LEARN JAVASCRIPT

🔥 Elevate Your JavaScript Game with These 10 Function Exercises! 🔥

Coding Questions and explanations!

What's Inside?

🛠️ From basic function declarations to advanced concepts like closures and arrow functions.

🎯 Hands-on exercises with complete explanations.

🧩 Challenges that test and expand your understanding of JavaScript functions.

Tackle these exercises to deepen your understanding and mastery of JavaScript functions - a fundamental building block in any JavaScript programmer's toolkit.

🏗️

Question: Write a function named add that takes two arguments and returns their sum.

Answer:

```
function add(a, b) {
  return a + b;
}
console.log(add(2, 3)); // 5
```

Explanation: This is a basic function declaration in JavaScript. The function add takes two parameters a and b and returns their sum. When add(2, 3) is called, it returns 5.

Question: How does a function expression differ from a function declaration?

Answer:

Function Declaration:

```
function hello() {
```

```
  console.log("Hello");
}
```

Function Expression:

```
const hello = function() {
  console.log("Hello");
};
```

Explanation: A function declaration defines a function with the specified parameters, whereas a function expression assigns a function to a variable. Function expressions are not hoisted, meaning they cannot be called before they are defined in the code.

Question: What is an Immediately Invoked Function Expression (IIFE) and provide an example?

Answer:

```
(function() {
  console.log("This function runs right away!");
})();
```

Explanation: An IIFE is a function that runs as soon as it is defined. The function is wrapped in parentheses, and the last two parentheses call the function immediately after it is defined.

Question: Write a function square that returns the square of a number and use it to calculate the square of 4.

Answer:

```
function square(num) {
  return num * num;
}
console.log(square(4)); // 16
```

Explanation: The square function takes a single argument num and returns its square by multiplying num by itself. square(4) returns 16.

Question: How can you create a function in JavaScript that remembers the state of variables from its last execution?

Answer:

```
function counter() {
  let count = 0;
  return function() {
    count += 1;
    return count;
  };
}
const myCounter = counter();
```

```
console.log(myCounter()); // 1
console.log(myCounter()); // 2
```

Explanation: This is an example of a closure. The counter function contains another function that increments and returns a count variable. The state of count is preserved between function calls due to the closure.

Question: What are default parameters in a function? Give an example.

Answer:

```
function greet(name = "Guest") {
   return "Hello, " + name;
}
console.log(greet("Alice")); // "Hello, Alice"
console.log(greet());       // "Hello, Guest"
```

Explanation: Default parameters allow functions to have predefined values if no values or undefined are passed. In the greet function, name defaults to "Guest" if no argument is provided.

Question: Write a function multiply that uses the rest parameter to take an arbitrary number of arguments and returns their product.

Answer:

```
function multiply(...numbers) {
  return numbers.reduce((acc, num) => acc * num, 1);
}
console.log(multiply(2, 3, 4)); // 24
```

Explanation: The rest parameter syntax (...numbers) allows us to represent an indefinite number of arguments as an array. The reduce method is used to multiply each element of the numbers array together.

Question: Explain arrow function syntax with an example.

Answer:

```
const add = (a, b) => a + b;
console.log(add(5, 3)); // 8
```

Explanation: Arrow functions provide a more concise syntax for writing function expressions. They do not have their own this, arguments, super, or new.target bindings, making them suitable for non-method functions.

Question: How do you create a function in JavaScript that cannot be used as a constructor?

Answer:

```
const myFunc = () => {
  console.log("This is a non-constructor function");
```

```
};
// new myFunc(); // TypeError: myFunc is not a constructor
myFunc(); // "This is a non-constructor function"
```

Explanation: Arrow functions cannot be used as constructors and will throw an error when used with new. This is because arrow functions do not have a prototype property.

Question: Write a function isEven that returns true if a number is even and false if it is not, using the ternary operator.

Answer:

```
function isEven(num) {
  return num % 2 === 0 ? true : false;
}
console.log(isEven(4)); // true
console.log(isEven(5)); // false
```

Explanation: The ternary operator (condition ? exprIfTrue : exprIfFalse) is used to return true if num % 2 equals 0 (the number is even), and false otherwise.

These exercises cover a variety of concepts related to JavaScript functions, including basic syntax, closures, arrow functions, and more, providing a practical understanding of how functions work in JavaScript.