CS 315

Programming Languages

Homework 01 Report

2022-2023 Spring

Emre Karataş
22001641
Section 01

Instructor: H. Altay Güvenir
Teaching Assistant: Dilruba Sultan Haliloğlu

Table of Contents

A) Operations of the Languages:

1. Dart

In Dart language, array elements are list objects. Literals in the Dart list (eg. Elements) are seperated by comma, which are enclosed by square brackets.

Note that below operations are tested on DartPad online editor.

1.1. Declare/Create an Empty List:

Code Snippet:

```
/* In void main()...
* declare a string list*/
List<String> emptyStringList = [];
```

Explanation:
This code snippet basically declares an empty String type list in the Dart language.

Output:
For right now, this code snippet will not generate any output.

1.2. Initialize a List with Some Values:

Code Snippet:

```
//famous CS people in the history list.
 List<String> famousCSPeople = ['Tim Berners-Lee', 'Alan Turing', 'Barbara Liskov'];
```

Explanation:

This code snippet written in Dart language declares list (famousCSPeople) in String type and initialize it with String type values, which are seperated by commas between square brackets. Note that similar to Java, C++, String type elements also represented inside quotes.

Output:

For right now, this code snippet will not generate any output.

1.3. Check If the List Is Empty or Not:

Code Snippet:

```
// checking whether the given list is empty or not.
bool famousCSPeopleChecking = famousCSPeople.isEmpty;

// also check emptyStringList declared above
bool emptyStringListChecking = emptyStringList.isEmpty;

// printing result of bool value computed above.
famousCSPeopleChecking ? print("famousCSPeople empty? :true"):
print("famousCSPeople empty? : false");

// printing result of bool value computed above.
emptyStringListChecking ? print("emptyStringList empty? :true"):
print("emptyStringList empty? :false");
```

Explanation:
IsEmpty method provided by Dart language is used to check whether the given list is empty or not. Bool (boolean) famousCSPeopleChecking and emptyStringListChecking variables hold the values of emptiness.

In the next line, ternary operators are used to print the result of the isEmpty variables. Ternary operator is a shorthand way of writing an if-else statement.

Output:
```
famous CS People empty? : false
emptyStringList empty? :true
```

1.4. Add a New Element to a List:

Code Snippet:
```
// adding new element to list. Lars Bak is the inventor of Dart language.
famousCSPeople.add('Lars Bak');
```

Explanation:
Add method is used to add a new element to the list, which is famousCSPeople in this example. The element variable name is provided inside paranthesis. As just String values are added, variable is provided to the method inside quation marks.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

1.5. Check if a Particular Element Exists In the List:

Code Snippet:
```
/*check whether list contains particular element or not
* Ada Lovalace is the inventor of ADA Language
* and it is not in the famousCSPeople list.*/
bool containsAdaLovalace = famousCSPeople.contains('Ada Lovalace');
```

```
// printing bool result by ternary operator
containsAdaLovalace ? print("famousCSPeople includes Ada Lovalace? :true"):
print("famousCSPeople includes Ada Lovalace? :false");
```

Explanation:
contains method is used to check whether the given list contains particular element inside itself or not. The result of that is hold inside a bool variable, containsAdaLovalace in this example. As just String values are checked, variable is provided to the method inside quation marks.

In the next line, ternary operators are used to print the result of the containsAdaLovalace variable. Ternary operator is a shorthand way of writing an if-else statement.

Output:
```
famousCSPeople includes Ada Lovalace? :false
```

1.6. Remove a Particular Element From the List:

Code Snippet:
```
/*remove a particular element from the list
Barbara Liskov is an element of famousCSPeople list*/
famousCSPeople.remove('Barbara Liskov');
```

Explanation:
This code removes a particular element from the list famousCSPeople, where the element being removed is 'Barbara Liskov', who is a famous woman computer scientist has contributed exceptionally to programming languages and distributed computing.

The remove() method is called on the famousCSPeople list, with the argument 'Barbara Liskov' passed to it. This method searches the list for the first occurrence of the given element and removes it from the list.

If the element is not found in the list, this method has no effect on the list.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

1.7. Get the head and the tail of a list:

Code Snippet:
```
// get head and tail of the famousCSPeople list
var listHead = famousCSPeople.first;
var listTail = famousCSPeople.sublist(1);

// prints the first element of the list
print("List head: $listHead");
// prints all elements of the list except for the first one
print("List tail: $listTail");
```

Explanation:
The first line of code retrieves the first element of the famousCSPeople list using the first property of the list, and assigns it to a new variable called listHead. This will be the head of the list.

The second line of code retrieves a sublist of the famousCSPeople list, starting from the second element (index 1) until the last element using the sublist() method. The resulting sublist is then assigned to a new variable called listTail. This will be the tail of the list.

Finally list head and list tail variables are printed by print statement.

Output:
```
List head: Tim Berners-Lee
List tail: [Alan Turing, Lars Bak]
```

1.8 Print all of the elements in the list:

Code Snippet:
```
print("List elements:");
// prints all elements in the famousCSPeople list
for (var elements in famousCSPeople)
{
   print(elements);
}
```

Explanation:
For loop is iterates through each element in the famousCSPeople list. In each iteration, the current element of the list is assigned to a variable called elements.

Within the body of the loop, the print() statement is used to output the value of the elements variable, which contains the current element of the list. This statement will print each element of the famousCSPeople list to the console on a new line, following the label printed in the first line of code.

Note that results of 1.4 and 1.6 are also visible in the output of that code snippet.

Output:
```
List elements:
Tim Berners-Lee
Alan Turing
Lars Bak
```

2. Go

In Go language, package container lists implement doubly linked list, which can be used to store and do operations on ordered sequence of elements. In this homework's scope, all elements in the list are in the type of String.

Note that below operations are tested on rextester online editor, which is provided on course website.

2.1. Declare/Create an Empty List:

Code Snippet:

```
//creating empty list
emptyList := list.New()

//creating famousCSPeople list.
famousCSPeople := list.New()
```

Explanation:
This code snippet basically declares an empty String type list in the Go language. In the package of container/list, lists are declared by the New() function. Obviously, both of the lists declared right now is empty.

Output:
For right now, this code snippet will not generate any output.

2.2. Initialize a List with Some Values:

Code Snippet:

```
//initialazing famousCSPeople list by elements
famousCSPeople.PushBack("Tim Berners-Lee")
famousCSPeople.PushBack("Alan Turing")
famousCSPeople.PushBack("Barbara Liskov")
```

Explanation:
This code snippet written in Go language uses list created in the previous step (famousCSPeople) in String type and initialize it with String type values by using the function provided by container/list package, PushBack(…) structure. In this way it adds elements one by one by calling PushBack(..) method. (*)

(*) Possible errors may occur for PushBack(…) method, but for the scope of this course and String type limitation, it is unlikely to occur for this particular case. If the one tries to use PushBack(..) method with the type other than the list provides, errors may occur. Also, if the concurrent proccessing is in the case, one should consider mutex locks to prevent it, but it is mentioned before, this error may not occur in the scope of CS 315 since that topic is in the scope of CS 342 – Operating Systems course.

Output:
For right now, this code snippet will not generate any output.

2.3. Check If the List Is Empty or Not:

Code Snippet:
```
//getting length of the famousCSPeople list to determine whether it is empty or not
famousCSPeopleChecking := famousCSPeople.Len() == 0

//getting the length of the emptyList list to determine whether it is empty or not
emptyListChecking := emptyList.Len() == 0

//printing results for emptyList
if emptyListChecking {
	fmt.Println("The emptylist is empty.")
} else {
	fmt.Println("The emptylist is not empty.")
}

//printing results for famousCSPeople
if famousCSPeopleChecking {
	fmt.Println("The famousCSPeople list is empty.")
} else {
	fmt.Println("The famousCSPeople list is not empty.")
}
```

Explanation:
Len() method provided by Go language is used to compute length of the given list. Then equalizing Len() method result to the variables (famousCSPeopleChecking and emptyListChecking), one can get the results for whether the given list is empty or not.

Different from Dart language, Go language does not support ternary operations. Therefore one can use traditional if-else statements to print out results of the variables. Fmt.println(…) is provided fmt to print out results to the console.

Output:
```
The emptylist is empty.
The famousCSPeople list is not empty.
```

2.4. Add a New Element to a List:

Code Snippet:
```
// adding new element to famousCSPeople list
famousCSPeople.PushBack("Lars Bak")
```

Explanation:

PushBack(…) method is used again to add a new element to the list, which is famousCSPeople in this example. The element variable name is provided inside paranthesis. As just String values are added, variable is provided to the method inside quation marks(*).

(*) Possible errors may occur for PushBack(…) method, but for the scope of this course and String type limitation, it is unlikely to occur for this particular case. If the one tries to use PushBack(..) method with the type other than the list provides, errors may occur. Also, if the concurrent proccessing is in the case, one should consider mutex locks to prevent it, but it is mentioned before, this error may not occur in the scope of CS 315 since that topic is in the scope of CS 342 – Operating Systems course.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

2.5. Check if a Particular Element Exists In the List:

Code Snippet:

```
//checking whether the particular element exists in the famousCSPeople list or not.
element := "Ada Lovelace"
containsAdaLovalace := false

for e := famousCSPeople.Front(); e != nil; e = e.Next() {
	if e.Value == element {
		containsAdaLovalace = true
		break
	}
}

// printing results for element existing result for the famousCSPeople list.
if containsAdaLovalace {
	fmt.Println("famousCSPeople contains Ada Lovalace")
} else {
	fmt.Println("famousCSPeople does not contain Ada Lovalace")
}
```

Explanation:
This code snippet basically uses variables and for loop to check whether the given element is in the list or not. One can define two variables, element to search and a ‘flag’ variable to keep whether the given element is found or not.

In the list, container/list packages enables user to iterate in the list from the beginning to the end. Front() method enables user to get first item in the list and it is assigned to the variable called e. By Next() method, iteration occurs until the e reaches end of the list, which is indicated by ‘nil’ by Go language. If the element searched is found, it changes value of the boolean variable. Then, result of the variable is printed to screen by fmt.Println(…). Note that Go language does not support ternary operations.

Output:

famousCSPeople does not contain Ada Lovalace

2.6. Remove a Particular Element From the List:

Code Snippet:

```
//removing particular element from the famousCSPeople list
elementToRemove := "Barbara Liskov"

for e := famousCSPeople.Front(); e != nil; e = e.Next() {
        if e.Value == elementToRemove {
                famousCSPeople.Remove(e)
                break
        }
}
```

Explanation:
This particular code snippet first identifies the element to remove by declaring it as variable. Then, iterating over elements of the list, if the iterating's item value is equal to elementToRemove variable value, then Remove(…) method is called to remove the element from the list.

Since these code is searching element to remove in the for loop, if it does not found the element nothing will happen. That is, for loop is making this code snippet safe. However, as mentioned above, concurreny problems may happen which is not in the scope of this course.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

2.7. Get the Head and the Tail of a List:

Code Snippet:

```
//getting head of the famousCSPeople list and printing it
if famousCSPeople.Len() > 0 {
   head := famousCSPeople.Front().Value
   fmt.Println("Head is:", head)
}

//getting tail slice of the famousCSPeople list and printing it
// slicing is used to get tail list of the famousCSPeople list.
if famousCSPeople.Len() > 1 {
    tail := make([]interface{}, 0)
    for e := famousCSPeople.Front().Next(); e != nil; e = e.Next() {
          tail = append(tail, e.Value)
    }
fmt.Println("Tail is:", tail)
}
```

Explanation:
This code snippet retrieves the head and tail of the famousCSPeople list and then prints them.Firstly, the program checks whether the famousCSPeople list has at least one element or not by comparing its length to 0 (famousCSPeople.Len() > 0). If the list is not empty, it retrieves the first element of the list using the Front() method and then accesses its value using the Value field (famousCSPeople.Front().Value). This value is assigned to the variable head. Finally, it prints the head value to the console using fmt.Println.

Secondly, retrieving tail operation happens if the famousCSPeople list has more than one element by comparing its length to 1 (famousCSPeople.Len() > 1). If the list has at least two elements, it initializes a new slice called tail with the type []interface{} and a length of 0(*).

(*)In this example, slicing and container lists are used together since container lists itself do not support getting all elements which are not the head of the list.

The for loop starts iterating from the second element of the list (famousCSPeople.Front().Next()) and continues until it reaches the end of the list (e != nil). In each iteration, the loop appends the current element's value (e.Value) to the tail slice using the append() function.

After the loop completes, the tail slice contains all elements of the famousCSPeople list except the first one (head). Finally, it prints the tail slice to the console using fmt.Println.

It’s worth to imply that the code checks for the list's length before accessing the head and tail elements, which prevents potential issues when trying to access an empty list.

Output:
```
Head is: Tim Berners-Lee
Tail is: [Alan Turing Lars Bak]
```

2.8. Print All of the Elements in the List:

Code Snippet:
```
//printing all elements in the famousCSPeople list
for e := famousCSPeople.Front(); e != nil; e = e.Next() {
		fmt.Println(e.Value)
}
```

Explanation:
This code snippet is a simple loop that iterates through the famousCSPeople list and prints the value of each element. It uses same functionalities of Front(), ‘nil’ Next() ‘Value’ compartments of Go language, which are explained above in detail.

Note that results of 2.4 and 2.6 are also visible in the output of that code snippet.

Output:
```
Tim Berners-Lee
Alan Turing
Lars Bak
```

3. Javascript

In Javascript, lists can be implemented using arrays. Note that implementations of Javascript language in this homework done on HTML file with using <script> </script> brackets (ie. JS implemented inside HTML file, not by using Node.js or any other JS component).

Note that below operations are tested on OneCompiler online editor and Firefox local machine browser (by selecting inspect button to see output).

3.1. Declare/Create an Empty List:

Code Snippet:
```
// Declare an empty list
let emptyList = [];
```

Explanation:
This code snippet basically declares an empty list in the JS. It's worth to note that in Javascript language, in declaration; the type (String in this homework) is not implied explicitly, which is different from some other language evaluated this homework such as Go(Golang).

Output:
For right now, this code snippet will not generate any output.

3.2. Initialize a List with Some Values:

Code Snippet:
```
// Initialize a list with some values
let famousCSPeople = ["Tim Barners-Lee","Alan Turing", "Barbara Liskov"];
```

Explanation:
This code snippet written in Javascript declares a list and populates that list(ie. Array) by declaring element values. Note that "let" token is used to declare arrays (lists) in the JS language(*). Also note that even though this homework just regards String type values, JS may hold any type of values in Arrays, providing that they are properly used later in that code(**).

However, there may be possible errors may occur due to this initialization is Javascript:

Index Errors (*): *If the list is accessed using an index that is out of bounds, it will result in an error. For example, if the list only has three elements and we try to access the fourth element, it will result in an error. Such example is:*

```
let famousCSPeople = ["Tim Barners-Lee","Alan Turing", "Barbara Liskov"];

if (famousCSPeople.length > 3) {
  console.log("The list has more than 3 elements.");
} else {
  console.log("The list has 3 or fewer elements.");
}

if (famousCSPeople[3]) {
  console.log(famousCSPeople[3]);
} else {
  console.log("The element at index 3 does not exist.");
}
```

Handling Data Type Errors (**):
To handle data type errors, one can use the typeof operator to check the data type of the elements in the list before performing any operations on them. Such example is:

```
let famousCSPeople = ["Tim Barners-Lee","Alan Turing", "Barbara Liskov", 315, true];

for (let i = 0; i < famousCSPeople.length; i++) {
  if (typeof famousCSPeople[i] !== 'string') {
    console.log(famousCSPeople[i] + " is not a string.");
  }
}
```

Output:
For right now, this code snippet will not generate any output.

3.3. Check If the List Is Empty or Not:

Code Snippet:

```
// Check if the emptyList is empty or not
console.log(emptyList.length === 0 ? "emptyList is empty" : "emptyList is not empty");
// Check if the famousCSPeople is empty or not
console.log(famousCSPeople.length === 0 ? "famousCSPeople is empty"
"famousCSPeople is not empty");
```

Explanation:
Length property provided by JS language (similar to Java array length property) is used to compute length of the given list. One can get the results for whether the given list is empty or not, by using equalition (===) operator provided by JS.

Similar to Dart language and some other imperative languages, JS language does support ternary operations.

Output:

```
emptylist is empty.
famousCSPeople is not empty.
```

3.4. Add a New Element to a List:

Code Snippet:

```
// adding new element to famousCSPeople list
famousCSPeople.push("Lars Bak");
```

Explanation:
Push(…) method is used to add a new element to the list, which is famousCSPeople in this example. The element variable name is provided inside paranthesis. As just String values are added, variable is provided to the method inside quation marks. It's worth to mention that data-type errors may occur if the push(…) function is not used properly, which is explained above.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

3.5. Check if a Particular Element Exists In the List:

Code Snippet:

```
// Check if a particular element exists in the list
if (famousCSPeople.includes("Ada Lovelace")) {
        console.log("Ada Lovelace exists in famousCSPeople");
} else {
        console.log("Ada Lovelace does not exist in famousCSPeople");
}
```

Explanation:
This code snippet basically uses includes(…) method to check whether the provided String occurs in the array or not. These explanation turns to be a bool operation, if the element exists in the array, necessary information is printed on the console.

Output:

```
Ada Lovelace does not exist in famousCSPeople
```

3.6. Remove a Particular Element From the List:

Code Snippet:

```
// Remove a particular element from the list
let index = famousCSPeople.indexOf("Barbara Liskov");
if (index !== -1) {
        famousCSPeople.splice(index, 1);
}
```

Explanation:
This particular code snippet first identifies the element’s index to remove. Then, if the element to remove has been found in the list(Arrays), by indicating if it is found the value of index will not be -1.These if statement handles error preventing since if the element of Array would not have been found in the array, then the splice method would have been removing last element of the array.

Then splice(… , …) method is used to remove desired element. Splice(… , …) method takes two elements: index of elements to remove, and the number of elements to remove.

Note that this code may contain some errors in particular cases and may need to be modified according to content of the array. Such cases are:

- Array may include more than one desired element to remove. In this case, first occurence of this element may be removed. To remove desired element’s all occurences, for loop iteration is needed.

- Array may not be declared some cases before these operations. Therefore one may check that whether the array is declared and initialized before applying these operations.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

3.7. Get the Head and the Tail of a List:

Code Snippet:
```
if (Array.isArray(famousCSPeople) && famousCSPeople.length > 0) {
        let head = famousCSPeople[0];
        let tail = famousCSPeople[famousCSPeople.length - 1];

        // Print the head and tail of the list with labels
        console.log("Head is: " + head);
        console.log("Tail is: " + tail);
} else {
        console.log("The famousCSPeople array is empty or not defined.");
}
```

Explanation:

This particular code snippet uses an if statement to check if the famousCSPeople variable is defined and is an array using the Array.isArray() method. This check helps prevent any reference errors that may occur if famousCSPeople is not defined or is not an array.

The code also checks if the array has at least one element using the length property. This check ensures that we do not try to access elements of an empty array, which would result in an index error.

The slice() method creates a new array containing a copy of a portion of the original array. It takes two arguments: the starting index of the slice (inclusive) and the ending index of the slice (exclusive).

In the code above, one can use the slice() method to extract a portion of the famousCSPeople array starting from index 1, which means all the elements of the array except for the first element (which is the head).

By default, if the ending index is not specified, slice() will extract all the elements from the starting index to the end of the array. In this case, one can only specify the starting index of the slice, so slice() will extract all the elements of the array from index 1 to the end.

The resulting array will contain all the elements of the famousCSPeople array except for the first element, which is the head.

Then the head and the tail is printed to the console if the array is actually array and it’s length > 0.

Output:

```
Head is: Tim Barners-Lee
Tail is: Alan Turing,Lars Bak
```

3.8. Print All of the Elements in the List:

Code Snippet:

```
//printing all elements of the array
if (Array.isArray(famousCSPeople) && famousCSPeople.length > 0) {
      console.log(famousCSPeople.join(", "));
} else {
     console.log("The famousCSPeople array is empty or not defined.");
}
```

Explanation:

This code snippet   basically uses Array.isArray() method is used to check if famousCSPeople is an array. If famousCSPeople is an array, the method will return true, and the code will proceed to the next condition. If famousCSPeople is not an array, the method will return false, and the code will skip to the else statement.

The && operator is used to check if the length of the famousCSPeople array is greater than 0. If the length is greater than 0, the code will execute the console.log() statement. If the length is not greater than 0, the code will skip to the else statement.

If both conditions are met, the console.log() statement is executed, which outputs the elements of the famousCSPeople array to the console, separated by a comma and a space.

If either condition fails, the else statement is executed, which outputs an error message to the console.

Note that results of 3.4. and 3.6. are also visible in the output of that code snippet.

Output:

```
Tim Barners-Lee, Alan Turing, Lars Bak
```

4. Lua

In Lua, the table data type is used to create associative arrays. These arrays allow indexing using not just numbers, but also strings and other Lua values, with the exception of nil. Tables have a flexible size, allowing you to dynamically add as many elements as needed. As the primary and sole data structuring tool in Lua, tables offer a significant level of versatility and functionality.

Note that below operations are tested on rextester online editor, which is provided on course website.

4.1. Declare/Create an Empty List:

Code Snippet:

```
-- Declare an empty list
local emptyList = {}
```

Explanation:
An empty table is created, and the reference is assigned to the variable emptyList. Lua tables can be used as lists.

Output:
For right now, this code snippet will not generate any output.

4.2. Initialize a List with Some Values:

Code Snippet:

```
--Initialize a list with some values
local famousCSPeople = {"Tim Barners-Lee", "Alan Turing", "Barbara Liskov"}
```

Explanation:
A table containing three strings is created and assigned to the variable famousCSPeople.

Output:
For right now, this code snippet will not generate any output.

4.3. Check If the List Is Empty or Not:

Code Snippet:

```
-- Check if the list is empty or not
print(#emptyList == 0 and "emptyList is empty" or "emptyList is not empty")
print(#famousCSPeople == 0 and "famousCSPeople is empty" or "famousCSPeople is not empty")
```

Explanation:
The length operator # is used to get the number of elements in the list. If the length is 0, the list is empty, otherwise not. The and and or operators are used to choose the appropriate message to print.

Output:

```
emptyList is empty
famousCSPeople is not empty
```

4.4. Add a New Element to a List:

Code Snippet:

```
-- Add a new element to a list
table.insert(famousCSPeople, "Lars Bak")
```

Explanation:
The table.insert function is used to add a new element "Lars Bak" at the end of the famousCSPeople list(*).

(*) No errors are expected in this operation as long as the table.insert function is used correctly, with a valid table and a value to insert.One can further modify the code and handle these situations such as by the code:

```
if type(famousCSPeople) == "table” then
--rest of the code
```

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

4.5. Check if a Particular Element Exists In the List:

Code Snippet:

```
-- Check if a particular element exists in the list
function contains(list, element)
```

```
    for _, value in ipairs(list) do
        if value == element then
            return true
        end
    end
    return false
end

if contains(famousCSPeople, "Ada Lovelace") then
    print("Ada Lovelace exists in famousCSPeople")
else
    print("Ada Lovelace does not exist in famousCSPeople")
end
```

Explanation:
The contains function iterates through the list using the ipairs iterator and checks if the given element exists in the list. If the element is found, the function returns true; otherwise, it returns false. The result is then used to print the appropriate message (*).

(*) Possible errors may occur due to that code snippet. If the input list is not a table, the ipairs iterator may cause issues. One should ensure that she/he is passing a valid table to the contains function. One can further modify the code to handle these operations:

```
if type(famousCSPeople) == "table" and #famousCSPeople > 0 then
--rest of the code
```

Output:

```
Ada Lovelace does not exist in famousCSPeople
```

4.6. Remove a Particular Element From the List:

Code Snippet:

```
-- Remove a particular element from the list
function removeElement(list, element)
    for i, value in ipairs(list) do
        if value == element then
            table.remove(list, i)
            break
        end
    end
end

removeElement(famousCSPeople, "Barbara Liskov")
```

Explanation:
The removeElement function iterates through the list, and when the element is found, it uses table.remove to remove the element at that index. The loop is then broken since the element has been removed(*).

(*) General case, no errors are expected in this operation. However, like the previous operation, if the input list is not a table, the `ipairs` iterator may cause issues. One should be ensured for passing a valid table to the `removeElement` function.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

4.7. Get the Head and the Tail of a List:

Code Snippet:
```
if #famousCSPeople > 0 then
   local head = famousCSPeople[1]
   local tail = {table.unpack(famousCSPeople, 2)}

   -- Print the head and tail of the list with labels
   print("Head is: " .. head)
   print("Tail is: " .. table.concat(tail, ", "))
else
   print("The famousCSPeople array is empty or not defined.")
end
```

Explanation:
This particular code snippet firstly checks if the famousCSPeople table (list) has elements in it. Then the # operator is used to get the length (number of elements) of the table. If the length is greater than 0, it means the table has elements in it. If the table has elements, it extracts the head and tail of the table. The first element of the table, famousCSPeople[1], is assigned to the variable head. The remaining elements of the table are extracted using the table.unpack function, starting from index 2, and then enclosed in a new table {} to create the tail variable.

The print function is used to display the head and tail of the table. The .. operator is used for string concatenation. For the tail, the table.concat function is used to join the elements of the tail table with a comma separator. If the table has no elements, print a message stating that the table is empty or not defined:

Output:
```
Head is: Tim Barners-Lee
Tail is: Alan Turing, Lars Bak
```

4.8. Print All of the Elements in the List:

Code Snippet:
```
-- Print all of the elements in the list
if #famousCSPeople > 0 then
   print(table.concat(famousCSPeople, ", "))
else
   print("The famousCSPeople array is empty or not defined.")
```

```
end
```

Explanation:
This code snippet checks if the famousCSPeople table has elements by using the length operator #. If there are elements in the table, it joins them with a comma separator using the table. concat function and prints the resulting string. If the table is empty or not defined, it displays a message indicating that the table is empty or not defined(*).

Note that results of 4.4. and 4.6. are also visible in the output of that code snippet.

(*) If `famousCSPeople` is not a valid table, the length operator `#` may cause an error. To prevent this, one should ensure that `famousCSPeople` is a valid table before performing the length operation.

Output:

```
Tim Barners-Lee, Alan Turing, Lars Bak
```

5. Python

In Python, lists are used to store multiple elements in one variable (list). In Python, list items are ordered elements which may occur duplicates of them in the same list.

Note that below operations are tested on rextester online editor (by Python3), which is provided on course website.

5.1. Declare/Create an Empty List:

Code Snippet:

```
# Declare an empty list
empty_list = []
```

Explanation:
An empty list is created, and the reference is assigned to the variable emptyList. To imply that the list is empty, square brackets used with their inside nothing inserted.

Output:
For right now, this code snippet will not generate any output.

5.2. Initialize a List with Some Values:

Code Snippet:

```
# Initialize a list with some values
famous_cs_people = ["Tim Barners-Lee", "Alan Turing", "Barbara Liskov"]
```

Explanation:
A list containing three strings is created and assigned to the variable famous_cs_people.

Output:
For right now, this code snippet will not generate any output.

5.3. Check If the List Is Empty or Not:

Code Snippet:
```
# Check if the list is empty or not
print("emptyList is empty" if not empty_list else "emptyList is not empty")
print("famousCSPeople is empty" if not famous_cs_people else "famousCSPeople is not empty")
```

Explanation:
This code snippet checks if two lists, empty_list and famous_cs_people, are empty or not and prints the corresponding messages. It uses conditional expressions (ternary operators) to achieve this.
For emptylist, code line checks if empty_list is empty by using the not keyword, which returns True if the list is empty and False otherwise. If the list is empty, it prints "emptyList is empty", otherwise it prints "emptyList is not empty". Same logic applies for the famous_cs_people list as well.

Output:
```
emptyList is empty
famousCSPeople is not empty
```

5.4. Add a New Element to a List:

Code Snippet:
```
# Add a new element to a list
famous_cs_people.append("Lars Bak")
```

Explanation:
The append(…) function is used to add a new element "Lars Bak" at the end of the famousCSPeople list(*).

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

5.5. Check if a Particular Element Exists In the List:

Code Snippet:
```
# Check if a particular element exists in the list
def contains(list, element):
    return element in list

if contains(famous_cs_people, "Ada Lovelace"):
    print("Ada Lovelace exists in famousCSPeople")
else:
    print("Ada Lovelace does not exist in famousCSPeople")
```

Explanation:
This code defines a function called contains to check if a particular element exists in a list and then uses it to check if "Ada Lovelace" is in the famous_cs_people list. It then prints a message based on the result.

Output:

```
Ada Lovelace does not exist in famousCSPeople
```

5.6. Remove a Particular Element From the List:

Code Snippet:

```
# Remove a particular element from the list
def remove_element(list, element):
    if element in list:
        list.remove(element)

remove_element(famous_cs_people, "Barbara Liskov")
```

Explanation:
This code defines a function called remove_element to remove a specific element from a list, and then uses it to remove "Barbara Liskov" from the famous_cs_people list.

Remove_element function takes two arguments, list and element. It checks if the given element is present in the list using the element in list expression. If the element is found in the list, it removes the element from the list using the list.remove(element) method(*).

(*) General case, no errors are expected in this operation. However, If the list parameter passed to the remove_element function is not a valid Python list or an iterable, using element in list and list.remove(element) would raise a TypeError. To avoid this, one could check if the provided object is a list or an iterable before executing the function's logic.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

5.7. Get the Head and the Tail of a List:

Code Snippet:

```
# Get the head and the tail of a list
if len(famous_cs_people) > 0:
    head = famous_cs_people[0]
    tail = famous_cs_people[1:]

    # Print the head and tail of the list with labels
    print("Head is:", head)
    print("Tail is:", ', '.join(tail))
else:
    print("The famousCSPeople array is empty or not defined.")
```

Explanation:
This code snippet checks if the famous_cs_people list is not empty, then retrieves the first element (head) and the rest of the elements (tail) from the list, and finally prints the head and tail with appropriate labels. If the list is empty, it prints a message indicating that the list is empty or not defined. Note that to print tail which may be consisted of multiple elements in the list, join(…) method is used to concanate all of the elements of the tail to the new copied array (tail).

Output:
```
Head is: Tim Barners-Lee
Tail is: Alan Turing, Lars Bak
```

5.8. Print All of the Elements in the List:

Code Snippet:
```
# Print all of the elements in the list
if len(famous_cs_people) > 0:
    print(', '.join(famous_cs_people))
else:
    print("The famousCSPeople array is empty or not defined.")
```

Explanation:
This code snippet first checks if the length of the famous_cs_people list is greater than 0 (i.e., the list is not empty). If the list is not empty, it prints all the elements in the list, separated by commas, using the join method on a string. If the list is empty, it prints a message indicating that the list is empty or not defined(*).

Note that results of 5.4. and 5.6. are also visible in the output of that code snippet.

(*)If famous_cs_people is not a list or iterable, using len(famous_cs_people) would raise a TypeError. To avoid this, one could check if the provided object is a list or iterable before proceeding with the rest of the code.

Output:
```
Tim Barners-Lee, Alan Turing, Lars Bak
```

6. Ruby

In Python, lists are used to store multiple elements in one variable by using arrays. Ruby arrays are not string compared to other languages' arrays, that is; they can grow automatically by adding elements to them.

Note that below operations are tested on rextester online editor, which is provided on course website.

6.1. Declare/Create an Empty List:

Code Snippet:

```
# Declare an empty list
empty_list = []
```

Explanation:
An empty list is created, and the reference is assigned to the variable emptyList. To imply that the list is empty, square brackets used with their inside nothing inserted.

Output:
For right now, this code snippet will not generate any output.

6.2. Initialize a List with Some Values:

Code Snippet:

```
# Initialize a list with some values
famous_cs_people = ["Tim Barners-Lee", "Alan Turing", "Barbara Liskov"]
```

Explanation:
A list containing three string item is created in Ruby language and assigned to the variable famous_cs_people.

Output:
For right now, this code snippet will not generate any output.

6.3. Check If the List Is Empty or Not:

Code Snippet:

```
# Check if the list is empty or not
puts empty_list.empty? ? "emptyList is empty" : "emptyList is not empty"
puts famous_cs_people.empty? ? "famousCSPeople is empty" : "famousCSPeople is
not empty"
```

Explanation:
This code checks if the empty_list and famous_cs_people lists are empty or not, and then prints out a message indicating their empty/non-empty status.

In Ruby, one can check if a list is empty by calling the empty? method on it. If the list is empty, this method returns true; otherwise, it returns false.

The ternary operator ? : is used to conditionally select the message to be printed based on whether the list is empty or not. If the list is empty, then "emptyList is empty" or "famousCSPeople is empty" is printed; otherwise, "emptyList is not empty" or "famousCSPeople is not empty" is printed.

Output:
```
emptyList is empty
famousCSPeople is not empty
```

6.4. Add a New Element to a List:

Code Snippet:
```
# Add a new element to the list, if it exists and is not frozen
if famous_cs_people && !famous_cs_people.frozen?
  famous_cs_people << "Lars Bak"
else
  puts "Error: famous_cs_people is not defined or is frozen."
End
```

Explanation:
This code block first checks if famous_cs_people exists and is not nil. If famous_cs_people is nil, then the condition famous_cs_people && !famous_cs_people.frozen? would evaluate to false, and the code in the else block would be executed.

If famous_cs_people is not nil, then the code checks if it is frozen using the frozen? method. If famous_cs_people is frozen, then the condition !famous_cs_people.frozen? would evaluate to false, and the code in the else block would be executed.

If famous_cs_people exists and is not frozen, then the code in the if block would be executed, and "Lars Bak" would be appended to the end of the list using the << operator.

The else block prints an error message indicating that famous_cs_people is either not defined or is frozen.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

6.5. Check if a Particular Element Exists In the List:

Code Snippet:

```
# Check if a particular element exists in the list
def contains(list, element)
  list.include?(element)
end

if contains(famous_cs_people, "Ada Lovelace")
  puts "Ada Lovelace exists in famousCSPeople"
else
  puts "Ada Lovelace does not exist in famousCSPeople"
end
```

Explanation:
This code defines a function called contains that takes two arguments: a list and an element. The function uses the include? method to check if element is present in list. The include? method returns true if element is found in list, and false otherwise. The contains function simply returns the result of the include? method.

After defining the contains function, the code calls this function with famous_cs_people as the list argument and "Ada Lovelace" as the element argument. If "Ada Lovelace" is found in famous_cs_people, then the code prints results.

Output:
```
Ada Lovelace does not exist in famousCSPeople
```

6.6. Remove a Particular Element From the List:

Code Snippet:
```
# Define the function to remove an element from the list, if it exists
def remove_element(list, element)
  if list && !list.frozen?
    list.delete(element)
  else
    puts "Error: The list is not defined or is frozen."
  end
end

# Call the function to remove an element from the list, if it exists
remove_element(famous_cs_people, "Barbara Liskov")
```

Explanation:
In this code snippet, a function called remove_element defined that takes two arguments: a list and an element. The function first checks if the list argument exists and is not frozen. If the list argument is nil or frozen, then an error message is printed. If the list argument exists and is not frozen, then the delete method is used to remove element from list.

Finally, we call the remove_element function with the famous_cs_people list and the string "Barbara Liskov" to remove "Barbara Liskov" from the list.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

6.7. Get the Head and the Tail of a List:

Code Snippet:

```
# Get the head and the tail of a list
if famous_cs_people.length > 0
  head = famous_cs_people.first
  tail = famous_cs_people.drop(1)

  # Print the head and tail of the list with labels
  puts "Head is: #{head}"
  puts "Tail is: #{tail.join(', ')}"
else
  puts "The famousCSPeople array is empty or not defined."
End
```

Explanation:
This code block retrieves the head and the tail of the famous_cs_people list, and prints them along with labels if the list is not empty.

The code begins by checking if the length of famous_cs_people is greater than 0, which indicates that the list is not empty. If the list is not empty, then the first method is used to retrieve the first element of the list, which is assigned to the variable head. The drop method is then used to retrieve all elements of the list except for the first one, which is assigned to the variable tail.

Finally, the code block prints the values of head and tail with labels. The join method is used to join the elements of tail into a comma-separated string before printing.

If the list is empty, then the code block prints a message indicating that the list is either empty or not defined.

Output:

```
Head is: Tim Barners-Lee
Tail is: Alan Turing, Lars Bak
```

6.8. Print All of the Elements in the List:

Code Snippet:

```
# Print all of the elements in the list
if famous_cs_people.length > 0
  puts famous_cs_people.join(', ')
```

```
else
  puts "The famousCSPeople array is empty or not defined."
end
```

Explanation:
This code snippet prints all of the elements in the famous_cs_people list, separated by commas, if the list is not empty. If the list is empty, it prints a message indicating that the list is either empty or not defined.

The code begins by checking if the length of famous_cs_people is greater than 0, which indicates that the list is not empty. If the list is not empty, then the join method is used to join the elements of famous_cs_people into a comma-separated string, which is then printed using the puts method.

If the list is empty, then the code block prints a message indicating that the list is either empty or not defined.

Note that results of 6.4. and 6.6. are also visible in the output of that code snippet.

Output:
```
Tim Barners-Lee, Alan Turing, Lars Bak
```

7. Rust

In Rust language, one can use Vec to implement ordered sequences (lists) to perform below operations. Vec is a heap-allocated array with changable size.

Note that below operations are tested on rextester online editor, which is provided on course website.

7.1. Declare/Create an Empty List:

Code Snippet:
```
// in fn main()…
// Declare/create an empty list
let mut empty_list: Vec<String> = Vec::new();
```

Explanation:
This code declares a mutable, empty Vec<String> list in Rust. The let keyword with mut creates a mutable variable, and Vec<String> specifies the list's type. Vec::new() is a static method that returns an empty Vec, initializing the variable.

Output:
For right now, this code snippet will not generate any output.

7.2. Initialize a List with Some Values:

Code Snippet:

```
/ Initialize a list with some values
let mut famousCSPeople = vec![String::from("Tim Barners-Lee"), String::from("Alan
Turing"), String::from("Barbara Liskov")];
```

Explanation:
A vec containing three string item is created in Rust language and assigned to the variable famous_cs_people. Same operations done in the above code, so their explanations have already given in 7.1.

Output:
For right now, this code snippet will not generate any output.

7.3. Check If the List Is Empty or Not:

Code Snippet:

```
// Check if the empty_list is empty or not
let is_empty_empty_list = empty_list.is_empty();
 println!("Is the empty list empty? {}", is_empty_empty_list);

 // Check if famousCSPeople is empty or not
 let is_empty_famousCSPeople = famousCSPeople.is_empty();
 println!("Is the famousCSPeople empty? {}", is_empty_famousCSPeople);
```

Explanation:
This code snippet checks if two Vec<String> lists (empty_list and famousCSPeople) are empty and prints the results. It calls the is_empty() method on each list, stores the boolean results in variables, and uses the println! macro to display the outcomes, replacing placeholders with the corresponding variables' values.

Output:

```
Is the empty list empty? true
Is the famousCSPeople empty? False
```

7.4. Add a New Element to a List:

Code Snippet:

```
// Add a new element to a list
famousCSPeople.push(String::from("Lars Bak"));
```

Explanation:
This code snippet demonstrates how to add a new element to a list (a Vec<String> in this case) in Rust. The push method is called on the famousCSPeople variable to add a new element to the list. The String::from("Lars Bak") expression creates a new String object containing the text "Lars Bak". The push method takes the new String object as an argument and appends it to the end of the famousCSPeople list (*).

(*) Possible errors that may occur:

Variable not mutable: *If famousCSPeople was not declared as mutable (mut keyword missing), one would get a compilation error when attempting to call the push method, as it would try to modify an immutable variable.*

Variable not in scope: *If famousCSPeople has not been declared or is not in scope when this line of code is executed, one will get a compilation error.*

Incorrect type: *If famousCSPeople is not of the type Vec<String>, one might get a compilation error or a runtime error, depending on the actual type of the variable.*

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

7.5. Check if a Particular Element Exists In the List:

Code Snippet:

```
// Check if a particular element exists in the list
let element = "Ada Lovalace";
let exists = famousCSPeople.contains(&element.to_string());
println!("Does the famousCSPeople contain {}? {}", element, exists);
```

Explanation:
This code snippet checks if an element exists in a Vec<String> list in Rust. It declares a variable element with the value "Ada Lovelace" and calls the contains method on the famousCSPeople list. The method returns a boolean value indicating if the element is found. The element.to_string() expression converts the &str to a String, and the result is stored in the exists variable.

Output:

```
Does the famousCSPeople contain Ada Lovalace? False
```

7.6. Remove a Particular Element From the List:

Code Snippet:

```
// Remove a particular element from the list
let element_to_remove = "Barbara Liskov";
famousCSPeople.retain(|x| x != element_to_remove);
```

Explanation:
This code snippet removes a specific element from a Vec<String> list in Rust by declaring a variable element_to_remove and calling retain on the famousCSPeople list. The retain method uses a closure that checks if each element is not equal to element_to_remove, keeping the element in the list if true, and removing it if false.

(*) Possible errors mentioned in 7.4. may also occur in this code snippet.

Output:
For right now, no output will be generated for this code snippet. In the coming sections, when the list is printed, output of that code snippet will be visible.

7.7. Get the Head and the Tail of a List:

Code Snippet:
```
// Get the head and the tail of a list
if let Some(head) = famousCSPeople.first() {
    println!("Head of the famousCSPeople: {}", head);
}
if famousCSPeople.len() > 1 {
    let tail = &famousCSPeople[1..];
    println!("Tail of the famousCSPeople: {:?}", tail);
 }
```

Explanation:
This code snippet demonstrates how to get the head (first element) and tail (all elements except the first one) of a Vec<String> list in Rust. The first() method is used to obtain an optional reference to the first element, and if there is one, it is printed as the head. To get the tail, the code checks if the list's length is greater than 1; if so, it creates a slice of the list from index 1 to the end and prints it as the tail.

Output:
```
Head of the famousCSPeople: Tim Barners-Lee
Tail of the famousCSPeople: ["Alan Turing", "Lars Bak"]
```

7.8. Print All of the Elements in the List:

Code Snippet:
```
// Print all of the elements in the list
println!("famousCSPeople elements: {:?}", famousCSPeople);
```

Explanation:
This code snippet prints all elements of the famousCSPeople list (a Vec<String>) using the println! macro. The {:?} placeholder in the formatted string is replaced by the debug representation of the list, which displays all elements within the list.

Note that results of 7.4. and 7.6. are also visible in the output of that code snippet.

Output:
```
famousCSPeople elements: ["Tim Barners-Lee", "Alan Turing", "Lars Bak"]
```

B) Evaluations of These Languages in Terms of Readability and Writability:

1. Dart

1.1. Readability:

Dart has a clean and expressive syntax that makes it easy to understand the code at a glance. It borrows concepts from other popular programming languages such as Java, Javascript, and C#, making it familiar to developers who have experience with these languages. For example, in the examples listed in this report, we used methods like isEmpty, add(), contains(), and remove(). These method names are self-explanatory and easy to understand.

1.2. Writability:

Dart's syntax allows for concise and straightforward code. It provides built-in methods and features that simplify common tasks, making it easier to write code. For instance, the isEmpty property checks if a list is empty, while the add() method appends an element to a list. These built-in methods minimize the need for verbose code, leading to more maintainable and efficient programs.

2. Go

2.1. Readability:

Go has a understandable and relatively clean syntax to work with. It uses function names and their functionalities similar to the imperative languages, mostly similar to C/C++ I believe. Additional to that point, Go uses familiar control structures like if, else, and for loops, which are common in many programming languages. This improves readability since developers can quickly understand the flow of the code.

However, compared to other programming languages, some of the cases in Go (nil for example) are new concepts to me which makes me to understand code hardly.

2.2 Writability:

Go has a strong typing system that helps catch errors at compile time. This encourages developers to write correct and reliable code. Above code uses PushBack, Front, Next, and Remove to manipulate the list. These functions are easy to understand and use, making the code more writable.Also, Go supports error handling through the error type, which can improve writability by allowing developers to handle different error scenarios. One possible error handling example may be inserted to above Go code snippet like this:

Code Snippet:

```
package main

import (
		"container/list"
		"errors" // to use error handling
		"fmt"
	)

	func removeElement(l *list.List, valueToRemove interface{}) error {
		for e := l.Front(); e != nil; e = e.Next() {
			if e.Value == valueToRemove {
				l.Remove(e)
				return nil
			}
		}
		return errors.New("element not found in the list") // error handling
	}

	func main() {
	// ... (The rest of your code before removing the element)

	// Removing a particular element from the famousCSPeople list
	elementToRemove := "Barbara Liskov"
	err := removeElement(famousCSPeople, elementToRemove)

	if err != nil {
		fmt.Println("Error:", err)
	} else {
		fmt.Println("Element removed:", elementToRemove)
	 }

	// ... (The rest of your code after removing the element)
	}
```

3. Javascript

3.1. Readability:

In terms of readabilty, Javascript code is generally easy to read due to its syntax being similar to other popular programming languages such as C++ and Java. Javascript supports comments, which can be used to explain the purpose of code and make it easier to understand. Javascript has a large number of built-in functions and libraries that make it easy to perform complex operations without writing a lot of code.

3.2. Writability:

In terms of writability, Javascript is a dynamic language, that is, variables can be assigned values of any type without having to declare their type explicitly. This makes it easier to write code quickly. Furthermore, Javascript has a lot of built-in functions and libraries that make it easy to perform complex operations with minimal code. Moreover, Javascript is an interpreted language, which means that developers can see the results of their code immediately without having to compile it.

4. Lua

4.1. Readability:

In terms of readabilty, Lua has a clean and minimal syntax, making it easy to read. It uses simple keywords and control structures (such as 'if', 'else', 'end') that are easy to understand. Lua is consistent in its use of conventions. For example, it consistently uses 'end' to close control structures and loops.
Also, variables and functions are given descriptive names, which improves readability.

4.2. Writability:

In terms of writability, Lua has a simple syntax that is easy to write. It requires minimal needed code, and its standard library is compact but powerful. Also, Lua has built-in support for common data structures like lists (arrays) and dictionaries (tables). Such idea explains that various operations on lists, such as adding, removing, and searching for elements, which can be performed with relatively few lines of code.

5. Python

5.1. Readability:

In terms of readabilty, a language with a clean and simple syntax, like Python, makes it easier to read and understand the code. Consistency in the use of conventions, such as indentation and variable naming, also plays a significant role in improving readability. Descriptive variable and function names help convey the purpose and functionality of the code, making it easier for others to comprehend its logic.

5.2. Writability:

In terms of writability, Python has a simple syntax that is easy to write, with minimal needed code. It features a rich standard library, which simplifies various programming tasks. Python allows various operations on lists, such as adding, removing, and searching for elements, which can be performed with relatively few lines of code.

6. Ruby

6.1. Readability:

In terms of readabilty, Ruby uses similar syntax structure to Java and the usage of operators are easily understandable due to similar approaches that we are familiar with other languages. It also has similar syntax rules to C++ by using << operator to include a new element to the list.

6.2. Writability:

In terms of writability, Ruby has a simple syntax that is easy to write, with minimal needed code. Conditional checks and the functions for common tasks are similar to the programmers who are working with imperative languages, which make Ruby language easy to code and write.

7. Rust

7.1. Readability:

In terms of readabilty, Rust has less readable since it uses lots of operators beyond the code such as :: which are also used in C/C++ family. Therefore readability is kind a complex structure in Rust language.

7.2. Writability:

In terms of writability, the same logic applies for Writability as mentioned in readability. One can struggle writting the operators correctly in the code which may lead unnecessary errors and wasting time for the developer. Even though it has complex error handling structure, due to syntax, one need to study more to be experienced in Rust language.

8. General Evaluation:

I believe that Python has the best structure to work with in terms of readability and writability of the code. It’s syntax rules are clear, supported by tons of packages growing everyday coded by millions of programmers worldwide. Function names, naming of the variables and the default functions provided by the Python 3 is clear and easy to understand.

C) Learning Strategy:

My learning strategy for this homework is to test and show with their examples about the required operations listed in the homework instructions with the potential errors which may occur due to these operations. I generally used my main sources for the languages that I listed below. Besides that, I also check regarding Stackoverflow posts to see and to determine how to act these possible errors. I also tried to make the examples simple enough to understand concepts better. Same list elements are used through the homework to see differantations between languages and their implementations.

Here are the main resources that I used during my research phase:

For Dart, I used the documentation which can be found at Dart website: https://api.dart.dev/be/180791/dart-core/List-class.html

For Go (Golang), I used the documentation which can be found GO developer website: https://pkg.go.dev/container/list

For Javascript, I used popular W3 schools tutorial website : https://www.w3schools.com/js/js_arrays.asp

For Lua, I used tutorialsPoint website documentation regarding Lua tables: https://www.tutorialspoint.com/lua/lua_tables.htm

For Python I used popular W3 schools tutorial website regarding Python lists: https://www.w3schools.com/python/python_lists.asp

For Ruby, I checked this online website which describes the operations of this homework briefly in their examples: https://www.digitalocean.com/community/tutorials/how-to-work-with-arrays-in-ruby

For Rust, I used formal documentation about Vec inside the Rust developer website: https://doc.rust-lang.org/std/vec/struct.Vec.html

The online compilers that I used during testing phase:

Dart: https://dartpad.dev/?

GO, Lua, Python, Ruby, Rust: https://rextester.com/

Javascript (as HTML): https://onecompiler.com/html

(One may use Firefox or Google Chrome for JS and may inspect the code result from the console.)