@CODE.CLASH

# This

JAVASCRIPT

# this keyword In JS.

Hey Devs! If you've ever come across the "this" keyword in JavaScript and found it confusing, don't worry.

- We've got you covered! Understanding "this" is crucial to unlocking the full potential of JavaScript.
- In this post, we'll unravel the mysteries of the "this" keyword and explore its various applications in different situations including arrow functions.

# The Global Context

Imagine you're in a vast playground called JavaScript. When you're outside of any function or object, the "this" keyword refers to the global object.

- In a browser environment, this global object is usually the window object.
- Let's take a look at an example to make it clearer:

```
console.log(this);
// Output:  the global object (window in the browser)
```

In this case, It's like looking at the entire playground from a distance.

# Object Method Context

When "this" is used inside a method of an object, it refers to the object itself. Think of it as the object speaking about itself.

- This allows you to access other properties and methods within the same object.
- Here's a simple example to illustrate this point:

```
const person = {
  name: "Imtiyaz",
  introduce: function() {
    console.log("Hello, my name is " + this.name);
  }
};

person.introduce();
// Outputs: Hello, my name is Imtiyaz
```

# Function Context

The behavior of "this" can get a bit tricky when used inside regular functions. Its value depends on how the function is called:

- If the function is called directly (as a standalone function), "this" refers to the global object (window in the browser).
- If the function is called as a method of an object, "this" refers to the object itself.
- If the function is called using the "new" keyword to create an instance of a constructor function, "this" refers to the newly created object.

```javascript
function sayHello() {
  console.log("Hello, " + this.name);
}


// 1. Standalone function call
sayHello()
// Output: Hello Undefined (refer to Global object)


// 2. Function called as a object method
const person = {
  name: "Imtiyaz",
  greet: sayHello
};

person.greet();
// Outputs: Hello, Imtiyaz


// 3. Constructor function context
function Person(name) {
  this.name = name;
}

const person3 = new Person("Imtiyaz");
console.log(person3.name);
// Outputs: Imtiyaz
```

# Event Handlers

In JavaScript, when an event is triggered, "this" often refers to the DOM element that triggered the event.

- It allows you to interact with or access properties of that particular element. Let's have a quick look:

```
const button = document.querySelector("button");

button.addEventListener("click", function() {
  console.log("Button clicked!");
  console.log(this); // Output: the button element
});
```

# Arrow Functions

Arrow functions have a different behavior compared to regular functions when it comes to the "this" keyword.

- They do not have their own "this" context but instead inherit it from the surrounding code.
- In simpler terms, "this" inside an arrow function refers to the value of "this" in the parent scope. Let's see an example:

```
const person = {
  name: "Imtiyaz",
  introduce: function() {
    const greet = () => {
      console.log("Hello, my name is " + this.name);
    };
    greet();
  }
};

person.introduce();
// Outputs: Hello, my name is Imtiyaz
```