Difference in Handling `arguments` Between Regular Functions and Arrow Functions in JavaScript

1. Regular Functions and `arguments`

- **Regular functions** in JavaScript have access to the `arguments` object. This object is array-like and contains all arguments passed to the function.
- The `arguments` object allows you to retrieve the parameters without explicitly listing them in the function definition.

Example:

```javascript
function logArguments() {
  console.log(arguments); // Logs the arguments passed to the function
}

logArguments(1, 2, 3); // Logs: [1, 2, 3]
```

- In the example above, `logArguments(1, 2, 3)` logs `[1, 2, 3]` because `arguments` refers to the function's passed parameters.

2. Arrow Functions and `arguments`

- **Arrow functions** do **not** have their own `arguments` object. If you try to use `arguments` inside an arrow function, it throws a `ReferenceError`.
- This is because arrow functions do not create their own `this` or `arguments` scope. They inherit `this` and `arguments` from their parent scope (lexical context).

Example:

```javascript
const logArguments = () => {
  console.log(arguments); // ReferenceError: arguments is not defined
};

logArguments(1, 2, 3); // Throws a ReferenceError
```

3. Solution Using Rest Parameters (`...args`)

- To access arguments in an arrow function, you can use **rest parameters** (`...args`), which is a modern feature in JavaScript. Rest parameters collect all function arguments into an array.

Example:

```javascript
const logArguments = (...args) => {
  console.log(args); // Logs the arguments as an array
};

logArguments(1, 2, 3); // Logs: [1, 2, 3]
```

- In this case, `args` is an array containing `[1, 2, 3]`.

Key Takeaways:

- **Regular Functions**: Can access the `arguments` object directly.
- **Arrow Functions**: Do not have their own `arguments` object. Use rest parameters (`...args`) to collect arguments in arrow functions.