What is the difference between anonymous function and arrow function javascript?

In JavaScript, both anonymous functions and arrow functions allow functions to be created in a way that can be more functionally useful depending on the context. However, they have distinct differences in syntax and behavior.

Topic	Anonymous Function	Arrow Function
Syntax	An anonymous function is simply a function without a name. Anonymous functions can be defined using the function keyword. They can be assigned to variables, passed as arguments to other functions, or used in any place where a function is allowed.	Introduced in ES6 (ECMAScript 2015), arrow functions provide a more concise syntax for writing function expressions. They use an arrow (=>) to denote the function, eliminating the need for the function keyword. Arrow functions are always anonymous, meaning they cannot have a named identifier.

Example	const square = function(x) {
	return x * x;
	};
		const square = x => x * x;

this Binding	Anonymous functions have their own this context, which depends on how and where the function was called. In traditional function expressions, this is dynamically bound based on the execution context.	One of the key features of arrow functions is that they do not have their own this context. Instead, this is lexically inherited from the outer function where the arrow function is defined. This is particularly useful in scenarios involving callbacks, where maintaining the context of this is important.

Example	function Person() {
	this.age = 0;
	
	setInterval(function growUp() {
	this.age++; // `this` does not refer to Person object but to global object (or undefined in strict mode)
	}, 1000);
	}
		function Person() {
		this.age = 0;
		
		setInterval(() => {
		this.age++; // `this` correctly refers to the Person object
		}, 1000);
		}

UseCases	More traditional approach and familiar to those coming from other programming languages. Useful when you need a function that has its own this context or when you need to access the arguments object, which is not available in arrow functions.	Great for short expressions where you need this to be bound to the context of the enclosing scope. Commonly used in functional programming patterns due to its succinctness and clarity, especially in map, filter, and reduce operations on arrays.
Constructors	Can be used as constructors, meaning you can use them with the new keyword.	Cannot be used as constructors and will throw an error if used with the ***new*** keyword.

Example	const Person = function(name) {
	this.name = name;
	};
	
	const person = new Person("Alice");
		const Person = name => {
		this.name = name; // Error: Arrow functions cannot be used as constructors
		};

In summary, while both can be used for similar purposes, arrow functions offer a more compact syntax and easier handling of ***this*** within callbacks and methods where maintaining the context of ***this*** is necessary. However, when you need a function that requires dynamic ***this***, or when you need to use a function as a constructor, traditional anonymous functions are more appropriate.