TASK 6 - JavaScript

1. What will be the output and why?

```
console.log(a);
var a = 10;
function foo() {
console.log(a);
var a = 20;
}
foo();
console.log(a);
```

Output:

```
undefined
undefined
10
```

Explanation:

The first console.log(a); outputs undefined because var a is hoisted but not initialized.
Inside foo(), console.log(a); outputs undefined due to a local hoisted a, which is initialized later.
The last console.log(a); outputs 10, as it references the global a, which was initialized to 10.

2. What will be the output? How can you modify the code so that it logs 0, 1, 2, 3, 4 instead?

```
for (var i = 0; i < 5; i++)
{
setTimeout(function(){
console.log(i);
}, 1000);
}
```

Output:

```
5
```

```
5
5
5
5
```

Modifying the Code to Log 0, 1, 2, 3, 4

Solution 1: Use let in the for loop

```
for (let i = 0; i < 5; i++) {
   setTimeout(function() {
      console.log(i);
   }, 1000);
}
```

3. What will be the order of the logs and why?

```
console.log('Start');
setTimeout(() => {
console.log('Middle');
}, 0);
console.log('End');
```

Output:

```
Start
End
Middle
```

Explanation:

Start and End are logged immediately because they are synchronous. Middle is logged last because setTimeout is asynchronous and moves to the event queue, executing after the main code completes.

4. What will be the output and why?

```
let arr = [1, 2, 3];
arr[10] = 5;
console.log(arr.length);
console.log(arr);
```

Output:

```
11
[1, 2, 3, <7 empty items>, 5]
```

Explanation:
Setting arr[10] = 5 creates a sparse array with 5 at index 10, expanding arr to have a length of 11.
The array will display as [1, 2, 3, <7 empty items>, 5] because indices 3 through 9 are uninitialized (they're "empty slots").

5. What will be the output of these expressions and why?

```
console.log([] + []);
console.log([] + {});
console.log({} + []);
```

Output: "" (an empty string)
Explanation: Both operands are empty arrays, which are converted to empty strings when used with +.

Output: "[object Object]"
Explanation: [] is converted to an empty string (""), and {} is converted to "[object Object]".

Output: 0
Explanation: When {} is the first item on a line, it's interpreted as an empty block of code, not an object literal.

6. What will be the output and why? How can you compare the content of two objects?

```
let a = { foo: 'bar' };
let b = { foo: 'bar' };
console.log(a == b);
console.log(a === b);
```

Output:

```
false
false
```

Explanation:
In JavaScript, objects are compared by reference, not by content. `a` and `b` refer to two different objects in memory, so `a == b` and `a === b` both return `false`.

How to Compare the Content of Two Objects:
To compare the content of two objects, you can use `JSON.stringify` or a deep comparison method:

```
JSON.stringify(a) === JSON.stringify(b);
```

7. What will happen when you run this code and why?

```
foo();
var foo = function() {
console.log('Function expression');
};
```

Output: TypeError - foo is not a function.
Explanation:
The variable foo is hoisted to the top, but only the declaration.
Before foo is assigned the function, it is undefined, so calling foo() results in a TypeError since undefined is not callable.

8. What will be the output and why?

```
const person = {
name: 'Alice',
age: 25,
address: {
city: 'Wonderland'
}
};
const { name, address: { city }, country = 'Unknown' } = person;
console.log(name, city, country);
```

Output: Alice Wonderland Unknown
Explanation:
name is destructured from person and assigned the value 'Alice'.
city is destructured from person.address and assigned the value 'Wonderland'.

country is assigned a default value of 'Unknown' because country does not exist in the person object.

9. What will be the output and why?

```
const promise = new Promise((resolve, reject) => {
console.log('Promise started');
resolve('Success');
});
promise.then(res => {
console.log(res);
});
console.log('Promise created');
```

Output:

```
Promise started
Promise created
Success
```

Explanation:

Promise started is logged immediately when the promise is created.
Promise created is logged next because .then() callbacks are asynchronous and wait until the main code finishes.
Finally, Success is logged from the .then() callback after the synchronous code completes.

10. What will be the output and why?

```
(function() {
var x = 10;
(function() {
console.log(x);
var x = 20;
console.log(x);
})();
})();
```

Output:

```
undefined
20
```

Explanation:

In the inner function, var x is hoisted, creating a local variable x that shadows the outer x. Initially, x is undefined, so the first console.log(x); prints undefined. Then x is assigned 20, so the second console.log(x); prints 20.