Day 1

JavaScript Output Based Question

(1)
```
function fruit() {

console.log(name);
console.log(price);

Var name = "apple";
let price = 20;

}

fruit ( );
```

Ans.) Inside the 'fruit' function:

- Hoisting of 'var' : The variable 'name' declared with 'var' is hoisted to the top of the function scope, but it's assignment ("apple") is not.

  This means 'name' is initialized as 'undefined' at the start of the function.

- 'Let' declaration : The variable 'price' declared with 'let' is not hoisted in the same way. It is in the "temporal dead zone" from the start of the function until the 'let' statement is executed, meaning it can't be accessed before it's declaration.

Therefore, the output will be :

- Undefined

- Reference Error : can't access 'price' before initialization.

(2)
```
for (var i=0 ; i<3 ; i++) {
    setTimeOut( () => console.log (i), 1);
}
```

Ans:- • Variable Declaration with 'var' : The Variable 'i' is declared with 'var', which has function scope (or global scope if not inside a function).

- Loop Execution : The loop runs three times, incrementing 'i' from 0 to 2.

- setTimeOut Function : For each iteration, 'setTimeOut' is called with a delay of 1 milisecond. The callback function inside 'setTimeOut' logs the value of 'i' to the scope.

Since 'var' is function-scoped (or global scoped) the same 'i' is shared across all iterations of the loop.

DELTA
Date / /
Pg No.

By the time the 'setTimeOut' callbacks execute (after the loop has completed), 'i' has been incremented to 3.

Therefore, the output will be—

- 3
- 3
- 3

(3) for (let i=0 ; i<3 ; i++) {
    setTimeOut (() ⇒ console.log (i), 1) ;
}

(3) Ans:> • Variable Declaration with 'let' : The variable 'i' is declared with 'let', which has block scope. This means a new 'i' is created for each iteration of the loop.

• loop Execution : The loop runs three times, incrementing 'i' from 0 to 2.

• 'setTimeOut' Function : For each iteration, 'setTimeout' is called with a delay of 1 milisecond. The callback function inside 'setTimeout' logs the current value of 'i' to the 'console'.

Because 'let' creates a new 'i' for each iteration, each callback function inside 'setTimeout' captures the values of 'i' at the time of that iteration.

Therefore, the output will be –

- 0
- 1
- 2

(4) console.warn(+true);
console.warn(typeOf +true);

Ans:- • First Statement →

• The unary plus operator ('+') attempts to convert it's operand to a number.

• In Javascript 'true' is converted to the number '1'.

• Second Statement →

• As explained, '+true' converts 'true' to the number '1'.

DELTA
Date / /
Pg No.

- The 'typeOf' operator returns a string indicating the type of the unevaluated operand.

Therefore, the output will be -

- 1
- number

(5) console.warn(!"Rajesh");
console.warn(typeOf("Rajesh"));

Ans:> First statement →

- The '!' (logical NOT) operator converts it's operand to a boolean value and then negates it.
- A non-empty string, such as "Rajesh", is considered 'truthy' in JavaScript.
- The negation of 'truthy' is 'false'.

Second Statement →

- The 'typeOf' operator returns a string indicating the type of the 'unevaluated' operand.
- "Rajesh" is a string.

Therefore, the output will be -

- false
- string