Babylonia'])

else:

   print("'Babylonia' key was not found.")

print("\nSearching for 'Rome' key: ")

if "Rome" in pythonDictionary:

   print("'Rome' key is found with the value: ", pythonDictionary['Rome'])

else:

   print("'Rome' key was not found.")
```

Output

```
Searching for 'Babylonia' key:

'Babylonia' key is found with the value: Babylon

Searching for 'Rome' key:

'Rome' key was not found.
```

Explanation

This code segment first searches for the keys ‘Babylonia’ and ‘Rome’. It is done by using **in** keyword in Python. If the key is found, its associative value is also printed.

Searching for a Value

```
print("\nSearching for 'Thebes' value: ")

if "Thebes" in pythonDictionary.values():

   print("'Thebes' value is found.")

else:

   print("'Thebes' was not found as a value.")

print("\nSearching for 'Athens' value: ")

if "Athens" in pythonDictionary.values():

   print("'Athens' value is found.")

else:

   print("'Athens' was not found as a value.")
```

Output

```
Searching for 'Thebes' value:

'Thebes' value is found.


Searching for 'Athens' value:

'Athens' was not found as a value.
```

Explanation

This code segment first searches for the values ‘Thebes’ and ‘Athens’. Again, this is achieved by **in** keyword and **values()** method which return the list of values in a dictionary. From this list, the existence of a specific key is checked.

Looping through Array

```
print("\nLooping through the array: ")

def foo(key, value):

   print(key, ": ", value)

for key in pythonDictionary:

   foo(key, pythonDictionary[key])
```

Output

```
Looping through the array:

Assyria :  Assur

Babylonia :  Babylon

Sumeria :  Ur

Egypt :  Thebes
```

Explanation

This code segment defines a function foo() that prints key-value pairs. Then, it calls the function foo() in a for loop to print each pair.

6. Ruby

Initialization

```
rubyDictionary = {"Assyria" => "Nineveh", "Babylonia" => "Babylon", "Sumeria" => "Ur", "Persia" => "Susa"}
```

Output

Since it is just the initialization process, there is no output yet.

Explanation

This code segment initializes associative array in Ruby.

Getting a Value

```
puts "Getting the value of 'Sumeria'"

val1 = rubyDictionary['Sumeria']

print "The value of 'Sumeria' is: ", val1

puts nil, nil
```

Output

```
Getting the value of 'Sumeria'

The value of 'Sumeria' is: Ur
```

Explanation

This code segment gets the associated value of ‘Sumeria’ and prints it. **Print** statement is used instead of **puts** for printing the value to make the second line more intact.

Adding a new Element

```
puts "Adding a new element 'Egypt': "

rubyDictionary['Egypt'] = 'Thebes'

puts rubyDictionary
```

Output

```
Adding a new element 'Egypt':

{"Assyria"=>"Nineveh", "Babylonia"=>"Babylon", "Sumeria"=>"Ur", "Persia"=>"Susa", "Egypt"=>"Thebes"}
```

Explanation

This code segment adds a new kay-value pair which is {‘Egypt’=> ‘Thebes’} and then prints the new array as Ruby supports printing of associative arrays.

Removing an Element

```
puts nil, "Removing the element 'Persia': "

removedElement = rubyDictionary.delete('Persia')

if (removedElement)

   puts "'Persia' is successfully removed."

   puts rubyDictionary

else

   puts "'Persia' does not exist"

end
```

Output

```
Removing the element 'Persia':

'Persia' is successfully removed.

{"Assyria"=>"Nineveh", "Babylonia"=>"Babylon", "Sumeria"=>"Ur", "Egypt"=>"Thebes"}
```

Explanation

This code segment first removes the value ‘Persia’ from the array by using the **delete()** method and assigns the deleted value to a variable. If this variable indeed exists, then a success message is displayed.

Modifying an Element

```
puts nil, "Modifying the element 'Assyria': "

rubyDictionary['Assyria'] = "Assur"

print(rubyDictionary)
```

Output

```
Modifying the element 'Assyria':

{"Assyria"=>"Assur", "Babylonia"=>"Babylon", "Sumeria"=>"Ur", "Egypt"=>"Thebes"}
```

Explanation

This code segment modifies the assocaited value for the key ‘Assyria’ from ‘Nineveh’ to ‘Assur’ and prints the new associative array.

Searching for a Key

```
puts nil, "Searching for 'Babylonia' key:"

foundBabylonia = false

rubyDictionary.each {|key, value|

   if key == 'Babylonia'

      foundBabylonia = true

      break

   end}

if (foundBabylonia)

  puts "'Babylonia' key is found."

else

  puts "'Babylonia' key was not found."

end

puts nil, "Searching for 'Rome' key:"

foundRome = false

rubyDictionary.each {|key, value|

   if key == 'Rome'

      foundRome = true

      break

   end}

if (foundRome)

  puts "'Rome' key is found."

else

  puts "'Rome' key was not found."

end
```

Output

```
Searching for 'Babylonia' key:

'Babylonia' key is found.


Searching for 'Rome' key:

'Rome' key was not found.
```

Explanation

This code segment first searches for the keys ‘Babylonia’ and ‘Rome’. Based on the result of this outcome, necessary message is displayed. The checking is done by comparing the values of keys in the associative array with the desired value.

Searching for a Value

```
puts nil, "Searching for 'Thebes' value: "

foundThebes = false
```

```
rubyDictionary.each { |key, value|
    if value == 'Thebes'
        foundThebes = true
        break
    end}
if (foundThebes)
  puts "'Thebes' value is successfully found."
else
  puts "'Thebes' was not found as a value."
end
puts nil, "Searching for 'Athens' value: "
foundAthens = false
rubyDictionary.each { |key, value|
    if value == 'Athens'
        foundAthens = true
        break
    end}
if (foundAthens)
  puts "'Athens' value is successfully found."
else
  puts "'Athens' was not found as a value."
End
```

Output

```
Searching for 'Thebes' value:
'Thebes' value is successfully found.
Searching for 'Athens' value:
'Athens' was not found as a value.
```

Explanation

This code segment first searches for the values 'Thebes' and 'Athens'. First, it creates a boolean value to indicate whether the value is found or not. After iterating through the associative array, if the value is present, foundThebes or foundAthens becomes true and the loop is exited. Based on the values of foundThebes and foundAthens, necessary output messages are displayed. The iteration is done by using the **each** method in Ruby.

Looping through Array

```
puts nil, "Looping through the array: "
def foo(key, value)
  puts "#{key}: #{value} "
end
rubyDictionary.each {|key, value|
      foo(key, value) }
```

Output

```
Looping through the array:
Assyria: Assur
Babylonia: Babylon
Sumeria: Ur
Egypt: Thebes
```

Explanation

This code segment defines a function foo() that prints key-value pairs. Then, it calls the function foo() in a for loop to print each pair with the help of built-in **each** method.

7. Rust

Initialization

```
let mut rust_dictionary = HashMap::new();

rust_dictionary.insert("Assyria", "Nineveh");

rust_dictionary.insert("Babylonia",
"Babylon");

rust_dictionary.insert("Sumeria", "Ur");

rust_dictionary.insert("Persia", "Susa");
```

Output

Since it is just the initialization process, there is no output yet.

Explanation

This code segment initializes associative array in Rust with HashMap class.

Getting a Value

```
println!("Getting the value of 'Sumeria': ");

let value1 = rust_dictionary.get("Sumeria");

match value1 {

    Some(value1) => println!("The value of
    'Sumeria' is: {:?}", value1),

    None => println!("The value of 'Sumeria'
    does not exist."),}

println!();
```

Output

```
Getting the value of 'Sumeria':

The value of 'Sumeria' is: "Ur"
```

Explanation

This code segment gets the associated value of ‘Sumeria’ and prints it. If the value is **None**, then it prints an error message meaning that the value actually does not exist.

Adding a new Element

```
println!("Adding a new element 'Egypt': ");

rust_dictionary.insert("Egypt", "Thebes");

println!("{:?}", rust_dictionary);
```

Output

```
Adding a new element 'Egypt':

{"Egypt": "Thebes", "Babylonia":
"Babylon", "Assyria": "Nineveh",
"Sumeria": "Ur", "Persia": "Susa"}
```

Explanation

This code segment adds a new kay-value pair which is {‘Egypt’: ‘Thebes’} and then prints the new array as Rust supports printing of associative arrays.

Removing an Element

```
println!();

  println!("Removing the element 'Persia': ");

  rust_dictionary.remove("Persia");

  println!("{:?}", rust_dictionary);
```

Output

```
Removing the element 'Persia':

{"Egypt": "Thebes", "Babylonia":
"Babylon", "Assyria": "Nineveh",
"Sumeria": "Ur"}
```

Explanation

This code segment first removes the value ‘Persia’ from the array by using the

remove() method. After that, it prints the array to see the changes.

Modifying an Element

```
println!();

println!("Modifying the element 'Assyria': ");

*rust_dictionary.get_mut("Assyria").unwrap() = "Assur";

println!("{:?}", rust_dictionary);
```

Output

```
Modifying the element 'Assyria':

{"Egypt": "Thebes", "Babylonia": "Babylon", "Assyria": "Assur", "Sumeria": "Ur"}
```

Explanation

This code segment modifies the assocaited value for the key ‘Assyria’ from ‘Nineveh’ to ‘Assur’ and prints the new associative array.

Searching for a Key

```
println!();

println!("Searching for 'Babylonia' key:");

if rust_dictionary.contains_key("Babylonia"){

    println!("'Babylonia' key is found.");}

else{

    println!("'Babylonia' key was not found.");}

println!();

println!("Searching for 'Rome' key:");

if rust_dictionary.contains_key("Rome"){

    println!("'Rome' key is found.");}

else{

    println!("'Rome' key was not found.");}
```

Output

```
Searching for 'Babylonia' key:

'Babylonia' key is found.


Searching for 'Rome' key:

'Rome' key was not found.
```

Explanation

This code segment first searches for the keys ‘Babylonia’ and ‘Rome’. This is easily achieved thanks to the **contains_key()** method of the Rust which return true if the key is present.

Searching for a Value

```
println!();

println!("Searching for 'Thebes' value");

let mut found_thebes = false;

for value in rust_dictionary.values() {

   if value == &"Thebes"{

      found_thebes = true;

      break;}}

if found_thebes{

    println!( "'Thebes' value is successfully found.");}

else {

    println!("'Thebes' was not found as a value.");}
```

```
println!();

println!("Searching for 'Athens' value");

let mut found_athens = false;

for value in rust_dictionary.values() {

   if value == &"Athens"{

         found_athens = true;

         break;}}

if found_athens{

      println!( "'Athens' value is successfully found.");}

else {

      println!("'Athens' was not found as a value.");}
```

Output

```
Searching for 'Thebes' value

'Thebes' value is successfully found.


Searching for 'Athens' value

'Athens' was not found as a value.
```

Explanation

This code segment first searches for the values 'Thebes' and 'Athens'. First, it creates a boolean value to indicate whether the value is found or not. After iterating through the associative array, if the value is present, found_thebes or found_athens becomes true and the loop is exited. Based on the values of found_thebes and found_athens, necessary output messages are displayed. The **values()** method is used in Rust to reach the list of values in the list.

Looping through Array

```
println!("Looping through the array: ");

fn foo(key: &&str, value: &&str){

       println!("{}: {}", key, value);}

for (key, value) in &rust_dictionary{

       foo(key, value);}
```

Output

```
Looping through the array:

Sumeria: Ur

Egypt: Thebes

Babylonia: Babylon

Assyria: Assur
```

Explanation

This code segment defines a function foo() that prints key-value pairs. Then, it calls the function foo() in a for loop to print each pair.

Language Evaluation

Dart

Dart was one of the better programming languages for this assignment. It had all the necessary functions to perform the operations such as containsKey(), containsValue() or remove(). Furthermore, accessing key-value pairs was also easy. In terms of readability, the language was clear and understandable. However, the necessity of putting $ signs before a variable in strings decreased the writability a little bit.

JavaScript

JavaScript was not one of the best programming languages for associative array declaration. First of all, it lacked built-in functions for printing the array or checking for the existence of a value. Nevertheless, it provided a method for checking the existence of a key which was hasOwnProperty() method. Readability was not good because of the methods like document.write instead of a print statement. Writability was also a big issue. Use of <br\> for newline characters was problematic and error-prone.

Lua

Lua was quite insufficient in terms of using associative arrays. The language does not contain any special methods for checking the existence of keys or values as well as a method for printing the array. To continue, array initialization was not very practical. Therefore, the language is one of the worse ones in terms of using associative array data structure. On the other hand, it is definitely one of the best readable languages. It avoided unnecessary punctuations like "{"or ";" which was good. Both if statements and for loops were crystal clear. The only problem for writability was the obligation to put **end** keyword at the end of an if statement or a for loop.

PHP

In terms of using associative array structures, PHP was again one of the better ones if not the best. It had all the necessary built-in functions except for printing the array. The methods array_search() and array_key_exists() were also quite helpful. However, PHP was probably the worst language in terms of both writability and readability. Putting a $ sign before variables both decreased writability and readability. Also, the use of echo instead of a print statement made the language very difficult to understand. The alternative use of <br> instead of \n was another problem.

Python

Python was definitely the best language for associative arrays. It had all the built-in functions including the ability of printing an associative array. The methods were also really easy to understand. Python was also the best in terms of readability and writability as everything was clear.

Ruby

Ruby was also helpful in using the associative array data structure. Declaration of the array was compact and nice. Furthermore, it supported printing of associative arrays which some languages like PHP lacked. However, accessing the key-value pairs by |key, value| inside of the each method was something unusual. So, writability was really poor as the format of the language was quite different and hard to remember. Finally, the use of puts also decreased the readability instead of a print statement.

Rust

For the use of associative array structures, Rust was the worst language. Even modifying the value was really difficult whereas it is quite easy in every other language. Moreover, the language did not have any special functions for value or key searching. Writability and readability were also very poor. The use of & at different places was confusing and hard to remember. The extra "!" at the end of println statements definitely decreased writability. Overall, it was a poor language.

Learning Strategy

For this homework, I started by learning the basic principles of some languages like Rust or Dart. I checked out function declarations, scoping or variable declarations. I usually gained these information from the official documentations of the languages. After that, I learned the associative arrays for each language. For instance, some were under the name of "dictionary" and some were called as "hash maps" in different languages. I learnt how each language handles these structures. Finally, I wrote and compiled the codes with the help of online compilers. When I stuck on a problem, I consulted my friends on whether they faced a similar problem and tried different algorithms for a solution. This mostly happened in searching for keys and values in languages with no specific built-in functions for them. Finally, from time to time, I ran small test programs to understand a concept better or see possible results of an algorithm.

URLs of Online Compilers

Dart: https://dartpad.dev/?

JavaScript: https://js.do/

Lua: https://www.tutorialspoint.com/execute_lua_online.php

PHP: https://www.w3schools.com/php/phptryit.asp?filename=tryphp_compiler

Python: https://www.programiz.com/python-programming/online-compiler/

Ruby: https://replit.com/languages/ruby

Rust: https://play.rust-lang.or