CS 315 Homework 1

Arrays in Dart, JavaScript, PHP, Python, and Rust

Student Name: Esad İsmail Tök

Student ID: 21801679

Section: 3

Dart Language

Design Issue 1: What types are legal for subscripts?

- Dart only supports integer subscript types and the other types such as double, char, string or boolean is not allowed as a subscript.

```
var list = [1, 2, 3, 4, 5];
for(int i = 0; i < list.length; i++){
  print(list[i]);
  // Belove subscript types are not allowed since they are not integers
  //print(list[true]);
  //print(list[1.0]);
  //print(list["X"]);
}
```

Figure 1: Legal subscripts in Dart language

- As it can be seen in the *Figure 1*, Dart only supports integer subscript type.

Design Issue 2: Are subscripting expressions in element references range checked?

- Dart arrays are reference checked. This check happens in run time and if an index that is out of bound is used as an index, then script error happens in run time.

```
for(int i = 0; i < list.length; i++){
  list[i] = 2 + i;
  // Below lines give script error because the indexes are out of bounds
  //list[-2] = 14;
  //list[7] = 14;
  //list[19] = 14;
}
```

Figure 2: Range checking in Dart language

- As it can be seen in the *Figure 2*, Dart checks the range of bounds in run time.

Design Issue 3: When are subscript ranges bound?

- In Dart language subscript ranges are bound in run time. Therefore it is dynamically bound.

Design Issue 4: When does allocation take place?

- Allocation takes place at run time in Dart language. Moreover the allocation is done in the heap instead of stack.

```
var names = new List(4);
// Even if we use new keyword in dart, the array is fixed sized and cannot grow and shrink. However it is still dynamically bound at run time.
names[0] = "Esad";
names[1] = "Başar";
names[2] = "Merve";
names[3] = "Sena";
print(names);
```

Figure 3: Subscript binding and allocation time in Dart language

- In *Figure 3* both the subscript binding and the allocation will happen at run time. Also the allocation will take place in heap.

Design Issue 5: Are jagged or rectangular multidimensional arrays allowed, or both?

- Dart language supports both rectangular and jagged arrays as multidimensional arrays. Namely in a multidimensional array subdimentions can both have same size or different sizes.

```
var firstDim = [5,4,3,2, 1];
var secondDim = [firstDim,firstDim,firstDim,firstDim];
for(int i = 0; i < secondDim.length; i++){
  for(int j = 0; j < firstDim.length; j++){
    print(secondDim[i][j]);
  }
}
```

Figure 4: Rectangular array in Dart language

```
var arr1 = [13, 4, 17, -5];
var arr2 = [-1,6];
var arr3 = [10];
var jaggedArr = [arr1,arr2,arr3];
for(int i = 0; i < jaggedArr.length; i++){
  for(int j = 0; j < jaggedArr[i].length; j++){
    print(jaggedArr[i][j]);
  }
}
```

Figure 5: Jagged array in Dart language

- *Figure 4* shows how to use rectangular arrays in Dart language and similarly *Figure 5* shows how to use jagged arrays in Dart Language.

Design Issue 6: Can array objects be initialized?

- In Dart, the array objects can be initialized.

```
var fixedSize = new List(10); // This array is fixed size and initialization is not done when we use this decleration
var arrInit = [1,2,3,4,5]; // Here we do array initialization and the size of the array is decşded according to the initialization
```

Figure 6: Array initialization in Dart language

- As it can be seen in *Figure 6*, we can initialize an array in the declaration step and by that, the arrays size is decided according to the number of elements in the initialization list.

Design Issue 7: Are any kind of slices supported?

- Dart supports slices by the sublist(firstIndex, secondIndex) function. In this function first index is included and the last index is excluded. This function returns a new list which is the sliced array.

```
var baseArr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var slicedArr = baseArr.sublist(1, 4);
print(slicedArr);
```

Figure 7: Slicing in Dart language

- *Figure 7* shows that slicing is possible and the returning sliced sublist in *Figure 7* is [2, 3, 4].

Design Issue 8: Which operators are provided?

- operator +(List<E> other) -> List<E> : Returns the concatenation of this list and other.
- operator ==(Object other) -> bool: Returns whether the current list is equal to the other list or not.
- operator [](int index) -> E: Returns the object at the given index.
- operator []=(int index, E value) -> void: Sets the value at the given index to the value at the parameter.

```dart
var list1 = [1, 2, 3];
var list2 = [4, 5, 6];
print(list1 + list2); // Op1
print(list1 == list2); // Op2
print(list2[0]); // Op3
list2[0] = 24; // Op4
print(list2[0]);
```

Figure 8: Operators in Dart language

```
[2, 3, 4]
[1, 2, 3, 4, 5, 6]
false
4
24
```

Figure 9: Outputs of operators in Dart language

- In *Figure 8* the usages of the operators are shown and the corresponding outputs are shown in *Figure 9*.

Python Language

Design Issue 1: What types are legal for subscripts?

- Python supports integer and boolean subscript types and the other types such as double, char, or string are not allowed as a subscript.

```
# Question 1: Subscript types
subscriptArr = np.array([1,2,3,4,5,6])
print(subscriptArr[0])
print(subscriptArr[True])
#print(subscriptArr[1.0])
#print(subscriptArr["M"])
print("-------------------------------
```

Figure 10: Legal subscripts in Python language

- Figure 10 shows that python only supports integer and boolean and true means index 1 and false means index 0.

Design Issue 2: Are subscripting expressions in element references range checked?

- Python does range checked at run time and if we try to access an element out of range, program gives list index out of range error. However negative integer indexes are valid and means the cursor goes at the end of the list.

```
rangeArr = np.array([1,2,3,4,5])
#print(rangeArr[15]); # Error
print(rangeArr[-1]); # last element
print(rangeArr[-3]); # 2 before the last element
print("-------------------------------")
```

Figure 11: Range checking in Python language

Design Issue 3: When are subscript ranges bound?

- In Python language subscript ranges are bound in run time. Therefore it is dynamically bound.

Design Issue 4: When does allocation take place?

- Allocation takes place at run time in Python language. Moreover the allocation is done in the heap instead of stack.

```
size = 4 * 2;
dyArr = np.ones(size);
print(dyArr);
```

Figure 12: Dynamic allocation and deallocation in Python language

- Figure 12 shows that the array can dynamically grow and shrink in python so the allocation takes place at run time.

Design Issue 5: Are jagged or rectangular multidimensional arrays allowed, or both?

- Python language supports both rectangular and jagged arrays as multidimensional arrays. Namely in a multidimensional array subdimentions can both have same size or different sizes.

```
# Question 5: Python allows both rectangular and jagged arrays
rectArr = np.array([[12, 15], [23, 1], [11, 11]])
for i in range(len(rectArr)):
    for j in range(len(rectArr[i])):
        print(rectArr[i][j])


jaggedArr= np.array([[12, 2, 4], [1, 2, 3, 4, 5, 6], [3]], dtype=object)
for i in range(len(jaggedArr)):
    for j in range(len(jaggedArr[i])):
        print(jaggedArr[i][j])
```

Figure 13: Rectangular and jagged multidimensional arrays in Python language

- Figure 13 shows that python can have both rectangular and jagged multidimensional arrays.

Design Issue 6: Can array objects be initialized?

- In Python, the array objects can be initialized.

```
initArr1 = range(5)
initArr2 = np.array([1, 2, 3, 4, 5])
```

Figure 14: Array initialization in Python language

- Figure 14 shows that range function can be used to initialize an array with values 0. And also an initialization list can be used as the second method.

Design Issue 7: Are any kind of slices supported?

- Python supports slicing by [firstIndex:secondIndex] operator and returns a sliced list. The first index is included and the second one is excluded.

```
baseList = np.array([1, 2, 3, 4, 5, 6])
slicedList = baseList[1:3]
print(slicedList)
```

Figure 15: Array slicing in Python language

- As Figure 15 shows, a sliced list is returned.

PHP Language

Design Issue 1: What types are legal for subscripts?

In PHP, arrays are ordered maps and the “key, value” matching works when reaching the array elements so as a “key”, any type of subscript is legal in PHP arrays.

```php
$ordredMapArray = [
  true => 1,
  "Test" => 2,
  1 => 3,
  2.0 => 4
];

echo $list[true], "\n";
echo $list["Test"], "\n";
echo $list[1], "\n";
echo $list[2.0], "\n";
```

Figure 16: Subscript Types in PHP language

- As Figure 16 shows any type of subscript is legal as a key value, since an array is an ordered map. String, boolean, int, and double are amongst the legal subscript types.

Design Issue 2: Are subscripting expressions in element references range checked?

- Yes the range check is done in PHP language, however when an index that is out of bound is encountered at run time the program does not crash and can execute the rest of the program.

```php
$arr = array(1, 2, 3, 4, 5); // Regular indexes has been assigned as the keys of the ordered map
echo $arr[0], "\n"; // Valid
echo $arr[1], "\n"; // Valid
echo $arr[2], "\n"; // Valid
echo $arr[12], "\n"; // Not valid but does not give error
```

Figure 17: Range checking in PHP language

Design Issue 3: When are subscript ranges bound?

- In PHP language subscript ranges are bound in the compile time if the array is static array. But an array in PHP language can also be a dynamic array and in that situation subscript range bound at run time.

Design Issue 4: When does allocation take place?

- In PHP language allocation takes place at the compile time if the array is static array. But an array in PHP language can also be a dynamic array and in that situation allocation takes place at run time.

```php
$dyArr = array();
var_dump($dyArr); // Array before the size growth
dyArr[] = 1;
dyArr[] = 2;
dyArr[] = 3;
var_dump($dyArr); // Array after the size growth
```

Figure 18: Subscript range binding and allocation in PHP language

- As Figure 18 shows, dynamic arrays can grow in size at run time.

Design Issue 5: Are jagged or rectangular multidimensional arrays allowed, or both?

- PHP language supports both rectangular and jagged arrays as multidimensional arrays. Namely in a multidimensional array subdimentions can both have same size or different sizes.

```php
// Question 5: Both rectangular and jagges arrays are allowed in PHP
$arrRectangular = [[2, 4, 6, 8], [4, 6, 8, 10], [1, 2, 3, 4]];
var_dump($arrRectangular);

$arrJagged = [[3], [5, 10, 15], [1, 2]];
var_dump($arrRectangular);
```

Figure 19: Rectangular and jagged multidimensional arrays in PHP language

Design Issue 6: Can array objects be initialized?

```php
$initArr = ["Esad", "Başar", "Hasan", "Merve"];
var_dump($initArr);
```

Figure 20: Initialization of arrays in PHP language

- In PHP, the array objects can be initialized.

Design Issue 7: Are any kind of slices supported?

- PHP supports slicing by array_slice(inputArray, firstIndex, secondIndex) method and returns the sliced array. The first index is included and the second one is excluded. Also if the secondIndex is not specified, the rest of the array is also returned. And negative indexing can be used to jump the cursor at the end of the array.

```php
$testArr = [1, 2, 3, 4, 5, 6, 7, 8];
$slicedArr = array_slice($testArr, 1, 4); // Returns [2, 3, 4]
$slicedArr = array_slice($testArr, 4); // Returns [5, 6, 7, 8]
$slicedArr = array_slice($testArr, -2, 1); // Returns [7]
```

Figure 21: Slicing of arrays in PHP language

Design Issue 8: Which operators are provided?

- Union: $a + $b -> Concatenation of $a and $b
- Equality: $a == $b -> Returns true if both $a and $b have the same key/value pairs
- Identity: $a === $b -> Returns true if both $a and $b have the same key/value pairs in the same order and of the same types.
- Inequality: $a != $b -> Returns true if $a and $b do not have the same key/value pairs
- Inequality: $a <> $b -> Returns true if $a and $b do not have the same key/value pairs
- Non-Identity: $a !== $b -> Returns true if $a is not identical to $b

```php
$arr1 = [1, 2, 3, 4, 5, 6];
$arr2 = [7, 8, 9, 10];
$arr3 = [1, 2, 3, 4, 5, 6];

var_dump($arr1 + $arr2); // Union
var_dump($arr1 == $arr2); // Equality
var_dump($arr1 == $arr3); // Equality
var_dump($arr1 === $arr2); // Identity
var_dump($arr1 === $arr3); // Identity
var_dump($arr1 != $arr2); // Inequality
var_dump($arr1 <> $arr2); // Inequality
var_dump($arr1 !== $arr2); // Non-Identity
```

Figure 22: Operators of arrays in PHP language

```
bool(false)
bool(true)
bool(false)
bool(true)
bool(true)
bool(true)
bool(true)
```

Figure 23: Outputs of boolean displays in Figure 22

JavaScript Language

Design Issue 1: What types are legal for subscripts?

- Javascript only supports integer subscript types and the other types such as double, char, string or boolean is not allowed as a subscript.

```
var arr = [1, 2, 3, 4, 5];
arr[0];
arr[1];
arr[2];
arr[3];
```

Figure 24: Legal Subscripts in Javascript language

Design Issue 2: Are subscripting expressions in element references range checked?

- Yes the range check is done in Javascript language, however when an index that is out of bound is encountered at run time the program does not crash and does not give an error at run time and can execute the rest of the program.

```
arr[100]; // Out of bound but does not give a run time error
```

Figure 25: Range checking in Javascript language

Design Issue 3: When are subscript ranges bound?

- In Javascript language subscript ranges are bound in the compile time if the array is static array. But an array in Javascript language can also be a dynamic array and in that situation subscript range bound at run time.

Design Issue 4: When does allocation take place?

- In Javascript language allocation takes place at the compile time if the array is static array. But an array in Javascript language can also be a dynamic array and in that situation allocation takes place at run time.

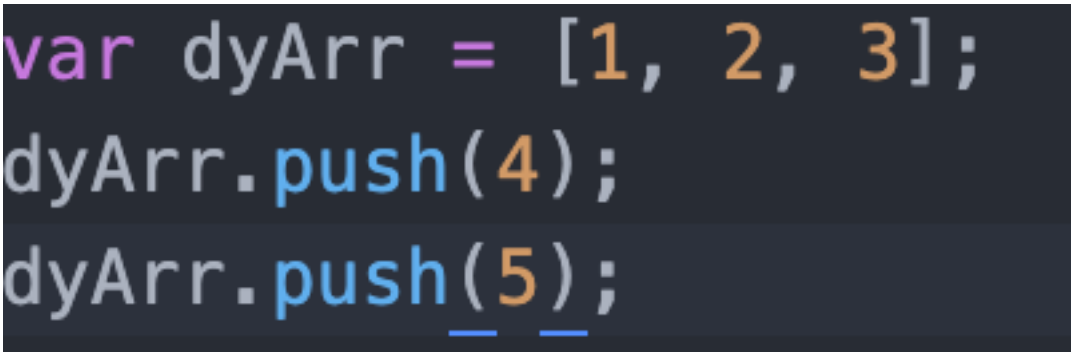

Figure 26: Dynamic range binding and allocation of arrays in Javascript language

Design Issue 5: Are jagged or rectangular multidimensional arrays allowed, or both?

- Javascript language supports both rectangular and jagged arrays as multidimensional arrays. Namely in a multidimensional array subdimentions can both have same size or different sizes.

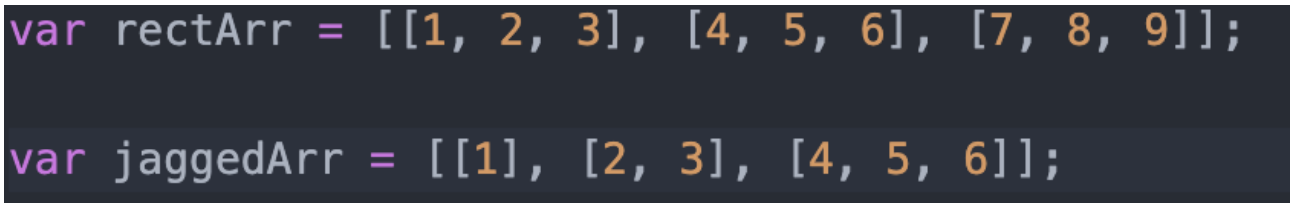

Figure 27: Rectangular and jagged arrays in Javascript language

Design Issue 6: Can array objects be initialized?

- Array objects can be initialized in Javascript.

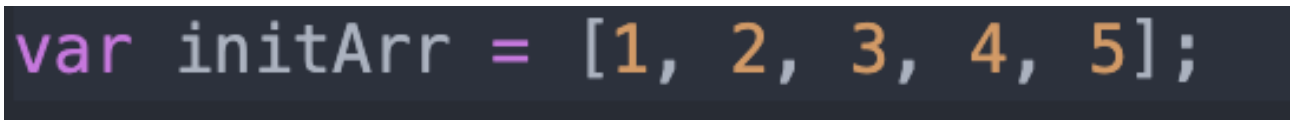

Figure 28: Initialization of arrays in Javascript language

Design Issue 7: Are any kind of slices supported?

- Javascript supports slicing by slice(firstIndex, secondIndex) method and returns the sliced array. The first index is included and the second one is excluded.

```
var baseArr = [10, 15, 20, 25];
var slicedArr = baseArr.slice(1,3); // [15, 20]
```

Figure 29: Slicing of arrays in Javascript language

Design Issue 8: Which operators are provided?

+ operator -> arr1 + arr2
== operator -> arr1 == arr2
=== operator -> arr1 === arr2
> operator -> arr1 > arr2
< operator -> arr1 < arr2

Discussion 1: From my point of view Dart is the best language among the others for array operations. The syntax of the language is similar to C based languages except the usage of “var” keyword. Also the language is more writable than C based languages in my opinion since the variable types are not statically determined but only the var keyword is used. So it makes it easy to construct arrays. Also array initialization can be done in Dart which is also increases both readability and writability. Dart only allows integer subscripts and I think it is more safer and increases reliability since there wont be any confusion about other data types for subscripts. Also range check is done and dynamic arrays are allowed in Dart which are the most important features in my view. In contrast with python which has a lot of operators for arrays, Dart has only a simple set of operators which may be decreases writability but it increases readability and reliability since this set of operators are only the necessary and simple ones. To conclude, even though some of the languages have similar kind of behaviours for array operations, how easy to use the language and how similar that the syntax of the language with C based languages affected my considerations of the use of that language in array operations and thats why I preferred Dart language.

Discussion 2: Learning a new programming language is itself a hard process. In this homework, the task was to be proficient at array operations in five different programming language so that we can demonstrate the required design issues about arrays. As a roadmap, I progressed 1 language per time and I tried to answer all questions about this programming language. First of all getting familiar with the environments of programming languages was a difficult part for me. I started with Dart language and I found an online code editor as an environment for testing the codes that I wrote. However after I was done with coding and everything was fine in the online editor, I started to encounter some problems in Dijkstra server. So I realized that the best practice for the coding part of this homework was to test the codes on the Dijkstra server directly after finishing each design issue in the language. That how I ease my work on debugging and making my code work on the Dijkstra server. Learning the basic syntaxes of each language was also a difficult process. First of all I was thinking to watch some tutorials for each language but then I realized that would take a lot of time and there was not enough time to complete the homework. So I decided to use the formal webpages, manuals, APIs, namely documentations of the languages. Their documentations was easy to understand but it was hard to find the correct information that I was looking for since those documentations are so big. Then I started to encounter some problems that does not included in the documentations such as which subscript types are legal for the languages. For those kinds of questions, it was also hard to find the solutions online. Therefore the only possible solutions for some design issues was to directly try those in the code and use the trial and error method. However, this is a difficult task since we try to run all those codes in Dijkstra server because in Dijkstra it is not possible to use an IDE other than the ones like VIM in the terminal. So each time I wanted to modify my code I needed to change the code on my machine using my own IDEs and then upload the files to the Dijkstra again and again. But overall, the most useful resources for me was the documentations of the languages. Also the APIs of the arrays of the languages was

extremely useful in terms of “Design Issue 8: Which operators are provided?” Because there is no better way to find all the array operators of a language other than checking the official website of the language. After I got proficient in the languages so that I could respond to the design issues in my codes, in which I tried to use comments as a reminder for me for the report of the homework, it was time to write the report. In the report part of the homework I basically expressed my findings about the design issues of all the languages that I experienced.

References

PHP array operators, https://www.php.net/manual/en/language.operators.array.php
PHP arrays, https://www.php.net/manual/en/language.types.array.php
Python arrays, https://data-apis.org/array-api/latest/API_specification/array_object.html
Dart Documentation, https://dart.dev/guides
Javascript documentation, https://devdocs.io/javascript/
Rust arrays, https://doc.rust-lang.org/rust-by-example/primitives/array.html
Sebesta, R. W. (2016). *Concepts of programming languages*. Pearson.