that → variable

this → Keyword → refer to object.
(cannot change value.)

Q) In object Method we this → object.

1) Alone → Global object

2) In function

3) In function → in Strict mode → this → undefined

4) In Event → refer to element → that → received the event (triggered)

5) Methods → call()
→ apply()
→ bind()
} → refer → this → to any object.

Q1)

```
const person = {
    firstName: "John",
    lastName: "Doe",
    id: 5556,
    full: function() {
        return this.firstName + " " +
               this.lastName;
    }
};
```

→ person.full();
→ o/p : John Doe

replace with

```
return this;
```

→ You get o/p
→ [object object]

Q2)

```
let x = this;
```

→ [object window]

```
function myFac() {
    return this;
}
```

refer → HTML element → that → received event

Q3)

```
<button onclick = "this.style.display = 'none'">Click
</button>
```

6) this precedence

1 bind() 2 apply() call() 3 Object Method 4 Global Scope

7) Is Arrow Function

});

mutual.then(function sucval(result){
console.log(result);
})
.catch(function errval(result){
console.log(result);
}
}

O/P
Promise Rejected

This keyword

1) Global Context :- ~~console.log~~ alert(this); → [object window]

2) Object Method context :- This used inside method of object it refers to object itself.

3) Function Context

JULY						2020
Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

* If function called directly → this → global object (window in browser)

* If function called as a Method of object → this → refers to object itself.

* If function called using new keyword → this → refers to newly created object.

Instance ← Create ← new Keyword

↓ of

Constructor function

Eg 1)

```
function SayAge() {
    console.log("Age is " + this.name);
}
SayAge();
```

O/P: Age is undefined

Event → (body)

```
<button class="button1">Click</button>
<script>
const but = document.querySelector("button1");
but.addEventListner("click", function() {
    console.log("Element Clicked");
    console.log(this.toString());
})
</script>
</body>
```

O/P

click → Element Clicked

[Object HTMLButton Element]

Arrow functions

→ No own "this" context; inherit → from → surrounding code

→ this → inside → Arrow function → refer → value of this → in → Parent scope;

Eg..

```
Const person = {
    Design: "D9001",
    age: 21,
    id: 210,
  introduce: function() {
        const greet = () => {
                console.log(this.Design);
        };
     greet();
   }
};
person.introduce();
```

O/P

D9001

→ simpler

Js this

- Imagine → var1 playground → Js — refer → global object
- When you're outside of any function/object → this keyword
- The global object in → Browser environment → Window
- Eg1)

```
<script>
    let one = document.getEle..ById("one");
    one.innerHTML = this;
</script>
```

O/p :: [object]Window

- When this → used → inside → method of an object — refers to object itself

Think of it as object speaking about itself.

Eg2)

```
<script>
let two = doc.getEleId("two");
let three = doc.getEleId("three");
//object
const personalDetails = {
    qualificId : 120899,
    age : 25,
    letterofId : function() {
        two.innerHTML = this;
        three.innerHTML = `My current id is ${this.qualificId}`
    }
};
personalDetails.letterofId();
</script>
```

If	description
function called directly	this → refers → global object
function called as method of an object	this → refer → object itself
function called using new keyword create instance of const function	this → refers → Newly created object

```
<script>
    let one = document.getElementById("one");
    let two = document.getElementById("two");
    let three = document.getElementById("three");

function sayHello() {
    console.log(this);
    one.innerHTML = `Hello ${this.name}`;
}

sayHello();

//-----------

function sayHello2() {
    console.log(this);
    two.innerHTML = `Hello ${this.name}`;
}

const person = {
    name: "Gulraz",
    greet2: sayHello2
};

person.greet2();

//-----------

function Person(name) {
    console.log(this);
    this.name = name;
}

const person3 = new Person("Gurvan");
three.innerHTML = person3.name;
</script>
```

Hello

Hello Gulraz

Gurvan

Console

Window { }

{name: 'Gulraz', greet2: f}

Person {}

7 Event Handler

When event is triggered → this → refers → DOM element that triggered event.

```
<script>
    let one = doc.getEleById("one");
    let two = doc.getEleById("two");

    let btn = doc.querySelector("button");

    btn.addEventListner("click", function(){
        one.innerHTML = "Button clicked!";
        two.innerHTML = this;
    });
</script>
```

[Click To get value of this] ← Button click

→ You get O/P → Button clicked! [object HTMLButtonElement]

Arrow Functions

They → not have → their own → this context. Instead inherit from → surrounding code. means → Value of this in the parent scope.

```
<script>
    let one = doc.getEle-Id("one");
    let two = doc.getEleId("two");
    const summer = {
        name : "Heat",
        destiny : function(){
            const khujali = () => {
                two.innerHTML = `This value: ${this}`;
                console.log(this);
            }, one.innerHTML = `Hello ${this.name}`;
summer.destiny(); }; } khujali();
```

Console

{name: 'Heat', destiny: f}

O/P

Hello Heat

The value: [object Object]