---

**Java Script**

**Arrow Functions**

New in ES6, arrow functions offer concise syntax and alternative `this` binding.

**Basic Syntax**

Arrow function syntax depends on parameters & return values:

1. `(param 1, ..., param N) => expression`  
   **Parameter list**  
   Equivalent to `{return expression;}`

2. `(param 1, ..., param N) => {statements}`

The `{}` are required when dealing with statements & multi-line function bodies.

**Variations**

Syntax also depends on context & stylistic preferences. A few common examples:

- If there is only one parameter name, the `{}` are optional: `param => expression`
- If the function takes no parameters, write the parameter list as `(): ( => expression`  
- If the function returns an object literal, wrap output in `{}`: `param => ({ zip: zap })`

**Common Use Cases**

- **Array Manipulation**  
  `let names = cacti.map(cactus => cactus.name)`
  `let cacti = plants.filter(plant => plant.type === "cactus")`
- **Promises/Callbacks**
  `doThis().then(() => doThat()).catch(() => fixIt()).finally(() => finish())`

---

**What About this?**

Arrow functions inherit `this` from their outer context, allowing you to persist the scope of the caller without using `bind` or `var self: this`.

- Need dynamic `this`? Use regular functions.
- Need lexical `this`? Use arrow functions.

---

**ES5 vs. ES6**

**ES5 Function Expression**

```
function foo(a, b) {
  return a + b;
}
```

**ES6 Arrow Function**

```
(a, b) => a + b;
```

- Remove the `function` keyword.
- When the only statement is `return`, remove the `return` keyword & the enclosing `{}`.
- Arrow functions cannot be used as constructors.
- Arrow functions do not come with an `arguments` object.

**Lexical Scope**

```
function cactus() {
  this.age = 0;
  setInterval(() => {
    this.age++;
    3, 1000
  },
}
```

- `this` is the same value as in the enclosing function.
- The methods `bind`, `apply` & `call` have no effect on arrow functions.