PAGE NO.:
DATE: / /

Assignment-4

Section:-A

Q1) What is a function in JS?

Ans: A function in JS is a reusable block of code designed to perform a specific task. Functions can take inputs (parameters), process them and return a result.

Q2) How do you invoke a function in JS?

Ans You invoke a function by writing its name followed by parentheses. For ex: functionName() or functionName(arguments) if it takes parameters.

Q3) What is JS object?

Ans A JS object is a collection of key-value pairs where keys are strings (or symbols) and values can be of any data type, used to model real-world entities.

Q4) What does the new keyword do in JS?

PAGE NO.:
DATE: / /

Ans:- The new keyword is used to create an instance of an object that has a prototype linked to a constructor function, initializing the object with properties and methods defined in the constructor.

Q5) How do you add a method to an existing JS object?

Ans You can add a method by assigning a function to a property of the object. For example, object. methodName = function { //code }

Section :- B

Q1) Explain the difference b/w primitive values and ~~of~~ objects in JS.

Ans Primitive values in JS include string, number, boolean, null, undefined, bigint & symbol. They are immutable and stored directly in the ~~memory~~ memory as simple data types.
For ex: let a = 5, let b = a;
a = 10;
console.log(b);

PAGE NO.:
DATE: / /

objects, on the other hand are complex data structures that store collections of data and functionality. They are mutuable and passed by reference, meaning changes to an ~~of~~ object affect all references to it.

Ex
```
let obj = {key: "value"},
let ref = obj;
ref. key = "new value";
consol. log (obj. key)
```

Primitive values are better for fixed ~~of~~ data, while objects are used for dynamic & complex data structures.

Q2) Describe the purpose of the call(), apply(), & bind() methods in JS

Ans: call(): invokes a function immediately with a specified this value and individual arguments

```
function greet~~ing~~(greeting) {
 consol. log(`${greeting}, ${this. name}`);
}
greet. call ({ name: "Alice"}, "Hello");
```

PAGE NO :
DATE : / /

- apply(): It is similar to call(), but takes arguments as an array.

greet.apply({ name: "Roy" }, ["Hi"])

- bind(): It creates a new function with a bound this value, which can be called later.

Const greetRJ = greet.RJ({ name: "RJ" });

greetRJ("Hiii");

Use call or apply for immediate invocation and bind for deferred execution with a fixed this.

Q3) What is a closure in JS, ~~and~~ and how can it be used in nested functions?

Ans A closure is a function that retains access to its lexical scope, even when the function is executed outside its original context. Closures enable nested functions to access variables declared in their outer functions.

PAGE NO :
DATE : / /

```
function outer(){
    let count=0;
  return function inner(){
      count++;
       return count;
   };
 }
const counter = outer();
console.log(counter());
console.log(counter());
```

Closures are useful for encapsulating private variables and creating function factories or event handlers.

Section :- C

Q1) Explain the different ways to execute functions in JS as a regular function, as a method, and as a constructor.

Ans As a regular function:- A function is called directly in the global or local Scope. The this value is undefined is strict mode or the global other wise.

```
function greet(){ console.log("Hello");
  } greet();
```

As a method: When a function is called as a property of an object, it acts as a method, and this refers to the object it belongs to.

. As a method: When a function is called as an property of an object, it acts as a method, and this refers to the object it belongs to.

```
const obj = {
 greet(){
 Console.log("Hello, ${this.name}");
 },
};
obj.greet();
```

. As a constructor. When called with the new keyword, a function acts as a constructor, creating a new object. this refers to the newly created object.

```
function Person(name){
     this.name = name;
 }
const person = new Person("Dev");
Consolelog(person.name);
```

PAGE NO :
DATE : / /

Q2) Describe how to create JS objects using both literal syntax and the new keyword. Compare the two approaches.

Ans: Objects can be created using curly braces with key-value pairs. This approach is simple, readable, and ideal for creating single objects.

```
const obj = {
  name: "Alex",
  age: 25,
  greet() {
console.log { `Hello, ${this.name}`};
 {
  obj.greet();
```

The new keyword is used with a constructor function to create multiple instances with shared prototypes.

```
function Person(name, age) {
   this.name = name;
   this.age = age;
 }
const arya = new Person("Arya", 2
Console.log (alice.name);
```

⇒ Comparison

- Use literal syntax for simple, one-off objects
- Use the new keyword for multiple instances with shared properties or methods.

Q3) Discuss the role of arguments in JS functions, including the arguments object and rest parameters.

Ans:- Arguments in JS functions allow passing data into functions. Functions can handle varying numbers of arguments using:

- arguments object: Array-like object availabel in non-arrow functions, holding all passed arguments.

```
function sum() {
  let total = 0;
  for(let i = 0; i < arguments.length;
                                  i++){
    total += arguments [i];
  }
  return total;  }
console.log (sum(1, 2, 3));
```

PAGE NO :
DATE : / /

• Rest parameters : A modern, array-like syntax for handling variable arguments.

```
function sum(... nums){
return nums. reduce((total, num) =>
    total + num, 0);
    }
console. log(sum (1, 2, 3));
```

=) Usage :

• Use arguments for backward ~~compa~~ compatability.
• Prefer rest parameters for readability and modern features.