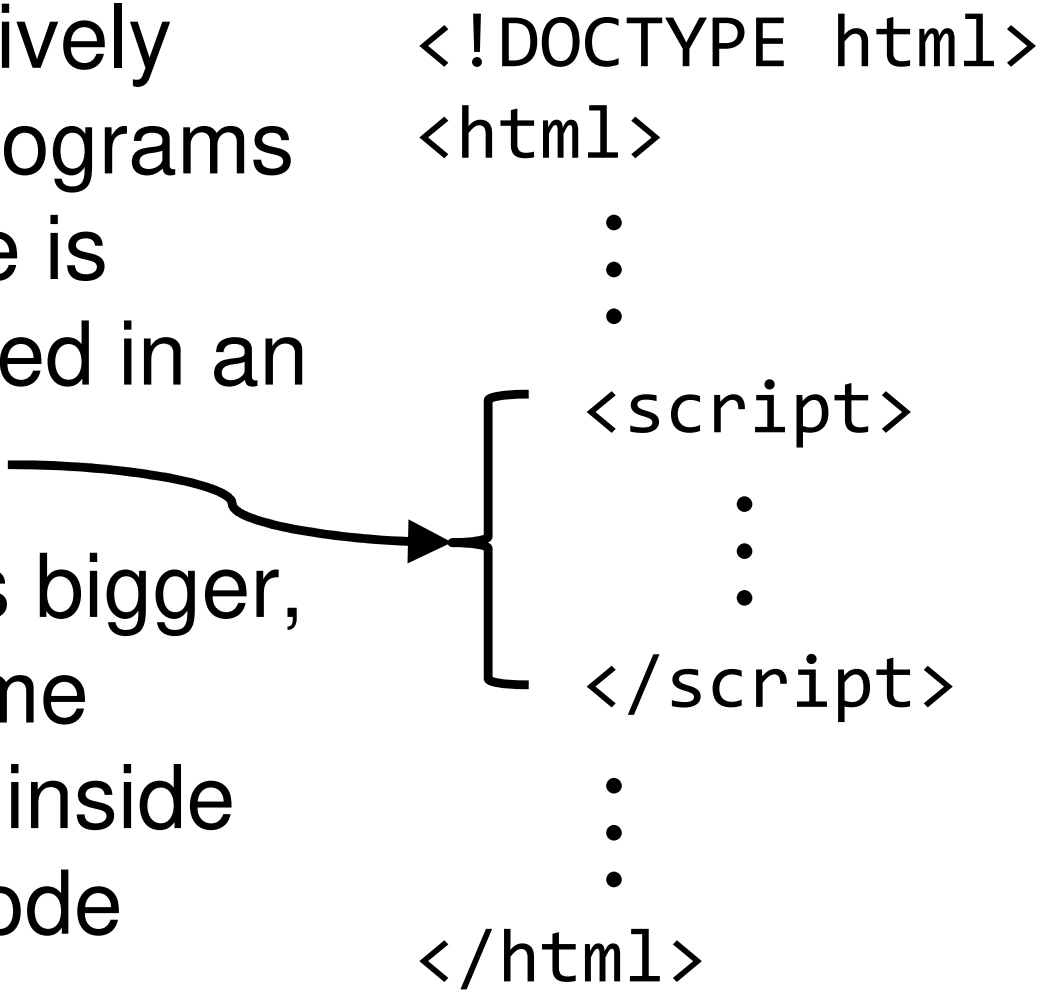
COMP4021
Internet Computing

Basic Module Pattern

Gibson Lam

Managing JavaScript Code

- We have built relatively short JavaScript programs so far and the code is mostly self-contained in an HTML script node
- If our program gets bigger, the code will become difficult to manage inside one single script node



Separating the Code

- One simple way is to separate the code into multiple script nodes or script files
- Or, a better way is to separate the code into modules / libraries so that:
  - You can manage the modules / libraries separately
  - Your code is better organized
  - You can re-use the modules / libraries in different JavaScript projects

Using Multiple JavaScript Files

- For example, the JavaScript code can be put into separate ‘module files’ and you can load them one by one, i.e.:

```
<!DOCTYPE html>
<html>
    ...
    <script src="module1.js"></script>
    <script src="module2.js"></script>
                        ⋮
    ...
</body>
</html>
```

Problem With Many JS Files

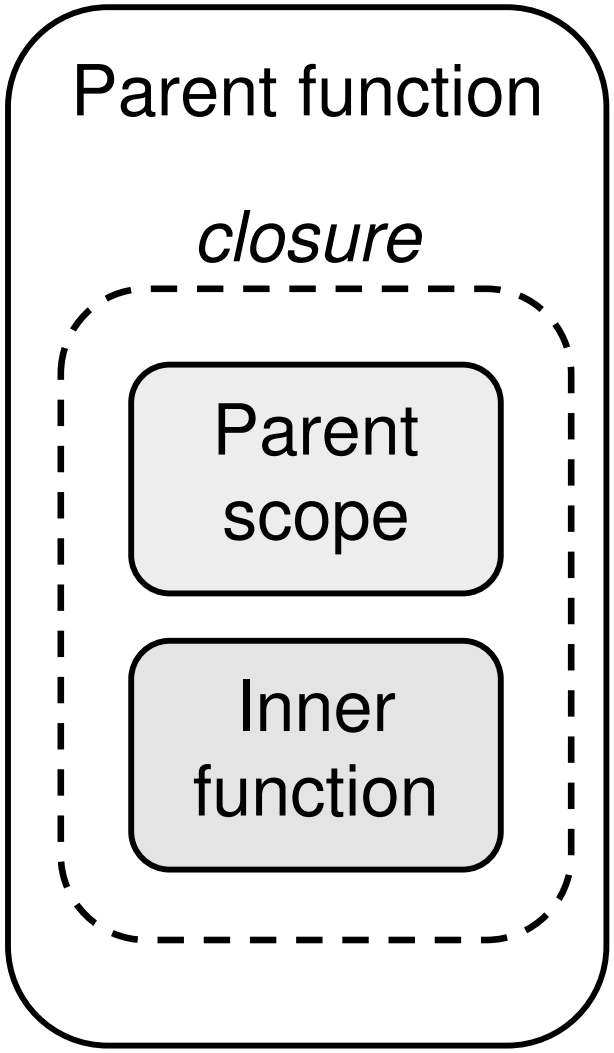
- Each module should contain its own variables and functions
- Let's assume each module file has 10 global variables
- Since global variables are shared within the same web page, there will be 50 global variables if you have 5 module files!

Using Module Pattern

- Module pattern is a JavaScript coding style that can avoid the problem of having too many global variables and functions
- It works similar to what a basic JavaScript object does, i.e. grouping things together
- But, in module pattern, you have a better management of properties and functions

Using Closures

- As you know, closures provide functions with a private working space
- Module pattern relies on the characteristics of closures to manage its variables and functions
- We will look at it in a step by step way by creating a `GameScore` module

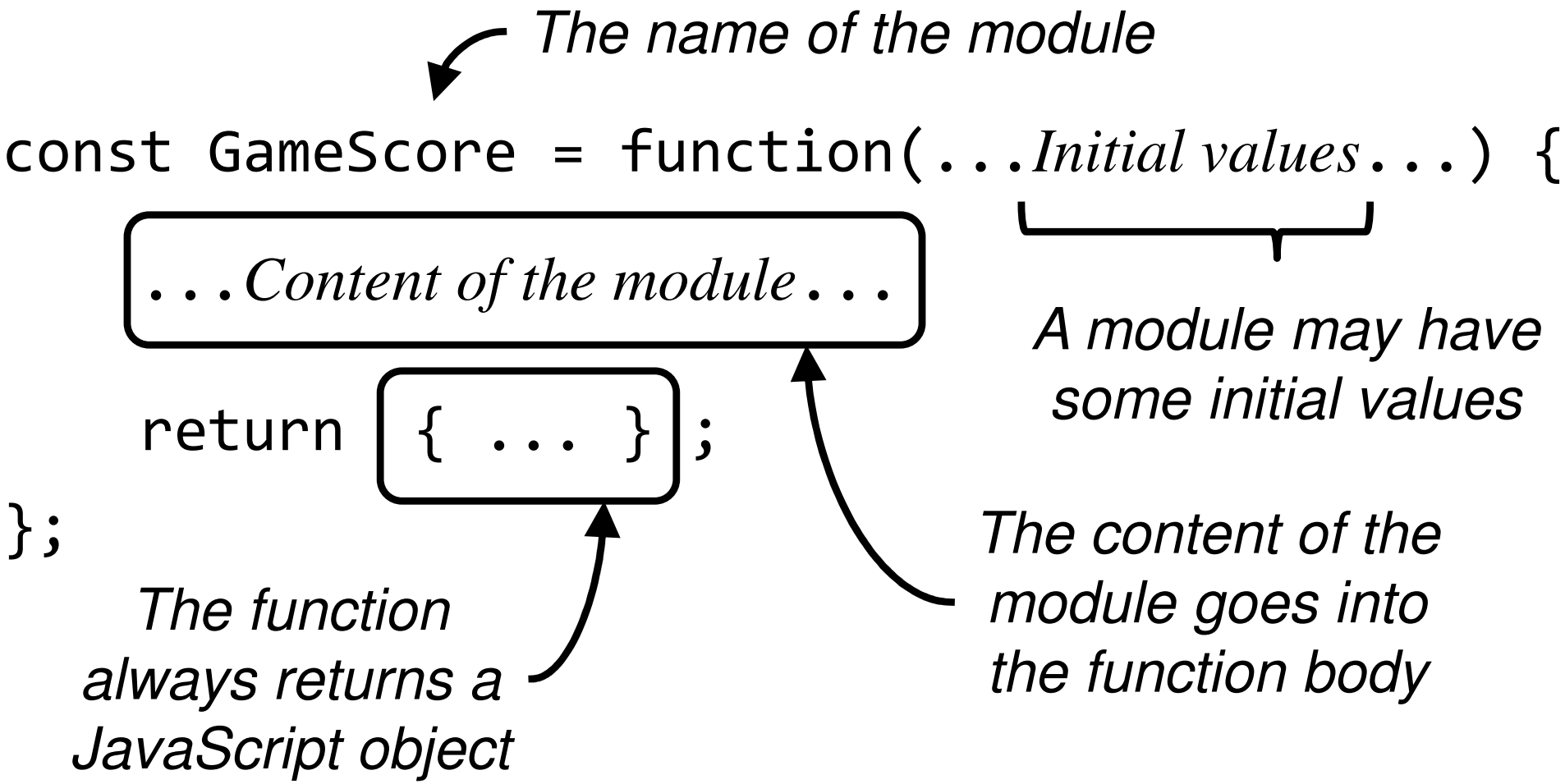


The `GameScore` Module

- Suppose that we need a module to help manage and show the score of a game
- We would want the module to:
  - Store the current score value
  - Let you increase / decrease the score
  - Update the score on the screen appropriately
- We will build this module using one of the module pattern designs

Building the Module

- To build a module, you use a JavaScript function structured like this:



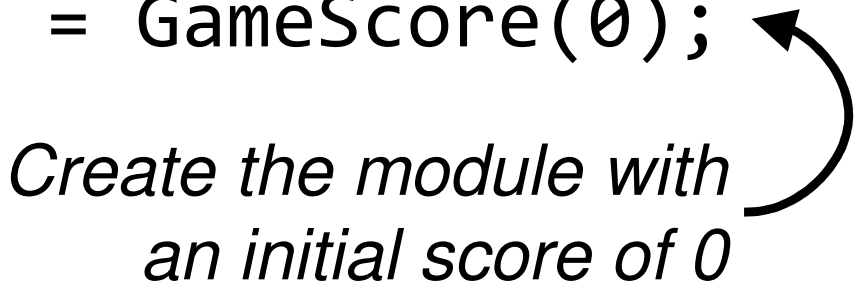
Initial Values

- You can put initial values into a module
- In the example, you may want to initialize the score with a number, i.e.:

```
const GameScore = function(initScore) {
    ...
};
```

- You then create the module using this code:

```
const gameScore = GameScore(0);
```



Content of the Module

- Once you set up the initial values, you can add some content into the module, which may include:
  - Initialization of the module
  - Properties of the module
  - Functions of the module
- We will show the properties and functions used by our `GameScore` module in the next few slides

The Module Properties

- We have two properties in the `GameScore` module: the current score and the HTML element for showing the score
- They are created inside the function as variables, i.e.:

Set the score to the initial score

```
let score = initScore;
const element = $("#score");
```

`const` *is used here because the element does not change*

Get and store the element to be used to show the score

The Module Functions

- The module has three functions:
  - `updateDisplay()` for showing the score in the web page whenever it is updated
  - `increase()` for increasing the score
  - `decrease()` for decreasing the score
- Here is the `updateDisplay()` function:

```
const updateDisplay = function() {
    element.text(score);
};
```

Show the current score in the HTML element

Increasing and Decreasing the Score

- The two functions to increase and decrease the score are shown on the right
- Note that both of them use the `updateDisplay()` function created in the previous slide

```
const increase = function() {
    score = score + 10;
    updateDisplay();
};

const decrease = function() {
    score = score - 10;
    if (score < 0)
        score = 0;
    updateDisplay();
};
```

The Module Content So Far

- Module properties and functions have been added inside the ‘module’
- However, you will not be able to use them!

All of them are local variables to the `GameScore` *function; they are inaccessible anywhere else*

```
const GameScore = function(initScore) {
    let score = initScore;
    const element = $("#score");

    const updateDisplay = function() {
        element.text(score);
    };

    const increase = function() {
        score = score + 10;
        updateDisplay();
    };

    const decrease = function() {
        score = score - 10;
        if (score < 0)
            score = 0;
        updateDisplay();
    };
};
```

The Returned Object

- To be able to use the module functions, you need to return them from the module
- This is done by putting them in a JavaScript object, for example:

```
return {
    increase: increase,
    decrease: decrease
};
```

Only two functions are returned, i.e. available for use

You can use the names of the module functions or any names you want

Using the Module

- As you have seen before, you create the module like this:

```
const gameScore = GameScore(0);
```

- You can then use the functions available from the returned object, for example:

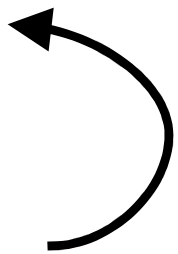
```
gameScore.increase();
```

Score: 0 ⇨ **Score: 10**

‘Private’ Data

- Unless the functions are returned by the module, you cannot use them, as shown here:

```
gameScore.updateDisplay();
```



Does not work as the function is not available outside the module

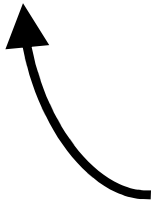
```
▶Uncaught TypeError: gameScore.updateDisplay
is not a function
```

- They become private data of the module which can only be accessed within the closure

IIFE

- Sometimes module pattern is initialized and created like this:

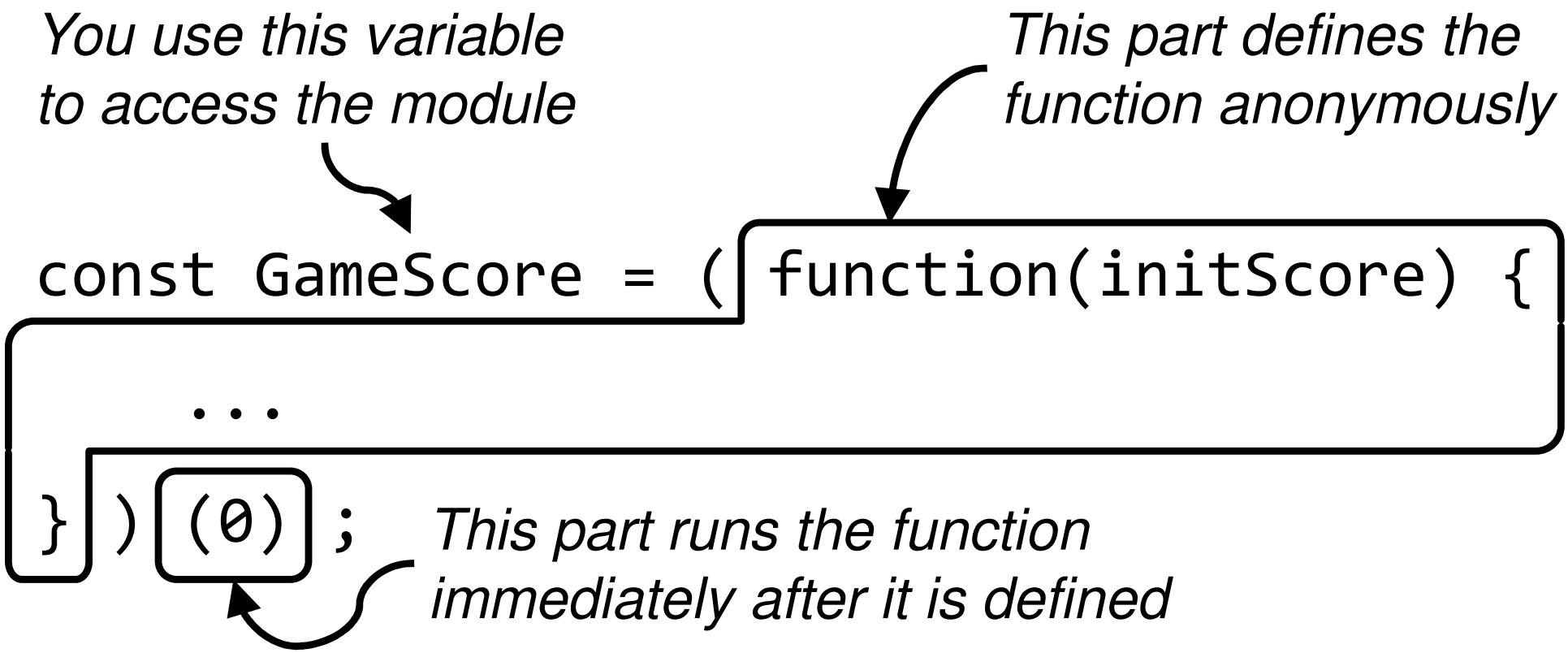
```
const GameScore = (function(initScore) {
    ...
})(0);
```



*This is the variable holding the module object, **not** the function*

- The function is created and run at the same time in one single statement
- This is called, IIFE (**I**mmediately-**I**nvoked **F**unction **E**xpression)

Using IIFE



- You use IIFE when you only use one instance of the module
- Only one variable is required to refer to the module