Concept Review: Copy-by-value

1. Explain the difference between copying a primitive variable versus copying an

object in JavaScript. Where does the value get stored (stack/heap)? How does

the process differ?

Ex. code

```
// copying for primitives

const a = 5;

const b = a;

// copying for objects

const arrA = [10];

const arrB = arrA;
```

A.The difference between copying a primitive variable and copying an object involves where and how the values are stored in memory (stack vs. heap) .

The stack is a memory which stores simple values and manages in a fashion Last In First Out(LIFO) as every variable has a separate space in stack. Primitives stored in Stack.

Heap is a dynamically allocated memory location for variables which can be stored and referenced by pointers. Objects stored in Heap.

When copying the primitive value we are copying the actual value i.e from eg:the variable “b” gets its own copy of value from ” a” later both becomes independent.

```
let a = 5;  // Primitive value stored in stack
let b = a;   // Copy of the value is created

console.log(a); // 5
console.log(b); // 5 so b value is 5 as it stores the copy of value of a

a = 35; // now a is assigned to another value by it doesn't effect on b as both
are assigned values and stored in stack
console.log(a); // 35
console.log(b); // 5

b = 20;       // Changing b doesn't affect a .
console.log(a); // 35
console.log(b); // 20
```

When arrB = arrA; is executed, the reference (pointer) to the object in the heap is copied to arrA. Both arrA and arrB now point to the same object in the heap. Any changes we made through either variable will reflect in the other because they refer to the same object.

```
const arrA = [10]; // Object reference stored in stack, object itself in heap
const arrB = arrA //Copy of the reference is created

console.log(arrA); //[10] //
console.log(arrB);//[10]

arrA[0] = 100;

console.log(arrA); //[100]
console.log(arrB);//[100]

arrB[0] = 200;
console.log("arrA",arrA);// 200 because both a and b have the same memory
reference.
console.log("arrB",arrB);//200
```

2. Explain what does “copy-by-value” mean?

The Copy by value means that copying the data where the actual value of variable is copied to new variable, after copied both variables are in dependents as these are stored in two different in stack. we can see from above eg changes made to variable b doesn’t not affect a and vice versa.

3. Explain what is a memory address or a memory reference?

Memory address is a unique identifier for a memory location where

data is stored in a RAM.

Memory Reference involves the a memory address to access data stored in memory.

For :

Let x = 10; //the variable x in declared an address is created let say “efrtyy” .the value 10 is stored in the address. When we try to access the data 10 by referncing the address is memory reference.

4. What is the memory stack? (You may google this, but your answer must be in

your own words based on your understanding)

A. Stack memory is used for static memory allocation ,including the functions and variables. Though it is fast in retrieving the data it is limited in size.

5. What is the memory heap? (You may google this, but your answer must be in

your own words based on your understanding)

A.Heap is dynamically allocated memory location as it is flexible

memory allocation but slower the data access.

Large data structures like array, linked lists, and tress are dynamically sized.

6. Does order matter in arrays, lists or vectors? Why or why not? How do you

search/ identify for an item in a list?

A Yes, order matters in array, lists or vectors.

Arrays: Elements stored in an array are in contiguous memory location and are strictly maintained in sequence manner as it determines the position of the element in an array.

List: Lists, which is a data structure, the order is important as each element points to the next element in sequence .Changing the order disrupts the entire structure.

Search a item in list:

Linear search : This simple method involves checking each element of the list line by line until we find the target item or reach the end of the item.

Binary search: In this we repeatedly divide the search interval

in half.

7. What’s the difference between methods vs. properties? How do you use each

one differently?

Properties are used to store data about an object. They define the state of an object which accessed using dot notation or bracket notation. Properties are accessed directly to get or set values.

Eg: here Car is an object ,we use car.make or car.year is to its properties

```
let car = {
      make: 'Toyota',
      model: 'Corolla',
      year: 2020
};

console.log(car.make);   // Output: Toyota
console.log(car['year']); // Output: 2020
```

Methods: Methods are used to perform operations on the data (properties) of the object. They define the behaviour of an object which accessed and invoked using dot notation, followed by parentheses. Methods are invoked to perform a task, often working with the object's properties.

Here we use start and getdetails functions to performan action on the properties of the car. s

Eg:
```
let car = {
      make: 'Toyota',
      model: 'Corolla',
      year: 2020,
      start: function() {
      console.log('Car is starting');
      },
      getDetails: function() {
      return `${this.make} ${this.model} (${this.year})`;
      }
};

car.start();   // Output: Car is starting
```

```
console.log(car.getDetails()); // Output: Toyota Corolla (2020)
```

8. Do you think primitives have methods and properties? Try it out for yourself in the

code editor.

A Primitives have no methods but still behave as if they do. When properties are accessed on primitives, JavaScript auto-boxes the value into a wrapper object and accesses the property on that object instead.

1.
```
let str = "Hello, World!";


// Accessing a method

console.log(str.toUpperCase()); // Output: "HELLO, WORLD!"

console.log(str.concat("welcome));// Hello, World!welcome

// Accessing a property (length)

console.log(str.length); //  13
```

2.
```
let num = 42;
```

```
console.log(num.toFixed(2)); // Output: "42.00"

console.log(num.valueOf()); // Output: 42


3.let bool = true;

console.log(bool.toString()); // Output: "true"
```