Bilkent University

CS 315- PROGRAMMING LANGUAGES

Ali Eren Günaltılı

21801897

Section 01

CS315- Homework 01

Topic: Arrays in Dart, Javascript, PHP, Python, and Rust

Javascript

1. What types are legal for subscripts?

Javascript arrays are non-homogenous. They can hold any type of objects. In javascript sucbcript operator only allowed to use int values. In javascript there is no specific type as integer unlike C type languages like C++ or Java. JavaScript does not define different types of numbers, like integers, short, long, floating-point etc. JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard. But although javascript stores the integer and double values similar, it only allows to put integer between brackets.

let arr = [“apple”, 6.7, 16, true];

console.log(arr[1]); //print 6.7

console.log(arr[1.3]); // print undefined

console.log(arr[“abc”]); // print undefined

Example: Incorrect Array Index

```
var stringArray = new Array();

stringArray["one"] = "one";
stringArray["two"] = "two";
stringArray["three"] = "three";
stringArray["four"] = "four";
```

In Javascript, non-associative arrays don’t allow string or another type for subcript operator. It won’t give syntax error or the program won’t crash but it prints undefined. In javascript ‘[]’ is used as subscript operator.

```
let array = [1,2,3,4,5]; // directly initialized
let str = "abc";
console.log(array[1]);
console.log(array[str]); // print undefined
console.log(array[3.2]); // print undefined
```

2. Are subscripting expressions in element references range checked?

It is reference checked but it won't cause a crash on program. Instead it prints undefined.

```
let array = [1,2,3,4,5]; // directly initialized
console.log(array[array.length]); //it prints undefined
let jmp = array[array.length]; // it prints []
//jmp becomes empty array after this assignment statement.
```

3. When are subscript ranges bound?

Javascript uses dynamic type binding. After the array is declared, it's size can be changed, or new elements can be inserted to array. Javascript supports heap dynamic arrays.

4. When does allocation take place?

Allocation takes place in run-time. JavaScript automatically allocates memory when objects are created and frees it when they are not used anymore (garbage collection). As arrays can grow in run-time and may need more memory cells after their declaration javascript allocates memory when it is needed. Therefore, it makes allocation in run-time.

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

As arrays are object and they can able to hold object. Any type of array is allowed in Javascript. TwoD arrays are created similar to C++.

```
let tworect = new Array(5);
console.log("Initial state of 2D array");
console.log(tworect); //it prints <5 empty item>
//rectArr function. creating rect array 5*5 and initialize it.
const rectArr =(arr) => {
    for(let i = 0; i < 5; i++){
        arr[i] = new Array(5); //every index of array points another
      //memory cell which are defined as array with size 5.
    }
    //filling the array
    for(let i = 0; i < 5; i++){
        for(let j = 0; j < 5; j++){
            arr[i][j] = (i + 1) * 10 + j;
        }
    }
    return arr;
```

```
}
let newTmp = rectArr(tworect);//rectArr fills the inside of createdTwoD array.
console.log("After filling up the 5x5 Array");
console.log(newTmp);
//output
// After filling up the 5x5 Array
[
  [ 10, 11, 12, 13, 14 ],
  [ 20, 21, 22, 23, 24 ],
  [ 30, 31, 32, 33, 34 ],
  [ 40, 41, 42, 43, 44 ],
  [ 50, 51, 52, 53, 54 ]
]
```

As javascript prints undefined when we create with let tworect = new Array(5); We can say that it won't directly initialize values of an array instead it keeps empty or garbage. We can create ragged array by using same property of javascript. We just have to write different value for every indices in array instead of saying arr[i] = new Array(5);.

```
let raggedArr = new Array(5);
ragArr(raggedArr);
console.log("----------Ragged Array--------------");
//ragged array initializer method
var ragArr = (arr) => {
    for(let i = 4; i >= 0; i--){
        arr[i] = new Array(i); //difference between rect array
    }

    for(let i = 0; i < 5; i++){
        for(let k = 0; k <= i; k++){
            arr[i][k] = (i + 1) * 10 + k;
        }
    }
    return 0;
}
console.table(raggedArr);
//output
----------Ragged Array--------------
┌─────────┬────┬────┬────┬────┬────┐
│ (index) │ 0  │ 1  │ 2  │ 3  │ 4  │
├─────────┼────┼────┼────┼────┼────┤
│    0    │ 10 │    │    │    │    │
│    1    │ 20 │ 21 │    │    │    │
│    2    │ 30 │ 31 │ 32 │    │    │
│    3    │ 40 │ 41 │ 42 │ 43 │    │
│    4    │ 50 │ 51 │ 52 │ 53 │ 54 │
└─────────┴────┴────┴────┴────┴────┘
```

Ragged and multi dimensional array can be directly created without using new keyword.

```
console.log("-----------TwoD Without New Keyword --------");
let twoD_2 = [[1,2,3], [4,5,6], [7,8,9], [10,11,'old']];
console.table(twoD_2);
//output
-----------TwoD Without New Keyword --------
┌─────────┬────┬────┬─────────┐
│ (index) │ 0  │ 1  │    2    │
├─────────┼────┼────┼─────────┤
│    0    │ 1  │ 2  │    3    │
│    1    │ 4  │ 5  │    6    │
│    2    │ 7  │ 8  │    9    │
│    3    │ 10 │ 11 │ 'old'   │
└─────────┴────┴────┴─────────┘
console.log("-----------Ragged Without New Keyword--------");
let ragged_2 = [[1,2,3,4],[5,6], [true], ['a', 'b', 'c', 'd', 'e']];
console.table(ragged_2);
//output
-----------Ragged Without New Keyword--------
┌─────────┬─────────┬─────┬─────┬─────┬─────┐
│ (index) │    0    │  1  │  2  │  3  │  4  │
├─────────┼─────────┼─────┼─────┼─────┼─────┤
│    0    │    1    │  2  │  3  │  4  │     │
│    1    │    5    │  6  │     │     │     │
│    2    │  true   │     │     │     │     │
│    3    │  'a'    │ 'b' │ 'c' │ 'd' │ 'e' │
└─────────┴─────────┴─────┴─────┴─────┴─────┘
```

6. Can array objects be initialized?

Array objects can be initialized once they are created if they are created without using new keyword. If they are created with new keyword, they are not initialized and if array is printed it prints <n empty items>.

```
let tworect = new Array(5);
console.log("Initial state of 2D array");
console.log(tworect); // print <5 empty items>
```

But if it is created without using new Array constructor, it had to be initialized and it had to be not equals to null.

```
let tmparr = [];
console.log(tmparr);
//it prints: []
```

```
let tworect = new Array(5);
console.log("Initial state of 2D array");
console.log(tworect); // print <5 empty items>
let tmparr = [1,2,3,4,5,6];
console.log(tmparr);
//it prints: [1,2,3,4,5,6]
tmparr[3] = 99;
console.log(tmparr);
//it prints: [1,2,3,99,5,6]
//values can be changed after initialization. It can hold any type of object.
```

7. Are any kind of slices supported?

Slices are allowed in javascript. Array can be copied by the help of slices or some part of it can be copied with slice function. Slice comes as a built-in function of array object in javascript.

```
console.log("----------Ragged Array---------------");
console.table(raggedArr);
//output
----------Ragged Array---------------
┌─────────┬────┬────┬────┬────┬────┐
│ (index) │ 0  │ 1  │ 2  │ 3  │ 4  │
├─────────┼────┼────┼────┼────┼────┤
│    0    │ 10 │    │    │    │    │
│    1    │ 20 │ 21 │    │    │    │
│    2    │ 30 │ 31 │ 32 │    │    │
│    3    │ 40 │ 41 │ 42 │ 43 │    │
│    4    │ 50 │ 51 │ 52 │ 53 │ 54 │
└─────────┴────┴────┴────┴────┴────┘
let rag_copy = raggedArr.slice(1, 3); //choosing the row with indices 1 and 2.
console.log("Slice of the ragged array");
console.table(rag_copy);
//output
Slice of the ragged array
┌─────────┬────┬────┬────┐
│ (index) │ 0  │ 1  │ 2  │
├─────────┼────┼────┼────┤
│    0    │ 20 │ 21 │    │
│    1    │ 30 │ 31 │ 32 │
└─────────┴────┴────┴────┘
console.log("Taking another slice");
//taking slice from column of 2d array.To do that reach every row of the 2d
array and then takes slice from them.
```

```
let rag_slice_column = new Array(raggedArr.length);
for(let i = 0; i < raggedArr.length; i++){
    rag_slice_column[i] = raggedArr[i].slice(0,1); }
console.table(rag_slice_column);
//output
Taking another slice
┌─────────┬────┐
│ (index) │ 0  │
├─────────┼────┤
│    0    │ 10 │
│    1    │ 20 │
│    2    │ 30 │
│    3    │ 40 │
│    4    │ 50 │
└─────────┴────┘
```

8. Which operators are provided?

Every operator provided for an object type in javascript are provided for arrays as they are considered as objects in javascript.

```
console.log("--------------------OPERATOR PART-----------------------");
console.log(tmparr);
//output
[ 1, 2, 3, 99, 5, 6 ]
console.log(decimal);
//output
[ [ 10, 20, 30, 40, 50, 60 ], [ 1, 2, 3, 4, 5, 6 ] ]
let result = tmparr + decimal;
console.log("Sum of two arrays");
console.table(result);
//output
1,2,3,99,5,610,20,30,40,50,60,1,2,3,4,5,6
//It considers two arrays as a string and put one of them to end of the other.
console.log(result != tmparr); //prints true
console.log(result == tmparr); //prints false
let identical = [1,2,3,99,5,6];
let copy_ident = identical;
console.log(copy_ident === identical); //prints true as they are identical === checks identical condition
console.log(identical > copy_ident); //print false as they are definitely the same. > checks the value of their address.
```

Python

Python doesn't provide arrays natively, numpy library had to be imported to use arrays in Python.

1-What types are legal for subscripts?

Only [] is allowed for subscript. Subscript operator only takes the integer as its parameter.

```
sorted = np.arange(15) #create array with size 15 and sorted ascending
order.
#print(sorted["1"]) print(sorted{1}) both cause error
```

2- Are subscripting expressions in element references range checked?

It checks the range of array when it tries to reach an item on array.

```
sorted = np.arange(15) #create array with size 15 and sorted ascending
order.
print(sorted[len(sorted) - 1])
print(sorted[len(sorted)]) #cause error: index is out of bound.

np.arange is built-in function of numpy library for creating a sorted
array.
```

3-When are subscript ranges bound?

Lists in python are heap-dynamic arrays. Therefore, subscript ranges bound happens in run-time. But as we use numpy.arrays as an array in our python program, they are not dynamic and created at a fixed size. Once you create the numpy array you cannot insert a new thing to index which is out of bound. Therefore, numpy arrays are static and their subscript ranges bound happens in compile-time when they are defined.

```
arr = np.array([1, 2, 3.7 , 18])
print(arr)
arr[len(arr)] = 19 #cause error: IndexError: index 4 is out of bounds for
axis 0 with size 4
print(arr)
```

4-When does allocation take place?

Variables and data structures whose sizes are predefined and known are allocated during compile time. As our numpy array's size had to be predefined, it

has to be allocated in compile-time. We can change the value of an index but we cannot change consumed memory by an array since we cannot change its size or free the memory block of a numpy array.

```
a = np.array(42) #size is 1
b = np.array([1, 2, 3, 4, 5]) #size is 5
#Once we define a and b, we cannot change their size.
a = np.array([11,22,33,44])
#We can only change it by creating a new object pointed by a.
#We cannot change a's size in run-time.
```

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

Both are allowed in python numpy array. Numpy array designed to make efficient mathematical operations. Thus, multidimensional arrays are very important component of arrays.

```
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
ragg = np.array([[[1,2,3],[4,5,6]], [[11,22,33],
[41,51,61],[71,81,91]]])
for i in range(len(ragg)):
        for j in range(len(ragg[i])):
                print(ragg[i][j])
#prints
#[[1, 2, 3]
[4, 5, 6]]
[[11, 22, 33],
[41, 51, 61],
[71, 81, 91]] #
```

In python we can reach multidimensional arrays by using bracket or nested brackets.

6-Can array objects be initialized?

Numpy arrays had to be initialized when it is created. If we don't initialize it we cannot create it. There is no thing like dynamic array in Javascript or C++ in numpy library. int arr = new int[5]; (C++) int arr = new Array(5);. When we create array arr = np.array([1,2,3,4,5]) . We have to put values inside of it. We can put integer, string or any object we want.

```
app = "apple"
arr1 = np.array([1, 2, 3.7, 18, "abc", app])
b = np.array([1, 2, 3, 4, 5]) #directly initialized
c = np.array([[1, 2, 3], [4, 5, 6]])
b[len(b) - 1] = 99
```

After we initialize the array we can change values for valid indices. If the index is invalid it will give error. We can also use np.empty() for creating uninitialized array with specific size.

```
e = np.empty(4)
print("Array E before initialization: ", end = "")
print(e)
for i in range(4):
    e[i] = 0
print("Array E after initialization: ", end = "")
print(e)
#before initialization: Array E before initialization: [0.00000000e+000
4.22805115e-307 1.33511562e-306 4.91235859e+163]
#after initialization: Array E after initialization: [0. 0. 0. 0.]
```

7-Are any kind of slices supported?

Slices are supported for any dimensional array in python from numpy. Syntax is very similar to reaching an element and it is very clear.

```
arr = np.array([1, 2, 3.7, 18])
print("Not sliced array: ", end = "")
print(arr)
tmp_arr = arr[1:3] #index 1 inclusive index 3 not. Basically from index 1
to 3 - 1 inclusively.
print("Sliced array: ", end = "")
print(tmp_arr)
#Outputs
#Not sliced array: [ 1.   2.   3.7 18. ]
#Sliced array: [2.  3.7]
print("Sliced array without typing two integer: ", end = "")
arrCopy = arr[:]
print(arrCopy)
#prints the same thing with not sliced array since : means take the all
part.
```

Two different syntax are available for taking slice you can either type **arrayName[:]**

Or **arrayName[startIndexInclusive:endIndexExclusive].**

Taking slice from a twoD array is also uses the same syntax. Only difference is the comma for different dimensions.

```
full2d = np.array([[1,2],[3,4],[5,6],[7,8],[9,10]])
print("Not sliced 2D array: ", end = "")
print(full2d)
slice = full2d[:, 0:1] #if : is writing without two integer which show
interval. It means take the all part. Take all rows but choose 0 to 1
columns. Which means take all rows and their crosssection with column[0]
print("Sliced array of 2D: ", end = "")
print(slice)

#outputs
Not sliced 2D array: [[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]]
Sliced array of 2D: [[1]
```

```
 [3]
 [5]
 [7]
 [9]]
```

8-Which operators are provided?

+(plus), *(multiply), **(exponent)

```
a = np.array(14)
b = np.array([1, 2, 3, 4, 5]) #directly initialized
b[len(b) - 1] = 99
f= b + 10
print("Summation of two array: " , end = "")
print(f)
#outputs
Summation of two array: [ 11  12  13  14 109]
```

You can make summation of two arrays if their sizes are same. Their dimension don't have to be same.

```
a = np.array([10,20,30,50,100])
b = np.array([[1, 2, 3, 4, 5],[6,7,8,9,10]]) #directly initialized
b[len(b) - 1] = 99
f= b + a
print("Summation of two array: " , end = "")
print(f)
#Output
Summation of two array: [[ 11  22  33  54 105]
 [109 119 129 149 199]]



g = b * a
print("Multiplication of two array: " , end = "")
print(g)
#Output
Multiplication of two array: [[  10   40   90  200  500]
 [ 990 1980 2970 4950 9900]]
h = a ** 2
print("Array A power of 2: " , end = "")
print(h)
#Output
Exponent of array: [  100   400   900  2500 10000]
#We can use two arrays for these operation or one numerical value and array
also valid for operation.
```

Equality(=) and not equal(!=) operators are also provided.

These two operators work without error if the sizes of array are same. Dimension is not matter. For different dimension comparison, row size and size had to be same.

A =([1,2,3,4,5])

B=([[1,2,3,4,5],[6,7,8,9,10]]). A and B can be compared without a problem since their row size is same. This will be the output when they are compared with equal operator **[[true true true true true][false false false false false]] .**

```
a = np.array([10,20,30,50,100])
b = np.array([[1, 2, 3, 4, 5],[6,7,8,9,10]]) #directly initialized
print(b !=a)
#output
[[ True  True  True  True  True]
 [ True  True  True  True  True]]
```

Notes About Python Numpy.ndarrays

NumPy arrays are homogeneous arrays, considering the mathematical operations that can be performed on them. But when we put different type of values inside the numpy.arrays it won't give errors instead it typecasting when prints.

```
arr = np.array([1, 2, 3.7, 18, "abc"])
print(arr) #print ['1' '2' '3.7' '18' 'abc']
arr = np.array([1, 2, 3.7, 18])
print(arr) #print [ 1. 2. 3.7 18.]
```

Rust

An array is a collection of objects of the same type in Rust language. They are stored in contiguous memory cells. Arrays use [] for creation and reaching its element. In Rust language size of an array had to be decided when it is created. Not only size but also the type of an array had to be decided too. Arrays are homogenous in rust language.

If array objects don't defined as mut (mutable), their values cannot be changed after the initialization.

1. What types are legal for subscripts?

Integer types is legal type for subscripts in rust language.

let xs: [i32; 5] = [1, 2, 3, 4, 5]; i32 means integer-32 byte. xs[0] is valid by this definition of array.

2. Are subscripting expressions in element references range checked?

// Out of bound indexing causes compile error

println!("{}", xs[5]);

3. **When are subscript ranges bound?**

   Subscript ranges are bound when the array is declared and created.
   let arr = [0; length];

   E0307 The length of an array is part of its type. For this reason, this length must be a compile-time constant. Rust gives error when some unconstant value tries to put as a length.

```
let stat_array: [i32; 5] = [1, 2, 3, 4, 5];
Ranges are bound in here as 0 to 5. Arrays are fixed-size in Rust.
```

4. **When does allocation take place?**

   Allocation takes place in compile-time. Arrays are created in stacks in compile-time.

5. **Are ragged or rectangular multidimensional arrays allowed, or both?**
   Both of them are allowed in Rust language.

```
    let mut multi_dim = [[0u8; 3]; 3];
    multi_dim[0][0] = 100;
    println!("array first item {}", multi_dim[0][0]);
    //outputs: array first item 100
    extern crate jagged_array;
      extern crate streaming_iterator;
      use std::iter::FromIterator;
      use jagged_array::{Jagged2, Jagged2Builder};
      use streaming_iterator::StreamingIterator;

      // Create a builder object for the array, and append some data.
      // Each `extend` call builds another row.
      let mut builder = Jagged2Builder::new();
      builder.extend(&[1, 2, 3]); // row 0 = [1, 2, 3]
      builder.extend(vec![4]);    // row 1 = [4]
      builder.extend(&[]);        // row 2 = []
      builder.extend(5..7);       // row 3 = [5, 6]
```

6. **Can array objects be initialized?** dArrays are directly initialized in Rust language when they are created since it allows only fixed-size static arrays.

```
let stat_array: [i32; 5] = [1, 2, 3, 4, 5];
   let mut array: [i32; 3] = [0; 3]; //all elements initialized to 0
```

7. Are any kind of slices supported?

Taking a slice from an array is available in Rust language.

```
 //taking slice from an array
     let full_arr = [1,2,3,4,5];
     let slice = &full_arr[1..4]; //Index 1 is inclusive 4 is exclusive
     println!("First index of sliced array {}", slice[0]);// prints 2 as
the new sliced array is {2,3,4}.
     println!("Last index of sliced array {}", slice[slice.len() - 1]);
// prints the last index of sliced array which is 4
```

8. Which operators are provided?

Only & and .. is provided for arrays to used in slice operation.

```
 //taking slice from an array
     let full_arr = [1,2,3,4,5];
     let slice = &full_arr[1..4]; //Index 1 is inclusive 4 is exclusive
```

PHP

1. What types are legal for subscripts?

[] and {} are legal for subscripts. But subscripts operator only takes integer value otherwise it gives error.

```
$array1 = array(1, 2, 3, 4, 5);
debug_to_console_wo_newLine("First element: ");
debug_to_console($array1[0]);
debug_to_console_wo_newLine("First element: ");
debug_to_console($array1{0});
//debug_to_console($array1["zero"]); //cause error:  Undefined index: zero in
// print same thing [] and {} do the same job.
```

2. Are subscripting expressions in element references range checked?

Yes, it is range checked.

```
$array1 = array(1, 2, 3, 4, 5);
//debug_to_console($array1[count($array1)]); gives error that means range is
checked.
```

3. When are subscript ranges bound?

Subscript ranges bound in run-time, it supports heap-dynamic arrays. It means you can add new things to index that is bigger than array size. It automatically grows.

```
$arr = array(1, 2, 3, 4, 5);
$arr[0] = 99;
debug_to_console("Full Array without Slice");
debug_to_console($arr);
$arr[10] = 100;
debug_to_console("After addition to new item in array");
debug_to_console($arr);
//output
Full Array without Slice
99,2,3,4,5
After addition to new item in array
99,2,3,4,5,100
```

4. When does allocation take place?

Allocation takes place in run-time.

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

Both of them are allowed and provided in PHP. As Php consider arrays as a storage for objects and arrays are also objects, it can store them in any way you want.

```
$multi_dim = array ( array ( 1, 2),
                  array ( 3,4 ),
                  array(5,6,),
                  array(7,8));
//multi dim array printing with nested for loops
debug_to_console("Multi Dim");
debug_to_console("----------------");
$i = 0;
$j = 0;
for($i = 0; $i < count($multi_dim); $i++){
    for($j = 0; $j < 2; $j++){
        debug_to_console_wo_newLine($multi_dim[$i][$j]);
    }
    debug_to_console(""); //for printing new line
}
```

```
//output
Multi Dim
----------------
1 2
3 4
5 6
7 8
----------------
debug_to_console("----------------");
$ragged = array ( array ( 1, 2, 3),
                  array ( 2,4 ),
                  array(6,7,8,9));
debug_to_console("--------------------------");
debug_to_console("Ragged Array");
$x = 0;
for($x = 0; $x < 3; $x++) {
    debug_to_console($ragged[$x]);
}
//output
-------------------------
Ragged Array
1,2,3
2,4
6,7,8,9
-----------------------
```

6. Can array objects be initialized?

Array objects can be initialized when they are created or they can be left empty to fill later. Empty things won't appear in the console in PHP. For example if we have an array with size 5 and we put some values into the index 15, if we try to print elements inside of the array, we will see 6 item; first 5 and index 15. PHP doesn't automatically put '0's or garbage values to unused memory cells. Thus it won't appear in console.

```
$array1 = array(1, 2, 3, 4, 5);  //can be initialized like this
$empty = array();
$empty[1] = 19;
debug_to_console_wo_newLine("After adding new item to empty: ");
debug_to_console($empty); //it prints only 19.
```

7. Are any kind of slices supported?

Slices from any dimensional arrays or ragged arrays are supported and very easy to use with built-in array_slice method.

```
$arr = array(1, 2, 3, 4, 5);
$arr[0] = 99;
$slice = array_slice($arr, 1, 4);
debug_to_console("Full Array without Slice");
debug_to_console($arr);
debug_to_console("Sliced Array");
debug_to_console($slice); // printing sliced array
//output
Full Array without Slice
99,2,3,4,5
Sliced Array
2,3,4,5
```

```
//slice from ragged array
$ragged = array ( array ( 1, 2, 3),
                  array ( 2,4 ),
                  array(6,7,8,9));
debug_to_console("-------------------------");
debug_to_console("Ragged Array");
$slicefromrag = array(0,0,0);
//slice from ragged array
$t = 0;
for($t = 0; $t < 3; $t++){
    $slicefromrag[$t] = array_slice($ragged[$t], 1, 2);
}
$x = 0;
for($x = 0; $x < 3; $x++) {
    debug_to_console($ragged[$x]);
}
//output
-------------------------
Ragged Array
1,2,3
2,4
6,7,8,9
----------------------
debug_to_console("----------------------");

debug_to_console("Taking slice from rag array's column with for loop");
$x = 0;
for($x = 0; $x < 3; $x++) {
```

```
    debug_to_console($slicefromrag[$x]);
}
//output
Taking slice from rag array's column with for loop
2,3
4
7,8
-------------------------
debug_to_console("-------------------------");
```

8. Which operators are provided?

+, ==, != , === , !==, <> are provided for array objects.

```
$x = array(11);
$y = array(9, 16); //non-homogenous
$z = $x + $y;
debug_to_console("Sum Array of two array");
debug_to_console($z);
debug_to_console("-----------");
//output
Sum Array of two array
11,16
-----------
//assoc arrays for comparison with operators.
$x = array("first" => 51);
$y = array("first" => "51");

var_dump($x);

// == (Equal)
var_dump($x == $y); //they are equal, type check doesn’t mattere. Php compare
them converting values in same type.

// === (Identical)
var_dump($x === $y); //they are not identical. Memory addresses are different.
// !=, <> (Not Equal)
var_dump($x != $y);
var_dump($x <> $y);

// !== (Non Identical)
var_dump($x !== $y);
//output
array(1) {
  ["first"]=>
  int(51)
```

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

Dart

1. What types are legal for subscripts?

[] are legal for subscripts. Only integer type is allowed between brackets.

```
List<int> arr = [1, 2, 3, 4, 5]; //only [] is allowed for subscript operator
  arr[0] = 11; //array index starts from zero
  //arr[1.2] = 13; cause syntax error -> The argument type 'double' can't be
       assigned to the parameter type 'int'.
  print("First element");
  print(arr[0]); //print 11
```

2. Are subscripting expressions in element references range checked?

Yes, it is range checked.

```
List<int> arr = [1, 2, 3, 4, 5]; //only [] is allowed for subscript operator
//Range checking
  /*for(int i = 0; i <= arr.length; i++){
    arr[i] = 15; //as we give equal condition at the last, arr[arr.length]
gives error.
    //range is checked by dart program.
    //Error message: RangeError (index): Invalid value: Not in range 0..4,
inclusive: 5
  }
  */
//We cannot reach arr[arr.length] since it is out of bounds.
```

3. When are subscript ranges bound?

Subscripts ranges are can be bound in compile-time or run-time since there are two different definition of array in Dart. Dart supports both static and dynamic arrays like C++;

First type = List<int> arr= [1,2,3,4,5] (non-fixed length arrays, you can add new items to this array, dynamic array typically)

Second type = var arr = new List(5); //you can't add new items to indices which are out of bounds, it cause error.

```
List<int> arr = [1, 2, 3, 4, 5]; //only [] is allowed for subscript operator
  //List<int> arr = []; is non-fixed size declaration.
  //But if we say var arr = new List(5); it is fixed length definition.
  arr[0] = 11; //array index starts from zero
  //arr[1.2] = 13; cause syntax error -> The argument type 'double' can't be
assigned to the parameter type 'int'.
  print("First element");
  print(arr[0]);
  print("Before insertion ");
  print(arr);
  print("After insertion");
  //arr[10] = 99; gives index out of bounds
  arr.add(99);
  print(arr);
  //output
First element
11
Before insertion
[11, 2, 3, 4, 5]
After insertion
[11, 2, 3, 4, 5, 99]
//second type array
var tmp = new List(2);
  tmp[0] = 100;
  tmp[1] = 200;
  print("Array with 2 elements");
  print(tmp);
  tmp.add(300);
  print("After addition");
  print(tmp);
//output
Array with 2 elements
[100, 200]
Unhandled exception:
Unsupported operation: Cannot add to a fixed-length list
```

4. When does allocation take place?

Allocation takes place in run-time or compile-time depends on the type of an array.

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

Both are allowed in Dart language. Two-D or ragged arrays can be defined in two different ways as I mentioned above.

```
intialized just like oneD.
  //Or they can declared as oneD then for every index we can put array object
inside of them.
  //Dart can holds object inside their array.
List<List<int>> twoD = [[1,2,3], [3,4,5], [5,6,7]]; //2d arrays can directly
  var twoD_2 = new List(3);
  print("Initial state ");
  print(twoD_2);
  for(int i = 0; i < twoD.length; i++){
    twoD_2[i] = new List(3);
  }

  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
      twoD_2[i][j] = 8;
    }
  }

  print("Two D Array");
  print(twoD);
  print("Two D Array after traversing all element");
  print(twoD_2);
  print("----------------------");
//output
Two D Array dynamic
[[1, 2, 3], [3, 4, 5], [5, 6, 7]]
Two D Fixed Size
[[8, 8, 8], [8, 8, 8], [8, 8, 8]]
----------------------

var ragged = new List(5);
  //ragged array creation
  for(int i = 0; i < 5; i++){
    ragged[i] = new List(i + 1);
  }
  for(int i = 0; i < ragged.length; i++){
```

```
    for(int j = 0; j < ragged[i].length; j++){
      ragged[i][j] = (i + 1) * 10 + j + 1;
    }
  }
  print("Ragged Array");
  print(ragged);
  print("----------------");
//output
Ragged Array
[[11], [21, 22], [31, 32, 33], [41, 42, 43, 44], [51, 52, 53, 54, 55]]
----------------
```

6. Can array objects be initialized?

They can be initialized by saying List<int> arr = [1,2,3,4,5]; otherwise it holds null value regarding to its size.

```
  var twoD_2 = new List(3);
  print("Initial state ");
  print(twoD_2);
//output
Initial state
[null, null, null]
```

7. Are any kind of slices supported?

Slices are supported via list.sublist method.

```
 //Slicing Part
  List<int> arr = [1, 2, 3, 4, 5]; //only [] is allowed for subscript operator

  //List<int> arr = []; is non-fixed size declaration.
  //But if we say var arr = new List(5); it is fixed length definition.
  arr[0] = 11; //array index starts from zero
  arr.add(99);
  print(arr); //prints: 11, 2, 3, 4, 5, 99
  var sliced = arr.sublist(1,4);
  print("Sliced Array");
  print(sliced); //prints 2, 3, 4
```

8. Which operators are provided?

+, ==, != are provided for Dart arrays. + logic is taking arrays as string and putting one of them to end of another.

```
print("Arr1 elements");
  print(arr1);
  print("Arr elements");
  print(arr);

  //operator provided for arrays
  //List<int> sum; Error Message: type 'List<dynamic>' is not a subtype of
type 'List<int>'
  List<int> sum = [];
  if(arr == arr1){
    sum = arr;
  }
  else{
    sum = arr + arr1;
  }
  print("Sum of arr and arr1");
  print(sum);
  print("Equality check");
  print(sum == arr);
  print(sum != arr);
//outputArr1 elements
[99, 20, 30, 40, 89]
Arr elements
[11, 2, 3, 4, 5, 99]
Sum of arr and arr1
[11, 2, 3, 4, 5, 99, 99, 20, 30, 40, 89]
Equality check
false
true
```

Question -1

Best Language for Arrays

The best language for array operation is javascript, since it allows to store non-homogenous values inside of it. It is also dynamic. You can add new things whenever you want. More than that, it allows to use of any memory cell for new addition, but if you try to reach a memory cell that is not allocated for array it prints undefined or shows some errors. As arrays are objects in javascript, they are come up with very useful functions and properties. Built-in functions such as filter(), table(), arrange() or sort() are very useful for array operations. Another edge is the writing flexibility of the language. In javascript, you can define arrays more than

one way. let arr = [1,2,3,4,5]; or let arr = new Array(5); both are possible definition. One of them directly initialize the array. Other is not. Javascript arrays can hold any type of object inside of it, therefore you can build n-dimensional or ragged arrays easily. One bad thing about javascript is it never gives an error when you try to reach a null object. It may be disadvantage in case of reliability. Instead it prints <empty item> or undefined. Also, arrays can easily traverse with [] or [][] unlike Rust. Type checking is not a problem for javascript unlike Rust too. Php and Rust language have a problem with writeability. They are not easy to understand or write for a beginner coder in those languages. Even printing is an issue. Python also has a more complicated syntax than javascript and it is less reliable than Javascript. When we make the array non-homogenous in python output will change. Dart language doesn't provide enough built-in functions as much as Javascript. Thus, javascript is the best.

Question – 2

My learning strategy

I started homework by dividing my workload into two; the familiar languages and non-familiar languages. In the beginning, I searched the syntax of every programming language from the internet. Then, categorize them from most familiar to less. It seems like javascript, python and maybe dart are the ones that I am more familiar with. Their array base syntax seems similar to C++ and Java. Also, if statements, loops, print statements seem particularly the same with C++ and Java. I started by writing a program instead of writing a report. I use visual studio code for Javascript. I started to write some console output by reading the document of javascript language. Whenever I have faced a problem, I searched online. For those moments, Stackoverflow will be my guide. In the case of javascript, I hadn't faced many problems. After I had enough practice with arrays, I started to implement what homework want from me. I debug my program from vscode easily, so javascript is not easy. In the case of the dart language, dart seems very goal-oriented and easy to understand. It is also not that hard, I use an online

dart compiler to avoid syntax errors but when I ran my program on an online compiler it gives a null safety error. Therefore, I write my program in an online compiler but run it on the Dijkstra machine. For Dart following the official documents was enough for writing array operations. Python was harder than I expected. Although I asserted that these three were familiar to me, I didn't have any experience with these three languages. I started by reading NumPy documentation. Then I try to implement using pycharm as it has a strong text editor and is easy to use debug. Python syntax was too hard for me even printing two things in the same line made me confused. Php is easier than my expectation since there are lots of sources online explaining PHP. I wrote my programs on vscode and run them on Dijkstra.

Writing the rust program was the hardest since there are not many sources on the internet and rust is a little different from than other four languages. Even the integers are different u-unsigned i-signed int. I use an online compiler for rust language. But working code on an online compiler won't work in the Dijkstra machine therefore I had to delete nested for loops and put iterative for.

When I write my report I used the documentation of languages to learn about their properties. How arrays are defined? Are they created in run-time or compile-time or are they dynamic or not? I took some notes about their properties. So that I can add some codes to answer a specific question in homework. The most helpful site was "https://www.w3schools.com/default.asp" for any language. Since it explains not giving the code snippets only. I decided to watch some python and javascript array tutorials from youtube to gain further information about them such as https://www.youtube.com/watch?v=Lf9rG5DsQYg. Therefore, in my report javascript and python parts are more detailed than the other three parts. The hardest part of the homework is understanding some questions on homework like questions 4 and 6. I used the lecture slides of chapter 6 to understand questions clearly. Another difficult thing was writing a report about rust and its main program. Since the language is too hard to learn for me in a short time.

References

Javascript web page for documentation: https://javascript.info/array

Official Python3 Documentation: https://docs.python.org/3/

Helpful website for various information about every language or concepts: https://www.geeksforgeeks.org/

Dart programming language list and array documentation: https://www.tutorialspoint.com/dart_programming/dart_programming_lists.htm

Official Rust documentation: https://doc.rust-lang.org/book/ch04-03-slices.html

Array documentation of Rust Language: https://doc.rust-lang.org/rust-by-example/primitives/array.html

Python NumPy library documentation: https://www.pythoninformer.com/python-libraries/numpy/index-and-slice/

Javascript Array explanation: https://www.tutorialsteacher.com/javascript/javascript-array

Site for documentation and information about various programming languages: https://www.w3schools.com/default.asp

Shell scripting information about Unix: https://www.tutorialspoint.com/unix/shell_scripting.htm