COMP4021
Internet Computing

About JavaScript Functions and Variables

Gibson Lam & David Rossiter

Deeper into JavaScript

- Let’s looks deeper at JavaScript
  - Functions – global and local
  - Variables – global and local
  - The window object
  - A closure function

Variable Scope

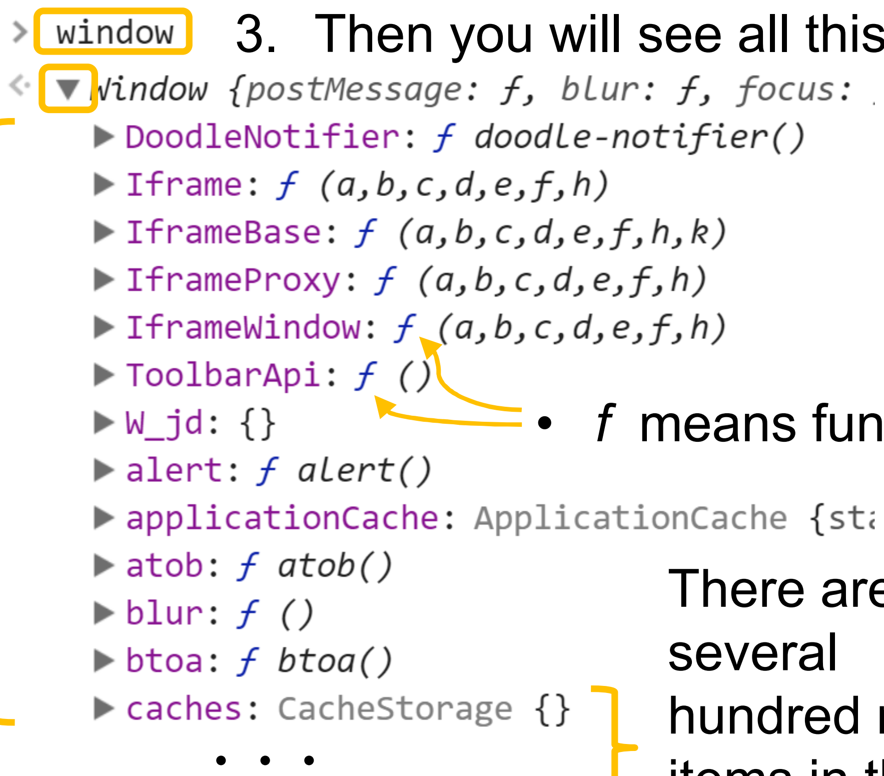
- Like other languages, in JavaScript you sometimes have different variable *scope*
- Variable scope basically means ‘where the variable works’
  - Global variables; variables created outside any function, these work everywhere
  - Local variables; variables created inside a function, these work only inside the function

The Window Object

- For JavaScript running inside a browser,
  the top level object is called `window`
- The `window` object actually means the browser itself
- A lot of the things that you used before, such as `document` and `console`, are in this `window` object
- Quick reminders:
  - `document.getElementById( ... )` searches for something
  - `console.log( ... )` shows information in the console

Showing the Window Object

1. Type this:
2. Select this:
3. Then you will see all this:



```
> window
< ▼ Window {postMessage: f, blur: f, focus:
    ▶ DoodleNotifier: f doodle-notifier()
    ▶ Iframe: f (a,b,c,d,e,f,h)
    ▶ IframeBase: f (a,b,c,d,e,f,h,k)
    ▶ IframeProxy: f (a,b,c,d,e,f,h)
    ▶ IframeWindow: f (a,b,c,d,e,f,h)
    ▶ ToolbarApi: f ()
    ▶ W_jd: {}
    ▶ alert: f alert()
    ▶ applicationCache: ApplicationCache {st
    ▶ atob: f atob()
    ▶ blur: f ()
    ▶ btoa: f btoa()
    ▶ caches: CacheStorage {}
          . . .
```

- *f* means function

There are several hundred more items in the window object

- You can see the content of the `window` object in the console window:
- Quick shortcut to show the console panel:
  Ctrl-Shift-J (PC)
  Command+Option+J (Mac)

Global Variables

- Any global variables that you create inside the browser are properties of the global window object

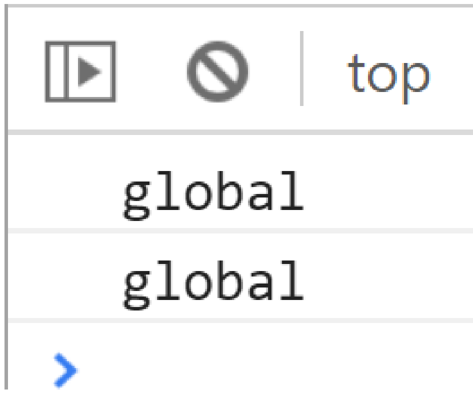
```
var myScope = "global";

window.onload = function() {
    console.log(window.myScope);
    console.log(myScope);
}
```

This is one way to set up what happens when the web page is loaded

Both these lines of code use the variable shown in the first line of code

The result of loading the page and running the code:



```
global
global
```

window.myScope means the myScope variable under the scope of window

Showing a Global Function

- It's the same for some functions

This is a global function

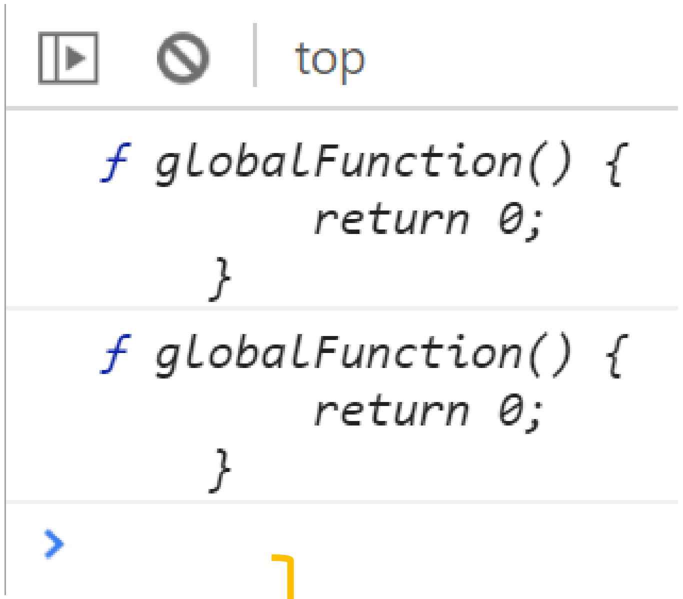
```
function globalFunction() {
    return 0;
}

window.onload = function() {
    console.log(window.globalFunction);
    console.log(globalFunction);
}
```

These both refer to the function shown above

This sets up what happens when the web page is loaded

The result of loading the page and running the code:



```
ƒ globalFunction() {
        return 0;
    }
ƒ globalFunction() {
        return 0;
    }
>
```

Making a JavaScript Variable

- To create a variable you can use `var` e.g.:

```
var a = 10;
```

- Or you can simply start using the variable without using any `var` , and it will be automatically created by JavaScript:

```
a = 10;
```

不加var的话就永远都是在glocal scope下操作变量

- But you have to understand the difference – next slide

Using and Not Using Var

- If you use `var` inside a function it will be a local variable:

```
function demo() {
    var a = 10;    // a local variable
}
```

- If you don't use `var` to create a variable before using it, that variable will be a **global variable**!

```
function demo() {
    a = 10;        // a global variable
}
```

Function Inside a Function

- You can also define functions inside other functions
- A function inside a function is called an ‘inner’ function
- For any inner function, it can access global variables and the local variables of the inner function (as usual)
- In JavaScript, an inner function can also access the local variables of its parent function – see the next few slides

Closure

- A *closure* is the name of an inner function
- In JavaScript, inner functions have access to the scope that is 'above' them

```
function greeting(name) {
    return function () {
        console.log("Hi " +
                    name + "!");
    };
}
```

This function is an inner function, which is called a closure

The inner function can access the variables of the parent function

Using a Closure

- One way to use a closure is to run the outer function i.e.:

use a variable to refer to an inner function

```
var sayHi = greeting("Dave");
```

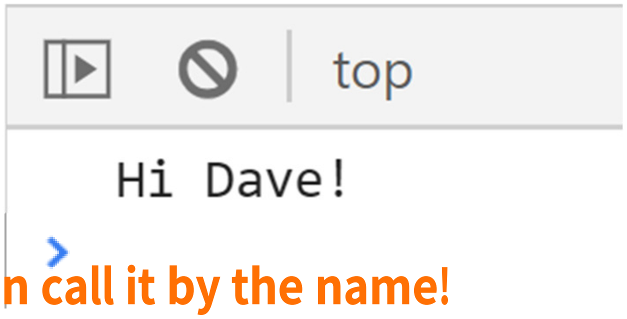
This uses the code shown on the previous slide

create了一个新的function，即便用的var，但是使用它的时候要用function call的形式

- You can then use the inner function as a normal function:

```
sayHi();
```

The result of running the code:



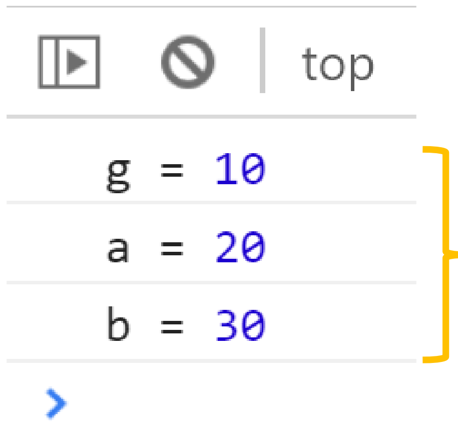
```
Hi Dave!
```

once the inner function gets a name, you can call it by the name!

Variable Scopes for Inner Functions

- Here is another example showing the scopes of variables that an inner function can see

The result of loading the page and running the code:



```
g = 10
a = 20
b = 30
```

```
var g = 10;

function dosomething() {
    var a = 20;

    function nested() {
        var b = 30;
        console.log("g =", g);
        console.log("a =", a);
        console.log("b =", b);
    }
    nested();
}

window.onload = function() {
    dosomething();
}
```

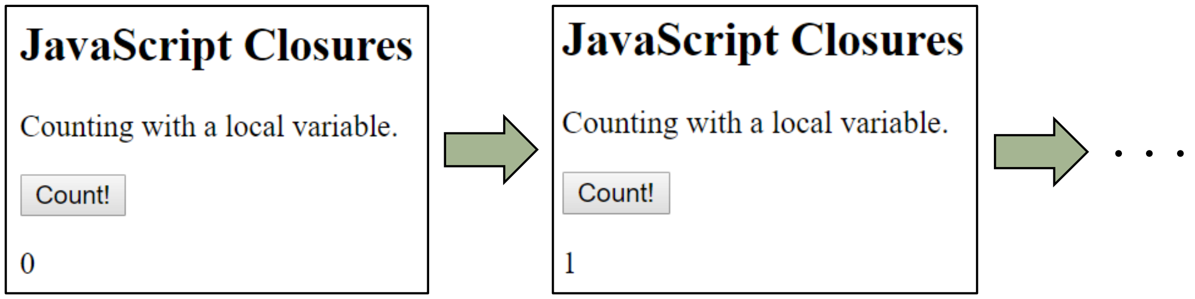
Execute the function defined above

```
<button type="button" onclick="myFunction()">Count!</button>

<p id="demo">0</p>

<script>
var add = (function () {
    var counter = 0;
    return function () { counter += 1; return counter;}
})();

function myFunction(){
    document.getElementById("demo").innerHTML = add();
}
</script>
```



JavaScript Closures

Counting with a local variable.

Count!

0

JavaScript Closures

Counting with a local variable.

Count!

1

. . .

This part only gets executed once

create the variable counter once, and then returns a function.

Starting from the second execution, this line only executes the inner function.

A Counting Example

- With this structure the variable `counter` can only be changed by `add()` , which protects `counter`
- That is good programming