**FAT ARROW FUNCTION**

**Fat Arrow Functions**  are relatively a new way of writing concise functions in javascript.

Fat arrow operator to quickly define javascript functions qith or without parameters.

Arrow functions allow us to write shorter function.

SYNTAX:=()=>{

};

They have been introduced by the ECMA Script 6 specifications and since then become the most popular ES6 feature.

We are able to omit the curly braces and the function and return keywordss when creating a new javascript function to write shorter function syntax.

They are not only time savers but also produce clean and more readable functions.

**EXAMPLE:**

function hello()

{

return "hi tejasswini";

}

let x=hello();

console.log(x);

**FAT ARROW FUNCTION:**

hello = () => {

return "hi tejasswini";

}

let x=hello();

console.log(x);

If the function has only one statement and the statement returns a value,you can remove the brackets and the return keyword:

hello = () => "hi tejasswini";

console.log(hello());

**THIS WORKS ONLY IF THE FUNCTION HAS ONLY ONE STATEMENT.**

**NO PARAMETERS:**

The simplest arrow function syntax is when function doesn’t hava any parameters.

Eg:

Function birthday()

{

Console.log(“happy birthday”);

}

Birthday();

With fat arrow function:

Eg:

let birthday=()=>{

console.log(“happy birthday”):

};

birthday();

the empty parantheses indicate that the function doesn’t have any parameters.

**PASSING ONE PARAMETER:**

function birthday(name){

console.log(“happy birthday”+name);

}

birthday(tejaswini);

**PASSING ONE PARAMETER BY USING FAT ARROW FUNCTION:**

**The arrow function syntax essentially does the same thing.we need to enclose the name parameter within the parantheses before the fat arrow function.**

let birthday=(name)=>{

console.log("happy birthday"+name);

}

birthday("tejaswini");

**PASSING TWO PARAMETERS:**

We can use arrow function with as many parameters as we want .for instance,here is the same example with two parameters,age and name.

Now the birthday() function requires two inputs to return the string.

Eg:

function birthday(age,name){

return “happy”+age+”th birthday,”+name;

}

let result=birthday(age,name);

console.log(“result”);

**PASSING TWO PARAMETERS WITH RETURN KEYWORD BY USING FAT ARROW FUNCTION:**

let birthday=(age,name)=>{

return "happy"+age+"th birthday,"+name;

};

console.log(birthday(25,"tejaswini"));

**ADVANTAGES :**

Shorter code/concise code.

Implicit return.

Minimal object-This keyword.

Function overriding conflicts will resolve.

**SHOTER CODE:**

Eg:

//function declaration

Function hello()

{

Console.log(“hello”);

}

//call the function

hello();

Eg:

Hello=()=>”hello”;

Console.log(hello());

**IMPLICT RETURN:**

If your function has only one return statement ,use implicit return .

Eg:

Let add=(a,b)=>{

return a+b;

}

Let result=add(10,20);

Console.log(result);

Eg:

let add=(a,b)=>a+b;

result=add(10,20);

console.log(result);

**MINIMAL OBJECT-This KEYWORD:**

**FUNCTION OVERRIDDING CONFLICT WILL RESOLVE:**

**Eg:**

Function wish()

{

Console.log(“gm”);

}

Wish();

Function wish(){

Console.log(“gn”);

}

Wish();

Output: gn

gn

Eg:

Function wish(){

Console.log(“gm”);}

Wish();

Function wish(){

Console.log(“gn”);}

Wish();

Output: gm

gn

**DISADVANTAGES:**

Although arrow functions have many use cases, they come with some limitations as well. There are some scenarios when we can’t or shouldn’t use the arrow function syntax.Most importantly, we can’t create constructors with arrow functions. If we use the new keyword together with the fat arrow syntax, the console will throw an error. This happens because JavaScript treats arrow functions in the same way as object methods. As a result, they can’t have their own methods, which would be an essential characteristic of JavaScript objects.

**CONCLUSION:**

Arrow functions provide developers with a useful shorthand to write function expressions in JavaScript. If used consistently, the new compact notation can save a lot of time and bandwidth, and improve code readability.