CS 315
Homework 1

Erhan Er

21801809

Section 3

Dart

1. What types are legal for subscripts?

Only integer values are legal. If the user enters other than integer, the compiler will give errors.

Example

```
var arr = [0, 1, 2, 3, 4];
var index = 2;
print("arr: $arr");

/* In the console, we will see the values at the specified indexes */
var value1 = arr[1];
var value2 = arr[index];

print("value1: $value1");
print("value2: $value2");

// The compiler will give an error for these accessing types
//var value3 = arr["a"];
//var value4 = arr[-1];
//var value5 = arr[true];
//var value6 = arr['a'];
```

Output

```
arr: [0, 1, 2, 3, 4]
value1: 1
value2: 2
```

2. Are subscripting expressions in element references range checked?

It checks the range from 0 to length of the array - 1. If the user enters a value bigger than or equal to length of the array or negative value, the compiler will give error.

Example

```
var rangeArr = [0, 1, 2, 3, 4, 5];
print("rangeArr: $rangeArr");

/* The compiler will give an index out of range error.
 * Therefore, Dart checks range. */
```

```
//var value7 = rangeArr[rangeArr.length + 1];
//var value8 = rangeArr[-2];
```

Output

```
rangeArr: [0, 1, 2, 3, 4, 5]
```

3. When are subscript ranges bound?

Dart allows adding new items to the arrays. Therefore, the ranges are bound in run time.

Example

```
var arr2 = [0, 1, 2, 3, 4, 5];
print("arr2: $arr2");

arr2.add(6);
arr2.add(7);

print("arr2: $arr2");
```

Output

```
arr2: [0, 1, 2, 3, 4, 5]
arr2: [0, 1, 2, 3, 4, 5, 6, 7]
```

4. When does allocation take place?

Compiler does not compile the whole program and gives an error. Therefore, allocation is done in compile time.

Example

```
/* Compiler gives an error and does not compile the program. **/
//print(allocationArr[1]);
var allocationArr = [0, 1, 2, 3, 4];
print(allocationArr[2]);
```

Output

```
allocationArr: [0, 1, 2, 3, 4]
2
```

5. Are ragged or rectangular multidimensional arrays allowed, or both?

Both of them are allowed in dart.

Example

```
// Ragged Array initialization
var raggedArr = [1, [2, 3], [4, 5, 6], [7, 8], 9];
print("raggedArr: $raggedArr");
var value9 = raggedArr[2];
var value10 = raggedArr[3];
print("raggedArr[2]: $value9");
print("raggedArr[3]: $value10");
var rectangularArr = [[1, 2], [3, 4], [5, 6]];
print("rectangularArr: $rectangularArr");
var value12 = rectangularArr[0][1];
var value13 = rectangularArr[2][1];
print("rectangularArr[0][1]: $value12");
print("rectangularArr[2][1]: $value13");
```

Output

```
raggedArr: [1, [2, 3], [4, 5, 6], [7, 8], 9]
raggedArr[2]: [4, 5, 6]
raggedArr[3]: [7, 8]
rectangularArr: [[1, 2], [3, 4], [5, 6]]
rectangularArr[0][1]: 2
rectangularArr[2][1]: 6
```

6. Can array objects be initialized?

Dart has native support for list comprehension.

Example

```
var objectArr = [for( var x = 0; x < 20; x++) x * 2 ];
print("objectArr: $objectArr");
var value11 = objectArr[5];
var value12 = objectArr[12];
print("objectArr: $value11");
print("objectArr: $value12");
```

Output

```
objectArr: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38]
objectArr: 10
objectArr: 24
```

7. Are any kind of slices supported?

Dart has a slice function “sublist(starting point, end point)” for arrays. It returns a new array and preserves the first function. If the user enters only one value, the function will slice the array up from the given index until the end of the array. If the user enters two values which are starting and end indexes, the function will slice the array from the index which is given first in the function call to the index which is given second in the function call. The function does not take negative indexes.

Example

```
var sliceArray = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
print("Slice Array: $sliceArray");

var slicedArray1 = sliceArray.sublist(5);
print("Sliced Array 1; Slice array is sliced from 5: $slicedArray1");

var slicedArray2 = sliceArray.sublist(2, 5);
print("Sliced Array 2; Slice array is sliced from 2 to 5: $slicedArray2");

// It does not allow us to enter negative values
//var slicedArray3 = sliceArray.sublist(2, -1);
//var slicedArray4 = sliceArray.sublist(-2);
print("Slice Array: $sliceArray");
```

Output

```
Slice Array: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Sliced Array 1; Slice array is sliced from 5: [5, 6, 7, 8, 9]
Sliced Array 2; Slice array is sliced from 2 to 5: [2, 3, 4]
Slice Array: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

8. Which operators are provided?

Dart provides “+”, “==”, “!=” operators. “+” operator merges two arrays and returns a new array. “==” operator

Example

```
var operationArray1 = [0, 1, 2, 3, 4];
var operationArray2 = [5, 6, 7, 8, 9];
var operationArray3 = [0, 1, 2, 3, 4];
var operationArray4 = [3, 1, 0, 2, 4];

var operation1 = operationArray1 + operationArray2;
print("operationArray1 + operation2: $operation1");
```

```
var operation2 = operationArray1 == operationArray2;
print("operationArray1 == operationArray2: $operation2");

var operation3 = operationArray1 != operationArray3;
print("operationArray1 != operationArray3: $operation3");
```

Output

```
operationArray1 + operation2: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
operationArray1 == operationArray2: false
operationArray1 != operationArray3: true
```

Javascript

1. What types are legal for subscripts?

Javascript allows every type for subscripts. However, other than integers, it will print undefined in the console and it also does not stop the program from execution.

Example

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

console.log("arr: " + arr);
/* In the console, we will see the values at the specified indexes */
console.log("arr[1]: " + arr[1]);
console.log("arr[index = 2]: " + arr[index]);

// In the console, we will see undefined for these subscriptings
console.log("arr[string]: " + arr["a"]);
console.log("arr[true]: " + arr[true]);
console.log("arr[char]: " + arr['a']);
```

Output

```
arr: 0,1,2,3,4,5
arr[1]: 1
arr[index = 2]: 2
arr[string]: undefined
arr[true]: undefined
arr[char]: undefined
```

2. Are subscripting expressions in element references range checked?

Javascript does not throw an error when the subscripting expression is not in the range. It only prints undefined to the console. Therefore, it does not check the range.

Example

```
var rangeArr = [0, 1, 2, 3, 4, 5];
console.log("rangeArr: " + rangeArr);
```

```
/* In the console, we will see undefined. Therefore, javascript does not check range */
console.log("rangeArr[rangeArr.length + 1]: " + rangeArr[rangeArr.length + 1]);
console.log("rangeArr[-1]: " + rangeArr[-1]);
```

Output

```
rangeArr: 0,1,2,3,4,5
rangeArr[rangeArr.length + 1]: undefined
rangeArr[-1]: undefined
```

3. When are subscript ranges bound?

Example EKLE

```
var arr2 = [0, 1, 2, 3, 4];
   console.log("arr2: " + arr2);

   arr2.push(5);
   arr2.push(6);

   console.log("arr2: " + arr2);
```

Output

```
arr2: 0,1,2,3,4
arr2: 0,1,2,3,4,5,6
```

4. When does allocation take place?

The code will work until the line where we are trying to access element of a not created array. Therefore, arrays are allocated in compile time.

Example

```
//console.log(allocationArr[1]); The code will work until this line.

var allocationArr = [0, 1, 2, 3, 4, 5];
console.log("allocationArr: " + allocationArr);
console.log("allocationArr[3]: " + allocationArr[3]);
```

Output

```
allocationArr: 0,1,2,3,4,5
allocationArr[3]: 3
```

5. Are ragged or rectangular multidimensional arrays allowed, or both?

In javascript, both types of arrays are allowed.

Example

```
// Ragged Array initialization
var raggedArr = [1, [2, 3], [4, 5, 6] , [7, 8], 9];

// Print whole array
console.log("Ragged Array: " + raggedArr);

// Get some values from array
console.log("raggedArr[2][1]: " + raggedArr[2][1]);
console.log("raggedArr[1][2]: " + raggedArr[1][1]);

// Rectangular Array initialization
var rectangularArr = [[1, 2], [3, 4], [5, 6]];

// Print whole array
console.log("Rectangular Array: " + rectangularArr);

// Get some values from array
console.log("rectangularArr[1][1]: " + rectangularArr[1][1]);
console.log("rectangularArr[0][1]: " + rectangularArr[0][1]);
```

Output

```
Ragged Array: 1,2,3,4,5,6,7,8,9
raggedArr[2][1]: 5
raggedArr[1][2]: 3
Rectangular Array: 1,2,3,4,5,6
rectangularArr[1][1]: 4
rectangularArr[0][1]: 2
```

6. Can array objects be initialized?

Javascript does not have native support for list comprehension.

7. Are any kind of slices supported?

In JavaScript, slice operation is supported. The function “slice(start, end)” will slice the specified array and create a new array object without breaking up the given array. Start and end points are optional. If the user does not specify a starting point, the function will slice the array from the beginning to the end. If the user enters a negative integer for the start point, the function will start from “array.length + given negative index” to the end of the array.

Example

```
var sliceArr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

console.log("Slice Array: " + sliceArr);

var slicedArr1 = sliceArr.slice();
console.log("Sliced Array 1: Slice Array is sliced without a starting point: " +
slicedArr1);

var slicedArr2 = sliceArr.slice(-3);
console.log("Sliced Array 2: Slice Array is sliced from -3: " + slicedArr2);

var slicedArr3 = sliceArr.slice(2, 5);
console.log("Sliced Array 3: Slice Array is sliced from 2 to 5: " + slicedArr3);

console.log("Slice Array: " + sliceArr);
```

Output

```
Slice Array: 0,1,2,3,4,5,6,7,8,9
Sliced Array 1: Slice Array is sliced without a starting point: 0,1,2,3,4,5,6,7,8,9
Sliced Array 2: Slice Array is sliced from -3: 7,8,9
Sliced Array 3: Slice Array is sliced from 2 to 5: 2,3,4
Slice Array: 0,1,2,3,4,5,6,7,8,9
```

8. Which operators are provided?

Javascript provides “+” ,”-”, “*”, “/”, “===”, “<”, “<=”, “>”, “>=” operators. However, “-”, “*”, “/” operators returns NaN. “+” operator returns a string that includes all of the elements in the array. “===” operator compares the address of the array in the memory. Therefore, if the pointers are different, operation will return false. “<”, “<=”, “>”, “>=” operations compare the arrays’ values. However, even if the arrays have only integer values, they compare the chars of these values. For example, if the first array starts with integer 23 and the second array starts with 100, “<”, “<=”, “>”, “>=” operations compare the first number of the first elements. Because the first array’s first element starts with 2 and the second array’s first element starts with 1, the operation tells 23 is bigger than 100.

Example

```
var operationArr1 = [0, 1, 2, 3, 4];
var operationArr2 = [1001, 6, 7, 8, 9];
var operationArr3 = [0, 1, 2, 3, 4];
var operationArr4 = operationArr1;
var operationArr5 = [21, 0, 0, 0, 0];

console.log("operationArr1: " + operationArr1);
console.log("operationArr2: " + operationArr2);
console.log("operationArr3: " + operationArr3);
console.log("operationArr4: " + operationArr4);
console.log("operationArr5: " + operationArr5);

var addedArr = operationArr1 + operationArr2;
var substractArr = operationArr1 - operationArr2;
var dividedArr = operationArr1 / operationArr2;
var multipliedArr = operationArr1 * operationArr2;

console.log("addedArr: " + addedArr);
console.log("Type of addedArr: " + typeof(addedArr));
console.log("------------------------------------------------------");

console.log("substractArr: " + substractArr);
console.log("Type of substractArr: " + typeof(substractArr));
console.log("------------------------------------------------------");

console.log("dividedArr: " + dividedArr);
console.log("Type of dividedArr: " + typeof(dividedArr));  console.log("-------------------
-----------------------------------");

console.log("multipliedArr: " + multipliedArr);
console.log("Type of multipliedArr: " + typeof(multipliedArr));  console.log("-------------
-----------------------------------------");

console.log("operationArr1 === operationArr3: " + (operationArr1 ===
operationArr3));   console.log("------------------------------------------------------");

console.log("operationArr1 !== operationArr4: " + (operationArr1 !== operationArr4));
console.log("------------------------------------------------------");

console.log("operationArr1 < operationArr5: " + (operationArr1 < operationArr5));
console.log("------------------------------------------------------");
console.log("operationArr5 > operationArr2: " + (operationArr5 > operationArr2));
console.log("---------------------------------------------------------");

console.log("operationArr5 >= operationArr2: " + (operationArr5 >= operationArr2));
console.log("---------------------------------------------------------");

console.log("operationArr1 < operationArr3: " + (operationArr1 <= operationArr3));
console.log("---------------------------------------------------------");
```

Output

```
operationArr1: 0,1,2,3,4
operationArr2: 1001,6,7,8,9
operationArr3: 0,1,2,3,4
operationArr4: 0,1,2,3,4
operationArr5: 21,0,0,0,0
addedArr: 0,1,2,3,41001,6,7,8,9
Type of addedArr: string
---------------------------------------------------------
substractArr: NaN
Type of substractArr: number
---------------------------------------------------------
dividedArr: NaN
Type of dividedArr: number
---------------------------------------------------------
multipliedArr: NaN
Type of multipliedArr: number
---------------------------------------------------------
operationArr1 === operationArr3: false
---------------------------------------------------------
operationArr1 !== operationArr4: false
---------------------------------------------------------
operationArr1 < operationArr5: true
---------------------------------------------------------
operationArr5 > operationArr2: true
---------------------------------------------------------
operationArr5 >= operationArr2: true
---------------------------------------------------------
operationArr1 < operationArr3: true
---------------------------------------------------------
```

PHP

1. What types are legal for subscripts?

Integers and booleans are allowed for subscripts. True values are considered as 1 and false values considered as 0.

Example

```
$arr1 = array(0, 1, 2, 3, 4);
$index = 1;

print "arr1: ";
var_dump($arr1);
/* In the console, we will see the values of the specified indexes */
echo "arr[0]: $arr1[0]\n";
echo "arr[index]: $arr1[$index]\n";
print "arr[true]: " . $arr1[true] . "\n";
print "arr[false]: " . $arr1[false] . "\n";

// In the console, compiler will give error
//print "5: " . $arr1["a"] . "\n";
//print "6: " . $arr1['a'] . "\n";
```

Output

```
arr1: array(5) {
  [0]=>
  int(0)
  [1]=>
  int(1)
  [2]=>
  int(2)
  [3]=>
  int(3)
  [4]=>
  int(4)
}
arr[0]: 0
arr[index]: 1
arr[true]: 1
arr[false]: 0
```

2. Are subscripting expressions in element references range checked?

PHP checks the range. It does not allow negative indexes and values bigger than or equal to length of the array.

Example

```
$rangeArr = array(0, 1, 2, 3, 4, 5);


print "rangeArr: ";
var_dump($rangeArr);
// The compiler will give error
//print "rangeArr[10]: " . $rangeArr[10] . "\n";
//print "rangeArr[-1]: " . $rangeArr[-1] . "\n";
```

Output

```
rangeArr: array(6) {
  [0]=>
  int(0)
  [1]=>
  int(1)
  [2]=>
  int(2)
  [3]=>
  int(3)
  [4]=>
  int(4)
  [5]=>
  int(5)
}
```

3. When are subscript ranges bound?

PHP allows adding new elements to an array. Therefore, the ranges are bound in run time.

Example

```
$arr2 = array("Bilkent", "University");
print "arr2: ";
var_dump($arr2);

array_push($arr2, "CS", 315);
print "arr2 after push operation: ";
```

```
var_dump($arr2);
```

Output

```
arr2: array(2) {
  [0]=>
  string(7) "Bilkent"
  [1]=>
  string(10) "University"
}
arr2 after push operation: array(4) {
  [0]=>
  string(7) "Bilkent"
  [1]=>
  string(10) "University"
  [2]=>
  string(2) "CS"
  [3]=>
  int(315)
}
```

4. When does allocation take place?

Example

```
//print "allocationArr[1]: " . $allocationArr[1] . "\n"; This will throw an error

$allocationArr = array(0, 1, 2, 3, 4, 5);
print "allocationArr: ";
var_dump($allocationArr);
print "allocationArr[3]: " . $allocationArr[3] . "\n";
```

Output

```
allocationArr: array(6) {
  [0]=>
  int(0)
  [1]=>
  int(1)
  [2]=>
  int(2)
  [3]=>
  int(3)
  [4]=>
  int(4)
  [5]=>
  int(5)
}
allocationArr[3]: 3
```

5. Are ragged or rectangular multidimensional arrays allowed, or both?

PHP allows both multidimensional arrays.

Example

```
// Ragged Array initialization
$raggedArr = array(1, array(2, 3), array(4, 5, 6) , array(7, 8), 9);

// Print whole array
print "Ragged Array: ";
var_dump($raggedArr);
print "------------ \n";

// Get some values from array
print "raggedArr[2][1]: " . $raggedArr[2][1] . "\n";
print "raggedArr[1][1]: " . $raggedArr[1][1] . "\n";

// Rectangular Array initialization
$rectangularArr = array( array(1, 2), array(3, 4), array(5, 6), array(7, 8) );

// Print whole array
print "Rectangular Array: ";
var_dump($rectangularArr);
print "----------------- \n";

// Get some values from array
print "rectangularArr[1][1]: " . $rectangularArr[1][1] . "\n";
print"rectangularArr[0][1]: " . $rectangularArr[0][1] . "\n";
```

Output

```
Ragged Array: array(5) {
  [0]=>
  int(1)
  [1]=>
  array(2) {
        [0]=>
        int(2)
		[1]=>
		int(3)
  }
  [2]=>
  array(3) {
		[0]=>
		int(4)
		[1]=>
		int(5)
		[2]=>
		int(6)
  }
  [3]=>
  array(2) {
		[0]=>
		int(7)
		[1]=>
		int(8)
  }
  [4]=>
  int(9)
}
------------
raggedArr[2][1]: 5
raggedArr[1][1]: 3
Rectangular Array: array(4) {
  [0]=>
  array(2) {
		[0]=>
		int(1)
		[1]=>
		int(2)
  }
  [1]=>
  array(2) {
		[0]=>
		int(3)
		[1]=>
		int(4)
  }
  [2]=>
  array(2) {
		[0]=>
		int(5)
```

```
        [1]=>
        int(6)
  }
  [3]=>
  array(2) {
        [0]=>
        int(7)
        [1]=>
        int(8)
  }
}
-----------------
rectangularArr[1][1]: 4
rectangularArr[0][1]: 2
```

6. Can array objects be initialized?

PHP does not have native support for list comprehension.

7. Are any kind of slices supported?

PHP supports slice operations with the function "array_slice(array, starting index, end index)". If the user enters a negative starting index, the function will slice the array from "length of array + negative index" to the end of the array. If the user enters positive starting point and negative end point, the function will slice the array from the starting index to "length of the array + negative index".

Example

```
$sliceArr = array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
print "Slice Array: ";
var_dump($sliceArr);
print "----------- \n";

$slicedArr1 = array_slice($sliceArr, 3);
print "Sliced Array 1: Slice Array is sliced from 3: ";
var_dump($slicedArr1);
print "------------------------------------------------------------ \n";

$slicedArr2 = array_slice($sliceArr, -3);
print "Sliced Array 2: Slice Array is sliced from -3: ";
var_dump($slicedArr2);
print "----------------------------------------- \n";

$slicedArr3 = array_slice($sliceArr, 2, 5);
```

```
print "Sliced Array 3: Slice Array is sliced from 2 to 5: ";
var_dump($slicedArr3);
print "--------------------------------------------- \n";

$slicedArr4 = array_slice($sliceArr, -2, 5);
print "Sliced Array 4: Slice Array is sliced from -2 to 5: ";
var_dump($slicedArr4);
print "--------------------------------------------- \n";

$slicedArr5 = array_slice($sliceArr, 2, -3);
print "Sliced Array 5: Slice Array is sliced from 2 to -3: ";
var_dump($slicedArr5);
print "--------------------------------------------- \n";

$slicedArr6 = array_slice($sliceArr, -2, -5);
print "Sliced Array 6: Slice Array is sliced from -2 to -5: ";
var_dump($slicedArr6);
print "--------------------------------------------- \n";

print "Slice Array: ";
var_dump($sliceArr);
print "----------- \n";
```

Output

```
Slice Array: array(10) {
  [0]=>
  int(0)
  [1]=>
  int(1)
  [2]=>
  int(2)
  [3]=>
  int(3)
  [4]=>
  int(4)
  [5]=>
  int(5)
  [6]=>
  int(6)
  [7]=>
  int(7)
  [8]=>
  int(8)
  [9]=>
  int(9)
}
-----------
Sliced Array 1: Slice Array is sliced from 3: array(7) {
  [0]=>
  int(3)
  [1]=>
  int(4)
  [2]=>
  int(5)
  [3]=>
  int(6)
  [4]=>
  int(7)
  [5]=>
  int(8)
  [6]=>
  int(9)
}
--------------------------------------------------------------
Sliced Array 2: Slice Array is sliced from -3: array(3) {
  [0]=>
  int(7)
  [1]=>
  int(8)
  [2]=>
  int(9)
}
------------------------------------------
Sliced Array 3: Slice Array is sliced from 2 to 5: array(5) {
  [0]=>
  int(2)
  [1]=>
  int(3)
  [2]=>
  int(4)
  [3]=>
  int(5)
  [4]=>
  int(6)
}
----------------------------------------------
Sliced Array 4: Slice Array is sliced from -2 to 5: array(2) {
  [0]=>
  int(8)
  [1]=>
  int(9)
}
---------------------------------------------
Sliced Array 5: Slice Array is sliced from 2 to -3: array(5) {
  [0]=>
  int(2)
  [1]=>
  int(3)
  [2]=>
  int(4)
  [3]=>
  int(5)
  [4]=>
  int(6)
}
---------------------------------------------
Sliced Array 6: Slice Array is sliced from -2 to -5: array(0) {
}
---------------------------------------------
Slice Array: array(10) {
  [0]=>
  int(0)
  [1]=>
  int(1)
  [2]=>
  int(2)
  [3]=>
  int(3)
  [4]=>
  int(4)
  [5]=>
  int(5)
  [6]=>
  int(6)
  [7]=>
  int(7)
  [8]=>
  int(8)
  [9]=>
  int(9)
}
```

----

8. Which operators are provided?

PHP provides “+”, “==”, “===”, “!=”, “!==”, “<>” operations. “+” operation adds to arrays and creates a new array from the given two arrays. It will take all of the first given array and check whether the second given array has different keys. If the second array has different keys, it adds these values to the new array. “==” operation checks whether given arrays’ key/value pairs match even if the order of elements are different. “===” operation checks whether given arrays key/value pairs match and in the same order. “!=” and “!==” operations are the opposite of “==” and “===” respectively. “<>” operation works the same as “!=” operation. However, if a key is string 0, PHP works with it in integer form. Therefore, if the key of an element in the array 1 is integer 0 and the other’s in the array 2 is string 0, “==”, “!=” and “<>” operators think that these keys are the same.

Example

```
$operationArr1 = array( "a" => 1, "b" => 2, "c" => 3, "d" => 4, "e" => 5 );
$operationArr2 = array( "e" => 5, "d" => 4, "c" => 3, "b" => 2, "a" => 1 );
$operationArr3 = array( "a" => 1, "b" => 2, "c" => 3, "d" => 4, "e" => 5 );
$operationArr4 = array( "f" => 5, "g" => 4, "h" => 3, "i" => 2, "j" => 1 );
$operationArr5 = $operationArr1;

$operation1 = $operationArr1 + $operationArr4;
print "operationArr1 + operationArr4: ";
var_dump($operation1);

$operation2 = $operationArr1 + $operationArr2;
print "operationArr1 + operationArr2";
var_dump($operation2);

$operation3 = $operationArr1 == $operationArr2;
print "operationArr1 == operationArr2: ";
var_dump($operation3);

$operation4 = $operationArr1 === $operationArr2;
print "operationArr1 === operationArr2: ";
var_dump($operation4);

$operation5 = $operationArr1 === $operationArr3;
print "operationArr1 === operationArr3: ";
var_dump($operation5);
```

```
$operation6 = $operationArr1 != $operationArr4;
print "operationArr1 != operationArr4: ";
var_dump($operation6);

$operation7 = $operationArr1 <> $operationArr4;
print "operationArr1 <> operationArr4: ";
var_dump($operation7);

$operation8 = $operationArr1 !== $operationArr2;
print "operationArr1 !== operationArr2: ";
var_dump($operation8);

$operation9 = $operationArr1 <> $operationArr5;
print "operationArr1 <> operationArr5";
var_dump($operation9);
```

Output

```
operationArr1 + operationArr4: array(10) {
  ["a"]=>
  int(1)
  ["b"]=>
  int(2)
  ["c"]=>
  int(3)
  ["d"]=>
  int(4)
  ["e"]=>
  int(5)
  ["f"]=>
  int(5)
  ["g"]=>
  int(4)
  ["h"]=>
  int(3)
  ["i"]=>
  int(2)
  ["j"]=>
  int(1)
}
operationArr1 + operationArr2array(5) {
  ["a"]=>
  int(1)
  ["b"]=>
  int(2)
  ["c"]=>
  int(3)
  ["d"]=>
  int(4)
  ["e"]=>
  int(5)
}
operationArr1 == operationArr2: bool(true)
operationArr1 === operationArr2: bool(false)
operationArr1 === operationArr3: bool(true)
operationArr1 != operationArr4: bool(true)
operationArr1 <> operationArr4: bool(true)
operationArr1 !== operationArr2: bool(true)
operationArr1 <> operationArr5bool(false)
```

Python

1. What types are legal for subscripts?

Only integer values are allowed. Negative indexes start from the end of the array. For example, index = -1 is the value at the end of the array.

Example

```
arr = np.array([0, 1, 2, 3, 4, 5])
index = 1

print("arr:", arr)
print("arr[3]:", arr[3])
print("arr[index]:", arr[index])
print("arr[-1]:", arr[-1])
#print("arr[true]:", arr[true]) Compile time error
#print("arr[a]:", arr["a"]) Compile time error
#print("arr[char]:", arr['a']) Compile time error
```

Output

```
arr: [0 1 2 3 4 5]
arr[3]: 3
arr[index]: 1
arr[-1]: 5
```

2. Are subscripting expressions in element references range checked?

Range is checked. The maximum index is “length of the array - 1” and the minimum index is “- length of the array”.

Example

```
rangeArr = np.array([0, 1, 2, 3, 4, 5])

print("rangeArr:", rangeArr)
#print("rangeArr[6]:", rangeArr[6]) Out of bounds
#print("rangeArr[-7]:", rangeArr[-7]) Out of bounds

print("rangeArr[-6]:", rangeArr[-6])
print("rangeArr[5]:", rangeArr[5])
#Therefore, it accepts integer values from -array.size to array.size - 1
```

Output

```
rangeArr: [0 1 2 3 4 5]
rangeArr[-6]: 0
rangeArr[5]: 5
```

3. When are subscript ranges bound?

Ranges are bound in compile time because the array size cannot be changed.

Example

```
addArr = np.array([0, 1, 2, 3, 4])
addArr2 = np.array([5, 6, 7, 8, 9])

print("addArr:", addArr)
print("addArr2:", addArr2)

addArr += 6
print("addArr:", addArr) #it does not add a new element to the array
addArr3 = np.insert(addArr, addArr.size, 12) #it does not add a new element to the
original array
print("addArr3:", addArr3)

addArr4 = np.concatenate((addArr, addArr2)) #it does not add a new elemet to the
original array
print("addArr4:", addArr4)

addArr2.resize(10, refcheck=False); #The only way to change the array size
#However, it deletes the original one and creates a new array.
addArr2[7] = 7;
print("addArr2: ", addArr2);
```

Output

```
addArr: [0 1 2 3 4]
addArr2: [5 6 7 8 9]
addArr: [ 6  7  8  9 10]
addArr3: [ 6  7  8  9 10 12]
addArr4: [ 6  7  8  9 10  5  6  7  8  9]
addArr2:  [5 6 7 8 9 0 0 7 0 0]
```

4. When does allocation take place?

Allocation takes place in run time.

Example

```
#Program executes until this line.
#print("allocationArr[1]:", allocationArr[1])

allocationArr = np.array([0, 1, 2, 3, 4])
print("allocationArr:", allocationArr)
print("allocationArr[2]:", allocationArr[2])
```

Output

```
allocationArr: [0 1 2 3 4]
allocationArr[2]: 2
```

5. Are ragged or rectangular multidimensional arrays allowed, or both?

Both of them can be created. However, ragged arrays are not supported by numpy natively as can be seen in the print part. Rectangular array is printed row by row or array by array. On the other hand, the ragged array is printed directly. If the notation “dtype=object” was deleted, the compiler would give a warning.

Example

```
raggedArr = np.array([0, np.array([1, 2]), np.array([3, 4, 5]), np.array([6, 7]), 8],
dtype=object)
print("raggedArr:", raggedArr)
print("raggedArr[1][1]:", raggedArr[1][1])
print("raggedArr[2][2]:", raggedArr[2][2])

rectangularArr = np.array([[0, 1], [2, 3], [4, 5]])
print("rectangularArr:", rectangularArr)
print("rectangularArr[1][0]:", rectangularArr[1][1])
print("rectangularArr[2][1]:", rectangularArr[2][1])
```

Output

```
raggedArr: [0 array([1, 2]) array([3, 4, 5]) array([6, 7]) 8]
raggedArr[1][1]: 2
raggedArr[2][2]: 5
rectangularArr: [[0 1]
 [2 3]
 [4 5]]
rectangularArr[1][0]: 3
rectangularArr[2][1]: 5
```

6. Can array objects be initialized?

Python has native support for list comprehension.

Example

```
objectArr = [x for x in range(100) if x % 5 == 0]
print("objectArr: ", objectArr)
print("objectArr[3]: ", objectArr[3])
print("objectArr[10]", objectArr[10])
```

Output

```
objectArr:  [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
objectArr[3]:  15
objectArr[10] 50
```

7. Are any kind of slices supported?

Slice is supported with the notation “[start:end:step]”. All of the points are optional.

Example

```
sliceArr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print("sliceArr: ", sliceArr)
# There are two ways of doing this
# First, slicing the array and putting it in an another variable
# Or directly print in it
# I used the first one for only the first example
# Others will be in second form.
slicedArr1 = sliceArr[2:]
print("slicedArr1: ", slicedArr1)
print("slicedArr2: ", sliceArr[2::2])
print("slicedArr3: ", sliceArr[:4])
print("slicedArr4: ", sliceArr[2:5])
print("slicedArr5: ", sliceArr[1:8:2])
print("slicedArr6: ", sliceArr[9:0:-3])
print("slicedArr7: ", sliceArr[-3:])
print("slicedArr8: ", sliceArr[-6:9])
print("slicedArr9: ", sliceArr[-8:-2])
print("slicedArr10: ", sliceArr[-1:-8:-2])

print("sliceArr: ", sliceArr)
```

Output

```
sliceArr:  [0 1 2 3 4 5 6 7 8 9]
slicedArr1:  [2 3 4 5 6 7 8 9]
slicedArr2:  [2 4 6 8]
slicedArr3:  [0 1 2 3]
slicedArr4:  [2 3 4]
slicedArr5:  [1 3 5 7]
slicedArr6:  [9 6 3]
slicedArr7:  [7 8 9]
slicedArr8:  [4 5 6 7 8]
slicedArr9:  [2 3 4 5 6 7]
slicedArr10:  [9 7 5 3]
sliceArr:  [0 1 2 3 4 5 6 7 8 9]
```

8. Which operators are provided?

All arithmetic operators, relational operators, equality operators and bitwise operators are allowed. “+” operator adds two arrays’ elements index by index. “-” operator subtracts two arrays’ elements index by index. “*” operator multiplies two arrays’ elements index by index. “/” operator divides two arrays’ elements index by index. “%” operator returns the remainder of the division of two arrays’ elements index by index. “^” and “**” operators return power of first arrays’ elements with respect to the second arrays’ elements. “+++...” operator works the same as the “+” operator. “---...” operator works the same as the “-” operator. However, if the number of “-” is even, it will work the same as the ”+” operator. “<”, “<=”, “>”, “>=”, “==” and “‘!=” operators compare elements in the same index of the two arrays. “&” and “|” operators do the bitwise operations.

Example

```
# <, <=, >, >=, ==, !=, +, -, *, /, %, |, &, ^, **, also (+++...) and (---...) works
operationArr1 = np.array([1, 2, 3, 4, 5])
operationArr2 = np.array([6, 7, 8, 9, 10])
operationArr3 = np.array([5, 2, 4, 3, 1])
operationArr4 = operationArr1
operationArr5 = np.array([0, 1, 2, 3, 4, 5, 6, 7]);

#operation = operationArr5 + operationArr1 It is not allowed

print("operationArr1 + operationArr2: ",operationArr1 + operationArr2)
print("operationArr1 - operationArr2: ",operationArr1 - operationArr2)
print("operationArr1 * operationArr2: ",operationArr1 * operationArr2)
print("operationArr4 / operationArr1: ",operationArr4 / operationArr1)
print("operationArr2 % operationArr1: ",operationArr2 % operationArr1)
print("operationArr2 ^ operationArr1: ",operationArr2 ^ operationArr1)
print("operationArr3 ** operationArr1: ",operationArr3 ** operationArr1)
print("operationArr1 ++ operationArr2: ",operationArr1 ++ operationArr2)
print("operationArr1 +++ operationArr2: ",operationArr1 +++ operationArr2)
print("operationArr1 -- operationArr2: ",operationArr1 -- operationArr2)
print("operationArr1 --- operationArr2: ",operationArr1 --- operationArr2)
print("operationArr1 < operationArr2: ",operationArr1 < operationArr2)
print("operationArr1 <= operationArr3: ",operationArr1 <= operationArr3)
print("operationArr1 > operationArr2: ",operationArr1 > operationArr2)
print("operationArr1 >= operationArr3: ",operationArr1 >= operationArr3)
print("operationArr1 == operationArr4: ",operationArr1 == operationArr4)
print("operationArr1 != operationArr3: ",operationArr1 != operationArr3)
print("operationArr1 | operationArr3: ",operationArr1 | operationArr3)
print("operationArr1 & operationArr3: ",operationArr1 & operationArr3)
```

Output

```
operationArr1 + operationArr2:  [ 7  9 11 13 15]
operationArr1 - operationArr2:  [-5 -5 -5 -5 -5]
operationArr1 * operationArr2:  [ 6 14 24 36 50]
operationArr4 / operationArr1:  [1. 1. 1. 1. 1.]
operationArr2 % operationArr1:  [0 1 2 1 0]
operationArr2 ^ operationArr1:  [ 7  5 11 13 15]
operationArr3 ** operationArr1:  [ 5  4 64 81  1]
operationArr1 ++ operationArr2:  [ 7  9 11 13 15]
operationArr1 +++ operationArr2:  [ 7  9 11 13 15]
operationArr1 -- operationArr2:  [ 7  9 11 13 15]
operationArr1 --- operationArr2:  [-5 -5 -5 -5 -5]
operationArr1 < operationArr2:  [ True  True  True  True  True]
operationArr1 <= operationArr3:  [ True  True  True False False]
operationArr1 > operationArr2:  [False False False False False]
operationArr1 >= operationArr3:  [False  True False  True  True]
operationArr1 == operationArr4:  [ True  True  True  True  True]
operationArr1 != operationArr3:  [ True False  True  True  True]
operationArr1 | operationArr3:  [5 2 7 7 5]
operationArr1 & operationArr3:  [1 2 0 0 1]
```

Rust

1. What types are legal for subscripts?

Only integer values are allowed. Other types will cause compiler error.

Example

```
let arr1 = [0, 1, 2, 3, 4];
let a = 2;
print!("arr1: ");
for x in arr1 {
        print!("{} ", x);
}
println!();
println!("arr1[1]: {} ", arr1[1]);
println!("arr1[a]: {} ", arr1[a]);
//print!("arr1[true]: {} ", arr1[true]);
//print!("arr1[string]: {}", arr1["a"]);
//print!(“arr1[char]: {}”, arr1[‘a’]);
```

Output

```
arr1: 0 1 2 3 4
arr1[1]: 1
arr1[a]: 2
```

2. Are subscripting expressions in element references range checked?

Range is checked. If the user enters a value for subscripting expression bigger than “length of the array - 1” or negative values, the compiler will not compile the program.

Example

```
let range_arr = [0, 1, 2, 3, 4, 5];
print!("range_arr: ");
for x in range_arr {
        print!("{} ", x);
}
println!();
println!("range_arr[6]: {}", range_arr[5]);
println!("range_arr[0]: {}", range_arr[0]);
//print!("range_arr[-1]: {}", range_arr[-1]); This will cause error
```

```
//print!("range_arr[6]: {}", range_arr[6]); This will cause error
```

Output

```
range_arr: 0 1 2 3 4 5
range_arr[6]: 5
range_arr[0]: 0
```

3. When are subscript ranges bound?

Subscript ranges are bound in compile time. The user cannot add, remove or resize arrays. Because in array initialization, the constructor only accepts lengths which are non-negative compile time constants.

Example

```
let arr2 = [0, 1, 2, 3, 4];
print!("arr2: ");
for x in arr2 {
        print!("{} ", x);
}
println!();
//arr2.push(5); It is not allowed
//arr2.resize(10); It is not allowed
```

Output

```
arr2: 0 1 2 3 4
```

4. When does allocation take place?

Allocation takes place in compile time. Because, if the user tries to access an element of a non-existing array, the compiler does not compile the program.

Example

```
/* println!("allocation_arr[1]: {} ", allocation_arr[1]);
 * Because of this, program does not compile */
let allocation_arr = [0, 1, 2, 3, 4, 5];
println!("allocation_arr[1]: {} ", allocation_arr[1]);
```

Output

```
allocation_arr[1]: 1
```

5. Are ragged or rectangular multidimensional arrays allowed, or both?

Only rectangular arrays are allowed. Because, by my observations, arrays only expect integers. If it encounters other types, it gives a compiler time error.

Example

```
let rectangular_arr = [[2; 6]; 6];
println!(“rectangular_arr: ”);
for x in rectangular_arr {
        for y in x {
        print!("{} ", y);
        }
        println!();
}
```

Output

```
rectangular_arr:
2 2 2 2 2 2
2 2 2 2 2 2
2 2 2 2 2 2
2 2 2 2 2 2
2 2 2 2 2 2
2 2 2 2 2 2
```

6. Can array objects be initialized?

Rust does not have native support for list comprehension.

7. Are any kind of slices supported?

Slice operations can be done with “[start..=end]” notation. If the user enters “[2..]”, the array will be sliced from index two to the end. If the user enters “[2..4]”, the array will be sliced from index two to index four, but it does not take the value at index four. If the user enters “[2..=4]”, the array will be sliced from index two to index four and it also includes the value at index four. Slice operation does not take negative values. It will cause a compile time error.

Example

```
let slice_arr = [0, 1, 2, 3, 4, 5];
print!("slice_arr: ");
for x in slice_arr {
        print!("{} ", x);
}
println!();

let sliced_arr1 = &slice_arr[2..4];
print!("sliced_arr1: ");
for x in sliced_arr1 {
        print!("{} ", x);
}
println!();

let sliced_arr2 = &slice_arr[1..=4];
print!("sliced_arr2: ");
for x in sliced_arr2 {
        print!("{} ", x);
}
println!();

let sliced_arr3 = &slice_arr[3..];
print!("sliced_arr3: ");
for x in sliced_arr3 {
        print!("{} ", x);
}
println!();
//let sliced_arr3 = &slice_arr[-1..3]; This will cause error
```

Output

```
slice_arr: 0 1 2 3 4 5
sliced_arr1: 2 3
sliced_arr2: 1 2 3 4
sliced_arr3: 3 4 5
```

8. Which operators are provided?

“==”, “!=”, “>”, “>=”, “<”, “<=” operators are allowed. “==”, “!=” operators will check whether the given arrays have the elements in the same order. “>”, “>=”, “<”, “<=” operators will check the relation between the elements of two given arrays. They will check the elements one by one in integer form. If they see a bigger element in the first array, “>”, “>=” will return true and “<”, “<=” will return false. If they see a smaller element in the first array, “>”, “>=” will return false and “<”, “<=” will return true.

Example

```
let operation_arr1 = [0, 1, 2, 3, 4];
let operation_arr2 = [100, 6, 7, 8, 9];
let operation_arr3 = [0, 1, 2, 3, 4];
let operation_arr4 = operation_arr1;
let operation_arr5 = [21, 0, 0, 0, 0];
let operation_arr6 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
print!("operation_arr1: ");
for x in operation_arr1 {
        print!("{} ", x);
}
println!();
print!("operation_arr2: ");
for x in operation_arr2 {
        print!("{} ", x);
}
println!();
print!("operation_arr3: ");
for x in operation_arr3 {
        print!("{} ", x);
}
println!();
print!("operation_arr4: ");
for x in operation_arr4 {
        print!("{} ", x);
}
println!();
print!("operation_arr5: ");
for x in operation_arr5 {
        print!("{} ", x);
}
println!();
print!("operation_arr6: ");
for x in operation_arr6 {
        print!("{} ", x);
}
println!();
println!("operation_arr1 == operation_arr2: {}", operation_arr1 == operation_arr2);
println!("operation_arr1 == operation_arr3: {}", operation_arr1 == operation_arr3);
println!("operation_arr1 == operation_arr5: {}", operation_arr1 == operation_arr5);
println!("operation_arr1 != operation_arr4: {}", operation_arr1 != operation_arr4);
println!("operation_arr2 > operation_arr5: {}", operation_arr2 > operation_arr5);
/* println!("operation_arr6 > operation_arr2: {}", operation_arr6 > operation_arr2)
 * It only accepts arrays that have the same size */
println!("operation_arr1 < operation_arr5: {}", operation_arr1 < operation_arr5);
println!("operation_arr1 <= operation_arr3: {}", operation_arr1 <= operation_arr3);
```

```
println!("operation_arr1 >= operation_arr3: {}", operation_arr1 >= operation_arr2);
```

Output

```
operation_arr1: 0 1 2 3 4
operation_arr2: 100 6 7 8 9
operation_arr3: 0 1 2 3 4
operation_arr4: 0 1 2 3 4
operation_arr5: 21 0 0 0 0
operation_arr6: 0 1 2 3 4 5 6 7 8 9
operation_arr1 == operation_arr2: false
operation_arr1 == operation_arr3: true
operation_arr1 == operation_arr5: false
operation_arr1 != operation_arr4: false
operation_arr2 > operation_arr5: true
operation_arr1 < operation_arr5: true
operation_arr1 <= operation_arr3: true
operation_arr1 >= operation_arr3: false
```

Best Language for Array Operations

Operation wise, Python has much more operation chances than other languages. In PHP, operations sometimes do not work as expected. In Javascript, most of the operations break the structure of the array. In Rust and Dart, we have a limited amount of operations. Slice operation wise, Python has many more different options with positive or negative start,end and step values. PHP has similar options, but it does not have step value. Rust also has similar options, but it does not have step value. In addition, Javascript and Dart also have similar options. However, they also do not have step options. Adding new elements to the array or removing elements from the array wise, Dart, Javascript and PHP are better since they allow the user to add new elements to the array or remove elements from the array. However, Python and Rust do not have this option. Creating big arrays faster wise, Python and Dart are better because they have simple list comprehensions. However, Javascript, PHP and Rust do not have list comprehensions. Therefore, among these criteria, Python is the best option for array operations.

Learning Strategy

When I was doing this homework, I experimented with most of the questions. Especially for question 8, I tried all of the operators by myself. I started with Javascript. I already knew how Javascript arrays work. Therefore, I only looked for the slice function [5]. In addition, I mainly used Firefox to check my code and also tried it on Chrome and Microsoft Edge. Then, I moved on to Dart. I did not know Dart. Therefore, I had to look for basic instructions such as print and variable initialization [2]. Then I worked on documentation of Dart [3]. I did not check every detail to directly learn everything but I experimented with some questions. Such as multidimensional arrays. Then I finally checked the whole operator list of Dart and tried some of them. Then I tried to understand what and how they were doing. I tried these codes on an online compiler [1]. After I finished most of the code, I tried it on dijkstra. Then, I moved on to Python. I read some parts of the documentation [11],[12]. I tried to add new elements to an array in different ways. However, I could not find a proper way to do it. Then, I researched how to change the size of an array and add new elements to an array [11], [13]. Then, I researched list comprehension [14]. When I reached question 7, I just read the parameters of the slice function. Then, before I wrote the code, I tried to think of different situations and their results. After that, I coded these different situations and read their results. I wrote code of Python in an online compiler [10]. Then I tried it on dijkstra. After that, I moved on to question 8. I tried all of the operators. Then, I tried to come up with an analysis of what they are doing by looking at the results. After that, I read the documentation of PHP [7][8][9]. I came up with different ideas when I was working on questions. I almost tried all of them by myself. In question 8, I tried the operators that I saw in the documentation. However, when I was checking the results, I realised that these operators do not work the same as written in the documentation. Therefore, I tried to understand exactly what they were doing. It was hard because they worked differently than I thought almost all the time. But in the end, I found out how they were working. I wrote PHP code on an online compiler[9]. Then, I tried it on dijkstra. Rust was the most difficult language for me. I could not really understand its documentation [15][16][17][19]. It was hard to write and understand the code at the beginning. When I got used to it, I started working on it. I mostly tried codes by myself because of the documentation. I wrote Rust code on an online compiler [18]. Then, I tried it on dijkstra.

References

[1] “Online Dart compiler”. https://dartpad.dev/?null_safety=true. [Accessed: Nov 22, 2021].
[2] “Print”. https://sites.google.com/site/dartlangexamples/api/dart-core/functions/print. [Accessed: Nov 22, 2021].
[3] “Lists”. https://dart.dev/guides/language/language-tour#lists. [Accessed: Nov 22, 2021].
[4] “Dart Operators”. https://www.w3adda.com/dart-tutorial/dart-operators. [Accessed: Nov 22, 2021].
[5] “Array.prototype.slice()”. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice. [Accessed: Nov 21, 2021].
[6] “Arrays”. https://javascript.info/array. [Accessed: Nov 21, 2021].
[7] “Arrays Manual”. https://www.php.net/manual/en/language.types.array.php. [Accessed: Nov 24, 2021].
[8] “Array Operators Manual”. https://www.php.net/manual/en/language.operators.array.php. [Accessed: Nov 24, 2021].
[9] “PHP Tryit Editor v1.2”. https://www.w3schools.com/php/phptryit.asp?filename=tryphp_compiler. [Accessed: Nov 24, 2021].
[10] “Online Python Compiler”. https://www.programiz.com/python-programming/online-compiler/. [Accessed: Nov 23, 2021].
[11] “the absolute basics for beginners - NumPy v1.21 Manual”. https://numpy.org/doc/stable/user/absolute_beginners.html#adding-removing-and-sorting-elements. [Accessed: Nov 23, 2021].
[12] “NumPy Array Slicing”. https://www.w3schools.com/python/numpy/numpy_array_slicing.asp. [Accessed: Nov 23, 2021].
[13] “Change the size of a numpy array in Python”. https://www.codespeedy.com/change-the-size-of-a-numpy-array-in-python/. [Accessed: Nov 23, 2021].
[14] Python - List Comprehension“”. https://www.w3schools.com/python/python_lists_comprehension.asp. [Accessed: Nov 23, 2021].
[15] “array - Rust”. https://doc.rust-lang.org/std/primitive.array.html. [Accessed: Nov 25, 2021].
[16] “Arrays and Slices - Rust By Example”. https://doc.rust-lang.org/rust-by-example/primitives/array.html. [Accessed: Nov 25, 2021].
[17] “Rust Variables”. https://www.w3adda.com/rust-tutorial/rust-variables. [Accessed: Nov 25, 2021].
[18] “Rust Playground”. https://play.rust-lang.org/. [Accessed: Nov 25, 2021].

[19] “Arrays and Slices in Rust”. https://dev-notes.eu/2019/07/arrays-slices-in-rust/. [Accessed: Nov 25, 2021].