JavaScript Essentials:

1. Arrow Functions

Arrow functions were introduced in 2015 and have had a huge impact on how JavaScript code is written.

I think this change was very positive, and now you rarely see the old way of writing functions in modern code.

It's a simple and nice change that lets you write functions in a shorter way, from:

Let me know if you'd like me to continue rewriting the rest of the text!

Syntax: const functionName = (parameters) => { code }

Example:

```
const add = (a, b) => a + b;
console.log(add(2, 3)); // Output: 5
```

- **this working in arrow function**

When defined as a method of an object, in a regular function this refers to the object, so you can do:

```
const person = {
  name: 'John',
  sayHello: function() {
    return `Hello, my name is ${this.name}!`
  }
}
```

2. Template Literals

Template literals are a new ES2015 / ES6 feature that allows you to work with strings in a novel way compared to ES5 and below.

Syntax: Use backticks ` and ${expression} for interpolation.

Example:

```
const name = 'John';
const greeting = `Hello, ${name}!`;
console.log(greeting); // Output: Hello, John!
```

3. Destructuring Assignment

Object Destructuring:

Given a dictionary, using the unpacking syntax you can extract just some values and put them into named variables:

```
person = {
  'name': 'John',
  'age': 30,
  'city': 'New York'
}

name, age = person.values()

print(name)   # John
print(age)    # 30
```

Array Destructuring:

Array destructuring is a way to extract values from an array and assign them to individual variables.

```
let arr = [1, 2, 3, 4, 5];

let [a, b, ...rest] = arr;

console.log(a);   // 1
console.log(b);   // 2
console.log(rest);   // [3, 4, 5]
```

In the example, the ... is called the "rest parameter" or "spread syntax".

4. Spread and Rest Operators

Spread Operator: Used to expand elements.

```
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5];
console.log(arr2);
// Output: [1, 2, 3, 4, 5]


If you think of assigning this way :

const arr2 = [arr1, 4, 5];

It means const arr2 = [[1,2,3], 4, 5];
```

Rest Operator: Used to collect multiple elements into an array.

```
const sum = (...numbers) => {
  return numbers.reduce((acc, number) => acc + number, 0);
};
console.log(sum(1, 2, 3, 4)); // Output: 10
```

5. Modules (import/export)

Exporting:

```
// module.js
export const greet = (name) => `Hello, ${name}!`;
```

Importing:

```
// main.js
import { greet } from './module.js';
console.log(greet('John')); // Output: Hello, John!
```

6. Classes and Inheritance

Class Declaration:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  greet() {
    return `Hello, my name is ${this.name}.`;
  }
}
```

Inheritance:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  greet() {
    return `Hello, my name is ${this.name}.`;
  }
}
```

7. Promises and Async/Await

Promises:

```
const fetchData = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Data fetched');
    }, 2000);
  });
};
fetchData().then((data) => console.log(data)); // Output: Data
fetched
```

Async/Await:

```
const fetchData = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('Data fetched');
    }, 2000);
  });
};
const getData = async () => {
  const data = await fetchData();
  console.log(data); // Output: Data fetched
};
getData();
```

8. Default Parameters

Syntax:

```
const greet = (name = 'Guest') => `Hello, ${name}!`;
console.log(greet()); // Output: Hello, Guest!
console.log(greet('John')); // Output: Hello, John!
```