

CS 315 HW1

BURAK YETİŞTİREN 21802608 SECTION-1



Table of Contents

<i>Dart</i>	3
1. Initialize.....	3
2. Get the value for a given key	3
3. Add a new element.....	4
4. Remove an element.....	4
5. Modify the value of an existing element	4
6. Search for the existence of a key	4
7. Search for the existence of a value.....	5
8. In a loop through an associative array, function foo is applied, which prints the key-value pair	5
<i>JavaScript</i>	6
1. Initialize.....	6
2. Get the value for a given key	6
3. Add a new element.....	7
4. Remove an element.....	7
5. Modify the value of an existing element	7
6. Search for the existence of a key	8
7. Search for the existence of a value.....	8
8. In a loop through an associative array, function foo is applied, which prints the key-value pair ..	10
<i>Lua</i>	11
1. Initialize.....	11
2. Get the value for a given key	11
3. Add a new element.....	11
4. Remove an element.....	12
5. Modify the value of an existing element	12
6. Search for the existence of a key	12
7. Search for the existence of a value.....	13
8. In a loop through an associative array, function foo is applied, which prints the key-value pair ..	14
<i>PHP</i>	15
1. Initialize.....	15
2. Get the value for a given key	15
3. Add a new element.....	16
4. Remove an element.....	16
5. Modify the value of an existing element	16
6. Search for the existence of a key	16
7. Search for the existence of a value.....	17

8. In a loop through an associative array, function foo is applied, which prints the key-value pair..	18
<i>Python</i>	19
1. Initialize.....	19
2. Get the value for a given key	19
3. Add a new element.....	20
4. Remove an element.....	20
5. Modify the value of an existing element	21
6. Search for the existence of a key	21
7. Search for the existence of a value.....	21
8. In a loop through an associative array, function foo is applied, which prints the key-value pair..	22
<i>Ruby</i>	23
1. Initialize.....	23
2. Get the value for a given key	23
3. Add a new element.....	24
4. Remove an element.....	24
5. Modify the value of an existing element	25
6. Search for the existence of a key	25
7. Search for the existence of a value.....	25
8. In a loop through an associative array, function foo is applied, which prints the key-value pair..	26
<i>Rust</i>	27
1. Initialize.....	27
2. Get the value for a given key	28
3. Add a new element.....	29
4. Remove an element.....	29
5. Modify the value of an existing element	30
6. Search for the existence of a key	30
7. Search for the existence of a value.....	31
8. In a loop through an associative array, function foo is applied, which prints the key-value pair..	32
Evaluation of the Languages in Terms of Readability & Writability	33
Learning Strategies That I Have Used	33
Compilers & Interpreters I Have Used.....	34

Dart

1. Initialize

```
var tolls = {'Ankara': 'Free', 'Kocaeli': 25, 'Istanbul': 45, 'Sakarya': 15};
```

This segment initializes the elements of the associative array using an initializer list. First and second elements are keys, and values respectfully. In Dart, keys and values do not have to be in the same type. They can be used heterogeneously.

2. Get the value for a given key

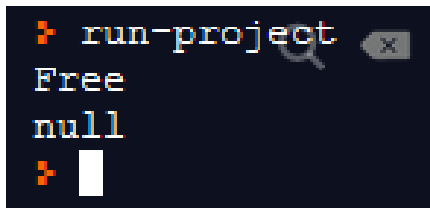
```
print(tolls['Ankara']);
```

This segment prints the value of the given key. “tolls[‘Ankara’]” statement returns the value of the key ‘Ankara’; print() function prints the returned value, Free. If the value of a key which is not in the associative array is given to be printed to the function, null is printed.

Input:

```
1  main()
2  {
3      var tolls = {'Ankara': 'Free', 'Kocaeli': 25,
4                  'Istanbul': 45, 'Sakarya': 15};
5      print(tolls['Ankara']);
6      print(tolls['Mersin']);
7  }
8  }
```

Output:



```
+ run-project
Free
null
+ 
```

3. Add a new element

```
tolls['Izmir'] = 65;
```

This segment adds a new key & value pair to the associative array. This type of segment can also be used to modify the value of a key if the key already exists in the array, which will be discussed in 5-th title.

4. Remove an element

```
tolls.remove('Izmir');
```

This segment removes the key & value pair if the key exists in the associative array. If a key which does not exist in the array is given as an argument to this function, nothing happens. Compiler does not give an error, also the array is not manipulated.

5. Modify the value of an existing element

```
tolls['Ankara'] = 10;
```

As stated in 3-rd title, if the key given to the tolls[] statement exists in the associative array, the old value of the key is replaced with the new value given. But, if the key does not exist in the array, a new key & value pair is created and added to the array.

6. Search for the existence of a key

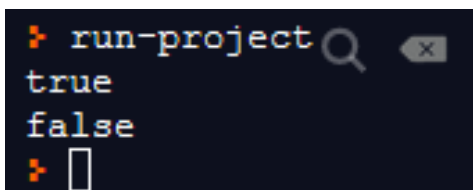
```
print(tolls.containsKey('Kocaeli'));
```

.containsKey() function takes a key as an argument and called to an associative array. If the key exists in the array it returns true, else it returns false. I have used print function to get an output.

Input:

```
1  main()
2  {
3  var tolls = {'Ankara': 'Free', 'Kocaeli': 25,
               'Istanbul': 45, 'Sakarya': 15}; //1
4
5  print(tolls.containsKey('Kocaeli'));
6  print(tolls.containsKey('Sinop'));
7  }
```

Output:



```
run-project true
false
>
```

7. Search for the existence of a value

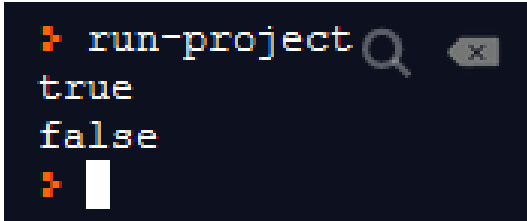
```
print(tolls.containsKey(45));
```

.containsValue() function is similar to the .containsKey() function. It takes a value as a parameter, called to an associative array. Returns true if the value exists in the array, else returns false. I have used print function to get an output.

Input:

```
1  main()
2  {
3  var tolls = {'Ankara': 'Free', 'Kocaeli': 25,
               'Istanbul': 45, 'Sakarya': 15}; //1
4
5  print(tolls.containsKey('Kocaeli'));
6  print(tolls.containsKey('Sinop'));
7  }
```

Output:



```
run-project
true
false
```

8. In a loop through an associative array, function foo is applied, which prints the key-value pair

```
void foo(key, value)
{
    print('$key:$value');
}

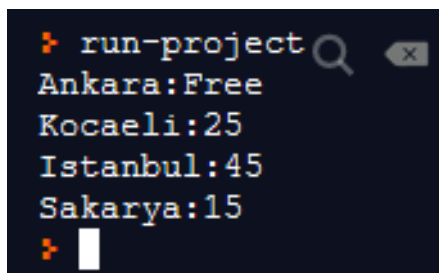
tolls.forEach(foo);
```

Defined function called foo() takes two arguments and passes them to the print function to print them in order. .forEach() loop is called to our associative array. .forEach() loop takes the function foo() as an argument and calls it to its elements.

Input:

```
1  main()
2  {
3    var tolls = {'Ankara': 'Free', 'Kocaeli': 25,
4                'Istanbul': 45, 'Sakarya': 15};
5    void foo(key, value)
6    {
7      print('$key:$value');
8    }
9
10   tolls.forEach(foo);
11 }
```

Output:



```
run-project
Ankara:Free
Kocaeli:25
Istanbul:45
Sakarya:15
run-project
```

JavaScript

1. Initialize

```
var tolls = { "Ankara": "Free", "Kocaeli": 25, "Istanbul": 45, "Sakarya": 15};
```

This segment initializes the elements of the associative array using an initializer list. First and second elements are keys, and values respectfully. In JavaScript, keys and values do not have to be in the same type. They can be used heterogeneously.

2. Get the value for a given key

```
document.write(tolls['Ankara'] + '<br>');
```

This segment prints the value of the given key. “tolls[‘Ankara’]” statement returns the value of the key ‘Ankara’; document.write() function prints the returned value, Free to the related webpage; the ‘
’ argument is to print an empty line. If the value of a key which is not in the associative array is given to be printed to the function, undefined is printed.

Input:

```
1 <script>
2     var tolls = { "Ankara": "Free", "Kocaeli": 25,
3                   "Istanbul": 45, "Sakarya": 15};
4
5     document.write(tolls['Ankara'] + '<br>');
6     document.write(tolls['Mersin'] + '<br>');
7 </script>
```

Java

Output:

Free
undefined

3. Add a new element

```
tolls['Izmir'] = 65;
```

This segment adds a new key & value pair to the associative array. This type of segment can also be used to modify the value of a key if the key already exists in the array, which will be discussed in 5-th title. This is similar to the Dart language.

4. Remove an element

```
delete tolls['Izmir'];
```

This segment deletes a key & value pair, taking the key as a parameter. If the key exists in the associative array, the key and the related value is deleted. If the key does not exists in the array nothing happens.

5. Modify the value of an existing element

```
tolls['Ankara'] = 10;
```

If the key given to the statement `tolls[]` exists in the associative array (`tolls` in this example) exists in the array, its value is changed with the provided value on the right side of the '=' sign of the statement. If the key does not exist in the array, then as stated in the 3-rd title, a new entry with the given key & value pair is created and added to the array.

6. Search for the existence of a key

```
document.write(tolls.hasOwnProperty('Kocaeli') + '<br>');
```

.hasOwnProperty() function is called to an associative array, taking the key as a parameter. If the key exists in the array, the function returns true, else false. document.write() function is used to print the returned value, '
' argument is to print an empty line.

Input:

```
1  var tolls = { "Ankara": "Free", "Kocaeli": 25,  
2  | | | | | "Istanbul": 45, "Sakarya": 15};  
3  
4  document.write(tolls.hasOwnProperty('Kocaeli') + '<br>');  
5  document.write(tolls.hasOwnProperty('Sivas') + '<br>');
```

Output:

```
true  
false
```

7. Search for the existence of a value

```
for(var i in tolls) //7  
{  
    if( tolls[i] == 45)  
    {  
        exists = true;  
        break;  
    }  
    else  
        exists = false;  
}  
document.write(exists + '<br>');
```

To check if a value exists in an associative array in JavaScript, the array should be iterated. I have used a for loop which iterates through the array. In the loop the variable i is assigned to be the key for each key present in the array. Then in the if statement, the value for the given key is checked with the expression 'tolls[i]'. If that value is equal to the value that is being searched for, then the variable "exists" is assigned to true, and the "break" statement is used to get out of the loop, else "exists" variable is assigned to false. After the loop is finished, document.write() function is used to print the output, '
' argument is to print an empty line.

Input:

```
1  var tolls = { "Ankara": "Free", "Kocaeli": 25,  
2      | | | | | "Istanbul": 45, "Sakarya": 15};  
3  
4  for(var i in tolls) //7  
5  {  
6      if( tolls[i] == 45)  
7      {  
8          exists = true;  
9          break;  
10     }  
11     else  
12         exists = false;  
13 }  
14 document.write(exists + '<br>');  
15  
16 for(var i in tolls) //7  
17 {  
18     if( tolls[i] == 55)  
19     {  
20         exists = true;  
21         break;  
22     }  
23     else  
24         exists = false;  
25 }  
26 document.write(exists + '<br>');
```

Output:

true
false

8. In a loop through an associative array, function foo is applied, which prints the key-value pair

```
function foo() //8
{
    document.write(i + ": " + tolls[i] + '<br>');
}

for(var i in tolls)
{
    foo();
}
```

Defined function called foo() uses the document.write() function to print the key & value pair. For loop is called to the associative array. foo() function is called in the for loop to print each key & value pair in the array.

Input:

```
1  var tolls = { "Ankara": "Free", "Kocaeli": 25,
2  | | | | | | | "Istanbul": 45, "Sakarya": 15};
3
4  function foo() //8
5  {
6  |   document.write(i + ": " + tolls[i] + '<br>');
7  | }
8
9  for(var i in tolls)
10 {
11 |   foo();
12 | }
```

Output:

```
Ankara: Free
Kocaeli: 25
Istanbul: 45
Sakarya: 15
```

Lua

1. Initialize

```
tolls = {Ankara = 'Free', Kocaeli = 25, Istanbul = 45, Sakarya = 15}
```

This segment initializes the elements of the associative array using an initializer list. First and second elements are keys, and values respectfully. In Lua, keys are strings which are written directly without an apostrophe or a quotation mark. The values are heterogeneous.

2. Get the value for a given key

```
print(tolls['Ankara'])
```

If the key exists in an associative array, the `tolls[]` statement with that key returns its value in Lua. If it doesn't exist, "nil" is returned. `print()` function is used to print the returned value.

Input:

```
1  tolls = {Ankara = 'Free', Kocaeli = 25,  
2          Istanbul = 45, Sakarya = 15}  
3  
4  print(tolls['Ankara'])  
5  print(tolls['Bursa'])  
6
```

Output:

```
Free  
nil  
➤
```

3. Add a new element

```
tolls["Izmir"] = 65
```

This segment adds a new key & value pair to the associative array. This type of segment can also be used to modify the value of a key if the key already exists in the array, which will be discussed in 5-th title.

4. Remove an element

```
tolls["Izmir"] = nil
```

In Lua, a key & value pair can be deleted by providing the “tolls[]” statement with the key and assigning it to “nil”. If a key which does not exist in the associative array is provided, nothing happens (no error is given, array is not manipulated).

5. Modify the value of an existing element

```
tolls['Ankara'] = 10
```

If the key given to the statement tolls[] exists in the associative array (tolls in this example) exists in the array, its value is changed with the provided value on the right side of the ‘=’ sign of the statement. If the key does not exist in the array, then as stated in the 3-rd title, a new entry with the given key & value pair is created and added to the array.

6. Search for the existence of a key

```
print(tolls['Kocaeli'] ~= nil)
```

The existence of a key is checked by the code segment above. If the value related the key is not “nil”, which means it exists, true is returned, else false is returned. print() function is used to print the returned value.

Input:

```
tolls = {Ankara = 'Free', Kocaeli = 25,  
         Istanbul = 45, Sakarya = 15}  
  
print(tolls['Kocaeli'] ~= nil)  
print(tolls['Erzincan'] ~= nil)
```

Output:

```
true  
false  
➤
```

7. Search for the existence of a value

```
local found = nil --7
for i in pairs(tolls) do
    if tolls[i] == 45 then
        found = true
        break
    end
end
print(found)
```

To check if a value exists in an associative array in Lua, the array should be iterated. I have assigned a variable called “found” as “nil”. Then, I have used a for loop which iterates through the array. In the loop the variable *i* is assigned to be the key for each key present in the array. Then in the if statement, the value for the given key is checked with the expression ‘tolls[i]’. If that value is equal to the value that is being searched for, then the variable “found” is assigned to true, and the “break” statement is used to get out of the loop, else it remains as “nil”. After the loop is finished, print() function is used to print the output.

Input:

```
1  tolls = {Ankara = 'Free', Kocaeli = 25,
2          Istanbul = 45, Sakarya = 15}
3
4  local found = nil
5  for i in pairs(tolls) do
6      if tolls[i] == 45 then
7          found = true
8          break
9      end
10 end
11 print(found)
12
13 local found = nil
14 for i in pairs(tolls) do
15     if tolls[i] == 100 then
16         found = true
17         break
18     end
19 end
20 print(found)
```

Output:

```
true
nil
>
```

8. In a loop through an associative array, function foo is applied, which prints the key-value pair

```
function foo(key, value)
    print(key, value)
end

for i in pairs(tolls) do
    foo(i, tolls[i])
end
```

Defined function called foo() takes two arguments and passes them to the print function to print them in order. for loop is called to our associative array loop takes the function foo() as an argument and calls it to its elements.

Input:

```
1  tolls = {Ankara = 'Free', Kocaeli = 25,
2          | Istanbul = 45, Sakarya = 15}
3
4  function foo(key, value) --8
5      print(key, value)
6  end
7
8  for i in pairs(tolls) do
9      foo(i, tolls[i])
10 end
```

Output:

```
Ankara Free
Istanbul 45
Sakarya 15
Kocaeli 25
✚
```

PHP

1. Initialize

```
$tolls = array("Ankara"=>"Free", "Kocaeli"=>25, "Istanbul"=>45, "Sakarya"=>15)
```

This segment initializes the elements of the associative array using an initializer list. First and second elements are keys, and values respectfully. In PHP, keys and values do not have to be in the same type. They can be used heterogeneously. Before initializing an associative array, keyword “array” must be used. Also in PHP, the initial character of variables is ‘\$’.

2. Get the value for a given key

```
print($tolls["Ankara"]);
```

This segment prints the value of the given key. “\$tolls[‘Ankara’]” statement returns the value of the key ‘Ankara’; print() function prints the returned value, Free. If the value of a key which is not in the associative array is given to be printed to the function, nothing is printed, and a PHP Notice is given.

Input:

```
<?php
$tolls = array("Ankara"=>"Free",
               "Kocaeli"=>25, "Istanbul"=>45,
               "Sakarya"=>15); //1
print($tolls["Ankara"]);
echo "<br>";

print($tolls["Van"]);
?>
```

Output:

https://PrivateReadyCertifications.bur.
Free

Console:

```
PHP 7.2.17-0ubuntu0.18.04.1 (cli) (built:
Apr 18 2019 14:12:38) ( NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.2.0, Copyright (c) 1998-201
8 Zend Technologies
[Sat Nov 28 18:35:15 2020] PHP Notice: Un
defined index: Van in /home/runner/Private
ReadyCertifications/index.php on line 11
[Sat Nov 28 18:35:15 2020] 172.18.0.1:4098
8 [200]: /
□
```


3. Add a new element

```
$tolls["Izmir"] = 65;
```

This segment adds a new key & value pair to the associative array. This type of segment can also be used to modify the value of a key if the key already exists in the array, which will be discussed in 5-th title.

4. Remove an element

```
unset($tolls["Izmir"]);
```

In PHP, unset() function is used to remove a key & value pair from the associative array. The parameter \$tolls["Izmir"] is provided to the function in this example. If the key exists in the array, the pair is removed, if the key does not exist nothing happens (no error is given, array is not manipulated).

5. Modify the value of an existing element

```
$tolls["Ankara"] = 10;
```

If the key given to the statement \$tolls[] exists in the associative array (\$tolls in this example) exists in the array, its value is changed with the provided value on the right side of the '=' sign of the statement. If the key does not exist in the array, then as stated in the 3-rd title, a new entry with the given key & value pair is created and added to the array.

6. Search for the existence of a key


```
if(array_key_exists("Kocaeli", $tolls))  
    print "true";  
else  
    print "false";
```

array_key_exists() function takes two arguments the key and the associative array, respectively. It returns 1 if the key exists, empty if the key does not exist. I have used an if statement to be able to print true or false properly. If the key exists in the array true is printed, else false is printed.

Input:

```
$tolls = array("Ankara"=>"Free", "Kocaeli"=>25, "Istanbul"=>45,  
"Sakarya"=>15);  
  
if(array_key_exists("Kocaeli", $tolls)) //6  
| print "true";  
else  
| print "false";  
echo "<br>";  
  
if(array_key_exists("Gaziantep", $tolls)) //6  
| print "true";  
else  
| print "false";  
echo "<br>";
```

Output:

 <https://PrivateReadyCertifications.burakyetistiren.repl.co>

true
false

7. Search for the existence of a value

```
$val = 45;  
if(in_array($val, $tolls))  
    print "true";  
else  
    print "false";
```

`in_array()` function takes two arguments the value and the associative array, respectively. It returns 1 if the key exists, empty if the key does not exist. I have used an if statement to be able to print true or false properly. If the value exists in the array true is printed, else false is printed.

Input:

```
$tolls = array("Ankara"=>"Free", "Kocaeli"=>25, "Istanbul"=>45, "Sakarya"=>15);

$val = 45;
if(in_array($val, $tolls)) //7
| print "true";
else
| print "false";
echo "<br>";

$val = 100;
if(in_array($val, $tolls)) //7
| print "true";
else
| print "false";
echo "<br>";
```

Output:

 <https://PrivateReadyCertifications.burakyetistiren.repl.co>

true
false

8. In a loop through an associative array, function foo is applied, which prints the key-value pair

```
function foo($key, $val){ //8
    print "$key : $val <br>";
}

foreach ($tolls as $key => $val){
    foo($key, $val);
}
```

Defined function called foo() takes two arguments and passes them to the print function to print them in order. .foreach() loop is called to our associative array. .foreach() loop takes the function foo() as an argument and calls it to its elements.

Input:

```
$tolls = array("Ankara"=>"Free", "Kocaeli"=>25, "Istanbul"=>45, "Sakarya"=>15);

function foo($key, $val){ //8
    print "$key : $val <br>";
}

foreach ($tolls as $key => $val){
    foo($key, $val);
}
```

Output:

<https://PrivateReadyCertifications.buraketistiren.repl.co>

```
Ankara : Free
Kocaeli : 25
Istanbul : 45
Sakarya : 15
```

Python

1. Initialize

```
tolls = {'Ankara': 'Free', 'Kocaeli': 25, 'Istanbul': 45, 'Sakarya': 15};
```

This segment initializes the elements of the associative array using an initializer list. First and second elements are keys, and values respectfully. In Python, keys and values do not have to be in the same type. They can be used heterogeneously.

2. Get the value for a given key

```
print(tolls['Ankara'])
```

This segment prints the value of the given key. “tolls[‘Ankara’]” statement returns the value of the key ‘Ankara’; print() function prints the returned value, Free. If the value of a key which is not in the associative array is given to be printed to the function, a KeyError occurs.

Input:

```
tolls = {'Ankara': 'Free', 45: 25, 'Istanbul': 45, 'Sakarya': 15}; #1  
  
print(tolls['Ankara'])  
print(tolls['Samsun'])
```

Output:

```
Traceback (most recent call last):  
  File "C:/Users/burak/PycharmProjects/pythonProject1/main.py", line 4, in <module>  
    print(tolls['Samsun'])  
KeyError: 'Samsun'  
Free
```

3. Add a new element

```
tolls['Izmir'] = 65
```

This segment adds a new key & value pair to the associative array. This type of segment can also be used to modify the value of a key if the key already exists in the array, which will be discussed in 5-th title.

4. Remove an element

```
del tolls['Izmir']
```

In Python, to delete a key & value pair from an associative array “del” keyword followed with the tolls[] statement (because our array is called tolls) with the key that is wanted to be removed is used. If the key exists in the array, the key & value pair is deleted, if the key does not exist, a KeyError occurs.

Input:

```
tolls = {'Ankara': 'Free', 45: 25, 'Istanbul': 45, 'Sakarya': 15};  
  
del tolls['Ankara']  
del tolls['Erzurum']
```

Output:

```
Traceback (most recent call last):  
  File "C:/Users/burak/PycharmProjects/pythonProject1/main.py", line 4, in <module>  
    del tolls['Erzurum']  
KeyError: 'Erzurum'
```

5. Modify the value of an existing element

```
tolls['Ankara'] = 10
```

As stated in 3-rd title, if the key given to the tolls[] statement exists in the associative array, the old value of the key is replaced with the new value given. But, if the key does not exist in the array, a new key & value pair is created and added to the array.

6. Search for the existence of a key

```
print(tolls.__contains__('Kocaeli'))
```

.__contains__() function can be called to an associative array with the key as a parameter. If the key exists in the array the function returns true, if not it returns false. The print() function is used to print the returned value.

Input:

```
tolls = {'Ankara': 'Free', 'Kocaeli': 25, 'Istanbul': 45, 'Sakarya': 15};  
  
print(tolls.__contains__('Kocaeli'))  
print(tolls.__contains__('Malatya'))
```

Output:

```
C:\Users\burak\Desktop\pythonProje  
True  
False  
  
Process finished with exit code 0
```

7. Search for the existence of a value

```
val = 45 #7  
print(val in tolls.values())
```

.values() function can be called to an associative array to return the values contained in the array. “in” keyword iterates through the returned values and looks for the provided variable before in. In our example, it looks for the val variable which is assigned to 45 beforehand. If the value is in the values true is returned, else false is returned.

Input:

```
tolls = {'Ankara': 'Free', 'Kocaeli': 25, 'Istanbul': 45, 'Sakarya': 15};
```

```
val = 45
```

```
print(val in tolls.values())
```

```
val = 100
```

```
print(val in tolls.values())
```

Output:

```
C:\Users\burak\Desktop\pythonProje
```

```
True
```

```
False
```

```
Process finished with exit code 0
```

8. In a loop through an associative array, function foo is applied, which prints the key-value pair

```
def foo(key, value): #8
    print( key,':', value)
```

```
for key in tolls:
    foo(key, tolls[key])
```

Defined function called foo() takes two arguments and passes them to the print function to print them in order. for loop is called to our associative array. Loop takes the function foo() as an argument and calls it to its elements.

Input:

```
tolls = {'Ankara': 'Free', 'Kocaeli': 25, 'Istanbul': 45, 'Sakarya': 15};

def foo(key, value): #8
    print( key,':', value)

for key in tolls:
    foo(key, tolls[key])
```

Output:

```
C:\Users\burak\Desktop\pythonProjec
Ankara : Free
Kocaeli : 25
Istanbul : 45
Sakarya : 15

Process finished with exit code 0
```

Ruby

1. Initialize

```
tolls = {"Ankara"=>"Free", "Kocaeli"=>25, "Istanbul"=>45, "Sakarya"=>15}
```

This segment initializes the elements of the associative array using an initializer list. First and second elements are keys, and values respectfully. In Ruby, keys and values do not have to be in the same type. They can be used heterogeneously.

2. Get the value for a given key

```
puts tolls['Ankara']
```

This segment prints the value of the given key. “tolls[‘Ankara’]” statement returns the value of the key ‘Ankara’; puts function prints the returned value, Free. If the value of a key which is not in the associative array is given to be printed to the function, an empty line is printed, representing “nil”; if the non-existing key is tried to be printed with a string an error occurs.

Input:

```
1  tolls = {"Ankara"=>"Free", "Kocaeli"=>25, "Istanbul"=>45, "Sakarya"=>15}
2
3  puts "Out: " + tolls['Ankara']
4  puts "Out: " + tolls['Konya']
```

Output:

```
❖ ruby main.rb
Out: Free
Traceback (most recent call last):
  1: from main.rb:4:in `<main>'
main.rb:4:in `+': no implicit conversion of nil into String (TypeError)
exit status 1
❖
```

3. Add a new element

```
tolls['Izmir'] = 65
```

This segment adds a new key & value pair to the associative array. This type of segment can also be used to modify the value of a key if the key already exists in the array, which will be discussed in 5-th title.

4. Remove an element

```
tolls.delete('Izmir')
```

In Ruby, to delete a key & value pair from an associative array `.delete()` function provided with the associative array. The key that is wanted to be removed is provided as a parameter to this function. If the key exists in the array, the key & value pair is deleted. If a key which does not exist in the associative array is provided, nothing happens (no error is given, array is not manipulated).

Input:

```
1  tolls = {"Ankara"=>"Free",
           "Kocaeli"=>25, "Istanbul"=>45,
           "Sakarya"=>15}
2
3  tolls.delete('Ankara')
4  tolls.delete('Amasya')
```

Output:

```
❖ ruby main.rb
❖
```

5. Modify the value of an existing element

```
tolls['Ankara'] = 10
```

As stated in 3-rd title, if the key given to the `tolls[]` statement exists in the associative array, the old value of the key is replaced with the new value given. But, if the key does not exist in the array, a new key & value pair is created and added to the array

6. Search for the existence of a key

```
puts tolls.key?('Kocaeli')
```

`.key?()` function can be called to an associative array with the key as a parameter. If the key exists in the array the function returns true, if not it returns false. The `puts` function is used to print the returned value.

Input:

```
1  tolls = {"Ankara"=>"Free",  
           "Kocaeli"=>25, "Istanbul"=>45,  
           "Sakarya"=>15}  
2  
3  puts tolls.key?('Kocaeli')  
4  puts tolls.key?('Adana')
```

Output:

```
❖ ruby main.rb  
true  
false  
❖
```

7. Search for the existence of a value

```
puts tolls.has_value?(45)
```

`.has_value?()` function can be called to an associative array with the value as a parameter. If the value exists in the array the function returns true, if not it returns false. The `puts` function is used to print the returned value.

Input:

```
1  tolls = {"Ankara"=>"Free",  
           "Kocaeli"=>25, "Istanbul"=>45,  
           "Sakarya"=>15}  
2  
3  puts tolls.has_value?(45)  
4  puts tolls.has_value?(100)
```

Output:

```
> ruby main.rb  
true  
false  
> 
```

8. In a loop through an associative array, function foo is applied, which prints the key-value pair

```
def foo(key, value) #8  
  puts key, value  
end  
  
for i in tolls  
  foo(i, tolls[i])  
end
```

Defined function called foo() takes two arguments and passes them to the puts function to print them in order. for loop is called to our associative array. Loop takes the function foo() as an argument and calls it to its elements.

Input:

```
1  tolls = {"Ankara"=>"Free",  
           "Kocaeli"=>25, "Istanbul"=>45,  
           "Sakarya"=>15}  
2  
3  def foo(key, value) #8  
4  |   puts key, value  
5  end  
6  
7  for i in tolls  
8  |   foo(i, tolls[i])  
9  end
```

Output:

```
❯ ruby main.rb  
Ankara  
Free  
  
Kocaeli  
25  
  
Istanbul  
45  
  
Sakarya  
15  
  
❯
```

Rust

1. Initialize

```
use std::collections::HashMap;  
let mut tolls = HashMap::new();  
  
tolls.insert("Ankara", 0);  
tolls.insert("Kocaeli", 25);  
tolls.insert("Istanbul", 45);  
tolls.insert("Sakarya", 15);
```

In Rust, operations with associative arrays are not as straightforward as the other programming languages discussed. The HashMap library should be imported to initialize & use the operations for the associative arrays (HashMap) in Rust. The variable for the array must be specified with the keyword “mut” stating that it is mutable. There is no initializer list like the other languages; .insert() function is used to insert the key & value pairs one-by-one. Unlike most of the programming languages discussed, the associative arrays cannot be heterogeneous. The types of the keys and values of the entries after the first one must be in the same type with the first one. The keys and the values should be same with the first one separately.

2. Get the value for a given key

```
println!("{}", tolls["Ankara"]);
```

This segment prints the value of the given key. “tolls[‘Ankara’]” statement returns the value of the key ‘Ankara’; puts function prints the returned value, 0. If the value of a key which is not in the associative array is given to be printed to the function, an error occurs (main function “panics” in Rust).

Input:

```
use std::collections::HashMap;
let mut tolls = HashMap::new();

tolls.insert("Ankara", 0);
tolls.insert("Kocaeli", 25);
tolls.insert("Istanbul", 45);
tolls.insert("Sakarya", 15);
//1

println!("{}", tolls["Ankara"]);
println!("{}", tolls["Edirne"]);
```

Output:

```
> rustc -o main main.rs
> ./main
0
thread 'main' panicked at 'no entry found for key', main.rs:12:18
note: run with 'RUST_BACKTRACE=1' environment variable to display a backtrace
exit status 101
> □
```

3. Add a new element

```
tolls.insert("Izmir", 65);
```

As discussed in the first title, in Rust an entry is made to an associative array is done by using the `.insert()` function called to the array. The key & value pair must be consistent to the other present entries in the array.

4. Remove an element

```
tolls.remove("Izmir");
```

In Rust, to delete a key & value pair from an associative array `.remove()` function provided with the associative array. The key that is wanted to be removed is provided as a parameter to this function. If the key exists in the array, the key & value pair is deleted. If a key which does not exist in the associative array is provided, nothing happens (no error is given, array is not manipulated).

Input:

```
use std::collections::HashMap;
let mut tolls = HashMap::new();

tolls.insert("Ankara", 0);
tolls.insert("Kocaeli", 25);
tolls.insert("Istanbul", 45);
tolls.insert("Sakarya", 15);

tolls.remove("Ankara");
tolls.remove("Trabzon");
```

Output:

```
➤ rustc -o main main.rs
➤ ./main
➤
```

5. Modify the value of an existing element

```
*tolls.get_mut("Ankara").unwrap() = 10;
```

To modify a value of a key in an associative array, two functions must be called in order to the array to access the value, then a value is assigned. Again, the assigned value must be consistent with the types of other values present in the array. `.get_mut()` function is called to access the given key as a parameter; `.unwrap()` function is used to make the value assignable. After the calling of these functions, a value is assigned. If the key does not exist in the array, a 'None' value is returned from the `.get_mut()` function, and an error occurs when the `.unwrap()` function is being tried to called to this 'None' value.

Input:

```
use std::collections::HashMap;
let mut tolls = HashMap::new();

tolls.insert("Ankara", 0);
tolls.insert("Kocaeli", 25);
tolls.insert("Istanbul", 45);
tolls.insert("Sakarya", 15);

*tolls.get_mut("Ankara").unwrap() = 10;
*tolls.get_mut("Mersin").unwrap() = 10;
```

Output:

```
> rustc -o main main.rs
> ./main
thread 'main' panicked at 'called `Option::unwrap()` on a `None` value', main.rs:15:2
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
exit status 101
> □
```

6. Search for the existence of a key

```
println!("{}", tolls.contains_key("Kocaeli"));
```

`contains_key()` function can be called to an associative array with the key as a parameter. If the key exists in the array the function returns true, if not it returns false. The `println!()` function is used to print the returned value.

Input:

```
use std::collections::HashMap;
let mut tolls = HashMap::new();

tolls.insert("Ankara", 0);
tolls.insert("Kocaeli", 25);
tolls.insert("Istanbul", 45);
tolls.insert("Sakarya", 15);

println!("{}", tolls.contains_key("Kocaeli"));
println!("{}", tolls.contains_key("Siirt"));
```

Output:

```
> rustc -o main main.rs
> ./main
true
false
> █
```

7. Search for the existence of a value

```
println!("{}", tolls.values().any(|&val| val == 45));
```

.values() function can be called to an associative array to return the values contained in the array. .any() function with the given parameters above iterates through the returned values and looks for the provided value. In our example, it looks for the val variable which is assigned to 45. If the value is in the values true is returned, else false is returned. The println!() function is used to print the returned value.

Input:

```
use std::collections::HashMap;
let mut tolls = HashMap::new();

tolls.insert("Ankara", 0);
tolls.insert("Kocaeli", 25);
tolls.insert("Istanbul", 45);
tolls.insert("Sakarya", 15);

println!("{}", tolls.values().any(|&val| val == 45));
println!("{}", tolls.values().any(|&val| val == 100));
```

Output:

```
> rustc -o main main.rs
> ./main
true
false
> █
```


8. In a loop through an associative array, function foo is applied, which prints the key-value pair

Function defined outside main:

```
fn foo(key: &str, value:i32)
{
    println!("{}", key, value);
}
```

For loop defined in main:

```
for(key, value) in &tolls{
    foo(key, *value);
}
```

Defined function called foo() takes two arguments and passes them to the println() function to print them in order. for loop is called to our associative array. Loop takes the function foo() as an argument and calls it to its elements.

Evaluation of the Languages in Terms of Readability & Writability

When the operations for the associative arrays are considered of the languages: Dart, JavaScript, Lua, PHP, Python, Ruby, and Rust, it can be said that, out of these languages, Dart, PHP, Python, and Ruby are more writable, in contrast to Lua, JavaScript, and Rust, and Dart, PHP, Python, Ruby, Lua, and JavaScript are more readable in contrast to Rust.

Rust is not writable, as there are many functions and complex operations are required to perform trivial actions to the associative arrays. Many different elements like functions, libraries, etc. must be known by the programmer, which makes Rust less writable. Also in Rust, all of the types of keys & values must be matching separately. This matter reduces writability drastically. For readability, again for the similar reasons, the language is not readable, at least for the associative array operations. Naming conventions of some functions are not easily understandable, operations with the references are complex, and difficult to understand.

JavaScript and Lua are more readable and writable when contrasted to Rust, still there are some conventions that have to be known by the programmer, but they are not as much in contrast to Rust. Also one downside of Lua is that it does not allow the usage of heterogeneous types of keys. The languages do not have primitive functions to check if a given value exists in the associative array which again makes them less writable.

Dart, PHP, Python, and Ruby provide straightforward and easy operations for the programmers to understand and implement. The naming conventions of the functions are easily understandable, which increases readability; the functions are easy to implement, which increases writability. Among these for languages Dart is most readable and writable language according to my own experience, as the operations were very straightforward, the naming conventions were natural, and close to the natural languages, as I nearly thought about the operations in my mind, without looking at any source, and they turned out to be true and working. I can say that after Dart, Ruby and Python were at the same level of readability and writability, as they required more need for resources in contrast for resources to learn about the operations, but not as much when compared to PHP.

Learning Strategies That I Have Used

I have used the information that is available online. I have read what the other programmers have done when they have faced the same problems that I had, I have gone through the available information that are published by the designers of the programming languages, that I am concerned about. After gaining the initial information needed about the operations, I have used the trial & error method: first I have got the required output, then I have given some invalid input to the programs to see how they would behave under those conditions.

In detail, the material I have used included the online forums, language definitions, and tutorial sites. I have used these materials to gain the initial knowledge about the operations I will be doing. I have tried to do the operations in online compilers for most of the languages Lua, Dart, Ruby, Rust, and PHP; programs in my own PC for the languages Python, and JavaScript. After I have gotten valid results in these compilers and interpreters, I have used the trial & error method I have discussed above.

Compilers & Interpreters I Have Used

Dart: <https://repl.it/languages/dart?v2=1>

JavaScript: Microsoft Edge web browser in my PC

Lua: <https://repl.it/languages/lua?v2=1>

PHP: <https://repl.it/@burakyetistiren/RadiantVainDoom#index.php> (*I had to sign up to use this, hence the link contains my username*)

Python: PyCharm IDE in my PC

Ruby: <https://repl.it/languages/ruby>

Rust: <https://repl.it/languages/rust?v2=1>