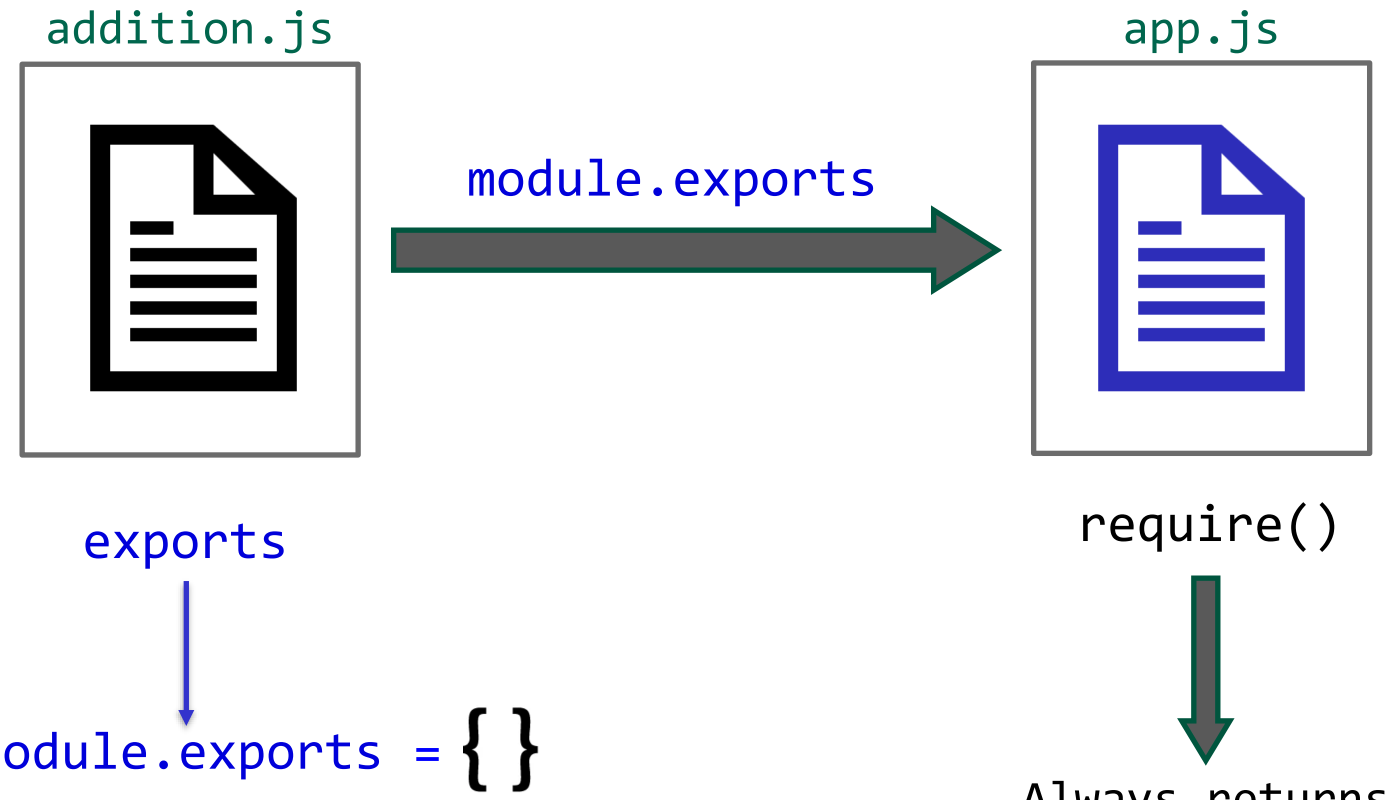
Module.exports vs exports

module.exports and exports

- The *module* is a plain JavaScript Object representing the current module.
- The module object has *exports* property which is a JavaScript variable, i.e. module.exports.
- The *exports* variable happens to be set to *module.exports*.
- When you assign something (literal, variable or function) to *exports* directly, you are not assigning it to the *exports* property of the *module* object.
- So *exports* is just a convenience feature so that the module authors (developers) can write less code.

Be very careful while using 'exports' directly!

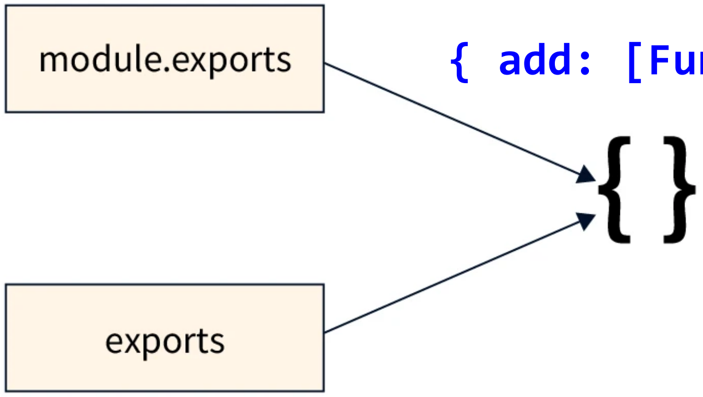
module.exports and exports



Initial value is
empty object

Always returns
module.exports

module.exports and exports

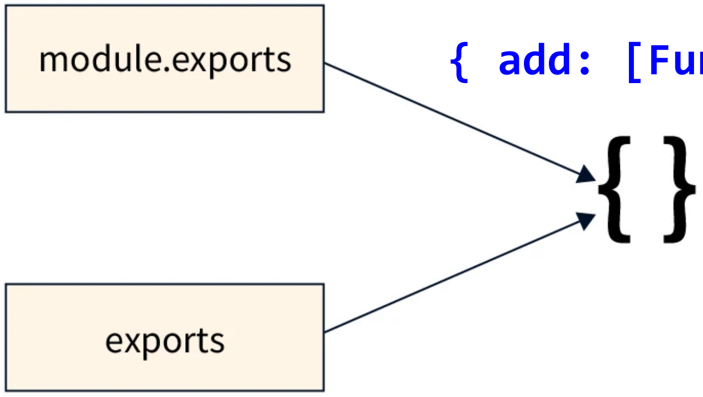


```
{ add: [Function: add] }
```

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

✓

```
module.exports.add = add;
```



```
{ add: [Function: add] }
```

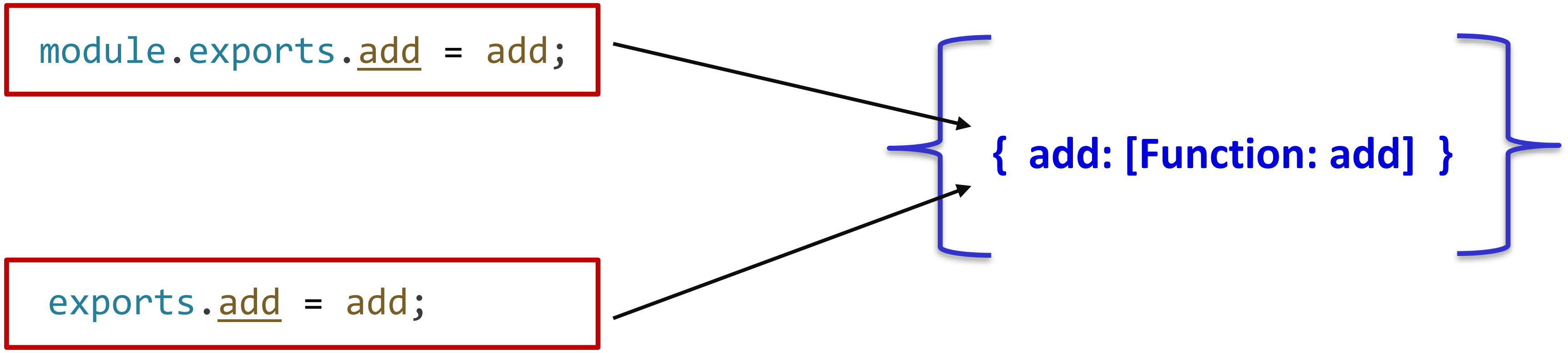
```
function add(a, b) {
    return a + b;
}
```

✓

```
exports.add = add;
```

module.exports and exports

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

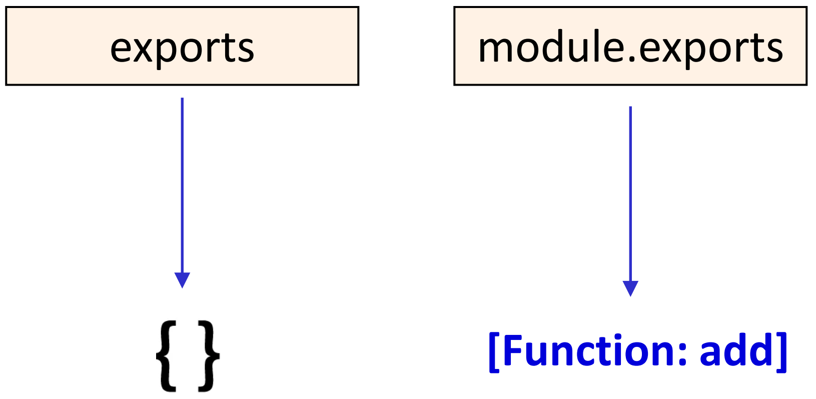


```
module.exports.add = add;
```

```
exports.add = add;
```

When we add properties to exports, we are actually adding properties to module.exports object, since exports points to module.exports.

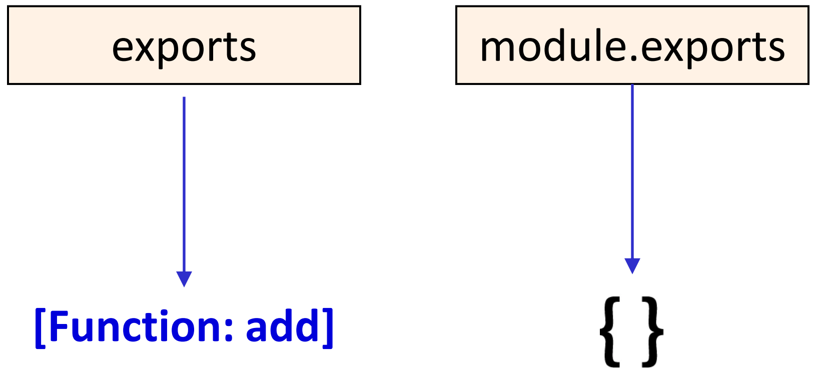
module.exports and exports



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

✓

```
module.exports = add;
```



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

✗

```
exports = add;
```

When we assign a new object to exports, the reference is broken and updating exports no longer updates module.exports

module.exports and exports

✓

- module.exports = {aFunction, anotherFunction}
- module.exports = aFunction
- module.exports.aFunction = aFunction
- module.exports.renamedFunction = aFunction

- exports.aFunction = aFunction
- exports.aFunction = {aFunction}

module.exports and exports

- exports = {aFunction, anotherFunction}
- exports = aFunction