dotnotes

JAVASCRIPT ARROW FUNCTIONS

⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒ ⇒

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

— ES5 vs. ES6 —

ES5 Function Expression

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

ⓘ Remove the `function` keyword.

ES6 Arrow Function

```
(a,b) ⇒ a+b;
```

ⓘ When the only statement is `return`, remove the `return` keyword & the enclosing `{}`.

— 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.

ⓘ Arrow functions cannot be used as constructors.

ⓘ Arrow functions do not come with an `arguments` object.

— 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.

Lexical Scope

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

`this` is the same value as in the enclosing function

ⓘ The methods `bind`, `apply` & `call` have no effect on arrow functions.