Lesson:

Arrow function

PW SKILLS

Topics Covered

- Introduction to Arrow functions.
- Arrow Function Syntax.
- Why should developers use arrow functions?

Introduction to Arrow Functions

Arrow functions, introduced in ES6, is a straightforward and compact way of writing JavaScript functions. They improve the structure and readability of our code by providing a more concise syntax.

Arrow functions, also known as lambda functions, are special functions in JavaScript that don't have a name and are not tied to a specific identifier. They don't return any values and can be declared without using the "function" keyword.

Syntax of an arrow function.

```JavaScript
(parameter1, parameter2, ..., parameterN )=>{
// function body
return expression; // optional
}
```

- They use a simplified syntax with the “=>” arrow.
- They don’t require the function keyword
- If the function body contains only a single expression, the curly braces `{}` and “return” statements can be omitted. The result of the expression is automatically returned.
- If the function has a single parameter, the parentheses around the parameter can be omitted. However, parenthesis are required when there are zero or multiple parameters.

Arrow functions can be written in different ways depending on the number of parameters they accept and the operation it performs.

Here are simple examples to illustrate the usage of arrow functions

```JavaScript
// 1. One parameter, and a single return statement

const square = x => x * x;

// 2. Multiple parameters, and a single return expression

const sumOftwoNumbers = (x, y) => x + y;

// 3. Multiple statements in function expression

const sum = (x, y) => {
  console.log(`Adding ${x} and ${y}`);
  return x + y;
};

// 4. Returning an object

const sumAndDifference = (x, y) => ({ sum: x + y, difference: x -
y });

// CALLING A FUNCTION
let output1 = square(5); //OUTPUT: 25
let output2 = sumOftwoNumbers(1, 2); // OUTPUT: 3
let output3 = sum(1, 2);
// OUTPUT: Adding 1 and 2
// returned 3

let output4 = sumAndDifference(5, 3);
// OUTPUT: { sum: 8, difference: 2 }
```

Features of Arrow Function Syntax

- Parentheses are optional in the case of a single parameter.
- Must use parentheses in case of multiple parameters.
- The return keyword is not required for a single return expression in the function body.
- The return keyword is required in case of multiple statements in the function.
- To return an object notation, wrap it with parentheses.

Arrow functions are typically used when a function is being passed as an argument to another function, or when a function is being assigned to a variable. They can also be useful for creating simple, one-line function expressions. However, they should not be used when defining methods on an object.

Limitation for arrow function

- Arrow functions cannot access the argument object.
- Arrow functions do not have the prototype property
- Arrow functions cannot be used with the new keyword.
- Arrow functions cannot be used as constructors.
- These functions are anonymous, and it is hard to debug the code.

Arrow function vs regular function

Syntax:

- **Regular Function:** Defined using the function keyword followed by a name, p[arameter list, and function body.
- **Arrow function:** Defined using a concise syntax with the "=>" arrow operator, omitting the function keyword and using a more compact structure.

```javascript
// regular function

function fun(n1,n2){
 // function body
}
fun(2,4)

// arrow function
const fun=()=>{

}
fun(n1,n2)
```

Arguments object

- **Regular function:** Has access to the arguments object, which contains all the arguments passed to the function.
- **Arrow function:** Does not have its own arguments object instead it inherits the arguments object from the enclosing scope.

```javascript

// regular function
function regularFunction() {
  console.log(arguments); // Output: [1, 2, 3]
}

regularFunction(1, 2, 3);

// Arrow function
const arrowFunction = () => {
  console.log(arguments); // Uncaught ReferenceError: arguments
is not defined
};

arrowFunction(1, 2, 3);
```

Binding of “this”

- **Regular function:** Has its, own **“this”** value, which is determined by how the function is called. The value of “this” can change dynamically depending on the context of invocation.
- **Arrow function:** Inherits the **“this”** value from the enclosing lexical scope. It does not have it's own **“this”** binding and captures the **“this”** value of the surrounding context.

```JavaScript
const person = {
   name: 'John',
   greet: function() {
      console.log('Hello, my name is ' + this.name);
   },
   greetTwo: () => {
      console.log('Hello, my name is ' + this.name);

   }
};

person.greet(); // Output: Hello, my name is John
person.greetTwo(); // Output: Hello, my name is
```

Note : “this” keyword will be explained in detail in advance javascript module

Use for new keyword

- **Regular function:** can be used as constructor function with the new keyword to create an instance of objects
- **Arrow function** cannot be used as a constructor function Attention to use new with an arrow function will result in a runtime error.

```JavaScript
// Regular function
function RegularFunction(name) {
  this.name = name;
}

const regularObj = new RegularFunction('John');
console.log(regularObj.name); // Output: John

// Arrow function
const ArrowFunction = (name) => {
  this.name = name;
};

const arrowObj = new ArrowFunction('John');
console.log(arrowObj.name); // Output: TypeError:
ArrowFunction is not a constructor
```