Day-14

JavaScript Output Based Question

gitHub → rajeshjha2000

(66)
```
const Obj = { };
Object.defineProperty (Obj, 'a', {
value :"Char" });
console.log (Obj.a);
console.log(Object.Keys(Obj));
```

Ans.> 1. 'Object.defineProperty( )' :⟶

- The 'Object.defineProperty( )' method defines a new property directly on an object, or modifies an existing property on an object, and returns the object.
- By default, properties added using 'Object.defineProperty()' are:

  → Non-enumerable : The property does not show up in enumerations such as 'for ... in' loops or 'Object.keys()'.

  → Non-writable : You can't change the value of the property.

  → Non-configurable : You can't delete the property or change it's attribute.

2. Property 'a' :⟶

- In our code, we defined a property 'a' on the 'Obj' object with the value "Char".
- Since we didn't specify any other attributes (such as 'enumerable', 'writable', or 'configurable'), the default settings apply.

3. First- `console.log(obj.a)`: →

- This will output the value of the property 'a', which is "Char".

4. Second- `console.log(Object.keys(obj))`: →

- The 'Object.keys()' method returns an array of a given Object's own enumerable property names.
- Since 'a' is non-enumerable by default, it will not appear in the array returned by 'Object.keys()'.

Therefore, the output will be →

- "Char"
- [ ]

(67)
```
function addItem(item, list){
  return list.push(item);
}
   const result = addItem('♡', ['👍']);
   console.log(result);
```

Ans.→ 1. 'addItem' Function →

- The 'addItem' function takes two parameters: 'item' and 'list'.
- It adds 'item' to the end of 'list' using the 'push()' method.
- The 'push()' method adds one or more elements to the end of an array and returns the new length of the array.

2. Calling `addItem('♡', ['👍'])` : ⟶

- Here, '♡' is the 'item' and '['👍']' is the 'list'.
- The 'push()' method adds '♡' to the array '['👍']', making it '['👍', '♡']'.
- The 'push()' method returns the new length of the array, which is 2.

Therefore, the output will be ⟶

- 2

(68)
```
function* generatorFn(i) {
    console.log('A');
    yield i;
   console.log('B');
   yield i* 2;
 }
 const gen = generatorFn(10);
 console.log(gen.next().value);
 console.log(gen.next().value);
```

Ans.→ 1. Generator Function(`function*`) : ⟶

- A generator function is a special type of function that can pause execution and later resume it.
- It returns an iterator object, which can be used to step through the code inside the function.
- The 'yield' keyword is used to pause the function and return a value.

2. Defining the Generator Function →
`generatorFn(i)`:

- The function logs 'A' to the console and then yields `i`.
- After that, it logs 'B' to the console and then yields `i*2`.

3. Creating the Generator Object →

- `const gen = generatorFn(10);`
- This creates an iterator object `gen` from the generator function. However, the function's body doesn't execute immediately; it's only prepared to be executed.

4. `console.log(gen.next().value)`: →

- The `next()` method resumes execution of the generator function from where it left off (or from the beginning if it hasn't started)
- The first `gen.next()` call starts the function, logs 'A' to the console and pauses at the first `yield`, returning the value `i` which is `10`.

5. `console.log(gen.next().value)`: →

- The second `gen.next()` call resumes the function from where it paused after the first `yield`.
- The function logs 'B' to the console and then yields `i*2` which is `20`

Therefore, the output will be →

- A
- 10
- B
- 20

(69) const {fName: "Rajesh"} = {fName : "Jha"};

console.log (fName);

Ans.-> 1. Object Destructuring →

• In JS, we can use Object destructuring to extract properties from an Object and assign them to Variables.

• The syntax `{fName : Rajesh}` means we are extracting the 'fName' property from the Object and assigning its value to a new Variable named 'Rajesh'.

2. What Happens Here →

• After destructuring, the Variable 'Rajesh' will hold the value `"Jha"`.

• However, there is no Variable named 'fName' declared in the code. The original 'fName' property is just used to extract the Value and doesn't create a new Variable 'fName'.

3. The Error →

• When we try to 'console.log (fName)' JS will look for a Variable named 'fName' in current Scope.

• Since 'fName' is not defined anywhere in code, a 'Reference Error' will be thrown

Therefore, the output will be →

• Reference Error : fName is not defined correct method →

- Const {fName: Rajesh} = {fName : "Jha"};
  Console.log (Rajesh);

→ If we wanted to log the value "Jha", we should log the variable 'Rajesh' instead

OR

```
Const {fName} = {fName : "Jha"}
console.log(fName);
```

(70)
```
function sum(n1, n2 = n1) {
    console.log (n1 + n2) ;
}
Sum(10);
```

Ans:→ 1. Function Def ('sum (n1,n2 = n1')): →

- The 'sum' function takes two parameters, 'n1' and 'n2'.
- The second parameter, 'n2', has a default value which is set to 'n1'.
- This means that if we don't provide a value for 'n2' when calling the function, 'n2' will automatically take the value of 'n1'.

2. Function Call ('sum(10)') : →
- Here, we are calling the 'sum' function with a single argument '10'.

Scanned with CamScanner

- Since 'n2' is not provided in the function call, it defaults to the value of 'n1', which is '10'.

3. Inside the function →
   - The function calculates 'n1 + n2'.
   - Since both 'n1' and 'n2' are '10', the result of addition is '20'.

Therefore, the output will be →

- 20

For more questions, Visit →

github → rajeshjha2000

Scanned with CamScanner