Arrow Function

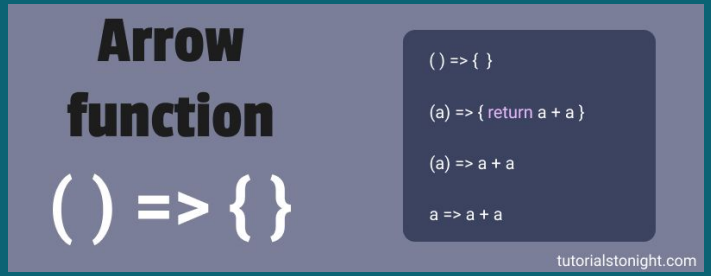
Presented By: Amna Shahzad
Rabia Yousuf

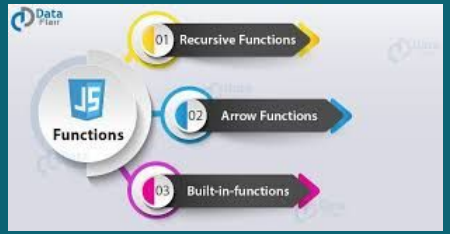
Topics

- Introduction
- Traditional function syntax
- What is arrow Function
- Arrow Function syntax
- Examples
- Advantages
- Uses
- Conclusion

Introduction

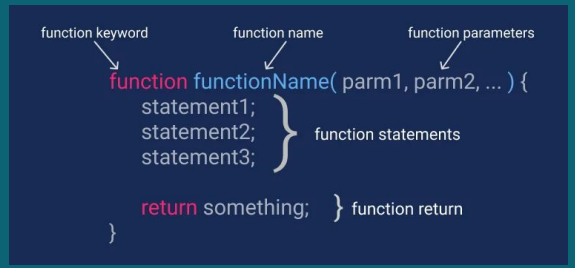
An arrow function expression is a compact alternative to a traditional function expression, with some semantic differences and deliberate limitations in usage: Arrow functions don't have their own bindings to this , arguments , or super , and should not be used as methods

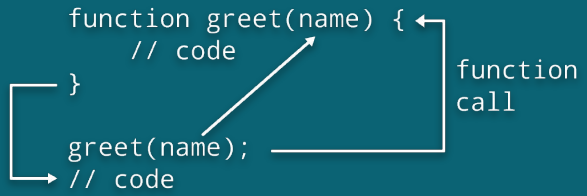


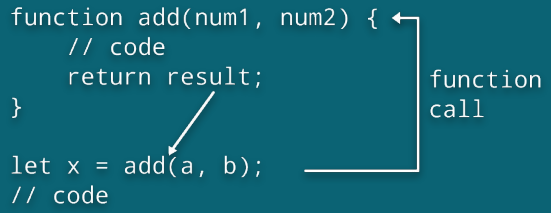


Traditional function Syntax

In a traditional function, there is an arguments local variable. If the number of arguments to our function is dynamic, using arguments allows us to easily do things like calculating the maximum number of passed arguments: function traditionalFn() { return Math.max(...arguments); }// 3







Example: Traditional function Syntax

```
function addNumbers(a, b) {

  return a + b;

}

var result = addNumbers(5, 3);

console.log(result);
```

// Output: 8

```
function greet(name) {
  console.log("Hello, " + name +
"!");
}

greet("John");
greet("Jane");
```

// Output: Hello, John!

What is Arrow Function

Arrow function is one of the features introduced in the ES6 version of JavaScript. It allows you to create functions in a cleaner way compared to regular functions. For example, This function // function expression let x = function(x, y) { return x * y; }

```
// arrow function expression
const add = (a, b) => {
    return a + b;
}
// very simple and concise syntax
const add = (a, b) => a + b;
// eliminates (), if it has single parameter
const square = a => a * a;
```

```
// ES5
var add = function (num1, num2) {
    return num1 + num2;
}

// ES6
var add = (num1, num2) => num1 + num2
```

Example: Arrow Function

```
const square = x => x * x;

console.log(square(5));
```

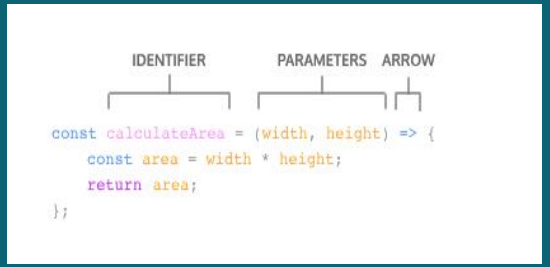
// Output: 25

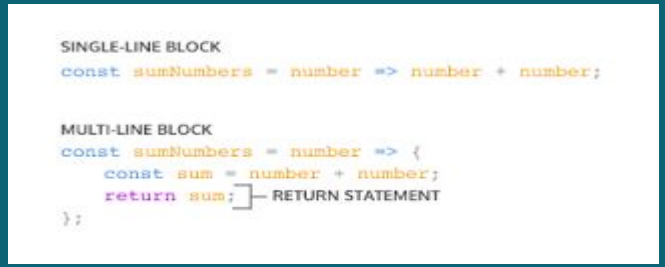
```
const obj = {
  name: 'John',
  sayHello: function() {
    setTimeout(() => {
      console.log(`Hello, ${this.name}!`);
    }, 1000);
  }
};

obj.sayHello();
```

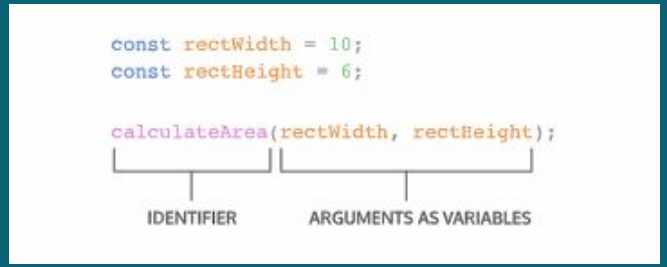
// Output: Hello, John!

Arrow Function Syntax

The syntax of the arrow function is: let myFunction = (arg1, arg2, ...argN) => { statement(s) } Here, myFunction is the name of the function. arg1, arg2, ... argN are the function arguments.







Example: Arrow Function Syntax

```
const multiply = (a, b) => {
  const result = a * b;
  return result;
};
console.log(multiply(4, 6));
```

// Output: 24

```
const add = (a, b) => {
  return a + b;
};

console.log(add(2, 3));
```

// Output: 5

Examples: Single Statement

```
let add = function ( a , b )
{
        return  a + b;
}
console.log( add( 2, 3 ) );
```

```
let add = ( a , b )   =>  { return a + b; }
```

OR

```
let add = ( a , b ) => a + b ;
// no need to write return, if not using {}
console.log( add( 2, 3 ) );
```

Output ======= 5

Examples: No Argument

```
let greet = function ()
{
        return  "Good Morning";
}
console.log( greet() );
```

```
let greet = ()   =>  {
        return "Good Morning";
        }
```

OR

```
let greet = ( ) => "Good Morning";
console.log( greet() );
```

Output ======= Good Morning

Examples: One Argument

```
let square = function ( x )
{
        return x * x;
}
console.log( square( 5 ) );
```

```
let greet = x => x * x;

console.log( square( 5 ) );
```

Output ======= 25

Examples: With Expression

```
let welcome = function ( age )
{
        if ( age < 18){
          console.log('Baby');
        } else {
          console.log('Adult');
        }
 }
let age = 5;
welcome( age );
```

```
let age = 5;
let welcome = (age < 18) ?
  () => console.log('Baby') :
  () => console.log('Adult');

welcome();
```

Output ======= Baby

Examples: Multiple Statements

```
let sum = function ( a , b )
{
        let result = a + b;
        return result;
 }
let result1 = sum( 5,7 );
console.log( result1 );
```

```
let sum = ( a, b ) => {
    let result = a + b;
    return result;
}
// must use {} if function has multiple statements
let result1 = sum( 5 , 7 );
console.log( result1 );
```

Output ======= 12

Examples: Objects Literal

```
let setColor = function ( color )
{
        return { value: color } ;
 }

let backgroundColor = setColor('Red');
console.log( backgroundColor.value );
```

```
let setColor = ( color ) => {
    return ( { value: color } );
}
// must used () when return Object
Otherwise it returns undefined!
let backgroundColor = setColor('Red');
console.log( backgroundColor.value );
```

Output ======= Red

Examples: this Keyword

Arrow function does not have its own this, it refers to its parent scope

```
let Person = {
    name: 'Jack',
    age: 25,
    sayName: function () {
        console.log( “Outer Fun = ”,this.age );
        let innerFunc = () => {
            console.log(“Inner Fun = ”, this.age );
        }
        innerFunc();
    }}
```

```
Person.sayName();
```

Output

Outer Fun = 25

Inner Fun = 25

Examples: Arguments Binding

You pass arguments to a regular function, access them using the arguments keyword. .

```
let x = function ( )
{
        console.log( arguments );
}

x( 4,6,7 )
```

Output:
// Arguments [4, 6, 7]

```
let x = () => {
    console.log( arguments );
}

x( 4,6,7 );
```

Output:
// ReferenceError: Can't find variable: arguments

Limitations

- Doesn’t have its binding to this or super.
- Cannot be used as a function constructor, use new keyword to create a new object.
- Doesn’t have arguments object, new.target keyword, and prototype property.

Use Case

- Array manipulation:

  used with array methods like `map`, `filter`, and `reduce` to perform concise operations on arrays.

- Callback functions:

  Passing short, inline functions as callbacks to other functions i.e. setTimeout().

- Event handlers:

  Clean and concise way to handle events in JavaScript.

Conclusion

- Concise, shorthanded and expressive way to write functions
- Arrow function for Single Statement

  (...args) => expression;
- Arrow function for Multiple Statements

  (...args) => { statements }
- Doesn’t have its binding to this or super.
- Cannot be used as a function constructor, use new keyword to create a new object from
- Doesn’t have arguments object, new.target keyword, and prototype property.


ANY
QUESTIONS?